



A3.D2.2 - S&D requirements for networks and devices (revised)

K. Dolinar, A. Fuchs, S. Gürgens, C. Rudolph

Document Number	A3.D2.2
Document Title	S&D requirements for networks and devices (revised)
Version	V1.0
Status	Final
Work Package	WP3.2
Deliverable Type	Report
Contractual Date of Delivery	30 June 2008
Actual Date of Delivery	6 August 2008
Responsible Unit	SIT
Contributors	SET
Keyword List	
Dissemination level	PU

Change History

<i>Version</i>	<i>Date</i>	<i>Status</i>	<i>Author (Unit)</i>	<i>Description</i>
V0.1	July 3rd, 2008	Draft	A. Fuchs, S. Gürgens, C. Rudolph (FhG-SIT)	A3.D2.1 corrected
V0.2	July 9th, 2008	Draft	A. Fuchs, S. Gürgens, C. Rudolph (FhG-SIT)	Added additional properties
V0.3	August 4th, 2008	Draft	S. Gürgens (FhG-SIT), K. Dolinar (SET)	Added archiving properties
V0.4	August 5th, 2008	Draft	S. Gürgens, C. Rudolph (FhG-SIT)	Some corrections
V1.0	August 6th, 2008	Draft	S. Gürgens, C. Rudolph (FhG-SIT)	Corrections according to quality check

Executive summary

This report provides a revised version of the specification language for the accurate description of S&D requirements for networks and devices. It also includes an extended set of S&D requirements specified in this language.

The main goal of this work is to provide the means to formulate exact specifications of security and dependability requirements. As necessary foundations to achieve this goal a formal framework has been developed that provides formal semantics for many S&D requirements. General S&D requirements covered by this framework include authenticity, proof of authenticity (different flavours of non-repudiation), authenticity with respect to a phase (session management, etc), confidentiality, integrity, enforcement of particular behaviours, availability, reliability, and maintainability.

In order to facilitate the use of the framework, a language for S&D requirements specification is defined that provides simplified access to the (rather complex) formal S&D requirements specifications. This language can be applied to a model of a system in terms of sequences of actions. Such a model can be derived from other standard description languages (e.g. message sequence charts and action diagrams).

This revised version of the requirements specification language provides an extended set of definitions for the formal semantics for S&D requirements. The new concept of durable proof of authenticity (see Section 2.2.4.) is relevant for long-term archiving solutions, and conditional confidentiality (see Section 2.2.6.) is useful in order to capture the evolution of knowledge. Furthermore, some dependability requirements were added that are relevant for the eGovernment scenario.

Table of Contents

1. Introduction	1
2. Formal specifications of S&D requirements	2
2.1. Preliminaries	2
2.1.1. <i>System behaviour specification and agents' knowledge about a system</i>	2
2.1.2. <i>Agents' knowledge about the global system behaviour</i>	2
2.1.3. <i>Agents' view of the global system behaviour</i>	3
2.2. Security requirements	5
2.2.1. <i>Authenticity</i>	5
2.2.2. <i>Authenticity with respect to a phase</i>	5
2.2.3. <i>Proof of authenticity</i>	6
2.2.4. <i>Durable proof of authenticity</i>	7
2.2.5. <i>A remark to integrity</i>	10
2.2.6. <i>Confidentiality</i>	11
2.2.7. <i>Enforcing certain system behaviour</i>	18
2.3. Dependability requirements	18
2.3.1. <i>Modelling a system for the specification of dependability requirements</i>	18
2.3.2. <i>Availability</i>	19
2.3.3. <i>Reliability</i>	19
2.3.4. <i>Maintainability</i>	20
3. A language for the specification of S&D requirements for networks and devices	21
3.1. Security requirements	22
3.1.1. <i>Authenticity requirements</i>	22
3.1.2. <i>Proof of authenticity requirements</i>	22
3.1.3. <i>Requirements with respect to a phase</i>	23
3.1.4. <i>Confidentiality requirements</i>	24
3.1.5. <i>Security requirements for enforcing behaviour</i>	25
3.2. Dependability requirements	27
4. Guidelines for requirements elicitation	29
4.1. Initial questions for requirements elicitation	29
References	30

1. Introduction

Devices and networks constitute the basic building blocks of modern IT systems. Confidential data can be stored, processed and transmitted, particular actions might require authenticity, devices have to be identified, etc.; a large variety of security and dependability (S&D) requirements can be directly specified on the level of devices and networks. The variety is particularly high in the context of ambient intelligence (AmI) systems where different types of devices exist and these devices can move from trusted to untrusted environments while using different communication links. Furthermore, complex S&D requirements specified on the level of business processes or work-flows can induce S&D requirements on devices and network communication. This report describes a language for the accurate specification of S&D requirements for networks and devices, taking into account the particular requirements of AmI systems.

The report is structured as follows: First, a formal approach to system behaviour specification is described which is then used to define the semantic foundations of the S&D requirements. General notions of security and dependability properties are defined within this specification framework. These notions are subsequently used to define the semantics for the S&D requirements language. Section 3. introduces the language for S&D requirements specification. Guidelines for the use of this language are given in Section 4..

2. Formal specifications of S&D requirements

2.1. Preliminaries

2.1.1. System behaviour specification and agents' knowledge about a system

The *behaviour* S of a discrete system can be formally described by the set of its possible sequences of actions (traces). Therefore $S \subseteq \Sigma^*$ 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 $\text{pre}(u)$; $\varepsilon \in \text{pre}(u)$ holds for every word u . We denote the set of letters in a word u by $\text{alph}(u)$.

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

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

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 $h^* : \Sigma^* \rightarrow \Sigma'^*$ 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 $h : \Sigma \rightarrow \Sigma' \cup \{\varepsilon\}$. 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 formalisations. 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 behaviour.

2.1.2. Agents' knowledge about the global system behaviour

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 behaviour 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 [2] 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, we extend the system specification by two components: *agents' knowledge* about the global system behaviour 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 behaviour the

agent can actually see. In the following paragraphs, these two components and their relations are explained in detail.

For any agent P its knowledge about the global system behaviour $W_P \subseteq \Sigma^*$ 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 behaviour generally does not include behaviour 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 behaviour. However, in general malicious behaviour 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 behaviour. Every behaviour which is not explicitly excluded by some W_P is allowed. Denoting a system containing malicious behaviour by S and the correct system behaviour by S_C , we assume $S_C \subseteq S \subseteq \Sigma^*$. We further assume $S \subseteq W_P$, i.e. every agent considers the system behaviour to be possible. This reflects the fact that an agent not having knowledge of any restrictions of the system considers all Σ^* to be possible. Security properties can now be defined relative to W_P . The relation between the system behaviour without malicious actions S_C , the system behaviour including malicious actions S , and W_P is graphically shown in Figure 1.

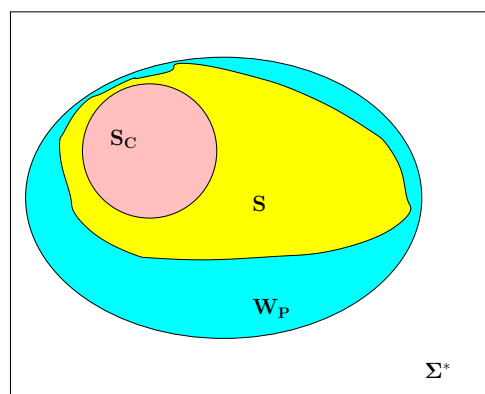


Figure 1: System behaviour and W_P

2.1.3. Agents' view of the global system behaviour

The set W_P describes what P knows initially. However, in a running system P can learn from actions that have occurred. Satisfaction of security properties obviously also depends on what agents are able to learn. After a sequence of actions $\omega \in S$ has happened, every agent can use its *local view* of ω to determine the sequences of actions it considers to be possible. 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 $\pi_P : \Sigma^* \rightarrow \Sigma^*_{/P}$ 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 other agents' actions without seeing the messages sent and received, respectively. In this case, P 's local view of ω will be equal to $(send, P, m1)(rec, Q)$. 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' state.

Thus, we generally denote the local view of an agent P on Σ by $\lambda_P : \Sigma^* \rightarrow \Sigma^*_P$. The local views of all agents together contain all information about the system behaviour S .

For a sequence of actions $\omega \in S$ 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 example with the projection on P being P 's local view, $\lambda_P^{-1}(\lambda_P(\omega))$ consists of sequences each of which contains an action $(send, P, (Q, m1))$. For some agent R that does not take part in ω , $\lambda_R^{-1}(\lambda_R(\omega))$ consists of sequences of actions of other agents, i.e. is equal to $(\Sigma \setminus \Sigma_{/R})^*$.

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 knowledge to reduce this set: $\lambda_P^{-1}(\lambda_P(\omega)) \cap W_P$ describes all sequences of actions P considers to be possible when ω has happened. The set $\lambda_P^{-1}(\lambda_P(\omega)) \cap W_P$ is similar to the possible worlds semantics that have been defined for authentication logics in the context of cryptographic protocols [1, 6]. 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 2.

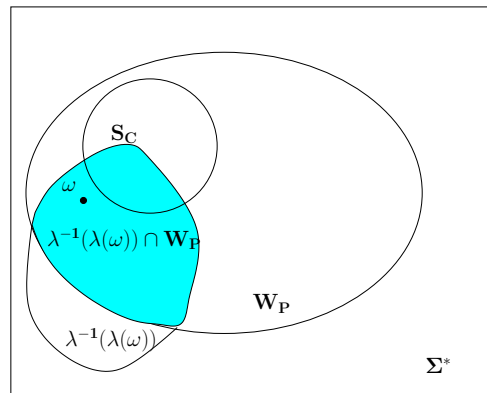


Figure 2: Local view of P

2.2. Security requirements

2.2.1. 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 possible 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 $\omega \in S$ with respect to W_P if $\text{alph}(x) \cap \Gamma \neq \emptyset$ for all $x \in \lambda_P^{-1}(\lambda_P(\omega)) \cap W_P$.*

Input needed for specifying authenticity:

- The action after which authenticity shall hold.
- The agent for which authenticity shall hold.
- The action or set of actions that shall be authentic.

