Trusted Computing Engineering for Resource Constrained Embedded Systems Applications

Deliverable 5.1

Formal Validation Approach

Project:	**TERESA**
Project Number:	**IST-248410**
Deliverable:	**D5.1**
Title:	**Formal Validation Approach**
Version:	**v1.2**
Confidentiality:	**Confidential**
Editors:	**Andreas Fuchs, Sigrid Gürgens**
Date:	**Dec 6, 2010**

Part of the Seventh Framework Program
Funded by the EC - DG INFSO

Control Sheet

Version History			
Version number	Date	Main author	Summary of changes
v0.1	4 Nov 2010	Sigi Gürgens (SIT)	Initial version
v1.0	23 Nov 2010	Andreas Fuchs, Sigi Gürgens (SIT)	almost final version
v1.1	26 Nov 2010	Andreas Fuchs, Sigi Gürgens (SIT)	version for internal review
v1.2	6 Dec 2010	Andreas Fuchs, Sigi Gürgens (SIT)	incorporated reviewer comments

Approval		
	Name	Date
Prepared	S. Gürgens	26 Nov 2010
Reviewed	All Project Partners	3 Dec 2010
Authorized	Antonio Kung	6 Dec 2010

Circulation	
Recipient	Date of submission
Project Partners	6 Dec 2010
European Commission	6 Dec 2010

Contents

List of Figures

1 Executive Summary

This document introduces the approach for formal validation of the TERESA S&D patterns. The approach is based on the Security Modeling Framework SeMF developed by Fraunhofer SIT which is explained in detail in Section 3. This section also introduces two different approaches for proving S&D properties for system models. Section 4 then shows how the first approach is used for formal validation of TERESA patterns, while Section 5 shows the application of the second approach for proving the correctness of pattern implementation.

2 Introduction

The objective of TERESA is to define, demonstrate and validate an engineering process for resource constrained embedded systems (RCES). One of the key ideas of the process is an S&D Pattern Repository. TERESA S&D Patterns are a means to capture security expert knowledge and make it available to application developers. A TERESA S&D Pattern contains all information relevant for a specific S&D solution: a static and a dynamic model of the solution itself, the S&D properties it provides, assumptions the environment must satisfy in order for the solution to provide the properties, the interfaces that are needed in order to integrate the solution into an application, and other useful information. Before a pattern is made available to application developers by including it in the Pattern Repository it is formally verified, i.e. on the basis of the information contained in the pattern, a proof is conducted that this solution indeed provides the S&D properties specified in the pattern when integrated into a system that satisfies the assumptions specified in the pattern. Once an application developer has identified S&D requirements that need to be fulfilled by his/her application, he/she can search the Pattern Repository for respective S&D Patterns that meet these requirements. From a security point of view only those patterns in the list that contain assumptions on the environment that are met by the particular application are appropriate for being integrated into the application. Other functional restrictions (concerning time or memory constraints, etc) may reduce the number of applicable patterns further.

This document introduces an approach for the verification of both the S&D Patterns and their implementation, i.e. their integration into a specific application. The approach is based on the Security Modeling Framework SeMF developed by Fraunhofer SIT. It provides a methodology for modeling a pattern system and for proving that in this model the solution specified in the pattern provides certain properties, given that the assumptions specified in the pattern hold in the system. It further provides a method to verify that for a concrete implementation of the pattern, i.e. when integrating the pattern into a concrete application, these assumptions are indeed satisfied.

The document is organized as follows. The next Section gives a detailed overview of the formal semantics of SeMF, introduces the most important security properties already defined, provides predicates as specific instantiations of these security properties, and introduces two different methods for proving properties of systems, Security Building Blocks and property preserving homomorphisms. Section 4 then explains the application of Security Building Blocks in order to prove security properties of S&D Patterns, while Section 5 shows how property preserving homomorphisms can be used to prove the correctness of pattern implementation. Finally, the document is concluded with an outlook on future work in WP5.

3 The Security Modeling Framework SeMF

3.1 Introduction

We will now describe in detail the Security Modeling Framework SeMF developed by Fraunhofer SIT that will be the basis for our validation approach in TERESA. In this framework, systems are specified in terms of sequences of actions, while the security properties of systems are specified as specific constraints regarding which sequences of actions can or can not occur. Security properties can be specified regardless of any specific abstraction level. SeMF provides two ways to prove that a specific system satisfies a specific property: By way of security preserving homomorphisms, and by using so-called Security Building Blocks (SeBBs). Both approaches will be described in Sections 3.6 and 3.7, respectively.

The basis SeMF elements are result of previous work, e.g. within the European project SERENITY [1]. In the European project EVITA [2], we defined specific instantiations of properties and developed the SeBB based verification approach. Within TERESA we will specify and use both property instantiations and Security Building Blocks that are particularly relevant for embedded systems.

The following section gives first a brief overview of formal languages as a means to system specification and then introduces our main additional concepts, an agent's *local view* and an agent's *initial knowledge*. We then give the formal definitions of the most important security properties based on these concepts (e.g. authenticity, confidentiality), and some instantiations relevant in TERESA. It is forseen that further property definitions and instantiations will be necessary in order to capture all S&D properties relevant in TERESA. In the last two sections we finally introduce our concepts of property preserving homomorphisms and Security Building Blocks (SeBBs).

3.2 Formal Languages

Properties of a concurrent system in the sense of Alpern and Schneider [3] are defined as sets of sequences of states. Similarly, the *behavior* B of a discrete system can be formally described by the set of its possible sequences of actions (traces). Therefore $B \subseteq \Sigma^*$ holds where Σ is the set of all actions of the system, and Σ^* is the set of all finite sequences of elements of Σ, including the empty sequence denoted by ε. This terminology originates from the theory of formal languages, where Σ is called the alphabet, the elements of Σ are called letters, the elements of Σ^* are referred to as words and the subsets of Σ^* as formal languages. Words can be composed: if u and v are words, then uv is also a word. This operation is called the *concatenation*; especially $\varepsilon u = u\varepsilon = u$. A word u is called a *prefix* of

a word v if there is a word x such that $v = ux$. The set of all prefixes of a word u is denoted by pre(u); $\varepsilon \in$ pre(u) holds for every word u. We denote the set of letters in a word u by $alph(u)$ and the number of occurences of any action of a set Γ in a word u by $card(\Gamma, u)$. If Γ consists of only one action a, we simply say $card(a, \omega)$.

Formal languages that describe system behavior have the characteristic that pre$(u) \subseteq B$ holds for every word $u \in B$. Such languages are called *prefix closed*. System behavior is thus described by prefix closed formal languages.

The set of all possible continuations of a word $u \in B$ is formally expressed by the *left quotient* $u^{-1}(B) = \{y \in \Sigma^* \mid uy \in B\}$.

Different formal models of the same application/system are partially ordered with respect to different levels of abstraction. Formally, abstractions are described by so called alphabetic language homomorphisms. These are mappings $h^* : \Sigma^* \longrightarrow \Sigma'^*$ with $h^*(xy) = h^*(x)h^*(y)$, $h^*(\varepsilon) = \varepsilon$ and $h^*(\Sigma) \subseteq \Sigma' \cup \{\varepsilon\}$. So they are uniquely defined by corresponding mappings $h : \Sigma \longrightarrow \Sigma' \cup \{\varepsilon\}$. In the following we denote both the mapping h and the homomorphism h^* by h. These homomorphisms map action sequences of a finer abstraction level to action sequences of a more abstract level.

Classical liveness and safety properties can easily be specified for such a system using well known formalizations. For security properties, we need to extend the system model by taking into account the agents' view of the system and agents' knowledge about the global system behavior.

3.3 Agents' Local Views and Initial Knowledges

Security properties can only be satisfied relative to particular sets of underlying system assumptions. Examples include assumptions on cryptographic algorithms, secure storage, trust in the correct behavior of agents or reliable data transfer. Relatively small changes in these assumptions can result in huge differences concerning satisfaction of security properties. Every model for secure systems must address these issues. However, most existing models rely on a fixed set of underlying assumptions (see for example [4] and [5]). Most of these assumptions are often implicitly given by particular properties of the model framework. Thus, it is very hard to verify whether a particular implementation actually satisfies all of these assumptions. Further, imprecise security assumptions might result in correct but useless security proofs and finally in insecure implementations. Therefore, a model for secure systems needs to provide the means to accurately specify underlying system assumptions in a flexible way.

In order to provide the required flexibility, SeMF extends the system specification by two components: *agents' knowledge* about the global system behavior and *agents' view*. The knowledge about the system consists of all traces that an agent initially considers possible, i.e. all traces that do not violate any system assumptions, and the view of an agent specifies which parts of the system behavior the agent can actually see. In the following paragraphs, these two components and their relations are explained in detail.

3.3.1 Agents' Knowledge about the Global System Behavior

For any agent P its knowledge $W_P \subseteq \Sigma^*$ about the global system behavior is considered to be part of the system specification.

We may assume for example that a message that was received must have been sent before. Thus an agent's W_P will contain only those sequences of actions in which a message is first sent and then received. All sequences of actions included in W_P in which a digital signature is received and verified by using some agent Q's public key will contain an action where Q generated this signature.

Care must be taken when specifying the sets W_P for all agents P in order not to specify properties that are desirable but not guaranteed by verified system assumptions. In a setting for example where we assume one time passwords are used, if P trusts Q, W_P contains only those sequences of actions in which Q sends a certain password only once. However, if Q cannot be trusted, W_P will also contain sequences of actions in which Q sends a password more than once.

The specification of the desired system behavior generally does not include behavior of malicious agents which has to be taken into account in open systems. An approach which is frequently used for the security analysis of cryptographic protocols is to extend the system specification by explicit specification of malicious behavior. However, in general malicious behavior is not previously known and one may not be able to adequately specify all possible actions of dishonest agents. In our approach, the explicit specification of agents' knowledge about system and environment allows to discard explicit specification of malicious behavior. Every behavior which is not explicitly excluded by some W_P is allowed. Denoting a system containing malicious behavior by B and the correct system behavior by B_C, we assume $B_C \subseteq B \subseteq \Sigma^*$. We further assume $B \subseteq W_P$, i.e. every agent considers the system behavior to be possible, as reasoning within SeMF primarily targets the validation and verification of security properties in terms of positive formulations, i.e. assurances the agents of the system may have. Other approaches that deal with malfunction, misassumptions and attacker models can not rely on this assumption.

