Threat Scenarios as a Means to Formally Develop Secure Systems

Volkmar Lotz
Siemens AG, Corporate Technology, ZT IK 3
D–81730 München
e–mail: Volkmar.Lotz@mchp.siemens.de

March 27, 1997

Abstract

We introduce a new method for the formal development of secure systems that closely corresponds to the way secure systems are developed in practice. It is based on Focus, a general-purpose approach to the design and verification of distributed, interactive systems. Our method utilizes threat scenarios which are the result of threat identification and risk analysis and model those attacks that are of importance to the system's security. We describe the adversary's behaviour and influence on interaction. Given a suitable system specification, threat scenarios can be derived systematically from that specification. Security is defined as a particular relation on threat scenarios and systems. Security relations covering different aspects as authenticity and availability are given. We show the usefulness of our approach by developing an authentic and available server component, based on standardized cryptographic protocols.

1 Introduction

When developing IT-systems for security critical applications, it is of particular importance to show that the proposed solution maintains security. Formal methods can be used to prove security on a mathematically sound basis according to the underlying semantic model, provided an appropriate formalization of security is given. However, there is no general notion of security: for each application, different aspects of security, as confidentiality, authenticity/integrity or availability, may be relevant. Though abstract security policies may be defined, the concrete security requirements are heavily influenced by the kind of attacks that are expected for the given system and the application domain.

Informal approaches that have been shown useful in practice are therefore based on threat identification and risk analysis, where the system and its environment are investigated in detail in order to determine the kind of possible attacks, their probability, and the loss in case of the attack being performed. Thus, critical system components are identified for which the associated risk cannot be tolerated, leading to application specific security requirements. [10] gives an overview of typical requirements on several security application domains. In general, mechanisms as for example access control, encryption or authentication protocols ([8]) have to be implemented to ensure security. It is the system designer's task to show that a specific set of mechanisms is suitable to meet the security requirements.

A formal method for the development of secure systems that is intended to be supportive in practice should be based on the above considerations. In particular, it should employ a definition of security that is independent of security mechanisms and is therefore suitable to show the effectiveness of a mechanism. It should allow the formalization of individual security notions. Additionally, and probably most important with respect to practice, such a method should offer the opportunity of integrating security analysis and functional system development by providing a clear formal relationship between security analysis results and system design specifications. The latter can be achieved by using a general-purpose system design and verification method.

Formal methods achieving all these goals are currently not available. Though many approaches have been proposed during the last twenty years, ranging from formal security models ([1, 9, 25], to mention but a few) to authentication logics ([7]) and other special-purpose protocol analysis techniques ([17, 24]), they all lack the desired flexibility and correspondence to system development. Recent approaches employing process algebras, CSP in particular, ([13, 20, 16, 22, 23]) come closer to our goals, but still are not completely satisfactory, as will be discussed later on.

In this paper, we introduce a new formal method for the development of secure systems that is intended to meet all of the requirements mentioned above. Since we are mainly interested in applications of communication systems, we utilize a general-purpose approach to the design and verification of distributed, interactive systems. Focus ([3, 5, 6]) models agents by stream processing functions and is compositional with respect to refinement. In our approach threat analysis results in the definition of threat scenarios. They are specified in Focus and can be easily derived from a system specification. Security analysis is then performed by checking the relationship between threat scenario and system specification. If the security relation holds, the threat scenario can be dropped, and system development proceeds as usual. Because of compositionality, further system refinements are secure with respect to the initial threat scenario.

Section 2 gives a brief overview of the Focus method and its basic notions. The properties of the semantic model of Focus are exploited in Sect. 3 to define threat scenarios and several notions of security that correspond to different seurity aspects. Using transmission media and typical attacks on them as example, we demonstrate how threat scenario templates can be defined. The usefulness of our approach is shown by example in Sect. 4, where we analyse a system utilizung a simple protocol based on ISO 9798-2 with respect to authenticity. It turns out, that, depending on protocol embedment, authenticity is achieved at the expense of losing availability if an attack occurs. Thus, a protocol variant is specified that considers timing aspects and preserves availability in case of the adversary obeying certain fairness conditions. In Sect. 5 we compare our approach to the advanced methods mentioned above.

2 System Specification and Development with Focus

In the following, we give a short introduction to the basic notions of Focus. We define the concepts and notations that are used in the remainder of the paper. For further reading we refer to [3] and [5]. The reader is expected to be familiar with set theory. We use N to denote the set of natural numbers, and $\mathsf{B} = \{0, 1\}$ to denote the set of bits. $\mathcal{P}(M)$ denotes the powerset of a set M.

2.1 Streams

In Focus, systems are viewed as consisting of components that communicate asynchronously with each other and their environment via named channels. The communication interface of a component is given by a set of (named) input and output channels. We will define the behaviour of a component by means of a mapping between input histories and output histories, thus describing the complete lifecycle of a system component. Communication histories of channels are modelled by *streams* of messages, where a stream is defined to be a finite or infinite sequence of messages. Given a set of messages M, we define $M^{\overline{\omega}}$, $M^{\overline{*}}$ and $M^{\overline{\infty}}$ to denote the set of streams, finite streams and infinite streams of messages from M, respectively. We have $M^{\overline{\omega}} = M^{\overline{*}} \cup M^{\overline{\infty}}$.

Streams can be viewed as functions mapping natural numbers to messages. For example, a finite stream $s \in M^{\overline{*}}$ of length $n \in \mathsf{N}$ is an element of the function space $[1..n] \rightarrow M$. With $\mathsf{dom}.s$ and $\mathsf{rng}.s$ we denote the domain and the range, respectively, of a function modelling a stream.

Let $\langle\rangle$ denote the empty stream, which is the unique finite stream that contains no messages, and $\langle m_1, m_2, \ldots, m_n\rangle$ denote the finite stream containing the n messages m_1, m_2, $\ldots$, m_n. We utilize a number of operations on streams:

- $s \frown t$ denotes the concatenation of two streams s and t. $s \frown t$ yields the stream that starts with s and proceeds with the elements of t, if s is finite. If $s \in M^{\overline{\infty}}$, we have $s \frown t = s$. We also use the concatenation operator for appending a single message to a stream and write $m \frown s$ instead of $\langle m\rangle \frown s$.

- $\#s$ denotes the length of a stream s with $\#s = \infty$ if $s \in M^{\overline{\infty}}$ and $\#s = n$ if $s = \langle m_1, \ldots, m_n\rangle$. Note that we also use the operator $\#$ to denote the number of elements of a set. This is not expected to cause confusion, since its interpretation will always be clear from the context.

- $A \copyright s$ denotes the stream generated from s by filtering away all elements not in A.

- For $s \in M^{\overline{\omega}}$ and $i \in \mathsf{N}$, $s.i$ denotes the i-th element of a stream s, if $i \leq \#s$. Otherwise, $s.i$ is undefined.

- $s \sqsubseteq t$ denotes the prefix relation on streams. We have $s \sqsubseteq t$ if and only if $\exists\, r \in M^{\overline{\omega}} : s \frown r = t$.

- $s|_i$ denotes the prefix of length i of a stream s, if $i \leq \#s$, otherwise it yields s.

- $\mathsf{map}(s, f)$ for a stream $s \in M^{\overline{\omega}}$ and a function $f : M \rightarrow A$, A being an arbitrary set, yields the stream resulting from applying f to all elements of s.

- s^n denotes the n-time iteration of the stream s. We have $s^0 = \langle\rangle$ and $s^{n+1} = s \frown s^n$. When applying the iteration operator to an explicitly given one-element stream, e.g. $\langle a\rangle$, we often leave out the delimiting brackets and write a^n instead of $\langle a\rangle^n$.

Some of the above operators are overloaded to tuples of streams in a straightforward way. In particular, $\#(s_1, \ldots, s_n) = min\{\#s_1, \ldots, \#s_n\}$ yields the length of the shortest stream in $(s_1, \ldots, s_n)$, and $A \copyright (s_1, \ldots, s_n) = (A \copyright s_1, \ldots, A \copyright s_n)$ filters each stream of $(s_1, \ldots, s_n)$ with respect to A. We use the operator $(A_1 \times \ldots \times A_n) \copyright (s_1, \ldots, s_n)$ to denote the substream of those $(s_1.i, \ldots, s_n.i)$ that are elements of $A_1 \times \ldots \times A_n$. To select the i-th element of a tuple, we use the projection function Π_i.

We use $s \trianglelefteq t$ ("s is a substream of t") for two streams s and t to denote the substream predicate, which is formally defined by $s \trianglelefteq t \equiv \exists h \in \mathsf{B}^{\overline{\omega}} : \mathrm{sel}(t, h) = s$ with sel being defined by $\forall x \in M^{\overline{\omega}}, h \in \mathsf{B}^{\overline{\omega}} :\ \mathrm{sel}(x, h) = \Pi_1((M \times 1)\textcircled{c}(x, h))$.

2.2 Timed Streams

If system behaviour depends on timing aspects, we need to be able to model the progress of time in order to describe and analyse them. For that purpose, we use so-called *timed streams*. In timed streams, the special symbol $\surd$ ("tick"), which is not an element of M, occurs. Each occurrence of $\surd$ denotes that a time unit of a particular length has passed. Messages occurring between two successive ticks are assumed to be communicated within the same time unit. Since time never halts, each infinite timed stream contains infinitely many ocurrences of $\surd$. By M^ω, M^* and M^∞ we denote the set of timed streams, finite timed streams and infinite timed streams of messages of M, respectively. We have $M^\omega = M^* \cup M^\infty$.

For timed streams, we may use all of the operators defined on (untimed) streams, with ticks interpreted as ordinary messages. Moreover, we use $s{\downarrow_j}$ to define the least prefix of S that contains j occurrences of $\surd$. If a timed stream s models a particular communication channel within a system, $s{\downarrow_j}$ describes the history of that channel up to the j-th time unit. The part of a stream beginning right after the j-th time unit is denoted by $s{\uparrow_j}$ and formally defined by $s{\uparrow_0} = s$ and, if $j > 0$, $\langle\rangle{\uparrow_j} = \langle\rangle$, $(m \frown s){\uparrow_j} = s{\uparrow_j}$, and $(\surd \frown s){\uparrow_j} = s{\uparrow_{(j-1)}}$. By $\mathsf{tm}(s, j)$ we denote the time unit at which the jth non-tick message occurs.

Abstraction from time is denoted by $\bar{s}$, where $\bar{s}$ results from s by removing all ocurrences of $\surd$. We further define a timed substream predicate $s \trianglelefteq_t t$ defining that s is a substream of t, such that each message of s occurs within the same time unit as it occurs in t. It is formally defined by $s \trianglelefteq_t t \equiv \exists h \in \mathsf{B}^{\overline{\omega}} : \mathrm{tsel}(t, h)) = s$ with tsel being defined by $\mathrm{tsel}(\langle\rangle, h) = \langle\rangle$, $\mathrm{tsel}(\surd \frown t, h) = \surd \frown \mathrm{tsel}(t, h)$, $\mathrm{tsel}(m \frown t, 0 \frown h) = \mathrm{tsel}(t, h)$, and $\mathrm{tsel}(m \frown t, 1 \frown h) = m \frown \mathrm{tsel}(t, h)$.

2.3 Stream Processing Functions

Focus models the bahaviour of deterministic system components by stream processing functions mapping the component's input history channels to its output history channels. In order to distinguish channels, stream processing functions usually work on named stream tuples instead of simple stream tuples. We define named stream tuples by assigning names to the input and output channels of a component, and define a mapping $\alpha \in Q \to M^\omega$, provided a set of channel identifiers Q is given. The operators on stream tuples that have been introduced so far are overloaded to named stream tuples, if necessary. In particular, time abstraction is lifted to named stream tuples, and denoted by $\bar{\alpha}$ for a named stream tuple α. If $Q \cap P = \emptyset$, we define $\alpha \uplus \beta$ to denote the union of the named stream tuples described by α and β. Formally, $\alpha \uplus \beta$ is the element of $Q \cup P \to M^\omega$ such that $c \in Q \Rightarrow (\alpha \uplus \beta)(c) = \alpha(c)$ and $c \in P \Rightarrow (\alpha \uplus \beta)(c) = \beta(c)$.

Moreover, we use $\vec{Q}$ as a shorthand for $Q \to M^\omega$. In Sects. 3 and 4 we often identify streams and channel names, if this is expected not to cause confusion.

We model a deterministic component C with input channels I and output channels O by a function $\tau \in \vec{I} \to \vec{O}$ that maps communication histories for the input channels to communication histories for the output channels.