2.2.2. 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 [3]. 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 behaviour within V as within S .

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

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

¹Please note that in order to achieve *authentication* one additionally needs assurance about the time of the occurrence of the action (see Section 2.2.2. *authenticity with respect to a phase*).

A phase can be a very complex part of the system (see for example the the definition of a phase by Grimm and Ochsenschläger [3]). However, often phases have well defined start and end actions. Therefore, for our purposes it is sufficient to consider only those phases that start with one specific action and end with one specific action. In other words, an element of such a phase begins with the start action and contains either no end action at all or contains just one end action at the end.

Definition 3 (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 $x \in S$ 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 $\text{alph}(v) \cap \Gamma \neq \emptyset$. Γ is currently authentic for P after x if $w = \varepsilon$.*

Input needed for specification of authenticity with respect to a phase

- Action with which P starts the phase between P and Q ,
- Action with which P ends the phase between P and Q ,
- The action after which authenticity shall hold.
- The agent for which authenticity shall hold.
- The action or set of actions that shall be authentic.

2.2.3. 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 P 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 P has executed an action from ΓP then Γ is authentic for P and P can forward the proof to any other agent using actions in ΓS .

Definition 4 (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 S with respect to $(W_P)_{P \in \mathbb{P}}$ if for all $\omega \in S$ and for all $P \in \mathbb{P}$ with $\text{alph}(\pi_P(\omega)) \cap \Gamma P \neq \emptyset$ the following holds:*

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

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

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

The following definition takes into account that in practise it is often not appropriate to expect that proofs that have been sent will always be received. Thus for the existence of a proof of authenticity it is not required that there be send and receive proof actions to continue with.

Definition 5 (Proof of authenticity) *A set $\Gamma P \subseteq \Sigma$ is called a set of proof receive actions of authenticity for a set $\Gamma \subseteq \Sigma$ on S with respect to $(W_P)_{P \in \mathbb{P}}$ if for all $\omega \in S$ and for all $P \in \mathbb{P}$, the set Γ is authentic for P after ω whenever $\text{alph}(\pi_P(\omega)) \cap \Gamma P \neq \emptyset$.*

Input needed for specifying proof of authenticity

- The agent for which authenticity shall hold.
- The action or set of actions that shall be authentic.
- The action or set of actions after which the agent shall own a proof of authenticity.

2.2.4. Durable proof of authenticity

In this section two different variants of the concept of durable proofs of authenticity are introduced. The first is based on Definition 5 introduced in the last section, the second is based on the concept of time.

Definition 5 tries to convey the idea that proof of authenticity means carrying out actions (e.g. using some technical means, as for example cryptography) in the following way: if an agent is involved in the proof it must have gotten an information of proof which he has acquired by doing a so called receive proof action; such an action completes the local view of that agent so that the conditions for authenticity of the set in question are actually satisfied (the knowledge enhances, the possible set of sequences of actions that resulted in current situation shrinks ...).

Definition 6 (Phase Actions) *For a phase V we define $\text{alph}[V] = \{y | \exists x \in V (y \in \text{alph}(x))\}$.*

Proposition 1 *Every system (prefix closed language) can be decomposed into phases.*

Proof. *Let S be a system and W a word so that $\max(W, S)$ is true and $\text{length}(W) = \max\{\text{length}(w) | w \in S\}$ and call $\text{length}(W)$ diameter of S , denote $\text{diam}(S)$. If a word $w = a_1 a_2 a_3 \dots a_n$ and $a_i \in \Sigma$ then let $(w)_i := a_i$.*

Let us take a set $V_{i < j} = \{(w)_i \dots (w)_k | w \in S \& i < k \leq j\}$, i.e a set of all the sequences of actions obtained by cutting every sequence from system S at the (inclusively) i -th letter/action and at the (inclusively) j -th or the nearest lower index if the original sequence from system S was shorter than j actions. If the original sequence from S was shorter than i actions then expression $(w)_i \dots (w)_k$ evaluates to the empty word.

We claim that $V_{i<j}$ is prefix closed. If it wasn't then there would exist $v \in V_{i<j}$ such that $V_{i<j}$ would not contain certain prefix p of v , i.e. there would exist two words $p, e \in \Sigma^*$ so that $v = pe$ but $p \notin V_{i<j}$. If we take it to the whole system S that would mean, according to the definition of set $V_{i<j}$, that there exists a sequence $uv \in S$ such that $up \notin S$ (i.e. as soon as S contains up , p is in $V_{i<j}$ by definition). The conclusion $up \notin S$ should mean that S is not prefix closed, since up is a prefix of $uv = upe$. This is contradiction as we know the system S is prefix closed.

It is not hard to see also that every non maximal sequence $v \in V_{i<j}$ has “exactly the same” continuation va , where $a \in \Sigma$ so that $va \in V_{i<j}$, as uv has in S , namely uva ; here uv and uva are the sequences of actions which, after cutting them at i -th and closest to j -th action, become v , respectively va , as in accord with definition for $V_{i<j}$.

Now, if we cut the set $V_{i<j}$ at the indexes so that $V_{i<j}$ contains at least one sequence of length 1 (i.e. at least one single action), then $V_{i<j}$ satisfies phase condition. If we take $V_{0<i_1}, V_{i_1+1<i_2}, \dots, V_{i_{f-1}+1<i_f}$, where $i_f = \text{diam}(S)$, so that every $V_{i_{l-1}+1<i_l}$ includes at least one single action (this is always possible), then this series is decomposition of the system S into phases. \square

Definition 7 (Phase Restrained Proof of Authenticity) A pair $(\Gamma S, \Gamma P)$ with $\Gamma S \subseteq \Sigma$ and $\Gamma P \subseteq \Sigma$ for V a phase of a system S is a phase restrained proof of authenticity for a set $\Gamma \subseteq \text{alph}[V]$ on system S with respect to $(W_P)_{P \in \mathbb{P}}$ if for all $\omega \in S$ and for all $P \in \mathbb{P}$ the set Γ is authentic for P after ω whenever $\text{alph}(\pi_P(\omega)) \cap \Gamma P \cap \text{alph}[V] \neq \emptyset$.

According to this definition a proof is phase restrained if the proof receive action only induces authenticity of the set Γ if it happens within a certain phase.

Definition 8 (Durable Proof of authenticity) A pair $(\Gamma \Upsilon, \Gamma \Pi)$ where $\Gamma \Upsilon$ and $\Gamma \Pi$ is a pair of sets of send and receive proof actions of authenticity for a set Γ is a durable proof of authenticity if for every phase decomposition V_1, V_2, \dots, V_k of system S there exist two series of subsets $\Gamma S_1, \Gamma S_2, \dots, \Gamma S_k \subset \Gamma \Upsilon$ and $\Gamma P_1, \Gamma P_2, \dots, \Gamma P_k \subset \Gamma \Pi$ and a series of phase restrained proofs $\{(\Gamma S_i, \Gamma P_i)\}_{i=1, \dots, k}$ such that $(\Gamma S_1, \Gamma P_1)$ is a phase restrained proof of authenticity for Γ with respect to phase V_1 and for every $i = 2, \dots, k$ $(\Gamma S_i, \Gamma P_i)$ is a phase restrained proof of authenticity for $\Gamma S_{i-1} \cup \Gamma P_{i-1}$ with respect to phase V_i .

The intention here is to express that durable proof has capacity to exceed limitations about time and to go beyond arbitrary time bound. Since in variant with phases the concept of time is defined in terms of phases we have to somehow make use of notion of phase in order to express exceeding the time limitation.

When in mathematics we say that something is unbounded we mean that this thing is beyond every conceivable bound. So, to define property of unboundedness we need a notion of bound which can be applied to the thing. In our case this means we have to introduce the fact that the proof of authenticity can be time limited in terms of phases: this we have done with phase restrained proof of authenticity. Now we can qualify what would mean that a certain proof of authenticity is unbounded in time and we did it with above definition. The definition simply means that the durable proof of authenticity will provide for a means which will survive every conceivable phase or even every conceivable succession of phases, where the succession essentially conveys the idea of passing time and if the proof survives every conceivable succession of phases it implies that it survives practically the every conceivable time bound.

A question is why are we not satisfied with the bare fact that a sequence of actions implies passing time. In a sequence every action happens after the previous and each action obviously demands certain

amount of time to accomplish. It is true that a sequence of actions implies the idea of time, however this gives no device to process (e.g. measure) the period of time the sequence lasts. For example, the best result we can hope to produce in terms of measuring an *unlimited* time, if we were to do it without phase decomposition, is in introducing a special pseudo phase / system which resembles the idea of infinity or, additionally, by saying that this pseudo system has infinite diameter. We could in a shrewd attempt then try to define the durable proof of authenticity as original SIT proof of authenticity which holds for any system S hoping that this means also the unlimited pseudo system as introduced above. The shrewd approach is somewhat changing the conditions of the original SIT proof of authenticity definition on the fly, but what if we need to stick to a particular system (as it's usually the case)? Moreover, the pseudo or infinite diameter systems don't seem feasible: an abstract pseudo system provides no properties to relate it to concrete theory (use it in relations, operations, mappings) because it is simply a symbolic system denoting our wishful idea of an unlimited system; similarly also a system with infinite diameter implies existence of an infinitely long sequence of actions, which is a hard requirement on a rather local problems we are facing in reality. On the other hand the decomposition of a system to phases looks rather transparent: every system is possible to decompose into phases. Phase decomposition gives us the device we need to measure time: we can slice the system into phases and measure discrete time intervals in terms of those phases. Then going beyond every bound means exactly enduring throughout all the phases (of every conceivable decomposition).