Security properties can now be defined relative to W_P. The relation between the system behavior without malicious actions B_C, the system behavior including malicious actions B, and W_P is graphically shown in Figure 3.1.

3.3.2 Agents' View of the Global System Behavior

The set W_P describes what P knows initially. However, in a running system P can learn from actions that have occured. Satisfaction of security properties obviously also depends on what agents are able to learn. After a sequence of actions $\omega \in B$ has happened, every agent can use its *local view* of ω to determine the sequences of actions it considers to have possibly happened after ω has happened. In order to determine what is the local view of an agent, we first assign every action to exactly one agent. Thus $\Sigma = \dot{\bigcup}_{P \in \mathbb{P}} \Sigma_{/P}$ (where $\Sigma_{/P}$ denotes all actions performed by agent P, and $\dot{\bigcup}$ denotes the disjoint union). The homomorphism

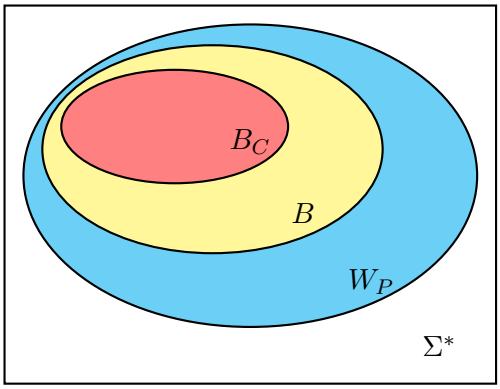

Figure 3.1: System behavior and W_P

$\pi_P : \Sigma^* \to \Sigma^*{}_{/P}$ defined by $\pi_P(x) = x$ if $x \in \Sigma_{/P}$ and $\pi_P(x) = \varepsilon$ if $x \in \Sigma \setminus \Sigma_{/P}$ formalizes the assignment of actions to agents and is called the $projection$ on P.

The projection π_P is the correct representation of P's view of the system if all information about an action $x \in \Sigma_{/P}$ is available for agent P and P can only see its own actions. In this case P's local view of the sequence of actions $\omega = send(P, m1)rec(Q, m1)$ for example is $send(P, m1)$. However, P's view may be finer. For example it may additionally note a message that was sent over a network bus without being able to see who sent it, in which case P's local view of $send(sender, message)$ e.g. is $send(message)$. P's local view may also be coarser than π_P. In a system the actions of which are represented by a triple (*global state, transition label, global successor state*), although seeing its own actions, P will not be able to see the other agents' states. Thus, we generally denote the local view of an agent P on Σ by $\lambda_P : \Sigma^* \to \Sigma_P^*$. The local views of all agents together contain all information about the system behavior B.

For a sequence of actions $\omega \in B$ and agent $P \in \mathbb{P}$, $\lambda_P^{-1}(\lambda_P(\omega)) \subseteq \Sigma^*$ is the set of all sequences that look exactly the same from P's local view after ω has happened. In the above network bus example all actions in $\{send(sender_1, message), send(sender_2, message), \ldots\}$ look identical for P. Depending on its knowledge about the system S, underlying security mechanisms and system assumptions, P does not consider all sequences in $\lambda_P^{-1}(\lambda_P(\omega))$ possible. Thus it can use its initial knowledge to reduce this set: $\lambda_P^{-1}(\lambda_P(\omega)) \cap W_P$ describes all sequences of actions P considers to have possibly happened when ω has happened. This set is similar to the possible worlds semantics that have been defined for authentication logics in the context of cryptographic protocols [6, 7]. Our notion is more general because for authentication logics λ_P and W_P are fixed for all systems, whereas in our approach they can be defined differently for different systems. The knowledge of P relative to a sequence of actions ω is graphically shown in Figure 3.2.

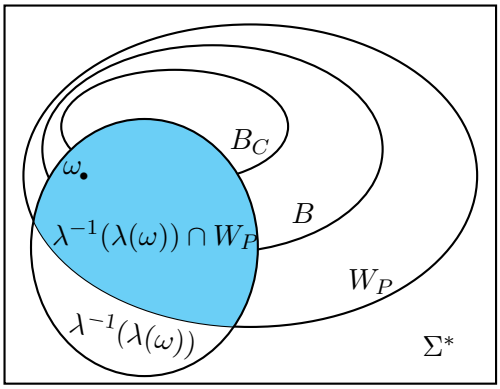

Figure 3.2: P's world after ω has happened

3.4 Basic Property Definitions

The concepts introduced in the previous section will now be used to define the most important security properties. Once the actual verification work starts, more property definitions will be added as necessary for TERESA.

3.4.1 Authenticity

Authenticity can be seen as the assurance that a particular action has occurred in the past.[1]

Since usually authenticity of some action for a certain agent is required, the definition has to refer in some way to the agent. Thus we call a particular action *a* authentic for an agent *P* if in all sequences that *P* considers to have possibly happened after a sequence of actions ω has happened, some time in the past a must have happened. By extending this definition to a set of actions Γ being authentic for P if one of the actions in Γ is authentic for P we gain the flexibility that P does not necessarily need to know all parameters of the authentic action. For example, a message may consist of one part protected by a digital signature and another irrelevant part without protection. Then, the recipient can know that the signer has sent a message containing the signature, but the rest of the message is not authentic. Therefore, in this case, Γ comprises all messages containing the relevant signature and arbitrary other message parts.

The formal definition for authenticity is as follows.

Definition 1 (Authenticity)**.** *A set of actions $\Gamma \subseteq \Sigma$ is authentic for $P \in \mathbb{P}$ after a sequence of actions $\omega \in B$ with respect to W_P if $alph(x) \cap \Gamma \neq \emptyset$ for all $x \in \lambda_P^{-1}(\lambda_P(\omega)) \cap W_P$.*

[1] Please note that in order to achieve *authentication* one additionally needs assurance about the time of the occurrence of the action (see Section 3.4.2 *authenticity with respect to a phase*).

3.4.2 Authenticity with Respect to a Phase

In many cases it is not only necessary to know *who* has performed a particular action, but also the specific *"time"* of the action is important. As our specification is discrete and does not model any real time properties, time is modeled in terms of relations between actions in a sequence. However, an explicit model of discrete time can be easily included in the model.

We use the definition of a $phase$ provided in [8]. A phase $V \subset \Sigma^*$ is a prefix closed language consisting only of words which, as long as they are not maximal in V, show the same continuation behavior within V as within B. Maximal words in a phase are those that lead out of the phase, i.e. those $v \in V$ for which exists $\omega, u \in B$ with $\omega = uv$ such that for all $a \in \Sigma$ with $\omega a \in B$ holds $va \notin V$.

Definition 2. *Let $B \subseteq \Sigma^*$ be a system. A prefix closed language $V \subset \Sigma^*$ is a phase in B if the following holds:*

1. $V \cap \Sigma \neq \emptyset$
2. $\forall \omega \in B$ *with* $\omega = uv$ *and* $v \in V \setminus (max(V) \cup \{\varepsilon\})$ *holds:* $\omega^{-1}(B) \cap \Sigma = v^{-1}(V) \cap \Sigma$

Thus a phase as defined above is essentially a part of the system behavior that is closed with respect to concatenation. In analogy to the maximal words of a phase we define the minimal words of a phase as all $v \in V$ with $|alph(v)| = 1$.

A phase can be a very complex construct. However, in many cases phases are of interest that can be defined by their starting and ending actions. Since an action can occur more than once in a word, it is not sufficient to identify the starting and terminating actions for determining where a particular phase starts and where it ends. The following definition takes this into account.

Definition 3. *Let $s_1, \ldots, s_k, t_1, \ldots, t_l \in \Sigma$ be actions. Then $V(\{s_1, \ldots, s_k\}, \{t_1(j_1), \ldots, t_l(j_l)\})$ (with $j_1, \ldots, j_l \in$ **N**) defines a phase in B that starts with actions $s_1, \ldots, s_k$ and terminates with actions $t_1, \ldots, t_l$ in the following sense:*

- *For all $\omega \in B$ for which exists $s_j \in \{s_1, \ldots, s_k\}$ such that $\omega s_j \in B$ follows $s_j \in V(\{s_1, \ldots, s_k\}, \{t_1(j_1), \ldots, t_l(j_l)\})$ (i.e. s_j is minimal in $V(\{s_1, \ldots, s_k\}, \{t_1(j_1), \ldots, t_l(j_l)\})$).*
- *For all $\omega \in B$ for which exists $t_i(j_i) \in \{t_1(j_1), \ldots, t_l(j_l)\}$ and $u, v \in \Sigma^*$ with $\omega = uvt_i$, $vt_i \in V(\{s_1, \ldots, s_k\}, \{t_1(j_1), \ldots, t_l(j_l)\})$, and $card(t_i, vt_i) = j_i$ follows that vt_i is maximal in $V(\{s_1, \ldots, s_k\}, \{t_1(j_1), \ldots, t_l(j_l)\})$.*

In the above definition, the starting action(s) need to be fixed so that starting from these the terminating actions occuring in the phase can be counted in order to identify those ones that actually terminate the phase. In the following definition, we conversely fix the termination action(s) and count the number of occurences of the starting action(s) backwards in the phase to identify the actual start(s) of the phase:

Definition 4. *Let $s_1, \ldots, s_k, t_1, \ldots, t_l \in \Sigma$ be actions. Then $V(\{s_1(i_1), \ldots, s_k(i_k)\}, \{t_1, \ldots, t_l\}) \subseteq B$ (with $i_1, \ldots, i_l \in$ **N**) defines a phase in B that starts with actions $s_1, \ldots, s_k$ and terminates with actions $t_1, \ldots, t_l$ in the following sense:*

- *For all* $\omega \in B$ *for which exists* $s_j(i_j) \in \{s_1(i_1), \ldots, s_k(i_k)\}$ *such that* $\omega s_j \in B$ *follows* $s_j \in V(\{s_1(i_1), \ldots, s_k(i_k)\}, \{t_1, \ldots, t_l\})$ *(i.e.* s_j *is minimal in* $V(\{s_1(i_1), \ldots, s_k(i_k)\}, \{t_1, \ldots, t_l\})$*) and* $card(s_j, v) = i_j$ *for all maximal words* $v \in V(\{s_1(i_1), \ldots, s_k(i_k)\}, \{t_1, \ldots, t_l\})$.
- *For all* $\omega \in B$ *for which exists* $t_i \in \{t_1, \ldots, t_l\}$ *and* $u, v \in \Sigma^*$ *with* $\omega = uvt_i$, $vt_i \in V(\{s_1(i_1), \ldots, s_k(i_k)\}, \{t_1, \ldots, t_l\})$, *follows that* vt_i *is maximal in* $V(\{s_1(i_1), \ldots, s_k(i_k)\}, \{t_1, \ldots, t_l\})$.