To correctly reflect the behaviour of real-life components, we require for each stream-processing function modelling a component, that its output at any time j is completely determined by its input received so far, which means up to time j. If additionally a possible delay of the component is considered, requiring the output at time $j+1$ being completely determined by the input up to time j, we call the function *strongly pulse driven.* The requirements on strongly pulse driven functions τ are formally described by

$$\alpha\downarrow_j = \beta\downarrow_j \Rightarrow \tau(\alpha)\downarrow_{j+1} = \tau(\beta)\downarrow_{j+1} \ .$$

The arrow $\overset{s}{\rightarrow}$ is used to model domains of strongly pulse driven functions.

2.4 Composition

Strongly pulse driven functions, and thus deterministic system components, can be composed using a number of different composition operators. For the outline of our approach, we need sequential composition, parallel composition, and feedback, which are depicted in Fig. 1 below.

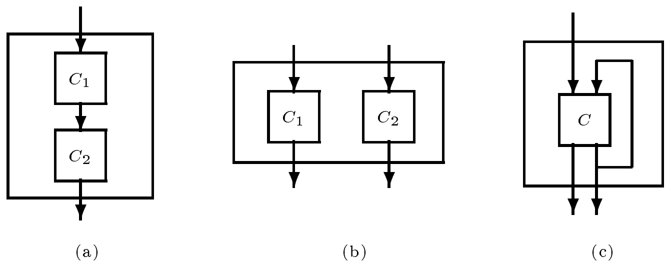

Figure 1: Composition: (a) sequential, (b) parallel, (c) feedback

Given two strongly pulse driven stream processing functions $\tau_1 \in \vec{I_1} \overset{s}{\rightarrow} \vec{O}_1, \tau_2 \in \vec{I_2} \overset{s}{\rightarrow} \vec{O}_2$, we use the operator " $\succ$ " to denote sequential composition if $O_1 = I_2$, and the operator "$\|$" to denote parallel composition if $I_1 \cap I_2 = O_1 \cap O_2 = \emptyset$. Formally, we have

$$\begin{aligned}(\tau_1 \succ \tau_2)(\alpha) &\overset{\text{def}}{=} \tau_2(\tau_1(\alpha)) \ , \\ (\tau_1 \parallel \tau_2)(\alpha) &\overset{\text{def}}{=} \tau_1(\alpha|_{I_1}) \uplus \tau_2(\alpha|_{I_2}) \ .\end{aligned}$$

where $\alpha|_Y$ denotes the restriction of the named stream tuple α to those channels contained in Y. The functions resulting from sequential and parallel composition of strongly pulse driven stream-processing functions are strongly pulse driven as well ([6]).

Given $\tau \in \vec{I} \overset{s}{\rightarrow} \vec{O}$ we define feedback by identifying a subset of τ's output channels with a subset of τ's input channels. Let $X \subset O$ and $r \in X \rightarrow I$ be a bijection that associates a subset of τ's input channels with X. We then define $\mu_X(\tau) \in (I \setminus \vec{r}(X)) \rightarrow \vec{O}$ recursively by

$$\mu_X(\tau)(\alpha) = \beta \qquad \text{where} \quad \beta = \tau(\alpha \uplus \beta|_{r(X)}) \ .$$

Because of the properties of strongly pulse driven stream-processing functions, it can be shown that for each α there is a unique β that satisfies the above equation. Moreover, $\mu_X(\tau)$ is itself strongly pulse driven ([6]).

So far we have introduced deterministic components, modelled by a single stream processing function, and their composition. In order to model nondeterministic network components as well, we now define a more general model, where components are modelled by sets of stream processing functions, with this set being a singleton, if the component is deterministic. Each function of the set describes a possible behaviour of the component. For a more operational view, we also introduce the notion of an input/output-behaviour describing a pair consisting of a particular input stream and a possible corresponding output stream. For a component $C \subseteq \vec{I} \overset{s}{\to} \vec{O}$ we formally define the set $C_{i/o}$ of input/output-behaviours by

$$C_{i/o} = \{(\alpha, \beta) \mid \exists \tau \in C : \tau(\alpha) = \beta\}.$$

The composition operators for stream processing functions are lifted uniformly to (nondeterministic) components. If C, C_1 and C_2 are appropriately defined, we have

$$\begin{array}{rcl} C_1 \succ C_2 & = & \{\tau \in \vec{I} \overset{s}{\to} \vec{O} \mid \forall \alpha : \exists \tau_1 \in C_1, \tau_2 \in C_2 : \tau(\alpha) = (\tau_1 \succ \tau_2)(\alpha)\} \ , \\ C_1 \parallel C_2 & = & \{\tau_1 \parallel \tau_2 \mid \tau_1 \in C_1 \wedge \tau_2 \in C_2\} \ , \\ \mu_X(C) & = & \{\tau \mid \forall \alpha \in (I \setminus \vec{r}(X)) : \exists \tau' \in C : \tau(\alpha) = \mu_X . \tau'(\alpha)\} \ . \end{array}$$

The specific kind of the definitions for sequential composition and feedback is provided in order to achieve full abstractness of the semantic model, see [5] and [6] for details.

2.5 Specifications

A component, semantically defined as a set of stream processing functions, can be specified by describing its communication interface (the input and output channels) and by stating properties of these functions. For example, a specification may describe a component in a state transition style, defining the state space and the state transitions allowed, or in a relational style by a set of arbitrary predicate logic formulæ. Focus provides a number of different specification formats. For our purposes, we are particularly interested in time-independent (ti) and time-dependent (td) specifications. Let I be a set of input channel names and O be a set of output channel names. The two specification formats are syntactically given by

$$S \equiv (I \rhd O) \overset{\mathsf{ti}}{::} R \ ,$$
$$S \equiv (I \rhd O) \overset{\mathsf{td}}{::} R \ ,$$

where S is the name of the specification, and R is a predicate logic formula with elements of I and O as its only free variables. Semantically, a specification is interpreted to describe the set of strongly pulse driven stream processing functions that "satisfy" R.

To formally define the semantics of a specification, we first define what it means for a named stream tuple to satisfy a predicate: For any named stream tuple $\alpha \in C \to M^{\infty}$ and formula P, whose free variables are contained in C, we define $\alpha \models P$ to hold iff P evaluates to true when each free variable c in P is interpreted as $\alpha(c)$. We then define the denotation of the time-independent and time-dependent specification format by

$$[\![\ S \ ]\!] \overset{\mathsf{def}}{=} \{\tau \in \vec{I} \overset{s}{\to} \vec{O} \mid \forall \alpha : \overline{(\alpha \uplus \tau(\alpha))} \models R\} \ ,$$

$$[\![\ S \ ]\!] \overset{\mathsf{def}}{=} \{\tau \in \vec{I} \overset{s}{\to} \vec{O} \mid \forall \alpha : (\alpha \uplus \tau(\alpha)) \models R\} \ ,$$

respectively. Note the use of the time abstraction operator for named stream tuples in the first line. For each time-independent specification, there is an equivalent time-dependent specification, resulting from substituting streams with their time abstractions.

Specifications can be composed using the same composition operators as defined for components. Since specifications describe components, the semantics of composite specifications is straightforward. Composite specifications can be syntactically given in an operator style, using the composition operators, or in a constraint style, using equations on named channels and renaming. Due to its better readability, the constraint style is often preferred in practice.

2.6 Refinement

When formally developing systems, the notion of refinement plays a central role. Focus offers a number of refinement techniques for components and specifications ([4]), of which only behaviour refinement is of interest for the following exposition. With respect to behaviour refinement, a system defined by a specification T is said to refine a system given by a specification S, if each function modelling a behaviour of T also describes a behaviour of S. If T refines S, we write $S \rightsquigarrow T$ and formally define

$$S \rightsquigarrow T \equiv [\![\, T \,]\!] \subseteq [\![\, S \,]\!] \ .$$

In order to prove that T is a refinement of S, it suffices to show that $R_T \Rightarrow R_S$.

3 System Security

3.1 Development of Secure Systems

As already stated in the introduction, the development of secure systems cannot be discussed without referring to general system development activities. As the key observation we notice that system development, starting from a requirement specification, goes through several design steps, in each of which the system is described on a less abstract level. We thus yield a sequence of design specifications S_1, S_2, $\dots$, S_n, where each S_i, $i \in \{2, \dots, n\}$ is a refinement of S_{i-1}. Since each refinement may introduce new components possibly being subject to an attack or specify new data types inducing additional security requirements, security analysis has in general to be performed at each single design step. With respect to a given design specification S_i, it is itself done in a stepwise manner, as depicted in Fig. 2. It is guided by a set of global security requirements, which, for example, describe the relevant security aspects and the kind of information considered to be security relevant. Global security requirements are often given in form of a system security policy. In general, S_i is not secure and has to be modified by introducing security mechanisms which counter those threats that have been identified as critical. The system resulting from this modification should be a refinement of S_i, since suitable security mechanisms are expected not to affect the specified system behaviour. Constructing a secure system is an iterative process, since security mechanisms, as other refinements performed within system development, introduce new components and/or data to the system which may themselves be subject to attack and have to be secured by further mechanisms. For example, considering a cryptographic mechanism that relies on secret keys, we need a mechanism to keep these keys confidential.

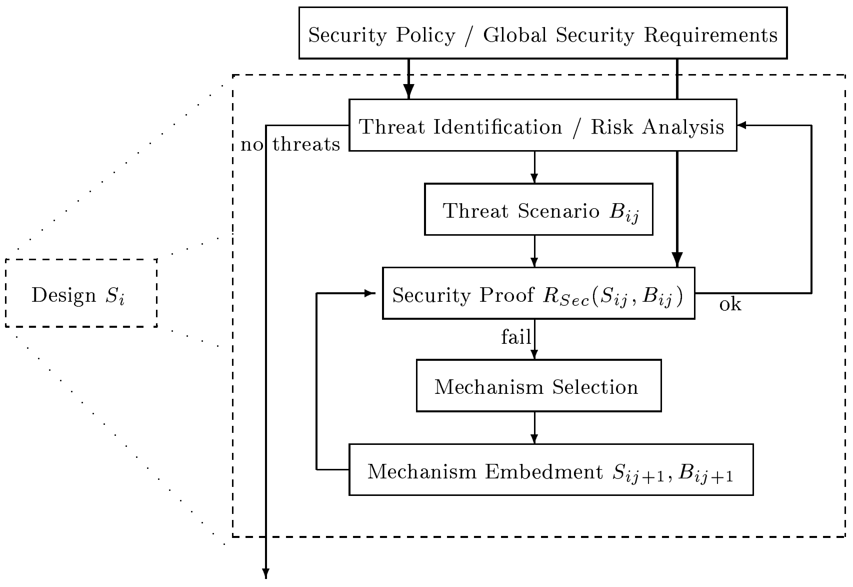
Security Policy / Global Security Requirements
Threat Identification / Risk Analysis
no threats
Threat Scenario B_{ij}
Design S_i
Security Proof $R_{Sec}(S_{ij}, B_{ij})$
ok
fail
Mechanism Selection
Mechanism Embedment S_{ij+1}, B_{ij+1}

Figure 2: Security Specific Development Steps

The single analysis steps, as shown in Fig. 2, are described as follows. Let $S_{i0} = S_i$ and $S_{ij}, j \in \{1, \ldots, m\}$, denote the system specifications resulting from each iteration within the security analysis of S_i.

1. **Threat Identification and Risk Analysis.** This is an application specific task that has to be carried out each time a security analysis is to be performed. Though classes of possible threats can be defined with respect to component types and application domains, the actual assessment of threats and associated risks heavily depends on the given specification S_{ij}. For example, if transmission media are considered, the associated risk depends, among others, on whether they are located in a secure or in a public area. Threat identification and risk analysis results in a classification of system components with respect to their criticality, and a description of the attacks that critical components may be subject to. Threat descriptions are concrete in the sense of referring to particular system components, and multiple occurences of the same kind of threat are possible (for example, if there are several communication links that are assumed to be eavesdropped).

2. **Definition of Threat Scenario.** The results of threat identification and risk analysis are used to specify a formal threat scenario B_{ij}, in which critical components are replaced by subsystems that specify the relevant attacks. Thus, B_{ij} models the system behaviour in a situation where all of the relevant attacks occur, which is the worst case with respect to security. Obviously, B_{ij} is not necessarily a refinement of S_{ij}.