There is also the dilemma of why couldn't we simply decompose the system into phases and tell that the original SIT proof of authenticity survives all those phases. The reason why not is rather technical: the original proof of authenticity doesn't exhibit explicit properties for affection to phases or for affection to time in general (except that it depends on sequences of actions, but we have seen that sequences themselves don't provide for a means to measure unlimited time). Now, if we are satisfied with saying that the original SIT proof of authenticity endures through all the phases of the system for every conceivable decomposition we should but notice that there is no property of the original SIT proof of authenticity which would be in any way affected by the decomposition. The original SIT proof of authenticity survives decomposing the system simply because it has nothing to do with decompositions.

The original SIT proof of authenticity is thus obviously invariant to phases; however, it is the same definition which is used for proofs that are not durable as well as for durable proofs. The original SIT proof of authenticity is an abstract prototype definition for all the proofs of authenticity, but we would like to derive a special definition for a durable proof of authenticity which would target the very property of durability not captured by other derived types of original SIT proof of authenticity. That would say we would like to differentiate a special definition which only holds if a proof really exhibits the durability property. The means we take for this in the above definition of durable proof of authenticity with phase decompositions are partial proofs of authenticity restrained to phases (and thus affected by phases) and the way the durability is assured is so that every phase restrained proof can proof that the send and receive actions of the preceding proof were valid at that phase to prove whatever those have been proving, be it another predecessor proof or initially the very set to durably proof the authenticity.

Variant with Real Time In this section we introduce the definition of durable proofs based on the notion of time.

Definition 9 (Action-in-Time) An action-in-time is an ordered pair (a, t) where a is an action and t is a real number denoting time parameter (e.g. a timestamp). $\Sigma[t] = \{(a, t) | a \text{ is an action and } t \in \mathbb{R}\}$ and $\Sigma[t_1, t_2] = \bigcup_{t_1 \leq t \leq t_2} \Sigma[t]$ and $\Sigma[t_1, t_2) = \bigcup_{t_1 \leq t < t_2} \Sigma[t]$. We redefine $\Sigma = \Sigma[0, +\infty)$.

Definition 10 (Time-of-Action) A function $\tau : \Sigma \rightarrow \mathbb{R}$ such that $\tau(a, t) = t$ is called time-of-action.

There is a restriction for words from Σ^* : for every $w = x_1x_2 \cdots x_n \in \Sigma^*$ where $x_i \in \Sigma$ it must hold that $\tau(x_1) \leq \tau(x_2) \leq \dots \leq \tau(x_n)$. This consequently constrains possible outcomes for concatenation operation.

Definition 11 (Restrained Proof of authenticity) A pair $(\Gamma S, \Gamma P)$ with $\Gamma S \subseteq \Sigma[t_1, t_2]$ and $\Gamma P \subseteq \Sigma[t_1, t_2]$ is time restrained proof of authenticity for a set $\Gamma \subseteq \Sigma[t_1, t_2]$ on system S with respect to $(W_P)_{P \in \mathbb{P}}$ if for all $\omega \in \Sigma[t_1, t_2]^*$ and for all $P \in \mathbb{P}$ with $\text{alph}(\pi_P(\omega)) \cap \Gamma P \neq \emptyset$ the following holds:

1. For P the set Γ is authentic after ω and
2. for each $R \in \mathbb{P}$ there exist actions $a \in \Sigma[t_1, t_2]_{/P} \cap \Gamma S$ and $b \in \Sigma[t_1, t_2]_{/R} \cap \Gamma P$ with $\omega ab \in S$.

Agent P can give proof of authenticity of Γ with respect to interval $[t_1, t_2]$ after a sequence of actions ω if 1 and 2.

The above definition is basically a rewrite of the original proof of authenticity for the variant with real time. Since every alphabet or language is constrained to a particular time interval this inherently means the original proof of authenticity is restrained to a particular time interval, which gives us the above definition. However, you should beware that the ω is taken out of $\Sigma[t_1, t_2]^*$ but the proof actions are taken out of $\Sigma[t_1, t_2]$ which means that in the point 2 in the above definition of time restrained proof of authenticity there is still a time interval from $[\tau(c), t_2]$, where c is the last action of ω , when a and b can still happen in succession.

If we release the upper bound for time in $\Sigma[t_1, t_2]$, i.e. $t_2 = \infty$, then we obtain an unrestrained proof of authenticity which we can regard as durable. Mind that in that case, as opposite to the discretionary approach of variants with phases, we don't need a special instrument for processing time, because there is no time invariance possible with the original proof of authenticity: every proof of authenticity is fundamentally associated to some time as time is factored in the very notion of action.

Definition 12 (Durable Proof of authenticity) A pair $(\Gamma S, \Gamma P)$ is durable proof of authenticity if it is time (un)restrained proof of authenticity on interval $[t_0, \infty)$ for some $t_0 \geq 0$.

2.2.5. A remark to integrity

Usually, integrity is viewed as a very important security requirement. It is certainly true that data integrity is a central security requirement. However, consider the following example. Data integrity can be provided by a checksum. Change of the data would be detected because the checksum is not correct any more. But without authenticity, i.e. without knowing who computed the checksum, the checksum mechanism is useless: A malicious agent may change the data and simply compute a new checksum. Integrity is satisfied (meaning that data has not been changed since the checksum has been computed) but the actual security requirement is not.

Thus in order to make sense, integrity of data requires some kind of anchor fixing the origin of the data that shall remain unchanged. When considering distributed systems, this anchor can only be provided by some authentic action in the past. Taking into account that integrity is implied by authenticity and that integrity without authenticity is valueless we have omitted a definition for integrity.

2.2.6. 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. 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 [4] provides this flexibility.

Various aspects are included in this definition. First, we have to consider an attacker C 's view of the sequence ω it has monitored and thus the set of sequences $\lambda_C^{-1}(\lambda_C(\omega))$ that are, from C 's view, identical. Second, C can discard some of the sequences from this set, depending on its knowledge of the system and the system assumptions, all formalized in W_C . For example, there may exist interdependencies between parameters in different actions, such as a credit card number remaining the same for a long time, in which case C considers only those sequences of actions possible in which an agent always uses the same credit card number.

So the set of sequences C considers possible after having monitored ω is reduced to $\lambda_C^{-1}(\lambda_C(\omega)) \cap W_C$. Third, we need to identify the actions in which the respective parameter(s) shall be confidential. Usually many actions are independent from these and do not influence confidentiality, thus need not be considered.

Essentially, parameter confidentiality is captured by requiring that for the actions that shall be confidential for agent C with respect to some parameter p , all possible (combinations of) values for p occur in the set of actions that an attacker considers possible. What are the possible combinations of parameters is the fourth aspect that needs to be specified, as we may want to allow C to know some of the interdependencies between parameters (e.g. C may be allowed to know that a message that was received must have been sent before).

The definition of parameter confidentiality as described in the following section captures all these different aspects.

Definition of the property We want to formalize the following property: An agent R that monitors a sequence of actions ω of a system S cannot distinguish between the possible values of a certain parameter (a certain part of the message, the agent performing the action, etc.) of a specific action or set of actions of the sequence, even if it knows the set of possible parameter values. Consider for example an application consisting of the following actions: a user requests a price for a certain service, the request is received by a service provider and then an offer for this service is sent and received. In this example, one critical parameter might be the price. The service provider might have different rates for different users and these rates can change. We assume the price is supposed to be confidential, i.e. no other agent shall be able to tell which price has been offered. In the remainder of this section the external agent (the attacker) is denoted R , the user U and the service provider SP . The actions in the system are *send-price-request*(U, SP), *rec-price-request*(SP, U), *send-offer*($SP, U, price$) and *rec-offer*($U, SP, price$). The first parameter denotes the agent executing the particular action. We assume that R can see all actions but not the parameters, i.e. R 's local view $\lambda_R(\text{send-price-request}(U, SP))$ for example is equal to *send-price-request*(U, SP), while $\lambda_R(\text{rec-offer}(U, SP, price)) = \text{rec-offer}(U, SP)$. In the sequences of actions that R considers possible after having observed ω , only the actions where a price is sent and received are of interest for the example. Thus we disregard all other actions, i.e. we map them with a suitably chosen homomorphism μ onto the empty word. From those actions not mapped onto ε , μ extracts the confidential parameter that occurs in the action. Generally not only the parameter itself but also the "type" of its occurrence has to be considered. This type can be, for example, that a certain user U has received an offer. The parameter associated with this type is the

price included in the offer. By considering only the type, actions from Σ are divided into classes the elements of which can be distinguished essentially by the parameter values. Each of these classes is represented by one type.

Hence $\mu(\lambda_R^{-1}(\lambda_R(\omega)) \cap W_R)$ is a set of sequences of actions that consist of the types of those actions that are of interest with respect to parameter confidentiality, paired with the respective parameter values being possible from R 's local view.

If Σ_t denotes the set of types of the parameter occurrences and if M denotes the set of parameter values then $\mu_{\Sigma_t, M} : \Sigma^* \rightarrow (\Sigma_t \times M)^*$ is a homomorphism. For simplicity we write μ if the related parameter set and the types are obvious. Such a homomorphism μ can be defined as follows:

$$\begin{aligned}\mu(\text{send-offer}(SP, U, \text{price})) &= (\text{Send}_{SP}, \text{price}) \\ \mu(\text{rec-offer}(U, SP, \text{price})) &= (\text{Rec}_U, \text{price}) \\ \mu(\text{send-price-request}(U, SP)) &= \mu(\text{rec-price-request}(SP, U)) = \varepsilon\end{aligned}$$

In order to explain our formalism, we use the price offer system S . Our aim is now to formalize that $\mu(\lambda_R^{-1}(\lambda_R(\omega)) \cap W_R)$ “contains all possible parameter values”.