In some cases we are not interested in how often each of the ending actions occurs within the phase but we want to fix the number of occurences of any of them.

Definition 5. *Let* $s_1, \ldots, s_k, t_1, \ldots, t_l \in \Sigma$ *be actions. Then* $V(\{s_1, \ldots, s_k\}, \{t_1, \ldots, t_l\}(j)) \subseteq B$ *(with* $j \in$ **N***) defines a phase in* B *that starts with actions* $s_1, \ldots, s_k$ *and terminates with actions* $t_1, \ldots, t_l$ *in the following sense:*

- *For all* $\omega \in B$ *for which exists* $s_j \in \{s_1, \ldots, s_k\}$ *such that* $\omega s_j \in B$ *follows* $s_j \in V(\{s_1, \ldots, s_k\}, \{t_1, \ldots, t_l\}(j))$ *(i.e.* s_j *is minimal in* $V(\{s_1, \ldots, s_k\}, \{t_1, \ldots, t_l\}(j))$ *).*
- *For all* $\omega \in B$ *for which exists* $t_i \in \{t_1, \ldots, t_l\}$ *and* $u, v \in \Sigma^*$ *with* $\omega = uvt_i$, $vt_i \in V(\{s_1, \ldots, s_k\}, \{t_1, \ldots, t_l\}(j))$, *and* $card(\{t_1, \ldots, t_l\}, vt_i) = j$ *follows that* vt_i *is maximal in* $V(\{s_1, \ldots, s_k\}, \{t_1, \ldots, t_l\}(j))$.

While in many cases we are able to identify the last action(s) of a phase, in some cases we may know only the first action(s) that occur outside the phase. The next definition allows to specify a phase using these actions.

Definition 6. *Let* $s_1, \ldots, s_k, t_1, \ldots, t_l \in \Sigma$ *be actions, and let* $p_1, \ldots, p_l$ *be parameters with values* in *and* ex. *Then* $V(\{s_1, \ldots, s_k\}, \{t_1(j_1, p_1), \ldots, t_l(j_l, p_l)\}) \subseteq B$ *(with* $j_1, \ldots, j_l \in$ **N***) defines a phase in* B *that starts with actions* $s_1, \ldots, s_k$ *and terminates with actions* $t_1, \ldots, t_l$ *in the following sense:*

- *For all* $\omega \in B$ *for which exists* $s_j \in \{s_1, \ldots, s_k\}$ *such that* $\omega s_j \in B$ *follows* $s_j \in V(\{s_1, \ldots, s_k\}, \{t_1(j_1, p_1), \ldots, t_l(j_l, p_l)\})$ *(i.e.* s_j *is minimal in* $V(\{s_1, \ldots, s_k\}, \{t_1(j_1, p_1), \ldots, t_l(j_l, p_l)\})$.
- *For all* $\omega \in B$ *for which exists* $t_i(j_i, in) \in \{t_1(j_1, p_1), \ldots, t_l(j_l, p_l)\}$ *and* $u, v \in \Sigma^*$ *with* $\omega = uvt_i$, $vt_i \in V(\{s_1, \ldots, s_k\}, \{t_1(j_1, p_1), \ldots, t_l(j_l, p_l)\})$, *and* $card(t_i, vt_i) = j_i$ *follows that* vt_i *is maximal in* $V(\{s_1, \ldots, s_k\}, \{t_1(j_1), \ldots, t_l(j_l)\})$.
- *For all* $\omega \in B$ *for which exists* $t_i(j_i, ex) \in \{t_1(j_1, p_1), \ldots, t_l(j_l, p_l)\}$ *and* $u, v \in \Sigma^*$ *with* $\omega = uvt_i$, $v \in V(\{s_1, \ldots, s_k\}, \{t_1(j_1, p_1), \ldots, t_l(j_l, p_l)\})$, *and* $card(t_i, v) = j_i$ *follows that* v *is maximal in* $V(\{s_1, \ldots, s_k\}, \{t_1(j_1), \ldots, t_l(j_l)\})$.

In other words, if the number of occurences of a terminating action is accompanied by the parameter in, the action is part of the phase. If it is accompanied by ex, it is the first action after the phase's termination.

Definition 7 (Authenticity with respect to a phase)**.** *A set of actions $\Gamma \subseteq \Sigma$ is authentic for agent $P \in \mathbb{P}$ after a sequence of actions $x \in B$ with respect to W_P and a phase V if it is authentic for P after x and for all $y \in \lambda_P^{-1}(\lambda_P(x)) \cap W_P)$ exists $u, v, w \in \Sigma^*$ such that $y = uvw$ and $v \in V$ and $alph(v) \cap \Gamma \neq \emptyset$. Γ is currently authentic for P after x if $w = \varepsilon$.*

The above definition can be used to specify the security property provided by e.g. an SSL channel: data origin authenticity and freshness. The concept of a phase is used to model the duration of the SSL channel: The phase starts with the establishment of the channel by receiving the last handshake message (from the server's point of view), and ends by terminating the SSL channel (e.g. by deleting the respective session key). Each message exchanged on the channel is both performed within the phase represented by the channel and authentically generated by the server and the client, respectively.

3.4.3 Proof of Authenticity

Some actions do not only require authenticity but also need to provide a proof of authenticity (non-repudiation of receipt etc.). Usually, some evidence of the occurrence of a particular action is provided as "proof". Different types of proofs are possible: transferable, non-transferable, proofs that can get lost, etc. The following describes transferable proofs with the additional assumption that one cannot lose the proofs. Other requirements can be defined in a similar way.

If agent Q owns a proof of authenticity for a set Γ of actions we assume it can send this proof to other agents, which in turn can receive the proof and be convinced of Γ's authenticity. In the following definition the set ΓP denotes actions that provide agents with proofs about the authenticity of Γ. If agent Q has executed an action from ΓP then Γ is authentic for Q and Q can forward the proof to any other agent using actions in ΓS.

Definition 8 (Proof of authenticity)**.** *A pair $(\Gamma S, \Gamma P)$ with $\Gamma S \subseteq \Sigma$ and $\Gamma P \subseteq \Sigma$ is a pair of sets of proof actions of authenticity for a set $\Gamma \subseteq \Sigma$ on B with respect to $(W_Q)_{Q \in \mathbb{P}}$ if for all $\omega \in B$ and for all $Q \in \mathbb{P}$ with $alph(\pi_Q(\omega)) \cap \Gamma P \neq \emptyset$ the following holds:*

1. *For Q the set Γ is authentic after ω and*
2. *for each $R \in \mathbb{P}$ there exist actions $a \in \Sigma_{/Q} \cap \Gamma S$ and $b \in \Sigma_{/R} \cap \Gamma P$ with $\omega ab \in B$.*

Agent $Q \in \mathbb{P}$ can give proof of authenticity of $\Gamma \subseteq \Sigma$ after a sequence of actions $\omega \in B$ if 1 and 2 hold.

In addition to agent Q and set of actions Γ we have to specify the set of actions after which proof of authenticity for Q shall hold.

Note that we do not specify actions for forwarding of proofs and receiving of forwarded proofs in our example because such actions might happen outside the specified system. Thus these actions are not explicitly included in the properties described below. However, in order to prove that these properties hold, the forwarding and receiving of proofs can easily be added.

3.4.4 Confidentiality

Confidentiality is required for data occurring in different types of actions. Among others these can be actions concerned with sending or receiving data or manipulating data stored on a particular device. We may also want to keep the agent acting confidential for privacy reasons. An adequate notion of confidentiality therefore has to provide the flexibility to define confidentiality for arbitrary parameters of the actions. The notion of *parameter-confidentiality* presented in [9] provides this flexibility.

Parameter Confidentiality essentially captures the following: Consider agent R has monitored a sequence of actions ω, and in some of these actions a parameter p occurs with a specific value. Then R must not be able to distinguish this sequence from any other sequence in which the parameter occurs with different values, even if knowing all possible values.

Consider as an example a system consisting of three active nodes and an end user. $Sensor_1$ and $Sensor_2$ are nodes deployed somewhere in the system. They perform measurements (e.g. measure the temperature inside and outside a house) and send the resulting data over the network. $Displ$ is the third node of the system. It receives data from the network and displays them to the end user *User* (e.g. the owner of the house). Let us assume that the communication channel between the display and the sensors is secured using a digital signature scheme. Hence we may want to require that a sensor's signature key shall be confidential to all other agents. Now assume that $Sensor_1$ generates and sends a signature and $Displ$ receives, verifies and accepts this signature, modeled for example by $send(Sensor_1, data, privK_1, sig_j(data))\, recv(Displ, data, pubK_1, sig_j(data))$. When monitoring this sequence, $Sensor_2$ and the display shall not be able to deduce the actual private key that was used even if knowing the key's length and thus all possible values. This property can be expressed with our notion of parameter confidentiality and we will explain it further using the requirement of $privK_1$ to be confidential to the display.