3. **Security Proof.** In order to proceed with system development with respect to functional requirements, we have to show that S_{ij} is secure, which is performed by proving that the security property, describing those deviations in system behaviour being permitted in case of an attack (and thus being a relation between system specification and

threat scenario), holds with respect to S_{ij} and B_{ij}. The concrete structure of the security property depends on the security policy and the security requirements, see Sect. 3.3 for details. If the proof fails, appropriate mechanisms have to be selected, otherwise it has to be checked whether the mechanisms introduced so far give rise to new relevant threats (i.e. return to step 1).

4. **Selection or Development of Mechanisms.** During this activity, suitable security mechanisms are selected or developed, where "suitable" means that the mechanisms are able to counter the threats as well as that they satisfy further criteria, including non-technical ones as, for example, cost and performance.

5. **Mechanism Embedment.** S_{ij} is extended by a specification of the selected mechanisms. We yield a system specification S_{ij+1} and, implicitly, a refined threat scenario B_{ij+1}. It has to be shown that S_{ij+1} is a refinement of S_{ij}. Next, the security proof (Step 3) has to be repeated with j replaced by $j + 1$.

The process is finished with a secure system S_{im} at design step i, if risk analysis does not identify further threats that have to be countered, the remaining threats are countered by non-technical mechanisms that are beyond the scope of our approach, or the remaining risk will be tolerated. Thus, step 1 must always follow step 3, which ensures that new threats resulting from the introduction of mechanisms are always considered. However, it often turns out to be useful to already include such new threats in the construction of B_{ij+1}, which, for example, is done in Sect. 4. Additionally, in most cases it is obvious that S_i is not secure, which allows to omit step 3 in the first iteration.

Our approach aims at the formal foundation of the development steps described above. However, risk analysis and selection of mechanisms are excluded, since they heavily depend on non-technical arguments and thus are out of reach of formal treatment. Since all of the formal work is performed within the Focus framework, at each time of security development there is a unique relationship to system development according to its functional specification. However, methodological issues of integrated functional and security development are beyond the scope of this paper, and further work will be dedicated to this subject.

3.2 Threat Scenarios

A threat scenario is a modification of a system specification that describes a situation in which the system is attacked by an adversary, according to the results of threat identification and risk analysis. In most application cases, the threat scenario can be derived systematically from the system specification: threat identification and risk analysis are typically performed on the basis of an architectural view of the system, which means that we have a compositional specification as starting point of security considerations. For each of the components, it can then be determined, whether it is likely to be subject to adversary actions. In the derivation of a threat scenario, the critical components will then be replaced by specifications modelling the adversary's influence on them.

Candidates for critical components can often be defined on the basis of an analysis of the application domain and the type of the component, or its role within the system specification. This offers the opportunity of defining templates describing abstract attacks on the component types of interest. Using instantiations of these templates for the modification of critical components identified by risk analysis, application specific threat scenarios can be easily constructed. Note that not necessarily each of the components of a given type has to be replaced, but if risk analysis leads to a specific component of that type being classified as critical, the template can be used.

In distributed communication systems and networks, it is mainly the communication medium rather than the communicating entities (users or computer systems) that are considered to be at risk (imagine logical communication channels being implemented by using public telephone lines). Therefore, in order to perform a risk analysis reasonably, we require the specification to explicitly model media as network agents, using an appropriate level of abstraction. However, this does not seem to cause problems in practice: if the medium is subject to further development, for example, if it is going to be implemented by a protocol working on an unreliable physical medium, it will be explicitly specified, otherwise it can be simply modelled by an agent behaving like the identity on its input. In the following, we provide a template for the construction of threat scenarios describing attacks on communication media. Given the results of the threat identification and risk analysis for a particular link of the system to be secured, the template can be easily instantiated, leading to an appropriate threat scenario for the given link. This will be demonstrated in Sect. 4.

Suppose M being a set of arbitrary messages, and MD being the specification of a medium transmitting messages of M, formally defined by

$$\mathrm{MD} \equiv (i : M \rhd o : M) :: \quad \mathrm{R}_{\mathrm{MD}} \ ,$$

with R_{MD} being an arbitrary predicate describing the communication behaviour of MD. If risk analysis identifies MD as critical, in the worst case an adversary is able to eavesdrop communication as well as to influence the transmission behaviour of the channel. Such an attack can be modelled by a network as depicted in Fig. 3, which is to replace MD in the threat scenario construction.

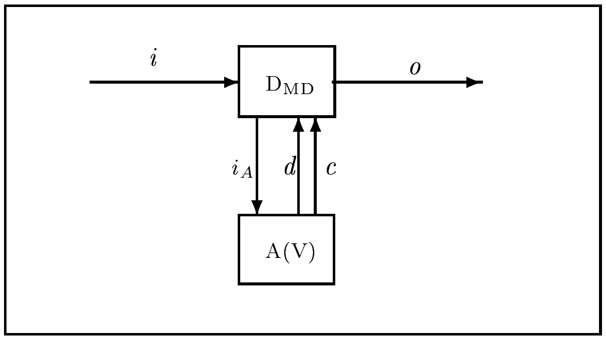

Figure 3: Threat scenario for communication channels

The threat scenario template is based on an explicit model of the adversary, together with the initial information available to her and the set of functions she can use to compute new information. As in [17] and [24], we use an explicit model of the adversary's influence on communication, based on the semantic model of FOCUS: the "data flow component" D_{MD} specifies how the adversary influences the behaviour of the transmission medium. For example, the adversary may insert or delete messages. Obviously, the specification of D_{MD} has to take into account properties of the medium MD, indicated by the index MD. A formal specification of the threat scenario $\mathrm{MD}_{\mathrm{Thr}}$, an instance of which is to replace each specification of a critical medium of the system analysed is given below. For better readability, the specification is given in constraint style.

$$\mathrm{MD}_{\mathrm{Thr}} \equiv (i : M \rhd o : M) ::$$

$$(i_A, o) := \mathrm{D}_{\mathrm{MD}}(i, d, c), \quad (d, c) := \mathrm{A(V)}(i_A) \ .$$

The two components represent the basic parts of the threat scenario specification: D_{MD} models the influence on communication with output channel i_A modelling those messages that leak through the medium and o modelling the output of the medium after possibly being modified by the adversary, and $\mathrm{A(V)}$ describes the adversary's abilities to generate new messages. Let U be a set of values, elements of which the adversary may use to perform her attacks. $V \subseteq U$ represents the set of values that are initially available to the adversary. Each time the adversary eavesdrops a message sent by a client, this set of values is extended according to the contents of this message and the set of functions the adversary may use to compute new values from already known ones. Let $F \subseteq (\bigcup_{n \in \mathsf{N}} U^n \rightarrow U) \times \mathsf{N}$ be a set of functions together with their arities that operate on messages, formally, if $n \in \mathsf{N}$ and $(f, n) \in F$, then $f \in U^n \rightarrow U$. The set of new messages C_F the adversary may get by stepwise computation from V using functions from F is then given by

$$C_F(V) = \bigcup_{n \in \mathsf{N}} C_F^N(n, V) \ ,$$

where $C_F^N(0, V) = V$ and $C_F^N(m+1, V) = \{x \in U \quad | \quad \exists (f, n) \in F, x_1, \ldots, x_n \in C_F^N(m, V) : x = f(x_1, \ldots, x_n)\}$.

Note that we are only interested in values satisfying the type constraints on MD's interface, since other values do not help the adversary in compromising the system. The formal specification of the adversary is given by

$$\mathrm{A(V)} \equiv (i_A : M \rhd d : M, c : C) \overset{\mathsf{ti}}{::}$$

$$\exists f : \quad d = f(\mathrm{V}, i_A)$$

$$\text{where} \quad \forall \quad W \subset U, i \in M^{\overline{\omega}}, i_0 \in M : \exists d_0, d_1 \in M^{\overline{*}} \cap W^{\overline{*}} :$$

$$\begin{array}{l} f(W, \langle\rangle) = d_0 \\ f(W, i_0 \frown i) = d_1 \frown f(C_F(W \cup \{i_0\}), i_A) \end{array}$$

Whenever the adversary is able to eavesdrop a message from i, modeled by the output i_A of D_{MD}, the set of messages known to her will be updated according to the functions in F. At any point, the adversary may output finitely many fraudulent messages taken from the set of values known to her at that point, described by the finite streams d_0, d_1. These messages are used to influence communication, e.g. by inserting them. The complete possibly infinite stream of fraudulent messages issued by the adversary is modeled by d. In some applications, it may turn out to be necessary to explicitly specify the influence of the adversary on the legitimate entity's communication, for example by determining the point of time at which a fraudulent message is inserted. We use c to model this kind of control, where data from a set of controls C are issued. Typically, we have $C = \mathsf{B}$. Within the template, we do not impose further restrictions on c, however, in an instantiation of the template further constraints can be introduced.

In our template for attacks on communication channels, the data flow component D_{MD} is not further specified, since the adversary's influence on communication is considered to be application specific. However, the syntactical interface of D_{MD} (legitimate messages on i, fraudulent messages on d, and controls on c) allows all kinds of possible attacks, as for example listed in [18], to be specified. Often, reliability aspects of the medium and specific attack descriptions can be separated, leading to a simple structure of D_{MD} with respect to its parameter MD:

$$\mathrm{D}_{\mathrm{MD}} \equiv (i, d, c \rhd i_A, o) :: \quad \mathrm{D}' \succ (\mathrm{ID} \parallel \mathrm{MD})$$

for some D' (with ID denoting the identity component, applied to input i_A). If, for example, the adversary may only insert new messages, without infinitely blocking legitimate messages, but is not able to determine the position where to insert, D' is given by the specification of the fair merge agent in [3], with the interface being adjusted.

This concludes the specification of the threat scenario template for transmission media. Its parameters are given by the adversary's initial set of values V, the set F of functions available, the type of control messages C, and the specification of the data flow component D. In addition, for some applications it may be suitable to further strengthen A. Sect. 4 shows a sample use of this template.

The kind of adversary model used in the threat scenario specification is close to the approach taken in [17] and [24], where it turned out to be useful for the analysis of cryptographic protocols. Differences occur, however, in the explicit modelling of the adversary's influence on communication, which in our approach can be tailored to the application at hand.

3.3 The Security Property

Given a system specification S and a threat scenario B that has been derived from S, security can be expressed using a particular binary relation R_{Sec} on specifications. If $R_{Sec}(S, B)$ holds, S is said to be secure with respect to the threats represented in B. However, the implications of $R_{Sec}(S, B)$ on the security of a system being implemented according to S depend heavily on the concrete definition of R_{Sec}. In the remainder of this section we want to introduce a number of variants of such a definition, which correspond to different kinds of security notions. Thus, our interpretation of security is split into two parts: a system specific part, which relates to vulnerabilities of the system under development, the specific abilities of an attacker to that system, and the environment of it, being modelled in a threat scenario, and a general part expressing common security requirements, being modelled using a particular security relation.

We start with the definition of the most restrictive type of security, in which adversary interference is expected to have no influence on the behaviour of the system. In this case, the threat scenario must be a refinement of the original system.

Definition 1 *A system S with syntactic interface (I,O) is called* absolutely secure *with respect to a threat scenario B, with the same interface, if* $R_S(\mathrm{S}, \mathrm{B})$ *holds, with* R_S *being defined by* $R_S(\mathrm{S}, \mathrm{B}) \equiv \mathrm{S} \rightsquigarrow \mathrm{B}$. □

In practice, absolute security is usually hard to achieve, and sometimes it is even not desired: if there are interactions that are not considered to be security relevant, then an adversary may influence these without compromising security.

If the security requirements on the application at hand are known exactly, we may use only these to define the system's security.

Definition 2 *Given a predicate P, a system S with syntactic interface (I,O) is called P-*secure *with respect to a threat scenario B, with the same interface, if $P(\mathrm{B})$ holds.* □

Formal definitions can be provided for certain common aspects of security, like integrity, authenticity, confidentiality, or availability. Using these definitions in a security analysis, the analyst need not formalise particular security requirements, but may only use the definition covering the aspects that are of importance to her application. Since in Sect. 4 we focus on authentication mechanisms and their impact on availability, we provide general definitions for authenticity and availability of a system.

We distinguish between a strong and a weak variant of authenticity.