(L, M) -Completeness For the following, we assume that R monitors all sequences of actions. Let us consider as an example the following sequence of actions:

$$\begin{aligned}\omega = & \text{send-price-request}(U, SP) \text{rec-price-request}(SP, U) \\ & \text{send-offer}(SP, U, \text{price}_1) \text{rec-offer}(U, SP, \text{price}_1) \\ & \text{send-price-request}(U, SP) \text{rec-price-request}(SP, U) \\ & \text{send-offer}(SP, U, \text{price}_2)\end{aligned}$$

Let us further assume that for $R \neq U, SP$ it shall be confidential which price was sent and received, respectively. Let $\{\text{price}_1, \text{price}_2\}$ be the set of possible prices, and Σ the set of resulting possible actions. As described above, R 's local view of this sequence is the following:

$$\begin{aligned}\lambda_R(\omega) = & \text{send-price-request}(U, SP) \text{rec-price-request}(SP, U) \\ & \text{send-offer}(SP, U) \text{rec-offer}(U, SP) \\ & \text{send-price-request}(U, SP) \text{rec-price-request}(SP, U) \\ & \text{send-offer}(SP, U)\end{aligned}$$

Now if R does not know which of the possible two parameters was sent, but does know that the same parameter that was sent was also received (specified in his initial knowledge W_R), R considers four different sequences of actions possible: two in which SP sends and U receives twice the same parameter (either price_1 or price_2), one in which first price_1 is sent and received and then price_2 , and one in which the parameters are sent and received in reverse order:

$$\lambda_R^{-1}(\lambda_R(\omega)) \cap W_R = \{ \begin{aligned} &send-price-request(U, SP)rec-price-request(SP, U) \\ &send-offer(SP, U, price_1)rec-offer(U, SP, price_1) \\ &send-price-request(U, SP)rec-price-request(SP, U) \\ &send-offer(SP, U, price_1) \\ &send-price-request(U, SP)rec-price-request(SP, U) \\ &send-offer(SP, U, price_1)rec-offer(U, SP, price_1) \\ &send-price-request(U, SP)rec-price-request(SP, U) \\ &send-offer(SP, U, price_2) \\ &send-price-request(U, SP)rec-price-request(SP, U) \\ &send-offer(SP, U, price_2)rec-offer(U, SP, price_2) \\ &send-price-request(U, SP)rec-price-request(SP, U) \\ &send-offer(SP, U, price_1) \\ &send-price-request(U, SP)rec-price-request(SP, U) \\ &send-offer(SP, U, price_2)rec-offer(U, SP, price_2) \\ &send-price-request(U, SP)rec-price-request(SP, U) \\ &send-offer(SP, U, price_2) \end{aligned} \}$$

The function μ now maps these sequences of actions onto sequences with types for the send and receive actions, each one being paired with the respective parameter. All other actions are mapped onto ε . This results in

$$\mu(\lambda_R^{-1}(\lambda_R(\omega)) \cap W_R) = \{ \begin{aligned} &(Send_{SP, price_1})(Rec_U, price_1)(Send_{SP, price_1}), \\ &(Send_{SP, price_1})(Rec_U, price_1)(Send_{SP, price_2}), \\ &(Send_{SP, price_2})(Rec_U, price_2)(Send_{SP, price_1}), \\ &(Send_{SP, price_2})(Rec_U, price_2)(Send_{SP, price_2}) \end{aligned} \}$$

If we want to describe a situation where R does not know any correlation between the parameter of a send and the respective receive action, the μ -image of the sequence of actions monitored by R contains eight different sequences of pairs (*type, parameter*) with no order on the parameters $price_1$ and $price_2$:

$$\mu(\lambda_R^{-1}(\lambda_R(\omega)) \cap W_R) = \{ \begin{aligned} &(Send_{SP, price_1})(Rec_U, price_1)(Send_{SP, price_1}), \\ &(Send_{SP, price_1})(Rec_U, price_1)(Send_{SP, price_2}), \\ &(Send_{SP, price_1})(Rec_U, price_2)(Send_{SP, price_1}), \\ &(Send_{SP, price_2})(Rec_U, price_1)(Send_{SP, price_1}), \\ &(Send_{SP, price_2})(Rec_U, price_1)(Send_{SP, price_2}), \\ &(Send_{SP, price_2})(Rec_U, price_2)(Send_{SP, price_1}), \\ &(Send_{SP, price_1})(Rec_U, price_2)(Send_{SP, price_2}), \\ &(Send_{SP, price_2})(Rec_U, price_2)(Send_{SP, price_2}) \end{aligned} \}$$

In general we have the requirement that in each group of actions that R knows to be correlated, it considers each of the parameters possible. In order to formalize this, we assign each group a number. We then built the μ -image of the sequences of actions that R considers possible after ω has happened, with the parameters being substituted by the number of the respective group they belong to. Then we check that when mapping these numbers arbitrarily onto possible parameters, this results in the μ -image of R 's inverse view of ω , i.e. we check that the μ -image is (L, M) -complete for a specific language L (containing the action types associated with numbers) and parameter set M .

For the formal definition of (L, M) -completeness, we need some notations: For $f : M \rightarrow M'$ and $g : N \rightarrow N'$ we define $(f, g) : M \times N \rightarrow M' \times N'$ by $(f, g)(x, y) := (f(x), g(y))$. The identity on M is denoted by $i_M : M \rightarrow M$, while $M^{\mathbf{N}}$ denotes the set of all mappings from \mathbf{N} to M .

Definition 13 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

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

In this definition, the set L consists of sequences of pairs (*action type, number*). The functions $f \in M^{\mathbf{N}}$ map the numbers to parameter values in M . Therefore, $(i_{\Sigma_t}, f)(L)$ consists of sequences of pairs (*action type, parameter value*). These sequences are in accordance with the correlations between parameter values defined by L . Now, K is (L, M) -complete if it consists of all possible sequences of pairs (*action type, parameter value*) derived by applying (i_{Σ_t}, f) to L for all possible mappings f from \mathbf{N} to M .

This property allows the formalization of any of the above described situations. Let us consider as an example again the sequence of actions ω .

For the set $\{Send_{SP}, Rec_U\}$ of relevant action types we now choose a numbering that assigns the same number to those actions that are correlated:

$$L_1 = \{(Send_{SP}, 1)(Rec_U, 1)(Send_{SP}, 2)\}$$

Having chosen the language L_1 in this manner and considering the set $M = \{price_1, price_2\}$ of parameter values, $\mu(\lambda_R^{-1}(\lambda_R(\omega)) \cap W_R)$ is (L_1, M) -complete if and only if

$$\begin{aligned} \mu(\lambda_R^{-1}(\lambda_R(\omega)) \cap W_R) = \{ & (Send_{SP}, price_1)(Rec_U, price_1)(Send_{SP}, price_1), \\ & (Send_{SP}, price_1)(Rec_U, price_1)(Send_{SP}, price_2), \\ & (Send_{SP}, price_2)(Rec_U, price_2)(Send_{SP}, price_1), \\ & (Send_{SP}, price_2)(Rec_U, price_2)(Send_{SP}, price_2)\} \end{aligned}$$

This exactly describes the situation in which R knows that the same price was received that was sent, but does not know which of the prices was sent. Note that if R considers more parameter values possible (i.e. if $\mu(\lambda_R^{-1}(\lambda_R(\omega)) \cap W_R)$ contains more than the above four sequences), R still does not know which parameter was sent.

If R shall not know that there is a correlation between send and receive actions, the action types have to be numbered differently: no actions are correlated. This results in

$$L_2 = \{(Send_{SP}, 1)(Rec_U, 2)(Send_{SP}, 3)\}$$

The requirement of (L_2, M) -completeness results in the above mentioned eight sequences of pairs (*type, parameter*). However, if R knows the correlation between *Send* and *Rec*, i.e. if $\mu(\lambda_R^{-1}(\lambda_R(\omega)) \cap W_R)$ contains only the four different sequences above (in which the same parameter value is sent and received), then $\mu(\lambda_R^{-1}(\lambda_R(\omega)) \cap W_R)$ is not (L_2, M) -complete: Using $f(1) = price_1$, $f(2) = price_2$ and any $f(3)$ we obtain

$$(Send_{SP}, price_1)(Rec_U, price_2)(Send_{SP}, price_1) \notin \mu(\lambda_R^{-1}(\lambda_R(\omega)) \cap W_R)$$

Thus by appropriately numbering the action types, i.e. by appropriately choosing the language L , we can formalize which correlations between actions R is allowed to know, or in other words, which sequences of actions have to be included in W_R . This gives rise to the following definition:

Let M be a parameter set, Σ a set of actions, Σ_t a set of types, $\mu : \Sigma^* \rightarrow (\Sigma_t \times M)^*$ a homomorphism, and $L \subseteq (\Sigma_t \times \mathbf{N})^*$. Then M is parameter-confidential for R after ω with respect to (L, M) -completeness if there exists an (L, M) -complete language $K \subseteq (\Sigma_t \times M)^*$ with $\mu(\lambda_R^{-1}(\lambda_R(\omega)) \cap W_R) \supseteq K$.

Instead of separately defining different languages L for different sequences ω and the resulting set of sequences $\mu(\lambda_R^{-1}(\lambda_R(\omega)) \cap W_R)$, it is sufficient to have an appropriate L and an (L, M) -complete language $K \subseteq (\Sigma_t \times M)^*$ serving for all $\omega \in S$. Using the function p_1 that denotes the projection on the first component of a tuple, we introduce the following definition:

Definition 14 Let M be a parameter set, Σ a set of actions, Σ_t a set of types, $\mu : \Sigma^* \rightarrow (\Sigma_t \times M)^*$ 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 $\mu(\lambda_R^{-1}(\lambda_R(\omega)) \cap W_R) \supseteq p_1^{-1}(p_1(\mu(\lambda_R^{-1}(\lambda_R(\omega)) \cap W_R))) \cap K$ for each $\omega \in S$.