Various aspects are included in this definition. First, one has to consider $Displ$'s view of the sequence ω it has monitored and thus the set $\lambda^{-1}_{Displ}(\lambda_{Displ}(\omega))$ of sequences that are, from $Displ$'s view, identical. We assume a network bus connection which allows all agents to see their own actions and the data and signature of actions performed by other agents, but not the sender and the signature key used. The display's local view λ_{Displ} of the sequence above is then $\lambda_{Displ}(send(Sensor_1, data, privK_1, sig_j(data))\ recv(Displ, data, pubK_1, sig_j(data))) = send(data, sig_j)\ recv(Displ, data, pubK_1, sig_j(data))$. The set $\lambda^{-1}_{Displ}(\lambda_{Displ}(send(Sensor_1, data, privK_1, sig_j(data))\, recv(Displ, data, pubK_1, sig_j(data))))$ of the sequences of actions that are from the display's view identical consists of sequences $send(P, data, privK_i, sig_j(data'))$ $recv(Displ, data, pubK_1, sig_j(data))$ with $privK_i$ denoting possible key values, P denoting any possible sender, and $data, data'$ denoting possible values of the parameter $data$.

Second, $Displ$ can discard some of the sequences from this set, depending on its knowledge of the system and the system assumptions, all formalized in W_{Displ}. There may for example exist interdependencies between parameters in different actions, such as the public key used for a signature's verification determining the private key used for its generation. In consequence, $Displ$ considers only those sequences of actions possible in which

its own receive action, including verification and acceptance of the signature, is always preceded by a signature generation and send action which uses the matching signature key and data. So the set of sequences $Displ$ considers to have possibly happened after ω has happened is reduced to $\lambda^{-1}_{Displ}(\lambda_{Displ}(\omega)) \cap W_{Displ}$. In the example this set consists of sequences $send(P, data, privK_i, sig_j(data))\, recv(Displ, data, pubK_1, sig_j(data))$, with $Displ$ knowing that there is a relation between the public key $pubK_1$ used for verification of the signature and $privK_i$ but not being able to deduce the actual value of $privK_i$.

Third, those actions have to be identified in which the respective parameter(s) shall be confidential, or in other words, the actions from which a monitoring agent can possibly learn the parameter's value. Usually many actions are independent from these and do not influence confidentiality, thus need not be considered. In the example, it is the send action that contains $privK_i$ and from which $Displ$ can learn its value but the sensing and display actions are not relevant regarding the confidentiality of $Sensor_1$'s signature key. We formalize this by using a homomorphism μ that maps all actions of interest for the particular confidentiality property onto what we call the *action type* while at the same time extracting the parameter to be confidential, and that maps all other actions onto the empty word. In the example, we define $\mu(send(Sensor_i, data, privK_i, sig_j(data))) = (send(Sensor_i, data, sig_j), privK_i)$ and $\mu(action) = \varepsilon$ for all other actions $action$.

Essentially, parameter confidentiality is captured by requiring that for actions that shall be confidential for an agent with respect to some parameter p, all possible (combinations of) values for p occur in this set. What are the possible combinations of parameters is the fourth aspect that needs to be specified, as one may want to allow the agent to know some of the interdependencies between parameters (e.g. it may be allowed to know the relation between signature and verification key). The notion of (L, M)–Completeness captures which are the allowed dependencies within a set of sequences of actions.

We define L to be a formal language that consists of sequences of pairs *(action type,number)* where *action type* are the types of actions that are kept by μ and the numbers are used to identify those actions whose relation between the parameters is allowed to be known. We extend the example and add a further send action by $Sensor_1$, resulting in $\omega = send(Sensor_1, data', privK_1, sig_i(data'))\, send(Sensor_1, data, privK_1, sig_j(data))\, recv(Displ, data', pubK_1, sig_i(data'))\ recv(Displ, data, pubK_1, sig_j(data))$. Now $Displ$ owns the sensor's public key and thus knows (and is allowed to know) that $Sensor_1$ always uses the same private key. On the other hand, $Sensor_2$ does not own $pubK_1$, and if we assume that it does not know that $Sensor_1$ always uses the same key, even if $data$ indicates who is the message's sender, $Sensor_2$ should not be able to distinguish one send action from another. Hence for describing the confidentiality requirement of the display we assign all send actions the same number, hence L_{Displ} contains sequences $(send(Sensor_1, data, sig_i), k)(send(Sensor_1, data, sig_j), k)$. Addressing the confidentiality requirement of $Sensor_2$ we assign different numbers to the send actions which results in a different language L_{Sensor_2} that contains sequences such as $(send(Sensor_x, data, sig_i), k)(send(Sensor_x, data, sig_j), l)$. Then functions f are used to map these numbers to the set M of possible parameter values. The resulting set of action sequences contains all possible combinations of parameter values with the constraint that actions related with respect to the the parameter contain the same parameter value. Such a set of action sequences is called (L, M)–complete.

For the formal definition of (L, M)–completeness, some additional notations are needed: For $f : M \longrightarrow M'$ and $g : N \longrightarrow N'$ we define $(f, g) : M \times N \longrightarrow M' \times N'$ by $(f, g)(x, y) := (f(x), g(y))$. The identity on M is denoted by $i_M : M \longrightarrow M$, while $M^{\mathbf{N}}$ denotes the set of all mappings from **N** to M, and $p_1 : (\Sigma_t \times M) \longrightarrow \Sigma_t$ is a mapping that removes the parameters.

Definition 9. *Let $L \subseteq (\Sigma_t \times \mathbf{N})^*$ and let M be a set of parameters. A language $K \subseteq (\Sigma_t \times M)^*$ is called (L, M)–complete if*

$$K = \bigcup_{f \in M^{\mathbf{N}}} (i_{\Sigma_t}, f)(L)$$

The definition of parameter confidentiality captures all the different aspects described above:

Definition 10 (Parameter Confidentiality)**.** *Let M be a parameter set, Σ a set of actions, Σ_t a set of types, $\mu : \Sigma^* \to (\Sigma_t \times M)^*$ a homomorphism, and $L \subseteq (\Sigma_t \times \mathbf{N})^*$. Then M is parameter-confidential for agent $R \in \mathbb{P}$ with respect to (L, M)-completeness if there exists an (L, M)–complete language $K \subseteq (\Sigma_t \times M)^*$ with $K \supseteq \mu(W_R)$ such that for each $\omega \in B$ holds*

$$\mu(\lambda_R^{-1}(\lambda_R(\omega)) \cap W_R) \supseteq p_1^{-1}(p_1(\mu(\lambda_R^{-1}(\lambda_R(\omega)) \cap W_R))) \cap K$$

Here $p_1^{-1} \circ p_1$ first removes and then adds again the parameters that shall be confidential, i.e. constructs all possible value combinations. (L, M)–completeness of K captures exactly that we require R to consider all combinations of parameter values possible except for those that we allow to be disregarded. Hence the right hand side of the inequality constitutes all sequences of actions we require R to consider to have possibly happened after ω has happened, while the left hand side constitutes those sequences R actually does consider to have possibly happened. For further explanations we refer the reader to [9, 10].

3.4.5 Enforcing system behavior

Some security requirements are concerned with enforcing specific system behavior, i.e. with not allowing certain other system behavior. Requiring authenticity of action a whenever action b has happened is one particular instantiation of enforcing system behavior. However, other required behavior needs a more general definition.

Definition 11 (enforce-behavior)**.** *For an alphabet Σ, let $L \subseteq \Sigma^*$ describe particular requirements and $B \subseteq \Sigma^*$ be the actual system. We say that L enforces its behavior on B, denoted by* enforce-behavior(L,B)*, if $B \subseteq L$.*

3.4.6 Trust

Trust assumptions play an important role for the verification of system properties. When for example using a PKI we need to trust in the authenticity of the Trusted Third Party's public key when verifying certificates. When using trusted computing technology we need to trust that the TPM indeed provides certain security properties, etc. Hence our Security Modeling Framework includes the concept of trust. Within SeMF, we will use the following trust concept:

The term ***trust*** refers to a relation from one agent in the system to another agent with respect to a property, or from an agent directly to a property in the system. Thus, agents can have three slightly different types of trust:

1. Agents can trust that some (global) property holds in a system.
2. Agents can trust that another agent behaves in a certain way, i.e. that a property concerning the behavior of this other agent is satisfied.
3. Agents can trust that another agent has a particular trust.

Being a relation, this notion of trust cannot be used to express different degrees of trust. Agents can either trust or not trust. See [11] for a more detailed discussion on our trust concept.

In order to provide the formal definition, we first introduce the definition of a system and a trusted system, both using the concepts introduced in Section 3.3.

Definition 12 (System)**.** *A system $S = (\Sigma, \mathbb{P}, B, \mathbb{W}, \mathbb{V})$ consists of a set $\mathbb{P}$ of agents acting in the system, a language $B \subseteq \Sigma^*$ over an alphabet of actions Σ describing the system behavior in terms of sequences of actions, a set $\mathbb{V} = \{\lambda_X : \Sigma^* \to (\Sigma_X)^* | X \in \mathbb{P}\}$ of agents' local views, and a set $\mathbb{W} = \{W_X \subseteq \Sigma^* | X \in \mathbb{P}\}$ of agents' initial knowledges.*

Here $(\Sigma_X)^*$ denotes the image of the homomorphism λ_X which has to be individually specified for each system. Which part of an action an agent can see depends on the specific system to specify and can contain any part of it, as indicated in the previous section.

An agent P's conception and understanding of a system S, denoted by S_P, may differ from the actual system. P may not know all about the system's behavior, thus from P's point of view the system's behavior consists of P's initial knowledge W_P. Further, P may not have all information with respect to the other agents' initial knowledges and local views, so P's conception of agents' initial knowledges (W_{XP}) and local views (λ_{XP}) may differ from the actual initial knowledges and local views of the system S. This motivates the following definition.