Definition 3 *A system S with syntactic Interface (I,O) is called* strongly authentic *with respect to a threat scenario B, with the same interface, if $R^s_{Ath}(\mathrm{S},\mathrm{B})$ holds, with R^s_{Ath} being defined by*

$$\begin{aligned} R^s_{Ath}(\mathrm{S},\mathrm{B}) \quad \equiv \quad & \forall f \in \vec{I} \overset{s}{\rightarrow} \vec{O} : \;\; f \in [\![\mathrm{~B~}]\!] \Rightarrow \\ & \forall x \in \vec{I} \quad \exists x' \in \vec{I}, f' \in [\![\mathrm{~S~}]\!] : x' \trianglelefteq x \;\;\wedge\;\; f'(x') = f(x) \;\; . \end{aligned}$$

□

The above definition states, that, if $(x, f(x))$ is an i/o-behaviour of B, then there is a substream x' of x such that $(x', f(x))$ is an i/o-behaviour of S. This means that each output of B is caused by a "legitimate" input, but we do not require the attacked system to respond to all legitimate inputs.

We yield a weaker variant of authenticity, if the above property is only required with respect to some message abstraction defined by an abstraction function *abs*. Thus, an adversary is allowed to manipulate some parts of a message that are considered irrelevant with respect to authenticity.

Definition 4 *A system S with syntactic Interface (I,O) is called* weakly authentic *with respect to a threat scenario B, with the same interface, and an abstraction function $abs : (\vec{I} \rightarrow M^{\overline{\omega}}) \rightarrow (\vec{I} \rightarrow S^{\overline{\omega}})$, with S being an arbitrary set of message abstractions, if $R^w_{Ath}(\mathrm{S},\mathrm{B})$ holds, with R^w_{Ath} being defined by*

$$\begin{aligned} R^w_{Ath}(\mathrm{S},\mathrm{B}) \quad \equiv \quad & \forall f \in \vec{I} \overset{s}{\rightarrow} \vec{O} : \;\; f \in [\![\mathrm{~B~}]\!] \Rightarrow \\ & \forall x \in \vec{I} \; \exists x' \in \vec{I}, f' \in [\![\mathrm{~S~}]\!] : abs(x') \trianglelefteq abs(x) \;\wedge\; f'(x') = f(x). \end{aligned}$$

□

With the weak authenticity definition we may, for example, formalize peer entity authentication, if messages allow the derivation of the entity identifier where they claim to come from, and the abstraction function is defined to extract this identifier from a given message.

Considering availability, we again distinguish between a strong and a weak variant. By strong availablity, we mean that for each legitimate input, there must be an appropriate system reaction.

Definition 5 *A system S with syntactic Interface (I,O) is called* strongly available *with respect to a threat scenario B, with the same interface, if* $R^s_{Aval}(\mathrm{S},\mathrm{B})$ *holds, with* R^s_{Aval} *being defined by*

$$\begin{aligned} R^s_{Aval}(\mathrm{S},\mathrm{B}) \equiv\ & \forall f \in \vec{I} \overset{s}{\to} \vec{O} : \ f \in [\![\mathrm{B}]\!] \Rightarrow \\ & \forall x \in \vec{I} \quad \exists f' \in [\![\mathrm{S}]\!] : f'(x) \trianglelefteq f(x) \ . \end{aligned}$$

□

Note that in the case of strong availability the system is not only required to somehow react to each legitimate input in case of an attack ocurring, but also to react in exactly the same way as in the non-attack case.

However, in many practical situations strong availability cannot be achieved nor is even desired: in these cases it is sufficient that at each point of time the system will eventually react to a legitimate input. If the input is provided by another component under the control of the system designer, this component may be specified to retransmit messages until the appropriate system reaction is observed.

To formalize weak availability, we have to switch to timed streams, in contrast to the definitions above which refer to untimed streams only.

Definition 6 *A system S with syntactic Interface (I,O) is called* weakly available *with respect to a threat scenario B, with the same interface, if* $R^w_{Aval}(\mathrm{S},\mathrm{B})$ *holds, with* R^w_{Aval} *being defined by*

$$\begin{aligned} R^w_{Aval}(\mathrm{S},\mathrm{B}) \equiv\ & \forall f \in \vec{I} \overset{s}{\to} \vec{O} : \ f \in [\![\mathrm{B}]\!] \Rightarrow \\ & \forall x \in \vec{I} : \#\bar{x} = \infty \Rightarrow \\ & \exists x' \in \vec{I}, f' \in [\![\mathrm{S}]\!] : \#\bar{x'} = \infty \ \wedge \ x' \trianglelefteq_t x \ \wedge \ \overline{f'(x')} \trianglelefteq \overline{f(x)} \ . \end{aligned}$$

□

Note that both availabilty definitions refer only to the existence of a response to a legitimate input, not to the amount of time between request and corresponding response.

3.4 Security Mechanisms

When threat identification and risk analysis is performed, systems, in general, turn out not to be secure. Therefore, we have to specify particular means, called security mechanisms, that are suited to counter the threats that have been identified as critical. We distinguish between technical mechanisms, which are given by a particular functionality of an IT system, and non-technical mechanisms, which are organisational or physical means located in the system's environment. As an example of non-technical means, take a messenger delivering a secret key, or a mechanical door lock preventing an intruder from accessing a computer system located in a particular room. In our approach, we only consider technical mechanisms, since they form a part of the system to be developed and can therefore be treated in the same way as functional requirements. However, assumptions based on non-technical mechanisms may influence the adversary model.

A lot of basic technical mechanisms suited to meet different security requirements have been proposed. [8] gives a representative overview. In general, for a given security problem, there are several mechanisms that are suited to meet the requirements, differing only with respect to non-functional criteria as performance, cost, and legal issues (patents, licences). Though these criteria may be of major importance to the application, they do not contribute to security analysis as described in the previous sections. Therefore, the selection problem is considered to be out of scope of our approach.

The mechanisms we are particularly interested in, include those based on cryptographic methods. They are based on concepts as common secrets, cryptographic keys, random numbers, nonces, and so on. In our approach, each of these concepts is modelled by a specific data type, where the adversary's abilities on the usage of elements of these data types are restricted. Consider, for example, the set of cryptographic keys and cryptograms in Sect. 4. The model of communication and the semantics of Focus allows to benefit from results of approaches specifically dedicated to the description of cryptographic systems, for example [17] or [24].

4 A Sample Development

In this section, we show the application of the method introduced above by giving a detailed example. We first give the specification of a simple system which, however, may occur in real-world applications in a similar form. Then, a threat scenario is described, which is ficticious but could as well have been achieved as the result of a real-world threat analysis. We show that our example system is not authentic without adding particular authentication mechanisms. We provide such a mechanism by specifying a challenge-response protocol with encrypted response which is a simplification of the ISO 9798-2 protocol ([11]). Specifications are given in state-transition style, which corresponds closely to the way cryptographic protocols are usually presented, and relational style, which gives a more abstract view of the protocol and is well suited for the conduction of correctness proofs. It is shown that the introduction of the authentication mechanism does not violate the original system specification, i.e. is a refinement of the original system. Given a simplified adversary model, we prove strong authenticity of the system. With a more complex adversary model, only weak authenticity can be shown.

With the proof of authenticity of the system including the authentication protocol, it turns out that availability is lost in case of an attack. We therefore have to modify our protocol specification by considering the timing of messages. The time-dependent protocol is then shown to be both authentic and available, with respect to some fairness assumptions on the adversary's behaviour.

In most cases, proof details are omitted. The reader may find all the details in the accompanying technical report [15].

4.1 A Simple Server

Our example provides the specification of a very simple, idealized server component that is able to receive requests submitted by a client via a transmission medium and to respond to those requests that have been issued by authorized clients by sending results using a different communication channel. Since the main focus of the example is on security analysis of the server, the detailed structure and contents of requests and results are not important. However, if looking for possible applications for servers of this kind, imagine an electronic door lock which is only released upon request, for example by inserting a smart card, or a mobile phone system, in which connect requests are received by a server and, possibly supplied with additional data about the requestor, forwarded to a switching center. We assume that there

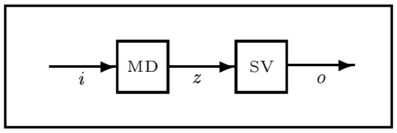

Figure 4: A simple server

are several clients using the same request channel, thus each request has to be tagged with the client's identifier. Figure 4 shows an abstract view of the server, consisting of a server component SV and the transmission medium MD. To formally specify the server in Focus, let *Id* be a set of identifiers, each of which is assumed to be authorized to sending requests, and *Req*, *Res* represent the set of requests and results, respectively. As argued above, *Req* and *Res* are not specified in detail. Using the operator style of specification, the server is described by

$$\mathrm{S} \equiv (i : \mathit{Id}\times\mathit{Req} \rhd o : \mathit{Res}) :: \quad \mathrm{MD} \succ \mathrm{SV} \ \ ,$$

with the component specifications given by

$$\mathrm{MD} \equiv (i : \mathit{Id}\times\mathit{Req} \rhd z : \mathit{Id}\times\mathit{Req}) \overset{\mathsf{ti}}{::} \quad z = i \ \ ,$$
$$\mathrm{SV} \equiv (z : \mathit{Id}\times\mathit{Req} \rhd o : \mathit{Res}) \overset{\mathsf{ti}}{::} \quad \#o = \#z \ \ .$$

Note that we assume an ideal transmission medium, resulting in the component MD being simply the identity on its input channel. This has been chosen in order to keep the simplicity of the example. Section 3.2 outlines how one may deal with more sophisticated media specifications.

SV states that each request of an authorized client, and only those, will be served. Because of the semantic model of time independent specifications in Focus, SV ist quite implicit: from the strong pulse-driveness constraint on functions satisfying SV it follows that requests are served in order of their receipt, and that no responses are issued in advance, anticipating future requests.

4.2 The Threat Scenario

In Sect. 3.2 we stated that each threat scenario is the result of an application specific threat identification and risk analysis, where templates can be used in the construction of the scenario. Since risk analysis heavily depends on non-technical arguments, for example consideration of associated financial loss, it is not completely covered by our method. For our example, we therefore assume that a risk analysis has been carried out, with the supposed result of the adversary being assessed as being able to eavesdrop the transmitted messages, to know about the set of client identifiers and requests, and to insert fraudulent messages. These assumptions are intended to completely describe the adversary's behaviour, particularly she cannot manipulate or delete messages on the input channel i in our example scenario.

Since MD models a transmission medium as discussed in Sect. 3.2, the template given there can be used to construct the threat scenario B. Thus,

$$\mathrm{B} \equiv (i : \mathit{Id}{\times}\mathit{Req} \rhd o : \mathit{Res}) :: \quad \mathrm{MD}_{\mathrm{Thr}} \succ \mathrm{SV} \ ,$$

with $\mathrm{MD}_{\mathrm{Thr}}$ as defined in Sect. 3.2, using the message set $M = \mathit{Id}{\times}\mathit{Req}$. Let $V = M$ and $F = \emptyset$, which states that the adversary knows the complete set of request messages that may be transmitted. Moreover, let $C = \mathsf{B}$ be the set of control messages. We assume the adversary to keep the consistency of her control and data output by adding the conjunct

$$\#1 \copyright c = \#d$$

to the specification of A in the scenario template $\mathrm{MD}_{\mathrm{Thr}}$ of Sect. 3.2, stating that for each message in d we have a corresponding 1 in c.

We still have to instantiate the data flow component D_{MD} of $\mathrm{MD}_{\mathrm{Thr}}$. Since we have decided to strengthen the adversary model A(V) by adding consistency requirements, we may use a quite general specification of D_{MD}, which will be suited for other analyses as well. We define

$$\mathrm{D}_{\mathrm{MD}} \equiv (i : M, d : M, c : \mathsf{B} \rhd i_A : M, z : M) \overset{\mathsf{ti}}{::}$$

$$i \sqsupseteq \Pi_1((M{\times}0) \copyright (z, c' \frown 0^\infty)) \quad \wedge \quad d|_n = \Pi_1((M{\times}1) \copyright (z, c')) \quad \wedge \quad i_A = i$$

$$\textsf{where} \quad n = \min(\#d, \#1 \copyright c) \quad \wedge \quad c' \sqsubseteq c \quad \wedge \quad \#1 \copyright c' = n \ .$$

The equation $i_A = i$ in the specifying relation states that all input messages may be eavesdropped by the adversary.