Applying the projection p_1 and then the inverse p_1^{-1} to $\mu(\lambda_R^{-1}(\lambda_R(\omega)) \cap W_R)$ results in sequences of actions where all parameter values occur and no grouping according to correlated actions has yet taken place. The intersection with the (L, M) -complete language K removes those sequences that do not match the respective grouping. Note that in the case of no correlation between actions, $p_1^{-1}(p_1(\mu(\lambda_R^{-1}(\lambda_R(\omega)) \cap W_R))) \cap K = p_1^{-1}(p_1(\mu(\lambda_R^{-1}(\lambda_R(\omega)) \cap W_R)))$.

More generally, the above equation can be used to define, for an arbitrary language $K \subseteq (\Sigma_t \times M)^*$ with $K \supseteq \mu(W_R)$, K -completeness of $\mu(\lambda_R^{-1}(\lambda_R(\omega)) \cap W_R)$. This allows to capture more sophisticated correlations between parameters.

A different property: M -rich In some cases there is no adequate language L to describe that R cannot recognize the respective parameters used. We introduce a new example to motivate a weaker confidentiality property and we show that this weaker property may not be adequate for our previous price example. Let us consider a system which models an auction. In this system we look at the bidding phase. For simplicity we assume there are only two bidders $U1$ and $U2$. The only possible action for bidders is *bid* with the parameters *bidder* and *amount*. In the same manner as above, agent R can monitor the bidding actions. We want to model the property that R may monitor the amount which a bidder has made but is not allowed to know which bidder has made which bid. In contrast to (L, M) -completeness of the price in the previous section, R is allowed to know which bids have been made by the same bidder. For example, R may know that bids have been made alternately by two agents, but it is not allowed to know which bidder has started and which bidder has placed the winning bid. The homomorphism μ for this example can be defined as follows (for simplicity we disregard the values of the amount):

$$\mu(\text{bid}(U, \text{amount})) = (\text{Bid}, U)$$

Now we consider the following sequence of actions:

$$\delta = \text{bid}(U2, \text{amount}_1) \text{bid}(U1, \text{amount}_2) \text{bid}(U2, \text{amount}_3) \text{bid}(U1, \text{amount}_4)$$

After having monitored δ , the following sequences of parameter values are possible in R 's view of the system:

$$\begin{aligned} \lambda_R^{-1}(\lambda_R(\delta)) \cap W_R = \\ \text{bid}(U2, \text{amount}_1) \text{bid}(U1, \text{amount}_2) \text{bid}(U2, \text{amount}_3) \text{bid}(U1, \text{amount}_4), \\ \text{bid}(U1, \text{amount}_1) \text{bid}(U2, \text{amount}_2) \text{bid}(U1, \text{amount}_3) \text{bid}(U2, \text{amount}_4) \end{aligned}$$

The μ -image now extracts types and parameters that shall be confidential:

$$\begin{aligned} \mu(\lambda_R^{-1}(\lambda_R(\delta)) \cap W_R) = \{ & (\text{Bid}, U2)(\text{Bid}, U1)(\text{Bid}, U2)(\text{Bid}, U1), \\ & (\text{Bid}, U1)(\text{Bid}, U2)(\text{Bid}, U1)(\text{Bid}, U2) \} \end{aligned}$$

This knowledge of R corresponds to the confidentiality property that R cannot tell which bidder has placed which bid while knowing that $U1$ and $U2$ bid alternately. However, it is not possible to find a language L such that $\mu(\lambda_R^{-1}(\lambda_R(\delta)) \cap W_R)$ is (L, M) -complete for $M = \{U1, U2\}$. Considering the correlations between actions known to R the following language $L3$ seems to be appropriate at first sight, as R knows that bidders place their bids alternately:

$$L_3 = \{(\text{Bid}, 1)(\text{Bid}, 2)(\text{Bid}, 1)(\text{Bid}, 2)\}$$

However, for the language L_3 chosen in this manner, $\mu(\lambda_R^{-1}(\lambda_R(\delta)) \cap W_R)$ is not (L_3, M) -complete as $f(1) = f(2) = U1$ results in

$$(\text{Bid}, U1)(\text{Bid}, U1)(\text{Bid}, U1)(\text{Bid}, U1) \notin \mu(\lambda_R^{-1}(\lambda_R(\delta)) \cap W_R)$$

Nevertheless R does not know which of the two parameter values occurred: for each of the bids it considers both bidders possible. To formalize this situation we need a property that does not consider the complete possible sequences of actions from R 's local view but that considers only a "cut" through all sequences at the respective interesting actions. The following property describes the fact that from R 's local view at each separate point in the sequence of actions, each of the parameter values is possible:

$$\forall u \in p_1(\text{pre}(\mu(\lambda_R^{-1}(\lambda_R(\delta)) \cap W_R))) : \\ p_2(\text{suf}_1(p_1^{-1}(u) \cap \text{pre}(\mu(\lambda_R^{-1}(\lambda_R(\delta)) \cap W_R)))) \supseteq M$$

Starting with the sequence of actions δ , we use λ_R , λ_R^{-1} and W_R to generate the set of possible sequences of actions that are identical to δ in R 's local view and which R considers possible. From these we extract, using the function μ , the relevant types with the respective parameter values, and pre generates all possible prefixes (i.e. we cut off the last action, the second last, etc.). With p_1 we disregard the parameters having been extracted by μ . Every u in a set of sequences generated in this manner is a sequence of types that correspond to those actions in δ that we are interested in, without the respective parameter values. p_1^{-1} again adds all parameter values in all possible combinations. The intersection with $\text{pre}(\mu(\lambda_R^{-1}(\lambda_R(\delta)) \cap W_R))$ disregards those sequences that R does not consider possible because of its knowledge about correlation of actions. From each of the resulting sequences, we consider only the last element by applying suf_1 (where $\text{suf}_i(\omega)$ returns the suffix of ω with length i). p_2 then determines those parameter values that R considers possible in the respective action. The resulting set must include the complete set M of parameter values.

For the above example we get

$$p_1(\text{pre}(\mu(\lambda_R^{-1}(\lambda_R(\delta)) \cap W_R))) = \{\varepsilon, \text{Bid}, \text{BidBid}, \text{BidBidBid}, \text{BidBidBidBid}\} \\ u = \text{BidBid} \text{ for example results in} \\ p_1^{-1}(u) = \{(\text{Bid}, U1)(\text{Bid}, U1), (\text{Bid}, U1)(\text{Bid}, U2), (\text{Bid}, U2)(\text{Bid}, U1), \\ (\text{Bid}, U2)(\text{Bid}, U2)\}$$

We now intersect this set with the set of sequences of types (with parameters) that R considers possible as described above.

$$p_1^{-1}(u) \cap \text{pre}(\mu(\lambda_R^{-1}(\lambda_R(\delta)) \cap W_R)) = \{(\text{Bid}, U1)(\text{Bid}, U2), (\text{Bid}, U2)(\text{Bid}, U1)\}. \\ \text{suf}_1 \text{ reduces the resulting } (\text{type}, \text{parameter}) \text{ sequences to the respective last } (\text{type}, \text{parameter}) \\ \text{(that is, to } (\text{Bid}, U_i) \text{), and finally } p_2 \text{ extracts the parameters that } R \text{ considers possible in this } (\text{type}, \text{parameter}). \\ \text{If this set does not include } M \text{ completely then } R \text{ knows more about the parameters that are possible in this action than it should know after having monitored } \delta. \\ \text{In our example, } p_2(\text{suf}_1(\{(\text{Bid}, U1)(\text{Bid}, U2), (\text{Bid}, U2)(\text{Bid}, U1)\})) = \{U1, U2\} = M.$$

Analogously to definition 13 we give the following general definition:

Definition 15 For a given set M of parameter values and a set Σ_t of action types we call the language $K \subseteq (\Sigma_t \times M)^*$ M -rich if

$$\forall u \in p_1(\text{pre}(K)) : p_2(\text{suf}_1(p_1^{-1}(u) \cap \text{pre}(K))) \supseteq M$$

Analogously to (L, M) -completeness, we can now define a different kind of parameter confidentiality:

Definition 16 Let M be a parameter set, Σ a set of actions, Σ_t a set of types, and $\mu : \Sigma^* \rightarrow (\Sigma_t \times M)^*$ a homomorphism. Then M is parameter-confidential for R with respect to M -richness if $\mu(\lambda_R^{-1}(\lambda_R(\delta)) \cap W_R)$ is M -rich for all $\omega \in S$.

For the price-offer example explained in Section 2.2.6., the property M -rich may be too weak. Consider for example a case with only two possible prices and SP offering alternately the two different prices to U . Now, if $\mu(\lambda_R^{-1}(\lambda_R(\omega)) \cap W_R)$ is M -rich, R cannot tell which price has been offered in which action. However, as R can observe that two prices have been offered alternately, R can calculate the average price offered to U , which may be undesirable.

For a given *type* sequence, (L, M) -completeness of a language $K \subseteq (\Sigma_t \times M)^*$ exactly determines the set of $(type, parameter)$ sequences which have to be in K . This is not true in the case of M -richness: As mentioned above, the equation

$$\mu(\lambda_R^{-1}(\lambda_R(\omega)) \cap W_R) = \{(Send_P, m1)(Rec_Q, m1)(Send_P, m2), \\ (Send_P, m2)(Rec_Q, m2)(Send_P, m1)\}$$

implies M -richness of $\mu(\lambda_R^{-1}(\lambda_R(\omega)) \cap W_R)$. But also the equation

$$\mu(\lambda_R^{-1}(\lambda_R(\omega)) \cap W_R) = \{(Send_P, m1)(Rec_Q, m1)(Send_P, m1), \\ (Send_P, m2)(Rec_Q, m2)(Send_P, m2)\}$$

would imply M -richness of $\mu(\lambda_R^{-1}(\lambda_R(\omega)) \cap W_R)$. So two different languages K express the required parameter variety.