Definition 13 (Trusted System)**.** *Agent P's conception of system S is defined by $S_P = (\Sigma, \mathbb{P}, W_P, \mathbb{W}_P, \mathbb{V}_P)$. Σ and $\mathbb{P}$ are the alphabet and set of agents, respectively, of both S and S_P, whereas P's initial knowledge (conception) $W_P \subseteq \Sigma^*$ of system behavior B constitutes the behavior of S_P. It further contains a set $\mathbb{V}_P = \{\lambda_{XP} : \Sigma^* \to (\Sigma_{XP})^* | X \in \mathbb{P}\}$ of agent P's conception of agents' local views of S, and a set $\mathbb{W}_P = \{W_{XP} \subseteq \Sigma^* | X \in \mathbb{P}\}$ of agent P's conception of agents' initial knowledges in S. We say that P* trusts *in system S_P (since it represents P's knowledge about system S).*

The definition of an agent's trusted system gives rise now to the definition of an agent's *trust in a property* holding in a system:

Definition 14 (Trusted Property)**.** *Let* prop *be any property that refers to a system as defined in Definition 12. An agent* $P \in \mathbb{P}$ *trusts in* prop *to hold in a system* S*, denoted by* trust(P, prop)*, iff* prop *is fulfilled in* S_P*.*

This notion of trust follows naturally from the different aspects that constitute the model of a system. If a property holds in the system as P perceives it (i.e. in S_P), then from P's point of view the property holds, i.e. P trusts in the property to hold in S. Further the notion of trust allows to specify precisely what it is an agent trusts in. An agent may have trust in one property but not in another. Of course, trust itself is a property of a system as well. Therefore the trust concept allows to model arbitrarily long trust chains such that e.g. the trust of an agent in another agent's trust in a property can be expressed.

3.5 Specific Property Instantiations

The previous section has introduced the basic notions that can be used for the accurate specification of security requirements. However, these notions are very general, and using these notions to specify more concrete security requirements can result in very complex expressions. As such, they are difficult to handle when it comes to including them into standard approaches for systems modeling and verification. Consequently, in the SERENITY project [12], we started the definition of a requirements language using refined notions that describe concrete instantiations of the above security requirements. This language has been further extended in the EVITA project [2] and will be further extended in the course of the TERESA project.

Definition 15 (auth)**.** *For* $P \in \mathbb{P}$ *and* $a, b \in \Sigma$*,* $auth(a, b, P)$ *holds in* B *if for all* $\omega \in B$ *that contain an action* b*, action* a *is authentic for agent* P *after* ω *corresponding to Definition 1.*

Note that in most cases, b will be in P's local view.

The following two security requirements are instantiations of *enforce-behavior*. The first one describes that in all sequences of actions, whenever action b happens, action a must have happened before, the second one describes the opposite (whenever b has happened, a must not have happened before).

Definition 16 (precede)**.** *Let* Σ *be a set of actions,* $a, b \in \Sigma$*. Let* $B, L \subseteq \Sigma^*$ *with* L *defined as follows:*

$$L = \Sigma^* \setminus ((\Sigma \setminus \{a\})^* \{b\} \Sigma^*)$$

Then $precede(a, b)$ *holds if* enforce-behavior(L, B) *holds.*

Definition 17 (not-precede)**.** *Let Σ be a set of actions, $a, b \in \Sigma$. Let $B, L \subseteq \Sigma^*$ with L defined as follows:*

$$L = \Sigma^* \setminus (\Sigma^*\{a\}\Sigma^*\{b\}\Sigma^*)$$

Then $not\text{-}precede(a, b)$ holds if enforce-behavior(L, B) *holds.*

For the specification of anonymity and privacy requirements, we can use the notion of *confidential* reflecting a specific confidentiality property according to Definition 10. Generally in order to specify confidentiality, we need to specify:

- the agents who are allowed to know the data to be confidential,
- the set M of possible values of the parameter that shall be confidential,
- the homomorphism μ that identifies all actions that contain the parameter that shall be confidential and that a malicious agent can use to gain knowledge about the parameter (μ maps these actions onto their "types" and at the same time extracts the parameter to be confidential: $\mu : \Sigma^* \longrightarrow (\Sigma_t \times M)^*$),
- the language L that characterizes the relations between actions that are allowed to be known by a malicious agent (e.g. the relation reflecting that the same message occurs in a specific send and receive action),
- the local views of malicious agents,
- the initial knowledges of malicious agents.

For the purposes of the security building blocks to be introduced in the next section, it is sufficient to define a particular instance of this general confidentiality property in which some of the above concepts are fixed:

- We do not fix the set of agents that are allowed to know the parameter that shall be confidential. This set is denoted by who.
- We fix the "types" of actions the homomorphism μ maps onto: Let $action(p_1, \ldots, p_j)$ be an action in Σ with p_i being the parameter to be confidential. W.l.o.g. let $p_i = p_1$. Then the type of this action is $(action, p_2, \ldots, p_j)$ and $\mu(action(p_1, \ldots, p_j)) = ((action, p_2, \ldots, p_j), p_1)$. In other words, μ keeps all parameters of the action except the one to be confidential which it extracts to form the second component of the action's image under μ. (Note that μ maps those actions that can not be used to extract knowledge about the parameter value to ε.)

  Having fixed this, the only information that is still needed is which is the parameter that we want to be confidential (the message m being sent/received, the data d being stored, the agent P performing a specific action, etc.), and which are the actions that a malicious agent can use to gain knowledge about the parameter to be confidential, We denote the parameter by par and the set of actions by $\mathcal{A}(par)$.

- Regarding the relations between actions that are allowed to be known by a malicious agent, we will use languages $\mathcal{L}$ to denote all dependencies that must be assumed to be known. We should for example assume that the agents know the relation between encryption and decryption with the same shared secret. Later we may fix specific languages $\mathcal{L}$.
- In this deliverable we keep the agents' local views and initial knowledges variable. However, when verifying concrete security properties of a TERESA pattern we may identify specific instances reflecting appropriately the respective pattern's environment.

Definition 18 (confidential)**.** *Let $who \subseteq \mathbb{P}$ be a subset of agents, par be the parameter whose value shall only be known by the agents in who, and $\mathcal{A}(par)$ be the set of actions from which a malicious agent can extract knowledge about the value of par. Let further $\mathcal{M}$ denote the possible values of par and $\mathcal{L}$ denote the language that captures all the allowed relation knowledge between actions in the sense explained above. Then $conf(\mathcal{A}(par), par, \mathcal{M}, \mathcal{L}, who)$ holds in B if $\mathcal{M}$ is parameter-confidential for all $P \in \mathbb{P} \setminus who$ with respect to $(\mathcal{L}, \mathcal{M})$–completeness according to Definition 10.*

In the course of the verification work we may find it useful to define even more concrete instantiations of parameter confidentiality.

3.6 Property Preserving Homomorphisms

As explained in the introduction, the Security Modeling Framework SeMF provides two possibilities to prove security properties for a system: Security Building Blocks, to be introduced in the next section, and property preserving alphabetic language homomorphisms, already introduced in Deliverable D7.1 [13]. The idea of the latter is the following: We specify the system that shall satisfy a specific security property in the necessary detail. Often we do not need to specify all agents' local views and initial knowledges but only those that are relevant for the property in question. We then specify an abstraction of this system by specifying a homomorphism that maps the concrete system onto the abstract one. Mostly the abstract system will only contain the images of those actions of the concrete system that are relevant for the property we want to prove, while all other actions are mapped onto the empty word. We add constraints to the specification of the abstract system that allow to prove easily that it satisfies the property we want to prove for the concrete system.

Finally, we prove that the homomorphism preserves some specific conditions that are proven to imply that the property holding in the abstract system induces the respective property to hold in the concrete system.

In the following we introduce the sufficient conditions already proven to preserve certain security properties and the respective theorems.

3.6.1 Preserving Authenticity

Definition 19. *Let $h : \Sigma^* \to \Sigma'^*$ be an alphabetic language homomorphism and for $P \in \mathbb{P}$ let $\lambda_P : \Sigma^* \to \Sigma_P^*$ and $\lambda'_P : \Sigma'^* \to \Sigma'^*_P$ be the homomorphisms describing the* local views *of P on Σ and Σ', respectively. The language homomorphism h preserves authenticity on S if for each $P \in \mathbb{P}$ exists a mapping $h'_P : \lambda_P(S) \to \lambda'_P(S')$ with $\lambda'_P \circ h = h'_P \circ \lambda_P$ on S.*

$f \circ g$ denotes the composition of functions f and g. The above property captures the fact that the homomorphism h has to be consistent with the local views of P on both abstraction levels in order to preserve authenticity. This means that the actions relevant for Γ' being authentic for P and their inverse images under h must be equally visible by P. In particular, the inverse image of the action being responsible for the authenticity of Γ' must not be mapped to ε by P's local view λ_P on the lower abstraction level. Figure 3.3 visualizes the condition the homomorphism must satisfy.

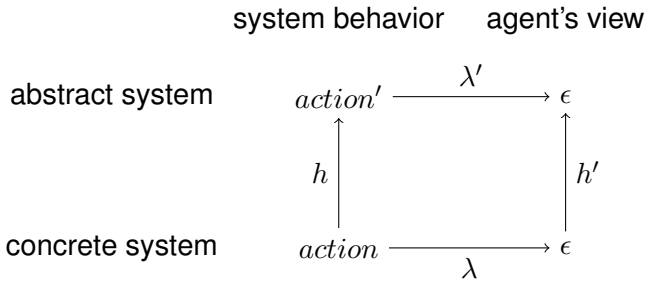

Figure 3.3: Preserving authenticity

Theorem 1. *If $\Gamma' \subseteq \Sigma'$ is authentic for $P \in \mathbb{P}$ after $\omega' \in h(S)$ with respect to W'_P, and if h preserves authenticity on S, then $\Gamma = h^{-1}(\Gamma') \cap \Sigma$ is authentic for $P \in \mathbb{P}$ after each $\omega \in h^{-1}(\omega') \cap S$ with respect to each W_P with $W_P \subseteq h^{-1}(W'_P)$.*

In [14] we have shown how to use the above theorem in order to prove that the specific integration of two security solutions satisfies a certain authenticity property. We have also discussed a naive integration of the two solutions that does not provide the property and have shown how this is reflected by properties of the mapping between the concrete and the abstract system.