Note that D_{MD} to some extent corresponds to the specification of a merge component, with the oracle partly determined by the control sequence c. Fairness of D_{MD} depends on the control sequence input: if, and only if, the control sequence allows the insertion of infinitely many messages, transmission of messages of i may be suspended for ever, this fact being reflected by using the prefix relation instead of equality with respect to i, and by extending the control sequence in the first conjunct. On the other hand, given an appropriate control sequence, each of the adversary's messages will indeed be inserted. Potential loss of fairness is intentional, since it does not seem to be reasonable to always assume a fair adversary. The auxiliary values n and c' are introduced to handle cases where the control sequence and the messages sent by the adversary do not fit together, meaning that there are less 1's in c than messages in d or vice versa. However, from our specification of A(V), we always have appropriate control sequences, simplifying the specifying relation of D_{MD} to

$$i \sqsupseteq \Pi_1((M{\times}0) \copyright (z, c \frown 0^\infty)) \quad \wedge \quad d = \Pi_1((M{\times}1) \copyright (z, c)) \quad \wedge \quad i_A = i \ .$$

So far, we have not introduced any fairness constraints on the adversary specification of our example, in fact, we need not assume fairness of the adversary in order to prove authenticity of the mechanism introduced below. However, fairness has to be considered when reasoning about availability in Sect. 4.3.3.

S as specified above, which means not containing any particular security mechanism, is not authentic with respect to B, as is shown in the following theorem.

Theorem 1 S *is not authentic w.r.t.* B, *i.e.* $R_{Ath}(\mathrm{S}, \mathrm{B})$ *does not hold.*

Proof. Choose $i = \langle\rangle$, $d = \langle(id_0, rq)\rangle$ for some $id_0 \in Id$, $rq \in Req$, and $c = \langle 1\rangle$ as existential witnesses. Then, $(i, (d, c))$ is a possible i/o-behaviour of A(V). In this case, by the definition of D, we have $z = \langle(id_0, rq)\rangle$, leading to $\#o = 1$ by the definition of SV. Since for all $x \in M^{\overline{\omega}}$, $x \trianglelefteq \langle\rangle \Rightarrow x = \langle\rangle$, authenticity of S would require (i, o) to be an i/o-behaviour of S, which is obviously not the case, because for all $f \in [\![\ S\ ]\!]$, we have $\#x = \#f(x)$. □

4.3 An Authentication Protocol

In order to specify an authentic server, we have to refine S by introducing an appropriate security mechanism. ISO proposes a simple challenge-response authentication protocol ([11]) that is considered to be suited for applications like our server. We give a specification of this protocol and analyse authenticity and availability in detail. A variant of this protocol proposed by [12] is discussed in [14] and [15].

4.3.1 Specification

Cryptographic Systems
The protocol is based on symmetric cryptoalgorithms and pseudo random number generators, and assumes that the server and each of the clients share a secret key not known to the adversary. To model cryptographic systems, a value space as for example defined in [24] is suited for our stream based communication model as well.

To describe the cryptographic system used in our example, let K be a set of cryptographic keys, Cr a set of cryptograms, and Ms a set of messages with $Cr \cap Ms = \emptyset$ meaning that messages and cryptograms can be distinguished. We have an encryption function $E : K \times (Cr \cup Ms) \to Cr$ and a decryption function $D : K \times (Cr \cup Ms) \to (Cr \cup Ms)$. In symmetric cryptosystems, we have

$$\begin{aligned} &D(k, E(k, x)) = x, && x \in Ms \cup Cr\ , \\ &E(k, x_1) = E(k, x_2) \Rightarrow x_1 = x_2, && k \in K, x_1, x_2 \in Ms \cup Cr\ . \end{aligned}$$

Further properties hold with high probability. Since FOCUS, like almost all other approaches to distributed systems design and verification, is not intended to deal with probabilities, we have to approximate them by predicate logic formulæ. A reasonable idealization is to take properties that hold with high probability for granted.

It is considered to be improbable that the adversary constructs cryptograms (by simply guessing or taking arbitrary keys and messages – which in good cryptosystems both are of nearly equal probability) that match cryptograms being issued by legitimate users. We model this fact by

$$\begin{aligned} &E(k_1, x) = E(k_2, x) \Rightarrow k_1 = k_2, && k_1, k_2 \in K, x \in Ms \cup Cr, \\ &E(k_1, m_1) = E(k_2, m_2) \Rightarrow k_1 = k_2 \wedge m_1 = m_2, && k_1, k_2 \in K, m_1, m_2 \in Ms, \end{aligned}$$

and assume that the adversary does not exploit the finiteness of the set of cryptograms. Note that the latter formula is modelled to only hold for messages of Ms, and requires that Ms is of considerably less cardinality than Cr.

The protocols also use random numbers. We choose a set R of values from which random numbers are taken. For each stream $r \in R^{\overline{\omega}}$ of random numbers, we at least require that no duplications occur, described by $PRN(r)$, with

$$PRN(r) \equiv \forall j \in \mathsf{dom}.r : r.j \notin \mathsf{rng}.(r|_{j-1})\ .$$

PRN obviously does not completely characterize random numbers, but is sufficient to show authenticity of the protocol based on [11].

State Transition Specification
We are now ready to specify the authentication protocol of [11]. Fig. 5 shows a structural view of the refined server SA.

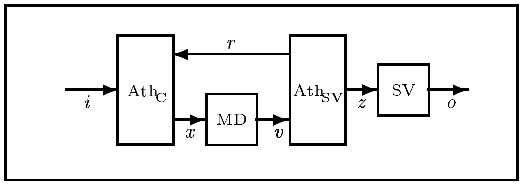

Figure 5: An authentication mechanism

Two components $\mathrm{Ath}_{\mathrm{C}}$ and $\mathrm{Ath}_{\mathrm{SV}}$ have been added to control protocol runs on the client and server side, respectively. Each time a request is received by $\mathrm{Ath}_{\mathrm{SV}}$, it issues a challenge on r and proceeds only in the case that the next message received is an appropriate response to the challenge. Otherwise, the request will be ignored. $\mathrm{Ath}_{\mathrm{C}}$ is responsible for passing on requests and suitable responses, if challenges are received.

For simplicity of the example, we specify a slight abstraction of the ISO-protocol by leaving out optional text fields and without considering the inclusion of the verifier's identifier in the response. With the latter, we lose protection against reflection attacks, which is, however, of less importance with respect to the demonstration of how our approach works. A formal specification in constraint style is given by

$$\begin{aligned}&\mathrm{SA} \equiv (i : \mathit{Id}{\times}\mathit{Req} \rhd o : \mathit{Res}) :: \\ &(x) := \mathrm{Ath}_{\mathrm{C}}(i, r),\;\; (v) := \mathrm{MD}(x)\;\; (z, r) := \mathrm{Ath}_{\mathrm{SV}}(v),\;\; (o) := \mathrm{SV}(z),\end{aligned}$$

where MD and SV are specified as in Sect. 4.1. For the specification of the new components, we assume that $\mathrm{Ath}_{\mathrm{SV}}$ will ignore requests, if they are not followed by an appropriate authentication token, and authentication tokens, if it is not waiting for them. For the moment, $\mathrm{Ath}_{\mathrm{C}}$ is specified to buffer all incoming challenges.

The first version of the specification is given in state transition style, for this style being the one corresponding most closely to common presentations of protocol descriptions like the one in [11]. Since so far we do not refer to timing of streams, a time independent specification will suffice. If each client shares a secret key with the server, meaning that there is a set $K_0 \subseteq K$ with $K_0 = \{k_{id} \mid id \in \mathit{Id}\}$, we have (with $M = \mathit{Id}{\times}\mathit{Req}$ as in Sect. 4.2)

$$\mathrm{Ath}_{\mathrm{C}} \equiv (i : M, r : R \rhd x : M \cup Cr) \overset{\text{ti}}{::}$$

$$\exists\ f_1, f_2 : \qquad x = f_1(i, r)$$

$$\text{where} \quad \forall i \in M^{\overline{\omega}}, r \in R^{\overline{\omega}}, (id, req) \in M, rn \in R :$$

$$\begin{aligned}
&f_1(\langle\rangle, r) = \langle\rangle \ , \\
&f_1((id, req) \frown i, r) = (id, req) \frown f_2(id, i, r) \ , \\[1em]
&f_2(id, i, \langle\rangle) = \langle\rangle \ , \\
&f_2(id, i, rn \frown r) = E(k_{id}, rn) \frown f_1(i, r) \ .
\end{aligned}$$

and

$$\mathrm{Ath}_{\mathrm{SV}} \equiv (v : M \cup Cr \rhd r : R, z : M) \overset{\text{ti}}{::}$$

$$\exists\ f_3, f_4 : \qquad (z, r) = f_3(v) \quad \wedge \quad PRN(r)$$

$$\text{where} \quad \forall v \in (M \cup Cr)^{\overline{\omega}}, (id, req) \in M, rn \in R, m, cr \in M \cup Cr :$$

$$\begin{aligned}
&f_3(\langle\rangle) = (\langle\rangle, \langle\rangle) \ , \\
&m \in M \Rightarrow \exists rn \in R : f_3(m \frown v) = (\langle\rangle, rn) \frown f_4(m, rn, v) \ , \\
&m \notin M \Rightarrow f_3(m \frown v) = f_3(v) \ , \\[1em]
&f_4(m, rn, \langle\rangle) = (\langle\rangle, \langle\rangle) \ , \\
&D(k_{id}, cr) = rn \Rightarrow f_4((id, req), rn, cr \frown v) = ((id, req), \langle\rangle) \frown f_3(v) \ , \\
&D(k_{id}, cr) \neq rn \Rightarrow f_4((id, req), rn, cr \frown v) = f_3(cr \frown v) \ .
\end{aligned}$$

In the above specification, sequences of protocol runs are treated by introducing states denoted by f_1, f_2 in $\mathrm{Ath}_{\mathrm{C}}$ and by f_3, f_4 in $\mathrm{Ath}_{\mathrm{SV}}$. If $\mathrm{Ath}_{\mathrm{C}}$ is in state f_2 waiting for challenges, any incoming request will be delayed until the authentication token has been constructed and $\mathrm{Ath}_{\mathrm{C}}$ set back in state f_1 waiting for requests. If $\mathrm{Ath}_{\mathrm{SV}}$ waits for a response in state f_4, anything except the response awaited will be rejected, with $\mathrm{Ath}_{\mathrm{SV}}$ returning to state f_3. If authentication tokens are received in state f_3, where there are no requests remaining to be authenticated, they are simply ignored.

Relational Specification
The state-transition specification given above closely follows the specification of [11], even in the sense of giving a rather operational view of both actors of the protocol. In order to gain a deep understanding of the protocol and to easily conduct correctness and security proofs, it is, however, often useful to take a more abstract view of the protocol by specifying those properties of the protocol that are considered to be essential in a relational style. In proofs, a relational specification often helps to avoid complex inductions or consideration of lots of irrelevant technical detail.

A more abstract, relational specification of our authentication protocol is given below, indicated by superscript R. We have the client's part of the protocol specified by

$$\text{Ath}^R_{\text{C}} \equiv (i : M, r : R \rhd x : M \cup Cr) \overset{\mathsf{ti}}{::}$$

$$
\begin{aligned}
& M \copyright x \sqsubseteq i && (1)\\
& \#r \geq \#i \Rightarrow \#M \copyright x = \#i && (2)\\
& \#r < \#i \Rightarrow \#M \copyright x = \#r + 1 && (3)\\
& \forall y : y \sqsubseteq x \Rightarrow \#Cr \copyright y \leq \#M \copyright y \leq \#Cr \copyright y + 1 && (4)\\
& \forall j \in \mathsf{dom}.x : x_{j+1} \in Cr \Rightarrow x_{j+1} = E(k_{\Pi_1(x_j)}, r_{\#M \copyright x|_j}) && (5)
\end{aligned}
$$

The first conjunct (1) states that the authentication component does not produce messages on its own. Each message being output has occurred in the input, and the sequence of messages is kept, denoted by the prefix operator. If there is a sufficient number of challenges, an authentication token can be constructed for each message, thus each input message will be output, as stated by (2). Otherwise, if there are not enough challenges, all messages, for which an authentication token can be computed, are output, plus the following message (formula (3)). In other words, the authentication component at the client's side sends a message received at i, and then waits for a challenge to construct the authentication token. If there are no further challenges, no more output is generated, otherwise the next challenge from the communication buffer is used. Messages and corresponding authentication tokens are output in an alternating way starting with a message and desribed by property (4). (5) then describes the structure of an authentication token corresponding to the immediately preceding message m: it is a cryptogram $E(k_{id}, c)$, with id being the identifier component of m, and c the corresponding challenge, where the nth challenge of r corresponds to the nth message sent along x.