Therefore, to bridge the formal gap between (L, M) -completeness and M -richness we need to consider the family \mathcal{K} of all languages $K \subseteq (\Sigma_t \times M)^*$ which are M -rich. Now $\mu(\lambda_R^{-1}(\lambda_R(\omega)) \cap W_R)$ is M -rich if and only if $\exists K \in \mathcal{K} : \mu(\lambda_R^{-1}(\lambda_R(\omega)) \cap W_R) = p_1^{-1}(p_1(\mu(\lambda_R^{-1}(\lambda_R(\omega)) \cap W_R))) \cap K$.

Conditional confidentiality Sometimes it is not sufficient to formulate a static confidentiality property that defines the role for each agent for the entire system lifetime. In order to model a system where one agent will transfer information to another agent only after the latter proves its trustworthiness (i.e. by way of TPM-Attestation) or to model that data will be retrieved from storage only if the agent's platform is not manipulated (i.e. by way of TPM-Seals), a more flexible definition of confidentiality is required:

Definition 17 (conditional-confidentiality) For a set of confidentiality condition actions Γ_P for an agent P where $\Gamma_P \subseteq \Sigma_{/P}$, conditional confidentiality with respect to Γ_P holds for P if $\forall \omega : \Gamma_P \cap alph(\omega) = \emptyset$ implies $\mu(\lambda_P^{-1}(\lambda_P(\omega)) \cap W_P) \supseteq p^{-1}(p(\mu(\lambda_P^{-1}(\lambda_P(\omega)) \cap W_P))) \cap K$

This definition describes that an agent P it is not allowed to know the contents of a certain data-block unless it has performed an action of Γ_P .

Specifying parameter-confidentiality In order to specify parameter-confidentiality, the following information is needed:

α : the action(s) for which some parameter(s) shall be confidential,

$par(\alpha)$: the parameter(s) that shall be confidential (some parameters of the action(s) may not require confidentiality and may be known by other agents),

μ : The actions agents can learn from.

\mathcal{P} : the set of possible values for the parameter(s),

\mathcal{Q} : a set of agents that is allowed to know the parameter value(s)

L : optional, a language that describes the interdependencies between parameter values.

Γ_P : optional, a set of actions that constitute the condition under which agent P is allowed to know certain data.

2.2.7. *Enforcing certain system behaviour*

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

Definition 18 (enforce-behaviour) *For an alphabet Σ , let $L \subseteq \Sigma^*$ describe particular requirements and $S \subseteq \Sigma^*$ be the actual system. We say that L enforces its behaviour on S , denoted by $\text{enforce-behaviour}(L,S)$, if $S \subseteq L$.*

This property is particularly useful to describe work-flows where certain sequences of actions have to occur. See Section 3.1. for instantiations of this property.

2.3. Dependability requirements

The term *dependability* has many different interpretation. In the context of IT systems, dependability requirements can include the following attributes:

- Availability (readiness for correct service),
- Reliability (continuity of correct service),
- Maintainability (restore a service within a given time after failure)
- Safety (absence of catastrophic consequences on the user(s) and the environment), and
- Security.

The notion of security used in the SERENITY project is much broader than what is usually subsumed under dependability. Therefore, security and dependability issues are separated in SERENITY. Security requirements are discussed in the previous section.

Safety in the sense of the absence of catastrophic consequences will not be considered on the level of security solutions for network and devices in SERENITY. It is assumed that the hardware of the devices is safe to use. One example of safety issues for AmI devices is the problem of defective rechargeable battery packs. For mobile phones as well as for laptops cases of exploding or burning battery packs are known. The focus of the work on networks and devices in SERENITY is not on such physical safety issues. Safety-related side effects of an application will also not be covered by the solutions on the network and devices level. It has to be assumed that such side effects are either known to the designer of AmI applications or have been identified on the levels of business processes and workflows and can therefore be prevented by more refined concrete security, availability or reliability requirements.

Thus, dependability issues of devices and networks in SERENITY concentrate on availability, reliability and maintainability.

2.3.1. *Modelling a system for the specification of dependability requirements*

Many dependability properties are concerned with the duration (time) or probability of particular actions or sequences of actions. Thus, in order to be able to reason about dependability, the system specification as presented in Section 2.1. needs to include information about time and probabilities.

Modelling time A discrete model of time is sufficient for the dependability requirements in SERENITY. Real-time issues will not be considered on the level of network and devices in SERENITY. At the level of requirements specifications the expected duration of particular actions is generally not known. In contrast, the duration of actions or sequence is part of the dependability requirements specification. Therefore, it would be not useful to explicitly include time and duration into the system specification. For the specification of timing requirements we need the following information:

- A discrete *time model* needs to be defined for each system. The time model consists of two sets:
 1. The set T_S describes the available time steps for a system S . This set can for example consist of natural numbers describing system ticks, seconds, etc. or a description of time in terms of hours, minutes, seconds.
 2. The set TI_S contains the values available for the description of time intervals.
- Further, a function $duration_S : \Sigma^* \rightarrow TI_S$ provides the duration of a particular sequence of actions. Please note that it may be that $duration_S(ab) \neq duration_S(a) + duration_S(b)$.

Modelling probabilities The model for probabilities is very similar to the time model explained above. Again, the probabilities of particular actions or sequences of actions are not previously known but are part of dependability requirements to be provided by dependability patterns. In contrast to the specification of time there is no need to specify a “probabilities model” for each system. We can define that probabilities are always numbers in the interval between 0 and 1. Thus, with $Pr_S : \Sigma^* \rightarrow [0, 1]$ we denote the probability for the occurrence of sequences of actions in S .

2.3.2. Availability

Availability is the property that a system is available for use when someone needs the service. We assume that in every sequence of actions we can identify the actions where a service is requested and the actions where successful access occurs. A service is *available* if the successful access occurs within a particular duration (time-out). Parts of sequences with access requests and no success within the defined time-out duration can be regarded as failure time where the user cannot access the service. We can now define availability in terms of the percentage of time in which the service is available:

$$Availability(S, Service) = \frac{available(S, Service)}{(available(S, Service) + failure(S, Service))} \%$$

Here, *available* is the sum of all durations where the Service is available in all sequences in S while *failure* is the sum of all failure times in all sequences in S . The concrete representation of these functions depends on the particular system model.

Now, availability of a service in a system S may be specified as $Availability(S, Service) > 99,9\%$. Another aspect of availability is concerned with the *system interface robustness*. A system interface should not cause a system failure when wrong input is provided. This includes user interfaces as well as communication interfaces.

2.3.3. Reliability

Reliability is the property that a system continuously provides the correct service. One important reliability requirement can be expressed as the mean time between failures. Other reliability requirements (e.g. failure rate or failure probability) can be formalized in a similar way.

Mean time between failures (MTBF) Mean time between failures (MTBF) expresses the time a system runs without any failures that affect the use of the system (note that failures can occur that are not noticed by any user of the system and that do not directly influence the functional behaviour of a system. These failures can be neglected for the MTBF but might, nevertheless, be important for the security of the system. Consider for example Trojan horses, key-loggers, bot-nets and other malicious software that does not directly influence the reliability of the system).

For a time duration t we define the mean time between failures as

$$MTBF(S, Service, t) = \frac{\text{operating-time}(S, Service, t)}{\text{number-of-failures}(S, Service, t)}$$

where *operating-time* expresses the total sum of all possible sequences with duration t and *number-of-failures* counts the failures that occur in these sequences. This definition can be refined by taking into account the probabilities of the sequences.

2.3.4. Maintainability

Maintainability is the property that a system can be restored to provide the services within a given time after failure. Maintainability can be specified by the following requirements. Other requirements considering physical failures, recover time by repair, etc. are out of the scope of a software framework.

Recover time after failure We assume that after a system failure occurred, a particular action a cannot be executed. The *recover time after failure* is the time that is needed in the maximum to achieve a system state where a can occur without any further delay. Thus, the recover time is specified in relation to a particular action. Please note that this is not a general requirement on the occurrence of a . There may be ordinary (non-failure) system states where a cannot occur for a time that is longer as the required recover time after failure.

Impact of system reset System reset frequently occurs in modern computing devices. Many causes for system reset exist, e.g. power failure, software or hardware errors, or system maintenance and security updates. System reset can occur suddenly without user interaction or might be induced by a user of the system. In any case, the impact of the system reset on the applications and services running on the system needs to be small. The following general requirements can be identified.

1. System reset shall not result in the loss of information about ongoing services.
2. System reset shall not result in the loss of data.
3. System reset shall not cause system function degradation.

For item 1 and 2 services and data that should not be affected by the system reset can be identified relative to sequences of actions. Item 3 is more difficult, because it is concerned with the possible continuations of the system behaviour after a particular sequence of actions with a system reset. Formally this is a liveness requirement.

3. A language for the specification of S&D requirements for networks and devices

The previous chapter has introduced the basic notions that can be used for the accurate specification of S&D requirements. However, these notions are very general. Therefore, using these notions to specify more concrete S&D requirements as needed for SERENITY pattern specifications can result in very complex expressions. Such expressions can be difficult to understand and the automatic processing of these expressions by the SERENITY framework would require complex reasoning mechanisms. Consequently, the S&D requirements language for networks and devices to be used in SERENITY needs to be based on refined notions of security and dependability, describing concrete requirements. To this end, we define a simple local view of agents, specific homomorphisms that extract those actions agents can learn from that are appropriate in the context of network and devices, and particular languages that capture dependencies between actions that again are relevant in the context of network and devices. Our refined S&D requirements will then be based on these definitions. More refined requirements can easily be added for other definitions of local view, etc.

— Agents' local view

The local view of agent P is restricted to those actions P performs itself and to actions **change-context** with the context that P can see:

$$\lambda_P(a) = \begin{cases} a & \text{if } a \in \Sigma_{/P} \\ \text{change-context}(C_P) & \text{if } a = \text{change-context}(C_{P_1}, \dots, C_{P_k}) \text{ and } P \in \{P_1, \dots, P_k\} \\ \varepsilon & \text{else} \end{cases}$$

The simplest case of a local view is that an agent can only see its own actions. However, in the context of AmI where agents move and change the environments they are operating in, some actions, although not performed by the agent itself, will be noticeable. For example, an agent that leaves a WLAN area will certainly notice that the internet connection terminates. The action **change-context** captures this, hence can be seen by the agent if its own context is involved.