3.6.2 Preserving Proofs of Authenticity

We now define properties for a homomorphism in order to preserve proofs. The first condition essentially states that whenever the homomorphic image of a sequence of actions $\omega \in S$ can in the abstract system be continued with a proof send action in $\Gamma S'$ and a proof receive action in $\Gamma P'$, there must be appropriate proof send and receive actions in the inverse image of $\Gamma S'$ and $\Gamma P'$, respectively, to continue the sequence of actions ω in S.

The second condition states that h must not map actions performed by and assigned to agent P on the low abstraction level onto actions assigned to a different agent on the high abstraction level.

Note that $((\Gamma S' \cap \Sigma'_{/P})(\Gamma P' \cap \Sigma'_{/R}))$ is the set of all words of length 2 with the first letter in $(\Gamma S' \cap \Sigma'_{/P})$ and the second letter in $(\Gamma P' \cap \Sigma'_{/R})$.

Definition 20. *Let $\Gamma S' \subseteq \Sigma'$ and $\Gamma P' \subseteq \Sigma'$. $h : \Sigma^* \to \Sigma'^*$ preserves $(\Gamma S', \Gamma P')$-proofs on S if h preserves authenticity on S and if the following holds:*

1. $h(\omega)^{-1}(h(S)) \cap ((\Gamma S' \cap \Sigma'_{/P})(\Gamma P' \cap \Sigma'_{/R})) \neq \emptyset$ *implies* $\omega^{-1}(S) \cap ((h^{-1}(\Gamma S') \cap \Sigma_{/P})(h^{-1}(\Gamma P' \setminus \Sigma'_{/\#}) \cap \Sigma_{/R})) \neq \emptyset$ *for each* $P, R \in \mathbb{P}$ *and* $\omega \in S$.
2. $h(\Sigma_{/P}) \cap (\Sigma' \setminus \Sigma'_{/\#}) \subseteq \Sigma'_{/P}$ *for each* $P \in \mathbb{P}$ *and* $h(\Sigma_{/\#}) \subseteq \Sigma'_{/\#}$.

Theorem 2. *Let $(\Gamma S', \Gamma P')$ be a proof action pair for $\Gamma' \subseteq \Sigma'$. Let $h : \Sigma^* \longrightarrow (\Sigma')^*$ be a homomorphism that preserves $(\Gamma S', \Gamma P')$-proofs on S, and let $\Gamma S = h^{-1}(\Gamma S') \cap \Sigma$, $\Gamma P = h^{-1}(\Gamma P' \setminus \Sigma'_{/\#}) \cap \Sigma$ and $\Gamma = h^{-1}(\Gamma') \cap \Sigma$. Then:*

1. *$(\Gamma S, \Gamma P)$ is a proof action pair for $\Gamma \subseteq \Sigma$.*
2. *If agent $P \in \mathbb{P}$ can give proof of authenticity of Γ' after $\omega' \in h(S)$ then P can give proof of authenticity of Γ after each $\omega \in h^{-1}(\omega') \cap S$.*

In [15] we have shown how this theorem can be used to prove that a specific system provides a specific proof of authenticity property.

3.6.3 Preserving Parameter Confidentiality

For simplicity, we assume that for a system $S \subseteq \Sigma^*$ and homomorphism μ' on $\Sigma'^* \supseteq h(S)$, the property for the concrete system is defined using the homomorphism $\mu = \mu' \circ h$ (where $f \circ g$ denotes the composition of functions f and g). Then the same language K can be used on both levels of abstraction to express the combinations of action types and parameter values. Definition 21 below can be easily transferred to the more general case with different type sets and different homomorphisms. However, the extended definition is more technical while the simplified version is suitable for many realistic scenarios.

Preservation of parameter confidentiality by a homomorphism h is concerned with the parameter values considered possible on the different levels of abstraction. It therefore depends on the local views λ_R and λ'_R as well as on the relation between the knowledge sets W_R and W'_R. These relations between the different levels of abstraction are shown in Figure 3.4, $\Lambda(\omega, W_R)$ abbreviating $\lambda_P^{-1}(\lambda_P(\omega)) \cap W_P$.

Definition 21 formulates a condition on language homomorphisms. We will show that homomorphisms satisfying this condition preserve parameter confidentiality, i.e. that if parameter confidentiality is satisfied in the homomorphic image of a system it is satisfied in the system as well.

Definition 21. *Let $h : \Sigma^* \longrightarrow \Sigma'^*$ and $\mu : \Sigma^* \to (\Sigma_t \times M)^*$ be language homomorphisms, $S \subseteq \Sigma^*$, $R \in \mathbb{P}$, and λ'_R the local view of agent R in Σ'^*. Then we call the homomorphism h* parameter confidential for R with respect to μ *if for all $\omega \in S$ there exists $A_\omega \subseteq \Sigma_t^*$ such that the following holds:*

$$\mu[\Lambda_R(\omega, W_R)] = \mu'[\Lambda'_R(h(\omega), h(W_R))] \cap p_t^{-1}(A_\omega)$$

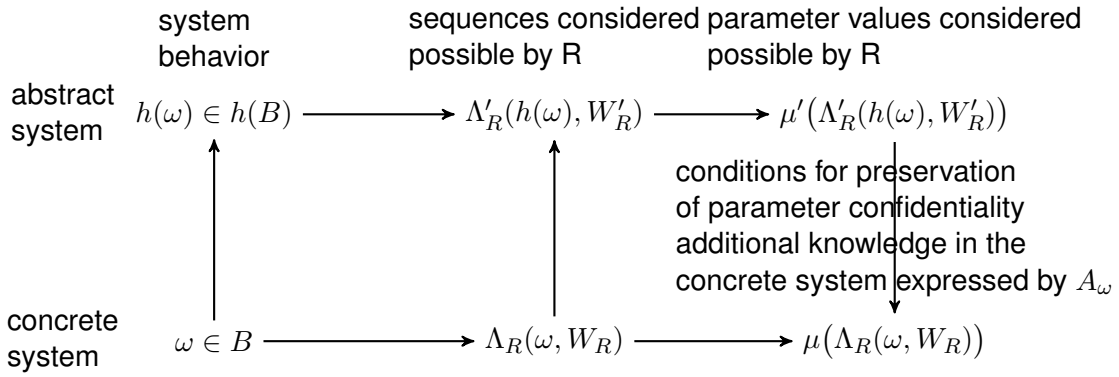

Figure 3.4: Preserving parameter confidentiality

So a parameter confidential homomorphism guarantees that the sequences of actions R considers possible after having monitored ω in S correspond to those R considers possible after having monitored $h(\omega)$ in S'. The definition takes into account that usually agents will be allowed to learn additional information in the refined system about the type of actions that have occurred. This additional knowledge is expressed by the set $A_\omega \subseteq \Sigma_t^*$. Applying p_t^{-1} adds all parameter values in M. Therefore, intersection with $p_t^{-1}(A_\omega)$ expresses that agents cannot gain any additional information on *parameter values*. In the case that agents do not learn any new information about action types, we simply have $A_\omega = \Sigma_t^*$ which leads to the much simpler condition $\mu[\Lambda_R(\omega, W_R)] = \mu'[\Lambda'_R(h(\omega), h(W_R))]$.

Theorem 3 now states that parameter confidential homomorphisms indeed preserve parameter confidentiality as defined in Definition 10.

Theorem 3. *Let $S' \subseteq \Sigma'^*$ be parameter confidential for agent $R \in \mathbb{P}$ with respect to some μ' and K. Let furthermore $S \subseteq \Sigma^*$ and homomorphism $h : \Sigma^* \longrightarrow \Sigma'^*$ such that $h(S) \subseteq S'$ and $h(W_R) = W'_R$. If h is parameter confidential with respect to R, then S is parameter confidential for R with respect to $\mu = \mu' \circ h$ and K.*

In [10] we have discussed an application of this theorem using a detailed example.

3.7 Security Building Blocks

In this section we introduce so-called *Security Building Blocks* (SeBBs) that will be used to prove security properties for TERESA patterns. A SeBB takes as input a set of *internal properties*, applies a rule (the *means*) to this set, and produces a set of *external properties* as implication of the internal properties.

Hence, a Security Building Block consists of three different parts, as illustrated in Figure 3.5:

- The *external properties* of a Security Building Block represent the security properties that the Security Building Block provides for the overall system.

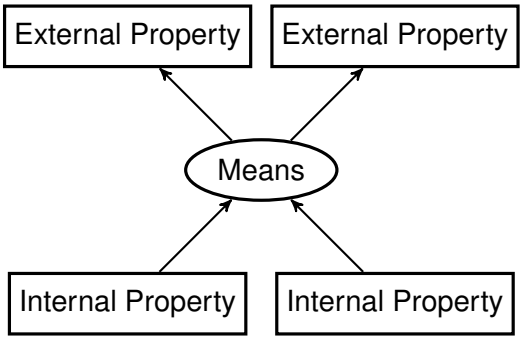

Figure 3.5: Structure of a Security Building Block

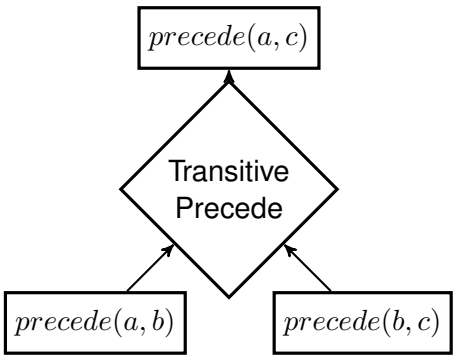

Figure 3.6: Example Building Block: Transitive Precede

- The *means* represents the mechanism or instrument by which the external properties can be achieved. These can be based on a proven theorem related to specific properties of the security requirements, or on expert knowledge related to the mechanism the SeBB addresses.
- The *internal properties* are those properties that must be fulfilled in order for the mechanism to provide the external properties.