A relational specification of the authentication component at the server's side looks as follows.

$$\text{Ath}^R_{\text{SV}} \equiv (v : M \cup Cr \rhd r : R, z : M) \overset{\mathsf{ti}}{::}$$

$$
\begin{aligned}
& \#r = \#M \copyright v && (6)\\
& PRN(r) && (7)\\
& \forall j \in \mathsf{dom}.z : \exists l \in \mathsf{dom}.v : z_j = v_l \wedge D(k_{\Pi_1(v_l)}, v_{l+1}) = rn && (8)\\
& z \trianglelefteq M \copyright v && (9)\\
& \#z = \#\{l \in \mathsf{dom}.v \mid v_l \in M \wedge l+1 \in \mathsf{dom}.v \wedge D(k_{\Pi_1(v_l)}, v_{l+1}) = rn\} && (10)
\end{aligned}
$$

$$\text{where} \quad rn = r_{\#M \copyright v|_l}$$

The specification states, that for each message received a challenge will be output (6), and that the stream of challenges satisfies the requirements on pseudo random numbers (7). From

Property (8) it follows that only those messages will be forwarded to the server component SV, that are correctly authenticated by the token immediately following the message in stream v. Correctly authenticated means that decryption of the token with k_{id}, id being the identifier component of the message, yields the challenge expected, which for the nth message in v is the nth challenge issued. The authentication component should preserve the sequence of messages as specified by (9). Assuming (9), (10) states that all correctly authenticated messages are indeed output.

From the relational specification of the authentication protocol, we get a relational variant SA^R of the specification of SA by the analoguous constraint specification

$$\begin{array}{l}\mathrm{SA}^R \equiv (i : Id \times Req \rhd o : Res) :: \\ (x) := \mathrm{Ath}^R_{\mathrm{C}}(i, r),\ \ (v) := \mathrm{MD}(x),\ \ (z, r) := \mathrm{Ath}^R_{\mathrm{SV}}(v),\ \ (o) := \mathrm{SV}(z).\end{array}$$

Proof of Refinement

Following the method of secure systems development as described in Sect. 3.1, the first step in order to show that the system indeed has become secure by introducing the authentication mechanism as specified above is to show that the introduction of the mechanism does not violate the functional requirements of the server. This is done by proving that SA, the system including the authentication protocol, is a refinement of S, the original server specification of Sect. 4.1. Since we gave a relational as well as a state-transition specification of $\mathrm{Ath}_{\mathrm{C}}$ and $\mathrm{Ath}_{\mathrm{SV}}$ and therefore of SA, our proof is twofold: We first show that SA is a (behavioural) refinement of SA^R and then prove that SA^R is a (structural) refinement of S.

Theorem 2 $\mathrm{SA}^R \rightsquigarrow \mathrm{SA}$, *i.e. for all* $f \in M^{\overline{\omega}} \to M^{\overline{\omega}}$ *we have* $f \in [\![\mathrm{SA}]\!] \Rightarrow f \in [\![\mathrm{SA}^R]\!]$.

Proof. The proof is performed separately for each of the properties of the relational specification, generally employing induction on the structure of the input streams. Details are given in [15]. □

Since we now have shown that the state-transition specification representing an operational view of the authentication protocol satisfies the properties given by the relational specification representing an abstract view, it remains to show that the relational specification is a structural refinement of the original server specification. Note that Theorem 2 contributes to the validation of both the state-transition and the relational specification.

Theorem 3 $\mathrm{S} \rightsquigarrow \mathrm{SA}^R$, *i.e. for all* $f \in M^{\overline{\omega}} \to M^{\overline{\omega}}$ *we have* $f \in [\![\mathrm{SA}^R]\!] \Rightarrow f \in [\![\mathrm{S}]\!]$.

Proof. The proof is given in [15]. □

Having now proved that the insertion of the authentication protocol does not violate the requirements on the server, we may turn our attention to authenticity.

4.3.2 Authenticity

The Threat Scenario Revisited

Since with the definition of the security mechanism additional channels and new message types have been introduced, it is appropriate to update the threat scenario parameters, as already argued in Sect. 3.1. For our example, we assume that challenges are transmitted via a secure channel (remember Fig. 5, where the threatened medium is only specified for the request and response channel), but that the adversary knows the set of possible random values R, and thus can guess one of them. In addition, she has some keys available, but not those of the legitimate clients, and may encrypt as well as decrypt. Formally, we have the threat scenario instantiation given by $V = M \cup R \cup K_A$ for some $K_A \subseteq K \setminus K_0$, $F = \{E, D\}$, D as defined in Sect. 4.2

In order to show the expressiveness of our approach with respect to reasoning about different adversary models, we will further distinguish between two different adversary characterizations.

First, we consider an adversary with limited capabilities. This kind of adversary only inserts fake requests and immediately tries to give an appropriate authentication response. This is an appropriate characterization of a door lock secured by a card reader, where the adversary tries to insert a fake card and therefore has to wait until the door is left unsupervised. We further refer to this kind of adversary model as the simple adversary model, formally defined by A strenghtened by

$$\exists\, h \in \mathsf{B}^{\overline{\omega}}, n \in \mathsf{N} \cup \{\infty\} : h = \langle 0, 1\rangle^n \wedge \mathrm{sel}(h, d) \in Cr^{\overline{\omega}} \wedge \mathrm{sel}(\bar{h}, d) \in M^{\overline{\omega}} \tag{11}$$

$$\exists\, i, j \in \mathsf{N}, k \in \mathsf{N} \cup \{\infty\} : c = (\langle 0\rangle^{2i} \frown \langle 1, 1\rangle \frown \langle 0\rangle^{2j})^k \ \ , \tag{12}$$

with $\bar{h}$ denoting the bitwise complement of a bitstream h. Note that the basis for this strengthening is the adversary specifcation A of Sect. 4.2, not the one from Sect. 3.2, which means that $\#1 \copyright c = \#d$ is still being asserted. Let $\mathrm{MD}^s_{\mathrm{Thr}}$ denote the specification of the threatened medium within the simple adversary model.

An advanced adversary model is given by the specification A of Sect. 4.2 without adding further constraints. An adversary which behaves according to that model may insert arbitray messages or cryptograms at each point of the original message stream, which may occur if messages and responses are transmitted via publicly accessible communication links, with mobile phone systems being an example. Let $\mathrm{MD}^a_{\mathrm{Thr}}$ denote the specification of the threatened medium within the advanced adversary model.

Authenticity with Simple Adversary Model

With BA_s denoting the threat scenario instantiation for the simple adversary model, we can show

Theorem 4 SA^R *is strongly authentic w.r.t.* BA_s*, i.e.* $R_{Ath}(\mathrm{SA}^R, \mathrm{BA}_s)$ *holds.*

Proof. From the definitions in Sect. 3.3 and the specification of SV, it follows that strong authenticity holds in case of

$$R_{\mathrm{Ath}^R_{\mathrm{C}}} \wedge R_{\mathrm{MD_{Thr}}} \wedge R_{\mathrm{Ath}^R_{\mathrm{SV}}} \Rightarrow \exists h \in \mathsf{B}^{\overline{\omega}} : \ \ z = \mathrm{sel}(h, i)$$

being valid. We show the assertion by contradiction. Assume that $\forall h \in \mathsf{B}^{\overline{\omega}} : z \neq \mathrm{sel}(h, i)$ (*) holds. Then, particularly we have $z \neq i$ by chosing $h = 1^{\infty}$. Three cases may occur.

Case 1: $z \sqsubset i$. But then we have $z = \mathrm{sel}(h, i)$ for $h = 1^{\#z} \frown 0^{\infty}$, which contradicts (*).

Case 2: There is a $j \in \mathsf{dom}.i \cap \mathsf{dom}.z$ with $z_j \neq i_j$. Let j_0 be the least such j, i.e. there is i' with $i = i' \frown i_{j_0} \frown i''$ and $z = i' \frown z_{j_0} \frown z''$. From specification properties (1), (8) and $z_{j_0} \neq i_{j_0}$, as well as (12) from the revised threat scenario, we conclude that there is an odd

l with $c_l = 1$, $c_{l+1} = 1$, $d_l = z_{j_0} = (id_0, req_0)$ for some $id_0 \in Id$ and $req_0 \in Req$, and $D(k_{id_0}, d_{l+1}) = r_{M©v|_l} = r_{j_0}$.

Without restricting generality, we may assume that this is the first attack, i.e. $l = 2j_0 - 1$. (Otherwise, a contradiction can be constructed following the argumentation below.) From the adversary specification and closure properties of C_F, it follows that $d_{l+1} \in C_F(V \cup \mathsf{rng}.x|_{2(j_0-1)})$. Two cases must be distinguished.

Case21: d_{l+1} is a eavesdropped cryptogram, i.e. $d_{l+1} \in \mathsf{rng}.Cr©x|_{2(j_0-1)}$, which from the specification of $\mathrm{Ath}_{\mathrm{C}}^{R}$ is equivalent to $d_{l+1} = E(k_{id}, r_j)$ for some $j < j_0$ and $id \in Id$. But from $PRN(r)$ it follows that $r_j \neq r_{j_0}$ for all $j < j_0$. This leads to a contradiction to the properties of the cryptographic system, since different challenges lead to different cryptograms.

Case 22: d_{l+1} is a cryptogram constructed by the adversary herself, i.e. $d_{l+1} = E(k, rn)$ for some $k \in K_A$ and $rn \in R$. But since then $k \notin K_0$, we have $d_{l+1} \neq E(k_{id}, r_{j_0})$ for all $id \in Id$ by the properties of the cryptographic system, which leads to contradiction of the assumption, that d_{l+1} is an appropriate authentication token wrt. some $id_0 \in Id$.

Case 3: $i \sqsubset z$. We have $z = i \frown z_{\#i+1} \frown z'$ The proof is analoguous to Case 2 with l set to $2\#i + 1$. □

The cruical point of the proof of Theorem 4 is the validity of the assumption, that the insertion of messages d_l, d_{l+1} is the first attack occurring. That the assumption does indeed hold, follows from

$$c = 0^{2j} \frown \langle 11 \rangle \frown 0^{\infty} \Rightarrow \#z \leq j$$

which states that after the first attack all forthcoming authentications, whether by a legitimate user or the adversary, will fail. We demonstrate the validity of the above formula by means of an example, assuming $j = 0$. A proof for arbitrary j can be obtained by induction on j and exploitation of pulse-driveness of functions satisfying the component specifications.

Let $c = \langle 11 \rangle \frown 0^{\infty}$. We show that $z = \langle\rangle$. From property (10), we have $\#z = \#S(v)$, with $S(v)$ denoting the set on the right-hand side of (10). From the particular definition of c, it follows that $v = v_1 \frown v_2 \frown x$, with v_1, v_2 being inserted by the adversary, and x having properties as specified by $\mathrm{Ath}_{\mathrm{C}}^{R}$. This means that for all even j we have

$$\begin{aligned} x_j &= E(k_{\Pi_1(x_{j-1})}, r_{\#M©x|_{j-1}}) \\ &= E(k_{\Pi_1(x_{j-1})}, r_{(\#M©v|_{j-1})-1}) \end{aligned}$$

Thus, for all j we have $v_j \in M \Rightarrow D(k_{\Pi_1(v_j)}, v_{j+1}) \neq r_{\#M©v|_j}$. Informally, the adversary's authentication fails for reasons already discussed in the proof of Theorem 4, and the legitimate clients' authentications fail, because they take the wrong challenge. Altogether, we have $\#S(v) = 0$, which leads to $z = \langle\rangle$.

The argumentation above, being driven by the conduction of the authenticity proof, shows that the protocol specified so far preserves authenticity at the expense of losing availability in case of an attack. This is essentially a consequence of the particular embedment of the protocol in the server environment, and could not have been detected by merely considering the protocol as given in [11]. Thus, it shows the importance of considering mechanism embedment as well as the ability to deal with different security aspects within our approach. We further consider availability below.

Authenticity with Advanced Adversary Model
Considering the advanced adversary model, given by the threat scenario instantiation including $\mathrm{MD}^a_{\mathrm{Thr}}$ of Sect. 4.2, the adversary is expected to insert single messages or authentication tokens at any position within stream x. In that case, we potentially lose strong authenticity, since an adversary may force the server to accept a fake request, as long as the identifier component of the fake request corresponds to the identifier of a legitimate and correctly authenticated message. The situation is illustrated by an example.