— Actions that can be learned from

As explained in the previous section, the homomorphism μ keeps those actions the agents can learn from, extracts the parameters that shall be confidential, and maps all other actions to the empty word. In the context of security properties on the level of communication and devices we have specifically considered confidentiality properties that refer to communication and those that refer to devices. Thus we distinguish two different homomorphisms that extract send and receive actions (μ_{com}) and store and retrieve actions (μ_{dev}).

$$\mu_{com}(a) = \begin{cases} (\text{send}(P, Q), m) & \text{if } a = \text{send}(P, Q, m) \\ (\text{receive}(P, Q), m) & \text{if } a = \text{receive}(P, Q, m) \\ \varepsilon & \text{else} \end{cases}$$

$$\mu_{dev}(a) = \begin{cases} (\text{store}(P), m) & \text{if } a = \text{store}(P, m) \\ (\text{retrieve}(P), m) & \text{if } a = \text{retrieve}(P, m) \\ \varepsilon & \text{else} \end{cases}$$

For actions with different parameters the definitions have to be adapted accordingly.

— Dependencies between actions

The language $L \subseteq (\Sigma_t \times M)^*$ describes which dependencies between actions agents are allowed to know. In the context of AmI scenarios on communication and device level we consider three different languages L :

1. L_{Zero} : Agents are not allowed to know any dependencies.
2. $L_{SendRec}$: Agents are allowed to know that what is received must have been sent.
3. $L_{StoreRetr}$: Agents are allowed to know that what is retrieved must have been stored.

Based on these definitions, the following two sections introduce the refined notions of S&D requirements. In the following it is assumed that every action is associated with exactly one agent executing this action. Further it is assumed that some actions are characterized as *send* and *receive* actions to be used for communication between agents. Names are used for actions, parameters or agents in the descriptions as placeholders for arbitrary actions, parameters and agent names.

3.1. Security requirements

3.1.1. Authenticity requirements

1. **authentic-send – Authenticity of sending for the recipient**

For a specific send action **send** by agent A and a specific receive action **rec** by agent B, **authentic-send(send,rec,B)** expresses the requirement that whenever B executes **rec** the action **send** is authentic for B corresponding to Definition 1.

2. **authentic-send-all(A,B)**

For all send actions **send** by agent A and receive actions **rec** by agent B shall the send action be authentic for the recipient B, i.e. **authentic-send(send,rec,B)** shall hold. Analogously, for all send actions **send** by agent B receive actions **rec** by agent A shall the send action be authentic for the recipient A, i.e. **authentic-send(send,rec,A)** shall hold.

3. **authentic-rec – Authenticity of receipt for the sender**

For a receive action **rec** by agent B and some action **confirm-rec** by agent A, **authentic-rec(rec,confirm-rec,A)** expresses the requirement that whenever A executes **confirm-rec** the action **rec** is authentic for A corresponding to Definition 1.

4. **authentic-local – Authenticity of local actions**

For actions **local-action** and **confirm-action** both executed by the same agent A, **authentic-local(local-action,confirm-action,A)** expresses the requirement that whenever A executes **confirm-action** the action **local-action** is authentic for A corresponding to Definition 1.

5. **authentic-remote – Authenticity of remote actions**

For an action **remote-action** by agent B and an action **confirm-action** by agent A, **authentic-remote(remote-action,confirm-action,A)** expresses the requirement that whenever A executes **confirm-action** the action **remote-action** is authentic for A corresponding to Definition 1.

3.1.2. Proof of authenticity requirements

The following requirements are based on Definition 4 that requires that proofs can always be forwarded. Analogous requirements can be defined based on Definition 5 that abstains from this requirement.

1. **non-rep-origin – Non-repudiation of origin**

Remark: Different non-repudiation requirements have to be distinguished, because the solutions have to be different. E.g. non-repudiation of origin can be achieved by digital signatures, non-repudiation of receipt might require a TTP, while non-repudiation for local actions is difficult to realise (possibility: logging).

For a send action **send** by agent A and receive action **rec** by agent B, **non-rep-origin(send,-rec,B)** expresses the requirement that whenever B executes **rec** the action **send** is authentic for B and there exist proof actions by B for action **send** corresponding to Definition 4.

2. **non-rep-rec – Non-repudiation of receipt**

For a receive action **rec** by agent B and an action **confirm-rec** by agent A, **non-rep-rec(rec,confirm-rec,A)** expresses the requirement that whenever A executes **confirm-rec** the action **rec** is authentic for A and there exist proof actions by A for action **rec** corresponding to Definition 4.

3. **non-rep-local – Non-repudiation of local actions**

For actions **local-action** and **confirm-action** both executed by the same agent A, **non-rep-local(local-action,confirm-action,A)** expresses the requirement that whenever A executes **confirm-action** the action **local-action** is authentic for A and there exist proof actions by A for action **local-action** corresponding to Definition 4.

4. **non-rep-remote – Non-repudiation of remote actions**

For an action **remote-action** by agent B and an action **confirm-action** by agent A, **non-rep-remote(remote-action,confirm-action,A)** expresses the requirement that whenever A executes **confirm-action** the action **remote-action** is authentic for A and there exist proof actions by A for action **remote-action** corresponding to Definition 4.

3.1.3. Requirements with respect to a phase

1. **auth-phase/non-rep-phase – Authenticity or non-repudiation in a particular phase**

For actions **start-phase,end-phase** and **phase-action** by arbitrary agents plus **confirm-action** by A, **auth-phase(phase-action,start-phase,end-phase,confirm-action,A)** expresses the requirement that whenever A executes **confirm-action** then **phase-action** is authentic in the phase starting with **start-phase** and ending with **end-phase** corresponding to Definition 3.

Remark: This requirement does not imply any information about the agent executing **phase-action**.

Note that it may be necessary to refine this requirement with respect to authenticity of send or receive actions or local or remote actions. More examples will provide more insight here.

2. **auth-phase/non-rep-phase – Authenticity or non-repudiation in a particular phase**

In addition to authenticity in a phase, **non-rep-phase(phase-action,start-phase,end-phase,confirm-action,A)** adds the requirement that A needs to be able to prove the occurrence of **phase-action** in the particular phase. Thus, adequate proof actions in accordance with Definition 4 are required.

3.1.4. Confidentiality requirements

1. **“General indistinguishability”** This is a very strong requirement that is useful for password or any other authorisation data. This implies that this particular data has to be indistinguishable while stored on a device as well as during transmission over any network links. This is probably only useful for the design phase. During runtime more concrete requirements taking into account the actual system are necessary.
2. **End-to-end confidentiality** This requirement is a requirement on data transfer: If data is confidential before the transfer it remains confidential (except for the recipient). No agents except those in **who** may learn the value of **data-to-be-conf** from the the actions in **actions-to-learn-from** although knowing the set **possible-values** that contains the possible values of the data to be confidential. The following more specific properties use the simplest case where the only actions agents can learn from are send and receive actions. Other properties with more actions to learn from can be specified analogously. Further, we only consider the case were there are no dependencies between actions allowed to be known and the case where the only dependency that is allowed to be known is that before receiving a message it must have been sent.

Hence the following definitions are based on the homomorphism μ_{com} and the languages L_{Zero} and $L_{SendRec}$ introduced at the beginning of this section.

Thus end-to-end-confidentiality has various different characteristics:

— **end-to-end-conf-DepZero(data-to-be-conf,possible-values,who)**

For all agents except those in **who**, **data-to-be-conf** shall be confidential. Agents only learn from send and receive actions and are not allowed to know any dependencies between actions.

— **end-to-end-conf-DepSendRec(data-to-be-conf, possible-values,who)**

For all agents except those in **who**, **data-to-be-conf** shall be confidential. Agents only learn from send and receive actions and are allowed to know the dependencies between sending and receiving of messages.

3. **end-to-end-conf-all-DepZero(A,B)**

For all send actions by agent A and respective receive actions by agent B and for all send actions by B and respective receive actions by A, the data sent shall be confidential for all other agents, i.e. **end-to-end-conf-DepZero(data-sent-by-A-or-B,possible-values, {A, B})** shall hold.

This security requirement assumes that the only actions agents can learn from are send and receive actions. It can be used if there are no dependencies between parameters allowed to be known. The following requirement can be used in case the agents are allowed to know the dependencies between sending and receiving of messages:

4. **end-to-end-conf-all-DepSendRec(A,B)**

For all send actions by agent A and respective receive actions by agent B and for all send actions by B and respective receive actions by A, the data sent shall be confidential for all other agents, i.e. **end-to-end-conf-DepSendRec(data-sent-by-A-or-B,possible-values, {A, B})** shall hold.

5. Confidentiality of data stored on a device

Nobody but the agents in the set **who** shall learn from the actions in **actions-to-learn-from** which values the data in **data-to-be-conf** stored on **device** have although knowing the set of possible values **possible-values**. The following more specific properties use the simplest case where the only actions agents can learn from are store and retrieve actions. Other properties with more actions to learn from can be specified analogously. Further, we again consider the case where there are no dependencies between actions allowed to be known and the case where the only dependency that is allowed to be known is that before receiving a message it must have been sent. Hence the following definitions are based on the homomorphism μ_{dev} and the languages L_{Zero} and $L_{StoreRetr}$ introduced at the beginning of this section.