A Security Building Block represents in itself a property of the system, namely the implication that if all internal properties hold, the external properties hold as well. These implications have to be formally proven, depending on the means that are applied. These can be categorized into two different classes:

- Implications originating from the properties of security properties themselves defined *within SeMF* allowing to relate them, which we will denote by *Formal SeBB* or *F-SeBB* for short,
- implications originating from security mechanisms like protocols and cryptographic primitives *external to SeMF*, denoted by *Mechanism SeBB* or *M-SeBB*.

F-SeBBs are those that originate from the definitions of the properties themselves. There are simple ones, such as the confidentiality regarding Bob and Alice being included in the confidentiality regarding only Bob, but also more complex ones, such as the chaining of trust assumptions regarding authenticity. The speciality with these properties is that they can be

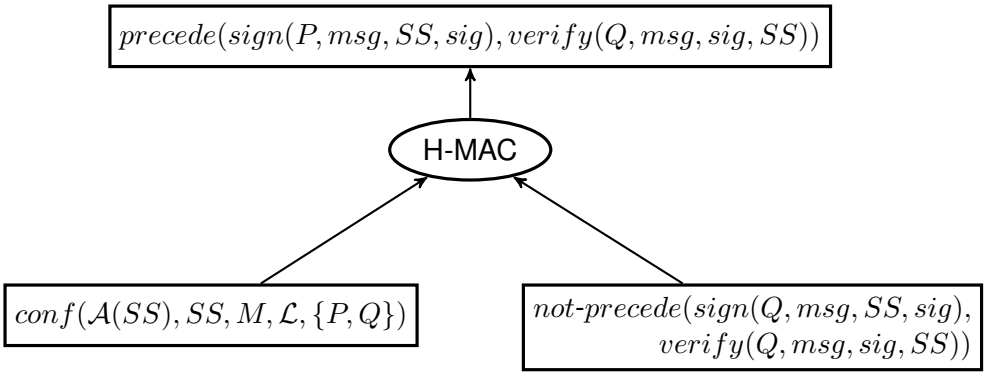

Figure 3.7: Example Building Block: HMAC-Building Block

proven within SeMF independent of any specific system, based only on the assumptions that are introduced by the internal properties. An example is given by the Security Building Block Transitive-Precede in Figure 3.6.

M-SeBBs in contrast cannot be proven completely internal to SeMF. They require assumptions that originate from mathematical proofs or expert knowledge that is external to SeMF. An example for the usage of HMAC-Signatures is provided in Figure 3.7.

In order to visualize the differences between means originating from within SeMF and those that require external assumptions, the former are represented as diamonds, whilst the latter are represented as circles (compare Figures 3.6 and 3.7).

An important special case of the applicability of SeBBs arises when a *Trust* property is part of the predicate to verify. This case can be easily addressed: Recall that a SeBB represents in itself a property for an arbitrary system, namely that the set of the SeBB's internal properties implies the SeBB's external properties. For example, the SeBB depicted in Figure 3.6 represents the implication $precede(a, c) \wedge precede(c, b) \rightarrow precede(a, b)$. Since this property holds in any system, it holds in particular in a trusted system S_P. By Definition 14, $precede(a, c) \wedge precede(c, b) \rightarrow precede(a, b)$ holding in S_P means that $trust(P, precede(a, c) \wedge precede(c, b) \rightarrow precede(a, b))$ holds in S, which implies $trust(P, precede(a, c) \wedge precede(c, b)) \rightarrow trust(P, precede(a, b))$.

4 The Pattern Verification Methodology

This chapter describes how the concepts of Security Building Blocks can be used for the verification of TERESA S&D patterns.

The usefullness of SeBBs is twofold. First, they can be used within a security engineering process which is not relevant for TERESA but worth mentioning: One starts with an abstract security property $prop$ and applies an adequate SeBB that has $prop$ as an external security property and uses a set of internal properties. One then decides for each of these internal properties whether it can be reasonably assumed to hold for the system to be designed. For each of those security properties that cannot be assumed to hold, one again applies a SeBB that has this security property as an external one and uses a set of internal security properties that need to hold. This process is repeated until all resulting internal security properties can be assumed to hold in the system. In the project EVITA we have demonstrated this approach, see [16] for more details.

In WP5 of TERESA we use SeBBs in the reverse way: We identify all assumptions that are included in an S&D pattern to be verified. For a pattern describing SSL for example we may have the assumption that the client's private key be confidential for the client. For each of the security requirements the pattern is supposed to provide, we then search for a way to repeatedly apply appropriate SeBBs leading from a subset of the assumptions identified by the pattern to the respective security requirement. The subset of assumptions serves as internal properties for the first round of SeBB application which produces a set of external properties. These are taken as internal properties for the next application of SeBBs, etc., until the last step in which the desired security requirement is (one of) the external propertie(s) of the applied SeBB. The process includes the following steps:

1. Modeling of the pattern system. This includes
    - to specify the agents and actions. We have two types of agents: those agents that are acting entities within the solution described by the pattern (e.g. the SSL client and server), and so-called *stereotype agents* that represent the application(s) on the one hand (according to the external interface of the pattern) and components used by the pattern (e.g. a crypto library) on the other hand (according to the internal interface of the pattern). These stereotype agents perform actions that are external to the solution but necessary for the solution to function. This can be for example the verification of a certificate for the establishment of an SSL channel. The solution may just perform the SSL handshake protocol and may assume that the server certificate is authentically provided by a trusted third party, while the stereotype agent represents the client browser, an agent external to the solution that actually verifies the certificate.

- to specify the local views of agents relevant for the security property to be proven. The local views will be specified according to the respective section in the pattern specification that identifies the communication means to be used by the pattern,
- to identify the initial knowledges of agents relevant for the security property to be proven. The initial knowledges will be specified according to the respective section in the pattern specification.

2. According to what is listed in the section 'Post-conditions' of the pattern, specify a security requirement R that shall be proven.
3. Specify the assumptions that need to hold in order for the pattern to satisfy R. These assumptions are listed in the section "Preconditions" of the pattern and constitute the starting set of internal properties.
4. Search for a SeBB that uses some of these assumptions as internal properties.
5. If R is not one of the external properties of the identified SeBB: add the SeBB's external properties to the set of internal properties then go back to step 3.
6. If R is contained in the external properties of the SeBB: finished.
7. If no more SeBBs can be applied, no proof can be found.

Note that there may be more than one way to find a path from a set of assumptions (internal properties) to the requirement. Once we have established such a path, we need to verify that the M-SeBBs we applied within our proof correspond to the security primitives used by the pattern. Having established this fact, the proof is finished.

Note that not finding a path from assumptions to requirement does not necessarily mean that the pattern does not provide the desired property. However, in most cases the specific external (intermediate) property that needs to be proven and for which no SeBB can be found to be applicable will point to security issues of the pattern. A redesign of the pattern taking these into account might then result in a provable pattern. Another reason for a failing proof may be that in all paths we find for a specific security requirement there is an M-SeBB that does not correspond to any of the mechanisms applied by the pattern. In this case the pattern needs to be redesigned as well. One possibility may be to introduce a further security mechanism in the appropriate part of the pattern, another possibility may be to add more assumptions to the pattern.

We have already successfully applied our verification method based on SeBBs to a complex security requirement within the EVITA project [16].

5 The Pattern Implementation Verification Methodology

An S&D pattern is the generic description of an S&D solution together with information concerning the environment in which the pattern can be used. This information concerns both the application into which the pattern can be integrated and the components that the pattern needs to use in order to function. In a concrete application development process the pattern environment is instantiated by the concrete functions of the application and components that the pattern uses, while the functionality of the solution itself is maintained. This is what we call the pattern implementation.

In this section we introduce a methodology that on the one hand proves the correctness of a specific implementation of a pattern, and on the other hand shall support the provision of guidelines for this implementation. Recall that the pattern verification methodology described in Section 4 uses specific assumptions identified within the pattern as a basis to prove that the pattern indeed provides the desired S&D properties. Hence when implementing the pattern, i.e. when integrating it into a specific application these assumptions must hold in this specific environment, otherwise the application might not satisfy the desired S&D property. The methodology to verify that the assumptions indeed hold is based on the use of property preserving homomorphisms described in Section 3.6. These homomorphisms "transport" specific S&D properties that hold for an abstract system to a concrete system, provided they satisfy certain conditions.

The idea is the following: Recall that for verifying the pattern we have already specified a system that models the behavior of the solution described by the pattern and the behavior of the external and internal interfaces that the solution needs in order to function. This additional behavior consists of so-called *stereotype agents* and their actions, i.e. agents and actions that are needed for the solution to function. We call this model the *pattern system model*, it constitutes the abstract system for our implementation verification methodology. We obtain the concrete system by modeling the solution together with the concrete application it is integrated into. While the model of the solution will be identical to the respective part of the pattern system model, the part of the model representing the application behavior will be a refinement of the stereotype agents and actions of the pattern system model. We further need to specify the local views and initial knowlegdes of the concrete system's agents.

Since the pattern verification is based on assuming that the pattern system satisfies certain assumptions, i.e. provides certain properties corresponding to the assumptions, in our abstract system all these properties hold. So in order to prove that the properties also hold in the concrete system (representing a concrete pattern implementation where the pattern is integrated into a concrete application), for each of them we define a homomorphism from the concrete application system to the abstract pattern system. With respect to the solution

behavior in the abstract pattern system and the concrete application system, respectively, the homomorphism will simply be the identity.

However, the homomorhism abstracts the concrete application agent(s) and its/their behaviour and maps them onto the stereotype agents and actions we used in the pattern system model. Finally, we prove that this homomorphism preserves the property in question, which by the Theorems introduced in Section 3.6 or similar ones still to be proven proves that the respective property holds in the concrete application system.