Let $i = \langle (id, req_1) \rangle$, $x = \langle (id, req_1), E(k_{id}, rn) \rangle$ and $v = \langle (id, req_1), (id, req_2), E(k_{id}, rn) \rangle$, then (x, v) is a possible I/O-behaviour of the advanced version $\mathrm{MD}^a_{\mathrm{Thr}}$. From the specification of $\mathrm{Ath}^R_{\mathrm{SV}}$, we yield $z = \langle (id, req_2) \rangle$ and (i, z) being an I/O-behaviour of $\mu\,(\mathrm{Ath}^R_{\mathrm{C}} \succ \mathrm{MD} \succ \mathrm{Ath}^R_{\mathrm{SV}})$, describing the system without the server component SV. Since in our example specification of SV we only refer to the length of the input, authenticity is not affected, but we will lose strong authenticity, if the output of SV differs between req_1 and req_2. However, weak authenticity is preserved in any case, if we take $f : Id \times Req \to Id$ with $f((id, req)) = id$ as abstraction function, since a fake request will only be successfully authenticated if there is a legitimate message (for which the authentication token has been originally constructed by $\mathrm{Ath}^R_{\mathrm{C}}$) with the same identifier.

The insertion of single authentication tokens by the adversary of the advanced model is of less criticality. As shown in the proof of Theorem 4, the adversary cannot construct authentication tokens corresponding to any legitimate message, and even eavesdropping legitimate tokens does not help, since from $PRN(r)$ and the properties of the cryptographic system it follows that all correct tokens are distinct. Therefore, the worst case that may occur is a fake token inserted immediately after a legitimate request, leading to the request being refused by the server. However, this does not affect authenticity.

From the above considerations we conclude, with $SA^R_a = \mu\,(\mathrm{Ath}^R_{\mathrm{C}} \succ \mathrm{MD} \succ \mathrm{Ath}^R_{\mathrm{SV}})$ denoting the system excluding the particular server component SV and $BA_a = \mu\,(\mathrm{Ath}^R_{\mathrm{C}} \succ \mathrm{MD}^a_{\mathrm{Thr}} \succ \mathrm{Ath}^R_{\mathrm{SV}})$ denoting the threat scenario corresponding to SA^R_a,

Theorem 5 SA^R_a *is weakly authentic w.r.t.* BA_a *and abstraction function* Π_1*, i.e.* $R^w_{Ath}(\Pi_1, \mathrm{SA}^R, \mathrm{BA}_s)$ *holds.*

The proof follows the argumentation above, but is omitted for reasons of space.

The advanced adversary model applies in situations, in which requests and authentication tokens are transmitted via publicly accessible communication links, with mobile phone systems being an example. Since the formal analysis shows that the protocol only provides peer entity authentication, but not message origin authentication, it is only suitable in application scenarios like the one described, if there is one type of requests (as in our example where the structure and/or value of requests is not referred to), or the given request can be checked with respect to context information. Such considerations have to be taken into account when, for a given application, security requirements are defined and the adversary model is constructed.

4.3.3 Availability

The reason for the potential loss of availability in the protocol as specified above lies in the fact that the protocol component on the client side buffers all incoming challenges, even if there is no actual request that requires the computation of an authentication token, and that in case of the construction of a new token the oldest challenge is used. Since the server cannot distinguish between legitimate and fraudulent messages, and therefore has to send a challenge whenever a request is received, the key to increased availability lies in the definition of what is considered to be the appropriate challenge for a token to be constructed by the client. It seems to be reasonable to not take a challenge that has been received at the client before the actual request has occurred, since such a challenge cannot be the appropriate one due to the non-zero delay, i.e. strong pulse-driveness, of both the medium and the server. Thus, the client has to take the next challenge that is received after the request. This, in fact, does not completely avoid taking the wrong challenge, but is a necessary condition for the achievement of availability.

In order to revise our specification of $\mathrm{Ath}_{\mathrm{C}}^{R}$ according to these arguments, we have to switch to the time-dependent format, which allows us to appropriately formalize the notion of "next challenge received". Besides replacing streams occurring in the specifying properties by their time abstractions, we only have to replace the description of the authentication token in property (5). The time-dependent specification $\mathrm{Ath}_{\mathrm{C}}^{T}$ ignores all incoming challenges until it has issued a new request and is given by

$$\mathrm{Ath}_{\mathrm{C}}^{T} \equiv (i : M, r : R \rhd x : M \cup Cr) \overset{\mathsf{td}}{::}$$

$$M \copyright \bar{x} \sqsubseteq \bar{i} \tag{13}$$
$$\#\bar{r} \geq \#\bar{i} \Rightarrow \#M \copyright \bar{x} = \#\bar{i} \tag{14}$$
$$\#\bar{r} < \#\bar{i} \Rightarrow \#M \copyright \bar{x} = \#\bar{r} + 1 \tag{15}$$
$$\forall y : \bar{y} \sqsubseteq \bar{x} \Rightarrow \#Cr \copyright \bar{y} \leq \#M \copyright \bar{y} \leq \#Cr \copyright \bar{y} + 1 \tag{16}$$
$$\forall j \in \mathsf{dom}.\bar{x} : \bar{x}_{j+1} \in Cr \Rightarrow \bar{x}_{j+1} = E(k_{\Pi_1(\bar{x}_j)}, (\overline{r{\uparrow}_{\mathsf{tm}(x,j)}})_1) \tag{17}$$

In property (17), $r{\uparrow}_{\mathsf{tm}(x,j)}$ describes the stream of challenges after that time unit in which the message $\bar{x}_j$ has been forwarded to the server, from which the first non-$\surd$ element is taken as the actual challenge.

In analogy to SA^{R} the time-dependent specification of the server is given by the constraint specification

$$\begin{array}{l}\mathrm{SA}^{T} \equiv (i : Id{\times}Req \rhd o : Res) :: \\ (x) := \mathrm{Ath}_{\mathrm{C}}^{T}(i,r),\ \ (v) := \mathrm{MD}(x),\ \ (z,r) := \mathrm{Ath}_{\mathrm{SV}}^{R}(v),\ \ (o) := \mathrm{SV}(z)\ \ .\end{array}$$

The time-dependent protocol specification still refines the original server specification SV.

Theorem 6 $\mathrm{S} \rightsquigarrow \mathrm{SA}^{T}$*, i.e. for all* $f \in M^{\overline{\omega}} \to M^{\overline{\omega}}$ *we have* $f \in [\![\ \mathrm{SA}^{T}\ ]\!] \Rightarrow f \in [\![\ \mathrm{S}\ ]\!]$.

Proof. Analogous to the proof of Theorem 3. □

Since taking the next incoming challenge is only a necessary, but not a sufficient condition for availability, we have to make further assumptions on fairness of the adversary in order to reason about availability. We first introduce a strong fairness condition that is sufficient for strong availability.

To estimate the time between an attack occurring and the challenge resulting from that attack being received by the client, we must know the maximum time delay caused by the server on its challenge output channel r. Let *dist* be an upper bound on that delay, we may add the property

$$\forall j \in \mathsf{dom}.\bar{r} : \mathsf{tm}(r,j) - \mathsf{tm}(M \cup \{\surd\} ©v, j) \leq dist \tag{18}$$

to the time-dependent version $\mathrm{Ath}^T_{\mathrm{SV}}$ of $\mathrm{Ath}^R_{\mathrm{SV}}$. Besides adding (18) to the specification, $\mathrm{Ath}^T_{\mathrm{SV}}$ is derived from $\mathrm{Ath}^R_{\mathrm{SV}}$ by replacing occurrences of streams with their time abstraction.

Considering the simple adversary model, a fair adversary is then given if there are more than *dist* time intervals between an attack and the next legitimate request, and an attack only occurs if there is no legitimate request pending. Formally, we add the following requirement to the adversary specification A of the time-dependent version of $\mathrm{MD}_{\mathrm{Thr}}$.

$$\forall j \in \mathsf{dom}.v : v_j \in M \wedge P_d(v,j,c) \Rightarrow \mathsf{tm}(x, l+1) - \mathsf{tm}(v, \#\overline{v|_j}) > dist \tag{19}$$

$$\text{where} \qquad l = \#0 © \bar{c}|_{\#\overline{v|_j}}$$

with l describing the number of legitimate requests and authentication tokens that have been forwarded by the threatened medium before the point of time at which v_j occurs, and

$$P_d(v,j,c) \equiv \bar{c}_{\#\overline{v|_j}} = 1 \wedge v_j \neq \surd$$

being valid, if the jth element of v is not a tick and has been inserted by the adversary. Informally, from the above formula being valid, it follows that each challenge that has been issued with respect to a fraudulent message is received by the client before the next authentication token for a legitimate request has to be computed.

Though the fairness requirement seems to be considerably strong, it is sufficient in many cases in which the simple adversary model applies, namely in the door lock scenario.

From the fragments above, we straightforwardly yield a time dependent threat scenario (with simple adversary model) BA^T_s, corresponding to SA^T and including a fair variant $\mathrm{MD}^{T,s}_{\mathrm{Thr}}$ of the threatened medium.

Theorem 7 *Assuming the fair adversary,* SA^T *is strongly available wrt. threat scenario* BA^T_s.

Proof. The proof is given in [15]. □

There may be application situations in which the strong fairness condition as it is assumed in the proof above cannot be asserted. The weak variant of availability as being defined in Sect. 3.3 can be shown with a weaker fairness condition holding: If for a variant of the timed threatened medium $\mathrm{MD}^{T,s}_{\mathrm{Thr}}$, infinite input x provided, it can be assumed that in the output v infinitely many times a situation occurs for which the above fairness constraint holds, then it can be shown that infinitely many legitimate requests are indeed being served, thus satisfying the weak availability definition of Sect. 3.3.

The time-dependent server specification SA^T keeps authenticity as the time-independent variant SV^R does. The proof follows the same line of argumentation as the proof of Theorem 4. We therefore have, with BA^T_s being the threat scenario instantiation for the timed server and the simple adversary model

Theorem 8 SA^T *is strongly authentic w.r.t.* BA_s^T, *i.e.* $R_{Ath}(\mathrm{SA}^T,\mathrm{BA}_s^T)$ *holds.*

Concerning the advanced adversary model, the authenticity considerations for the time-independent case apply to the time-dependent case as well. With respect to availability, additional fairness properties have to be specified in order to deal with the insertion of fake authentication tokens leading to a failing authentication for a legitimate request.

4.4 Discussion of the Example

By the conduction of the example above, including the specification of a server component, the introduction of a security mechanism (a challenge response protocol based on [11]) in order to achieve authenticity, and the development of a variant of the protocol offering availability as well, it has been shown that the approach outlined in Sect. 3 is well suited for the formal treatment of those tasks that occur within the development of secure systems. In particular, it turned out that the approach allows a fine-grained analysis with respect to different adversary characterizations and security notions. The example points out the consequences of the adversary's behaviour to the security of the system: assuming the simple adversary model, stronger security properties have been proved than within the advanced model. Thus, the critical role of threat identification and risk analysis is reflected in our approach. For example, it has been clearly pointed out that the protocol provides only peer entity authentication, but that in case of the advanced adversary model message origin authentication is necessary to provide strong authenticity.

Two different styles have been utilized in the formal specification of the protocol: the state transition style allows protocol specification from an operational point of view that can be straightforwardly derived from an informal specification as for example given in the standard documents, whereas the relational style provides a more abstract view that is well suited for analysis and proof. Providing these different views, protocol design as well as analysis is supported, with the formal relationship between them given by the Focus refinement notions. Within our method, both styles of specification have to consider mechanism embedment. The example demonstrates that details of mechnanism implementation are of equal importance to security as protocol design itself: The loss of availability coming along with the first protocol variant is a consequence of the particular implementation, namely the buffering of challenges.

The loss of availability emphasises the need of consideration of the interdependence of different security aspects instead of concentrating on single aspects: Though the first variant perfectly satisfies authenticity requirements, it will only be of little use in practice. It is important to notice that the conduction of the authenticity proof has turned our attention to availability considerations.

The definition of the threat scenario template of Sect. 3.2 has turned out to be advantageous in our example: The definition of the simple and the advanced adversary model have been defined using the template, where the added properties only refer to the distinguishing properties of the different adversary characterizations.

5 Related Work

The formal treatment of security aspects in system design shows quite a long history: in the early seventies, Bell and LaPadula presented a first model covering confidentiality aspects ([1]), which since then has been followed by numerous formal security model proposals, for example the non-interference model [9] and the Terry-Wiseman model [25]. All these models are similar in that they provide an abstract system description, often in terms of a state transition system, and express security properties in terms of the abstract system model, for example by specifying secure states or secure state transitions. In general, they concentrate on particular security aspects, confidentiality in most cases ([1, 9], [25] covers confidentiality and a rudimentary notion of integrity), or even include specific mechanisms in the system model (for example, access control lists in [1]). Thus, they lack the desired flexibility with respect to the analysis of application specific security requirements and threat models within a general framework. Additionally, formal security models only show a vague relationship to system design and implementation. The use of an abstract system model intended to be kept as simple as possible, though covering a whole range of possible systems, and of specific description techniques lead to the security model being isolated from functional system development, thus raising the need for explicitly defining the relationship to a given system, which is only rarely done by the model designers (one of the few exceptions is given by [2]). In our approach, security analysis is immediately based on a specification describing the application at hand, with the process being closely integrated to functional system development.