Again, depending on existing or not existing dependencies between the parameters, we have two different characteristics of this requirement:

- **device-conf-DepZero(device,data-to-be-conf,possible-values,who)**
For all agents except those in **who**, **data-to-be-conf** shall be confidential. Agents only learn from store and retrieve actions and are not allowed to know any dependencies between actions.
- **device-conf-DepStoreRetr(device,data-to-be-conf,possible-values,who, set-of-send-rec-pairs)**
For all agents except those in **who**, **data-to-be-conf** shall be confidential. Agents only learn from store and retrieve actions and are allowed to know that the same parameter is stored that was retrieved:

6. Conditional Confidentiality

Analogously to the above we can define specific properties based on the Definition 17 of conditional confidentiality as introduced in Section 2.2.6..

- **end-to-end-conf-cond-Lzero(data-to-be-conf,possible-values,who,conditions)**
- **device-conf-cond-Lzero(device,data-to-be-conf,possible-values,who,conditions)**

3.1.5. Security requirements for enforcing behaviour

1. **access-allowed(who,device,data,access-type)** is the requirement that only agents in the set **who** can have access of the type **access-type** to **data** on **device**.

This implies that whoever tries to have access to the data has to authenticate itself as being member of **who**. Thus in the refined system that describes how this is achieved, **access-allowed** is an instantiation of **enforce-behaviour**: Whenever an access action by any agent occurs, some action must have occurred before that authenticates the agent as being member of **who**. What is considered access to a specific device depends on the **access-type**. For example, read and write access have to be distinguished and will require different solutions that comply with the requirement. We may even distinguish between different types of read access, e.g. between just copying a certain file without understanding its content, and reading a file with the consequence of understanding its content.

Definition 19 (access-allowed) Let $who = \{P_1, \dots, P_i\}$ be a set of agents, Σ a set of actions, and $access(P_i, device, data, access-type) \in \Sigma$ the action of P_i to access data on device with some access type. Let $auth(P_i, who) \in \Sigma$ be the action that authenticates P_i as being a member of who . Let further $S \subseteq \Sigma^*$ be a system. We define a language L as follows:

$$L = \Sigma^* \setminus \bigcup_{P_1, \dots, P_i} (\Sigma \setminus \{auth(P_i, who)\})^* \{access(P_i, device, data, access-type)\} \Sigma^*$$

Then **access-allowed**($who, device, data, access-type$) holds if **enforce-behaviour**(L, S) holds, i.e. if $S \subseteq L$.

2. sequence(a,b,c)

This requirement describes that if action a is followed by action c (denoted by $a < c$), an action b must have occurred in between, i.e. $a < b < c$ must hold. This can be used for example to describe the requirement that when having installed a program, before using it some license agreement must be acknowledged.

Definition 20 (sequence(a,b,c)) Let Σ be a set of actions, $a, b, c \in \Sigma$. Let $S, L \subseteq \Sigma^*$ with L defined as follows:

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

Then **sequence**(a, b, c) holds if **enforce-behaviour**(L, S) holds.

3. no-sequence(a,b)

This requirement describes that after action a , action b must not occur anymore. This can be used for example to describe the requirement that when having sold an upgrade to a program, the upgrade can not be used anymore.

Definition 21 (no-sequence(a,b)) Let Σ be a set of actions, $a, b \in \Sigma$. Let $S, L \subseteq \Sigma^*$ with L defined as follows:

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

Then **no-sequence**(a, b) holds if **enforce-behaviour**(L, S) holds.

4. bounded-no-sequence(a,b,c)

If a sequence occurs which is bounded by a starting Action a and an ending Action c (with neither a nor c as part of the sequence-body), then no b is allowed within.

Definition 22 (bounded-no-sequence(a,b,c)) Let Σ be a set of actions. Let $a, b, c \in \Sigma$. Let $S, L \subseteq \Sigma^*$ with L defined as follows:

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

Then **bounded-no-sequence**(a, b, c) holds if **enforce-behaviour**(L, S) holds.

3.2. Dependability requirements

This section provides language constructs for the specification of relevant dependability requirements as defined in Section 2.3..

1. **High-availability(S,a)** holds if $Availability(S, a) > 99,9\%$. It is assumed that a models a particular service.
2. **High-MTBF(t)** holds if the MTBF for a system is higher than t for an average running time of the system.
3. **recover-time(a,t)** holds if after every failure in all continuations of the system behaviour, action a is possible after duration t .

4. Time-Synchronization

This property describes that the internal system clocks of every agent in A is in sync with every other agent's system clock with respect to a certain variance ϵ :

Definition 23 (time-sync(A,ε)) $\forall \omega \in S \forall u, v, w \in \Sigma^* \forall P, Q \in A : uxvwyw = \omega \wedge x = thetime(P, t_1) \wedge y = thetime(Q, t_2) \rightarrow t_1 < t_2 + \epsilon$

5. Behavior-Transparency

Especially when applying dependability patterns it is often an important feature to provide transparency of the actual behavior of the system towards the client. This requirement is targeted in the areas of

- Load balancers and session failover mechanisms ensuring the uniform behavior of the server-farm
- Transparent (caching) proxies that do not alter any content being forwarded
- Power generators providing electricity in the case of a power outage.

It might also be a desirable property for security patterns that preserve the user experience. The transparency property for all of these cases can be translated into a confidentiality property, utilizing the formal methods as described in Section 2.2.6.:

Definition 24 (uniform-cluster-behavior(Server,Cluster,Client)) *The property holds if for the receive actions of a Client represented by*

$$\mu_{uni}(a) = \begin{cases} (receive(Client, m), Server) & \text{if } a = receive(Client, Server, m) \\ \epsilon & \text{else} \end{cases}$$

*and the language L_{Zero} the property **confidentiality** holds.*

This definition refers to the very strong property that a client shall never know anything about the internals of the cluster, thus the cluster having session-failover mechanisms installed. In order to provide a weaker property that only covers the loadbalancing aspect one would have to construct a language L_{LB} that allows clients to recognize steady servers per session.

6. Load Balancing

A word on load balancing:

Load balancing can be understood as a means to achieve low response-times. Therefor it does not contribute to dependability by itself, unless response-times become so high, that time-outs occur. Though in case high response times result in session-failure, it becomes an availability issue and is covered by the according property.

Nevertheless, we will provide the definition of two algorithms implementing load balancing, providing the possibility to compare various patterns with respect to this:

- **Round-Robin** denotes algorithms that forward incoming opensessions in a naive way cycling the available servers.

Definition 25 (Round-Robin-Load-Balancing(Cluster)) $\forall P, Q \in Cluster \subseteq \mathbb{P}$ and $P \neq Q : sequence(receive(P, opensession), receive(Q, opensession), receive(P, opensession))$

- **Least-Loaded** denotes algorithms that forward incoming opensessions to the least loaded server within the cluster .

Definition 26 (Least-Loaded-Load-Balancing(Cluster)) Let $Cluster \subseteq \mathbb{P}$, $P, Q, R \in Cluster$ with $P \neq Q$. Then **Least-Loaded-Load-Balancing(Cluster)** holds if **bounded-no-sequence(leastloaded(P), opensession(Q), leastloaded(R))** holds.

4. Guidelines for requirements elicitation

4.1. Initial questions for requirements elicitation

Questions 1-4 are concerned with a slightly refined view on a scenario, Questions 4 and 5 cover general requirements for devices and networks, and the remaining question covers requirements related to particular actions/parameters in the scenarios. Mainly, plain English can be used to describe the requirements.

1. **What are the ACTORS of the scenario?** Actors can be humans, devices or any other entity that can “act” in the scenario. If concrete devices and network components can be named at this level of scenario description, requirements can be formulated for these.
2. **What are the ACTIONS in the system?** For every ACTOR name the main actions in the scenario. Proposed notation: **action-name** (ACTOR, parameters). The parameters can express data, configuration, current context, network addresses, etc..
3. **What are typical sequences of actions?** One sequence might already provide a good understanding of the scenario.
4. **Are there any generally satisfied assumptions for the scenario?** These can describe properties of the scenario, devices, actors’ behaviour etc. that generally can be assumed.
5. **Are there any general requirements for devices?** This should **not** include requirements covered by the usual baseline protection (i.e. regular backup, frequent security updates, etc). Examples for “good” requirements:
 - Corporate data on John’s laptop has always to be confidential for John, i.e. nobody else is allowed to learn any information on this data.
 - A service on a particular server has to be always available (might induce a general requirement on the network connection to the server).
6. **Are there any general requirements for network communication?** Example: Every wireless communication between two devices needs message confidentiality (i.e. nobody but sender and recipient should get any additional information on the content of the message).
7. **What are the requirements for single actions or particular sequences of actions or *phases* beginning and ending with a particular action?** Security requirements can be stated by using the notions of authenticity, proof of authenticity, authenticity with respect to a phase, and confidentiality as described below. If possible, all information in the paragraphs “Input needed...” should be included. Dependency requirements and any other requirements that do not fit into these notions should be provided in plain English. Please try to give explanations and maybe reasons for the requirement and avoid to use only keywords (like “must be dependable”).

References

- [1] M. Abadi and M.R Tuttle, 1991. A Semantics for a Logic of Authentication. In *Tenth Annual ACM Symposium on Principles of Distributed Computing, Montreal, Canada*, pages 201–216.
- [2] M. Burrows, M. Abadi, and R. Needham, 1990. A Logic of Authentication. *ACM Transactions on Computer Systems*, 8.
- [3] R. Grimm and P. Ochsenschläger, 2001. Binding Cooperation, A Formal Model for Electronic Commerce. *Computer Networks*, 37:171–193.
- [4] S. Gürgens, P. Ochsenschläger, and C. Rudolph, 2003. Parameter confidentiality. In *Informatik 2003 - Teildagung Sicherheit*. Gesellschaft für Informatik.
- [5] L.C. Paulson, 1996. Proving Properties of Security Protocols by Induction. Technical Report 409, Computer Laboratory, University of Cambridge.
- [6] G. Wedel and V. Kessler, 1996. Formal Semantics for Authentication Logics. In *Computer Security - Esorics 96*, volume 1146 of *LNCS*, pages 219–241.