Figure 5.1 shows the idea of our approach.

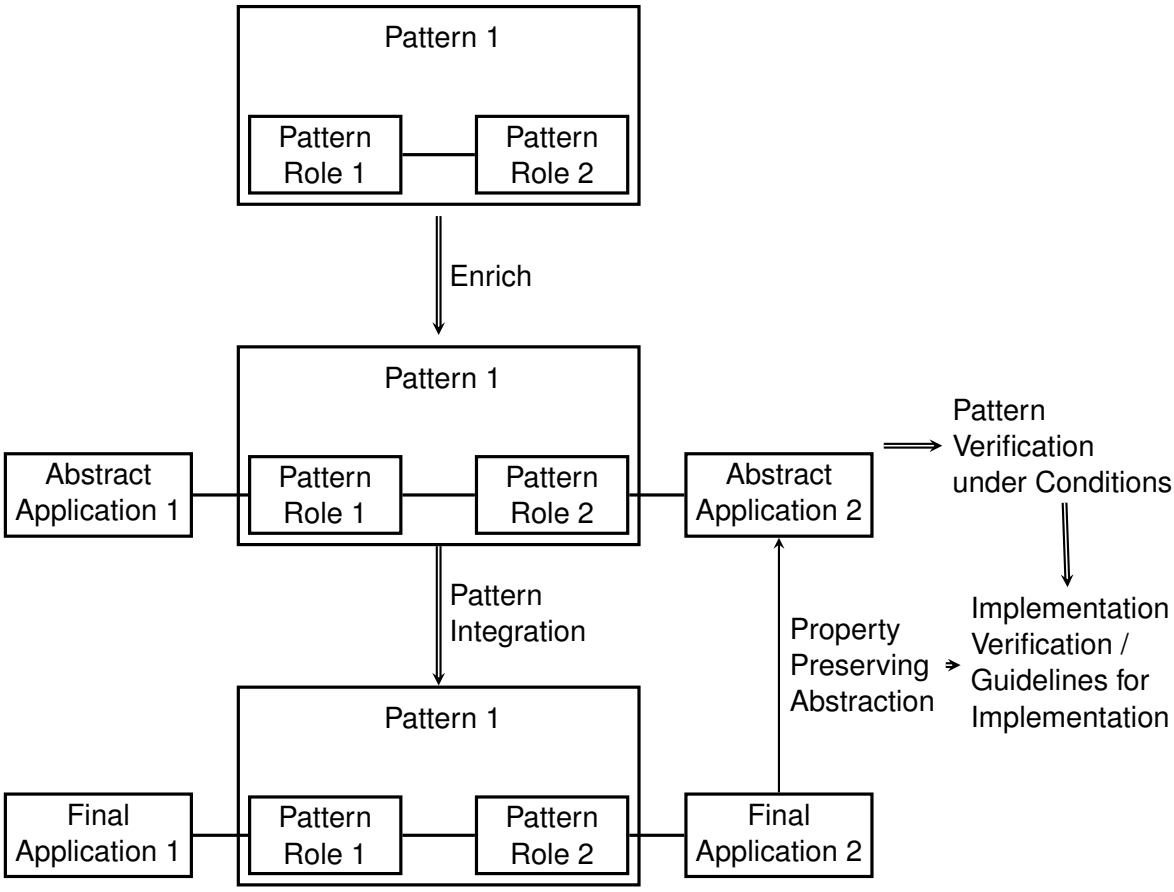

Figure 5.1: Preserving parameter confidentiality

5.1 Guidelines for Pattern Implementation

The above described method is not meant to be the means by which application developers verify a concrete pattern implementation, the method is more suitable to be applied by security experts. However, the conditions that a homomorphism must satisfy in order to preserve a certain security property are expected to lead to guidelines that can be followed by the application developer. The conditions to be satisfied for preserving authenticity for example

are concerned with the compatibility of the homomorphism with the agents' local views in the abstract and concrete system, respectively. Since the local views correspond to the nature of communication (e.g. wired vs. wireless network), the communication means forseen in the pattern specification will imply certain restrictions for communication means of the application when implementing the pattern. Further, in [11] we have investigated cases in which a mapping from the concrete to the abstract system violates the condition for preserving authenticity. For each application domain we will therefore investigate specific pattern implementations in order to extract domain specific implementation guidelines for these patterns.

Together with the other WP5 partners IRIT and Uni Siegen we will then transform these guidelines into other modeling domains such as UML/OCL. As can be seen in [16], some security properties of our Security Modeling Framework SeMF can be broken down to very basic properties such as an action never happening in the system, or one action always being preceded by another action. These basic properties shall be specified as constraints of the system and formulated in Object Constraints Language (OCL, see [17]), a language specified by the Object Management Group (OMG) for specifying constraints for UML models. As a result, security properties that are sufficient for a specific pattern implementation to provide the desired external security properties are transformed to OCL constraints which can then be verified by standard UML/OCL tools.

6 Outlook

In this document we have described the TERESA approach for the verification of both S&D patterns and their implementation, i.e. their integration into a specific application. The approach is based on the Security Modeling Framework SeMF developed by Fraunhofer SIT. It provides a methodology for modeling a pattern system and for proving that in this model the solution specified in the pattern provides certain properties, given that the assumptions specified in the pattern hold in the system. It further provides a method to verify that for a concrete implementation of the pattern, i.e. when integrating the pattern into a concrete application, these assumptions are indeed satisfied.

In the next phase of the work, we will apply our approach for pattern verification to concrete examples. These examples originate from patterns specified in Deliverable D2.1 [18] but contain a more precise specification that is suitable for verification purposes (such as the Secure Software Download pattern of the metrology domain). While many useful Security Building Blocks (SeBBs) are already specified (see [16]), verification of TERESA example patterns may lead to the specification of more SeBBs that are specifically relevant for embedded systems.

A further part of the work will consist in developing the approach for formulating guidelines for platform dependent pattern implementation. More precisely, we will on the one hand specify sufficient conditions of language homomorphisms for preserving further security properties (precede, trust, ...). On the other hand, together with the the other WP5 partners IRIT and Uni Siegen we will transform these conditions into other modeling domains. For example, we will refine, where possible, security properties representing assumptions for patterns into very basic properties and formulate these as OCL constraints refering to a UML model of specific application domains.

The end phase of the work will then consist in verifying TERESA S&D patterns to be included in the repository (see WP4), verifying concrete pattern implementations in the various TERESA domains, and deriving from these verifications general guidelines for platform dependent implementation.

7 Bibliography

[1] "Serenity, system engineering for security & dependability," www.serenity-project.org, 2006.

[2] "Evita,e-safety vehicle intrusion protected applications," www.evita-project.org/, 2010.

[3] B. Alpern and F. B. Schneider, "Defining liveness," *Information Processing Letters*, vol. 21, no. 4, pp. 181–185, 7 Oct. 1985.

[4] M. Burrows, M. Abadi, and R. Needham, "A Logic of Authentication," *ACM Transactions on Computer Systems*, vol. 8, 1990.

[5] L. Paulson, "Proving Properties of Security Protocols by Induction," Computer Laboratory, University of Cambridg, Tech. Rep. 409, 1996.

[6] M. Abadi and M. Tuttle, "A Semantics for a Logic of Authentication," in *Tenth Annual ACM Symposium on Principles of Distributed Computing, Montreal, Canada*, August 1991, pp. 201–216.

[7] G. Wedel and V. Kessler, "Formal Semantics for Authentication Logics," in *Computer Security - Esorics 96*, ser. LNCS, vol. 1146, 1996, pp. 219–241.

[8] R. Grimm and P. Ochsenschläger, "Binding Cooperation, A Formal Model for Electronic Commerce," *Computer Networks*, vol. 37, pp. 171–193, 2001.

[9] S. Gürgens, P. Ochsenschläger, and C. Rudolph, "Parameter confidentiality," in *Informatik 2003 - Teiltagung Sicherheit*. Gesellschaft für Informatik, 2003.

[10] ——, "Abstractions preserving parameter confidentiality," in *European Symposium On Research in Computer Security (ESORICS 2005)*, 2005, pp. 418–437.

[11] A. Fuchs, S. Gürgens, and C. Rudolph, "A Formal Notion of Trust – Enabling Reasoning about Security Properties," in *Preceedings of Fourth IFIP WG 11.1 International Conference on Trust Management*, 2010.

[12] K. Dolinar, A. Fuchs, S. Gürgens, C. Rudolph, "A3.D2.2 - S&D requirements for networks and devices," SERENITY-Project, Tech. Rep., 2008.

[13] B. Hamid, N. Desnos, D. Weber, C. Bodenstedt, A. Fuchs, S. Gürgens, "D7.1 - Analysis of Challenges S&D requirements for networks and devices," TERESA-Project, Tech. Rep., 2010.

[14] A. Fuchs, S. Gürgens, and C. Rudolph, “On the Security Validation of Integrated Security Solutions,” in *Emerging Challenges for Security, Privacy and Trust - 24th IFIP TC 11 International Information Security Conference, SEC 2009, Cyprus*, ser. IFIP Advances in Information and Communication Technology, D. Gritzalis and J. Lopez, Eds., vol. 297. Springer, 2009.

[15] S. Gürgens, P. Ochsenschläger, and C. Rudolph, “Authenticity and provability - a formal framework,” in *Infrastructure Security Conference InfraSec 2002*, ser. Lecture Notes in Computer Science, vol. 2437. Springer Verlag, 2002, pp. 227–245.

[16] A. Fuchs, S. Gürgens, R. Rieke, L. Apvrille, “D3.4.3 - Final Architecture and Protocols Verification,” EVITA-Project, Tech. Rep., 2010.

[17] “Object Constraint Language (OCL),” http://www.omg.org/spec/OCL/.

[18] M. Blanco, D. Gonzales, “D2.1 - Use Cases Application Viewpoint,” TERESA-Project, Tech. Rep., 2010.