We view formal security models as being helpful with respect to discussing security notions and analysing abstract security policies, but in general they are not suited to meet the requirements on a practically applicable and useful method to the design of secure systems. This is emphasized by the fact that security models have not been heavily used in commercial practice.

A lot of research has been performed in order to formally analyse a particular class of security mechanisms: cryptographic protocols. This work has to be considered with respect to our approach, since cryptographic protocols are among the most important security mechanisms relevant to our desired application field of communication systems. Authentication logics originating from [7] are the most popular technique being used for authentication protocols. They use modal logic techniques to derive the knowledge and beliefs of the protocol participants that allow the achievement of the authentication goals. Their practical relevance is due to the ease of analysis and the high degree of possible automization. Thus they are suited for the efficient analysis of protocols. On the other hand, they use a restricted communication and adversary model which allow only certain classes of attacks to be identified, and they do not cover confidentiality issues.

Further approaches, for example [17] and [24], address more complex attacks, including interleaved protocol execution, and confidentiality of key material. [17] models protocols as state transition systems, with transitions being enabled by the protocol entities or the adversary. The adversary's knowledge at each state ist analysed by exploiting the term-rewriting properties of the underlying cryptographic system. Security analysis is then by performed by analysing reachability within the state transition system. [24] uses higher order logic to model each protocol entity's view of communication. Both [17] and [24] allow the partly automization of proofs.

Despite their technical differences, security protocol analysis techniques satisfy the demand on strict separation of the mechanisms to be analysed and the seurity requirements the mechnanism is expected to satisfy. However, all of them use particular formalisms and/or communication and adversary models, and do not explicitly address embedment and system development issues. By their nature, they only address particular security aspects. Thus, they can only be viewed as an ingredient of a method meeting the requirements as stated in Sect. 1.

As shown in Sect. 4, our approach is able to deal with authentication protocols as well, assuming flexible and complex adversary models. However, we expect proofs to be more complex in a general setting like ours than within a tailored approach, thus it seems to be worthwile to use specific protocol analysis techniques for a quick analysis of a proposed authentication mechanism and use the results as part of a relational specification of a mechanism in our method. Further analysis with respect to embedment and those security aspects that are intentionally not covered by the protocol analysis methods is then performed within the FOCUS security development approach.

A lot of work covering similar topics as ours has been performed using process algebras, CSP in particular. Like in FOCUS, different security aspects and mechanisms can be analysed within CSP, ranging from non-interference ([13, 20]), authenticity ([16, 22]) and general confidentiality [22] to anonymity ([23]). Besides utilizing a well-known and established specification and verification technique, this work is remarkably characterized by treating security as a property of the system specification itself, without referring to an external security model.

The main technical difference between this work and ours occurs with respect to the communication model: CSP is based on synchronous communication, whereas the FOCUS semantics employs asynchronous communication. The synchronous model often allows easy and highly automated proofs, as is shown in [16], where model checking techniques are employed. However, there are complexity limitations, leading in [16] to only a reduced variant with single protocol entities of the Needham-Schroeder public key [19] protocol being automatically proved, the extension to the full variant has to be performed manually. We consider the asynchronous approach as advantageous: It offers a higher degree of abstraction, which makes the approach especially suitable for security analysis in early development phases, and more flexibility with respect to the specification of the adversary's influence on communication (e.g. the deletion of messages, though not included in the example of Sect. 4). Though in our example the proofs have been performed using pencil and paper, and some details had to be omitted for reasons of presentation, we are even able to conduct proofs formally, since there is automated proof suppart available within FOCUS ([21]).

We also consider the explicit provision of a threat scenario as useful with respect to further system development. Once security analysis is finished, the threat scenario can de dropped, and system development proceeds as usual. In [22]'s confidentiality considerations, for example, the adversary process is an integral part of the system specification, thus increasing specification complexity. Moreover, threat scenarios allow a uniform treatment of different security aspects, whereas the CSP papers employ several techniques, for example inference functions in [13] and certain system abstractions in [20] in order to express non-interference properties.

6 Conclusion and Further Work

We have introduced a new approach to the formal development of secure systems that is based on a procedure being established in practice and aims at a mechanism independent security notion, flexibility with respect to security aspects as well as integration of security analysis and development according to the functional requirements on the system. Application specific security requirements, as a result of threat identification and risk analysis, are formally modelled by threat scenarios which specify the anticipated behavior of the adversary, in particular her influence on communication. Security is defined as a relation on threat scenarios and systems.

The main focus of this paper has been to show the basic principles of our approach by conducting a comprehensive sample development of an authentic and available server. For purposes of presentation, our example has been simplified: we provide a simple protocol, and restrict the behaviour of the adversary (for example, by not considering attacks possibly leading to deletion of messages). However, our example is of practical relevance, since the protocol is only a slight abstraction of a standard protocol ([11]) and the adversary characterization seems to be reasonable for certain application situations (for example, a secure door lock). In [15], the approach has also been used to analyse the protocol variant of [12], with the subtle differences between these variants clearly shown.

The example shows a number of promising results that raise evidence that the approach is well-suited to support the formal development of secure systems in practice. By forcing to specify mechanism embedment as well, our method turns out to be suitable for the analysis of effects resulting from multiple executions of protocols and particular properties of communication, because the semantic model guarantees the consideration of the whole lifetime of the system instead of just a single protocol run. Additionally, it offers the opportunity to reason about different security aspects. Formal definitions of several security notions have been given.

Applicability of our method is supported by dividing the security notion in an application specific part (threat scenario) and a general part (security relation). In common applications, threat scenarios may be derived systematically from compositional system specifications, which has been shown for components modelling transmission media in communication systems.

Our approach particularly benefits from choosing Focus as the basis of formalization. Since Focus is a general purpose formal development method, it offers the opportunity to continue system development from those specifications that result from security analysis. On the other hand, security analysis can be performed at each stage of the system development. Systematic derivation of threat scenarios is supported: information flow to the adversary is modelled by simply adding (logical) channels to the system specification.

However, a lot of work remains to be done: the approach has to be generalized by defining further security relations, corresponding, for example, to confidentiality. Effects of multiple attacks, which may occur if an adversary is able to simultaneously attack several critical components, and of interleaving of protocol runs have to be investigated. To improve practicability, it is important to provide a set of threat scenario templates that can be instantiated for a variety of common threat analysis results, and a set of basic mechanism specifications. The approximation of cryptographic algorithms has to be further improved. A notion of compositionality with respect to different threats and threatened components is desirable.

Even in its initial state, our approach provides significant progress with respect to a formal method that reaches the aims mentioned above. With further work being performed, we will get close to a method that can be profitably applied in practice.

Acknowledgements

This work has been performed at Technical University of Munich as part of the author's omgoing PhD studies. The author is grateful to Manfred Broy, Ursula Hinkel, Christoph Hofmann, Walter Fumy, Volker Kessler, Helmut Kurth, Michael Munzert and several anonymous referees for helpful comments on earlier versions of this paper. The author is also grateful to Manfred Broy and Heribert Peuckert for motivating the work on this subject.

References

[1] D.E. Bell, L. LaPadula: Secure Computer Systems: Mathematical Foundations (NTIS AD-770 768), A Mathematical Model (NTIS AD-771 543), A Refinement of the Mathematical Model (NTIS AD-780 528), MTR 2547 Vol. I-III, ESD-TR-73-278, Mitre Corporation, Bedford MA, 1973

[2] D.E. Bell, L. LaPadula: Secure Computer Systems: Unified Exposition and Multics Interpretation, NTIS AD-A023 588, MTR 2997, ESD-TR-75-306, Mitre Corporation, Bedford MA, 1976

[3] M. Broy, F. Dederichs, C. Dendorfer, M. Fuchs, T.F. Gritzner, R. Weber: The Design of Distributed Systems – An Introduction to FOCUS – Revised Version, Technical Report TUM-19202-2, Technische Universität München, 1993

[4] M. Broy: (Inter-)Action Refinement: The Easy Way, in: *Program Design Calculi* (M. Broy, ed.), NATO ASI Series F, Vol. 118, Springer Verlag, New York, 1993

[5] M. Broy: Advanced Component Interface Specification, in: *Theory and Practice of Parallel Programming – Proceedings TPP '94* (T. Ito, A. Yonezawa, eds.), LNCS 907, Springer Verlag, New York, 1995

[6] M. Broy, K. Stølen: Interactive System Design, Book Manuscript, 1996

[7] M. Burrows, M. Abadi, R. Needham: A Logic of Authentication, Report 39, Digital Systems Research Center, Palo Alto, 1989

[8] O. Fries, A. Fritsch, V. Kessler, B. Klein (Hrsg.): Sicherheitsmechanismen: Bausteine zur Entwicklung sicherer Systeme, REMO Arbeitsberichte, Oldenbourg Verlag, München, 1993 (in German)

[9] J.A. Goguen, J. Meseguer: Security Policies and Security Models, in: *Proceedings of the 1982 IEEE Symposium on Research in Security and Privacy*, IEEE Computer Society Press, Oakland, CA, 1982, pp. 11–20

[10] S. Herda, S. Mund, A. Steinacker (Hrsg.): Szenarien zur Sicherheit informationstechnischer Systeme, REMO Arbeitsberichte, Oldenbourg Verlag, München 1993 (in German)

[11] ISO/IEC CD 9798: Information Technology – Security Techniques – Entity Authentication Mechanisms, Part 2: Entity Authentication Using Symmetric Techniques, 1992

[12] ISO/IEC DIS 10181-2.2: Information Technology – Open Systems Interconnection – Security Framework for Open Systems: Authentication Framework, 1993

[13] J.L. Jacob: Specifying Security Properties, in: *Developments in Concurrency and Communications* (C.A.R. Hoare, ed.), Addison-Wesley, Reading, MA, 1990

[14] V. Lotz: Threat Scenarios as a Means to Formally Develop Secure Systems, in: *Computer Security – ESORICS '96* (E. Bertino, H. Kurth, G. Martella, E. Montolivo, eds.), LNCS 1146, Springer Verlag, New York, 1996

[15] V. Lotz: Threat Scenarios as a Means to Formally Develop Secure Systems, Technical Report TUM-I9709, Technical University Munich, 1997 (available at http://wwwbroy.informatik.tu-muenchen.de/reports)

[16] G. Lowe: Breaking and Fixing the Needham-Schroeder Public Key Protocol Using CSP and FDR, in: *Tools and Algorithms for the Construction of Systems – TACAS '96* (T. Margaria, B. Steffens, eds.), LNCS 1055, Springer Verlag, New York, 1996

[17] C. Meadows: The NRL Protocol Analyzer: An Overview, *Journal of Logic Programming*, Vol. **19**, 1994

[18] S. Mund: Sicherheitsanforderungen – Sicherheitsmaßnahmen, in: *VIS '93* (P. Horster, G. Weck, Hrsg.), Vieweg Verlag, 1993 (in German)

[19] R. Needham, M. Schroeder: Using Encryption for Authentication in Large Networks of Computers, *Communications of the ACM*, **21**(12), 1978

[20] A.W. Roscoe, J.C.P. Woodcock, L. Wulf: Non-interference through Determinism, in: *Computer Security – ESORICS '94* (D. Gollmann, ed.), LNCS 875, Springer Verlag, New York, 1994

[21] R. Sandner, O. Müller: Theorem Prover Support for the Refinement of Stream Processing Functions, in: *Proceedings of TACAS '97*, Springer Verlag, New York, 1997

[22] S. Schneider: Security Properties and CSP, in: *Proc. of the 1996 IEEE Symposium on Research in Security and Privacy*, IEEE Computer Society Press, Oakland, CA, 1996

[23] S. Schneider, A. Sidiropoulos: CSP and Anonymity, in: *Computer Security – ESORICS '96* (E. Bertino, H. Kurth, G. Martella, E. Montolivo, eds.), LNCS 1146, Springer Verlag, New York, 1996

[24] E. Snekkenes: Formal Specification and Analysis of Cryptographic Protocols, PhD thesis, 1995

[25] P. Terry, S. Wiseman: A 'New' Security Policy Model, in: *Proc. of the 1989 IEEE Symposium on Research in Security and Privacy*, IEEE Computer Society Press, Oakland, CA, 1989, pp. 215–228