Pattern-based Confidentiality-Preserving Refinement

Holger Schmidt

University Duisburg-Essen, Germany, Faculty of Engineering, Department of Computer Science and Applied Cognitive Science, Workgroup Software Engineering, holger.schmidt@uni-duisburg-essen.de

Abstract. We present an approach to security requirements engineering, which makes use of special kinds of problem frames that serve to structure, characterize, analyze, and solve software development problems in the area of software and system security.
In this paper, we focus on confidentiality problems. We enhance previously published work by formal behavioral frame descriptions, which enable software engineers to unambiguously specify security requirements. Consequently, software engineers can prove that the envisaged solutions provide functional correctness and that the solutions fulfill the specified security requirements.

1 Introduction

As a consequence of an increasing demand for *security*, software engineers are not only confronted with functional requirements, but also with security requirements, although they are not experts in *security engineering*. In the early phases of software development, functional as well as security requirements have to be elicited and analyzed. This task alone is difficult enough, but the software engineers are then faced with realizing the requirements. Clearly, they need methods and techniques that help them to elicit, analyze, specify and finally realize security requirements in a *feasible* and *correct* way.

In earlier publications (cf. [4, 5, 6, 7]), we introduced a security engineering process that focuses on the early phases of software development. The basic idea is to make use of special patterns defined for structuring, characterizing, and analyzing *problems* that occur frequently in security engineering. Similar patterns for functional requirements have been proposed by Jackson [11]. They are called *problem frames*. Accordingly, our patterns are named *security problem frames*. Furthermore, for each of these frames, we define a set of *concretized security problem frames* that take into account generic security mechanisms to prepare the ground for solving a given security problem.

In this paper, we concentrate on the (concretized) security problem frames that deal with confidentiality. We present the following enhancements of our security requirements engineering approach:

- We underlay the (concretized) security problem frames with a *formal behavior description* to gain an unambiguous comprehension of the frames, and to clarify their semantics.

– As a prerequisite for software development based on *stepwise refinement*, we prove that the step from security problem frames to concretized security problem frames is a functionally correct refinement, which preserves the confidentiality requirement.
– We provide a point of contact to the *formal probabilistic (and possibilistic) security requirement descriptions* by Santen [18]. This allows software engineers to express security requirements in a well-defined way.

Furthermore, the work in this paper constitutes a basis to analyze the instantiation process of the (concretized) security problem frames and to investigate necessary applicability conditions for the frames.

The bottom line of these enhancements is a security engineering approach that focuses on the early phases of secure software development. Furthermore, it yields a formal specification of the software to be built, which constitutes a starting point for software design and implementation.

In the following, we first present Jackson's problem frames as well as security problem frames and concretized security problem frames in Sects. 2 and 3. In Sect. 4, we briefly introduce the formal specification language CSP (Communicating Sequential Processes) [9], which we subsequently use to create formal behavior descriptions of (concretized) security problem frames for confidential data transmission (using encryption). Furthermore, we analyze these formal models with respect to confidentiality-preserving refinement in Sect. 5. Section 6 discusses related work, and the paper closes with a summary and perspectives in Sect. 7.

2 Problem Frames

Problem frames are a means to analyze and classify software development problems. Jackson [11] describes them as follows: "A problem frame is a kind of pattern. It defines an intuitively identifiable problem class in terms of its context and the characteristics of its domains, interfaces and requirement." Problem frames are described by *frame diagrams*, which basically consist of rectangles and links between these (see frame diagrams in Figs. 1 and 2). The task is to construct a *machine* that improves the behavior of the environment it is integrated in.

Plain rectangles denote *domains* (that already exist), a rectangle with a single vertical stripe denotes a *designed domain* physically representing some information, and a rectangle with a double vertical stripe denotes the machine to be developed. *Requirements* are denoted with a dashed oval. The connecting lines represent interfaces that consist of *shared phenomena*. Shared phenomena may be events, operation calls, messages, and the like. They are observable by at least two domains, but controlled by only one domain. For example, if a user types a password to log into an IT-system, this is a phenomenon shared by the user and the system, which is controlled by the user. A dashed line represents a requirements reference, and the arrow shows that it is a *constraining* reference. Furthermore, Jackson distinguishes *causal* domains that comply with

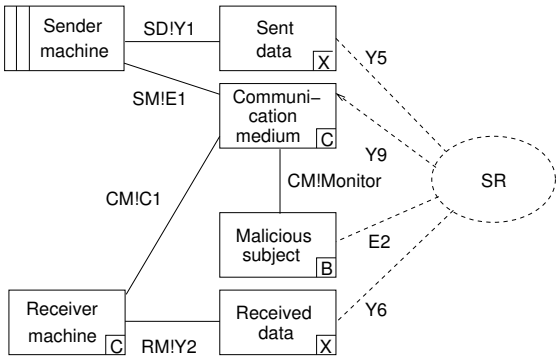

Fig. 1. Security problem frame for confidential data transmission

some physical laws, *lexical* domains that are data representations, and *biddable* domains that usually are people.

In the frame diagram depicted in Fig. 1, a marker "X" indicates a lexical domain, "B" indicates a biddable domain, and "C" indicates a causal domain. The notation "SM!E1" means that the phenomena of interface E1 are controlled by the machine domain Sender machine.

Problem frames greatly support developers in analyzing problems to be solved. They show what domains have to be considered, and what knowledge must be described and reasoned about when analyzing the problem in depth. Developers must elicit, examine, and describe the relevant properties of each domain. These descriptions form the *domain knowledge*.

The domain knowledge consists of *assumptions* and *facts*. Assumptions are conditions that are needed, so that the requirements are realizable. Usually, they describe required user behavior. For example, it must be assumed that a user ensures not to be observed by a malicious user when entering a password. Facts describe fixed properties of the problem environment regardless of how the machine is built.

Requirements describe the environment, the way it should be, after the machine is integrated. In contrast to the requirements, the *specification* of the machine gives an answer to the question: "How should the machine act, so that the system, i.e., the machine together with the environment, fulfills the requirements?" Specifications are descriptions that are sufficient for building the machine. They are implementable requirements.

3 (Concretized) Security Problem Frames

To meet the special demands of software development problems occurring in the area of security engineering, we introduced security problem frames (SPF) [4, 5]. SPFs are a special kind of problem frames, which consider *security requirements*. The SPFs we have developed strictly refer to the *problems* concerning security. They do not anticipate a solution. For example, we may require the confidential

transmission of data without mentioning encryption, which is a means to achieve confidentiality.

Solving a security problem is achieved by applying generic security mechanisms (e.g., encryption to keep data confidential), thereby transforming security requirements into *concretized security requirements*. The generic security mechanisms are represented by concretized security problem frames (CSPF). The benefit of considering security requirements without reference to potential solutions is the clear separation of problems from their solutions, which leads to a better understanding of the problems and enhances the re-usability of the problem descriptions, since they are independent of solution technologies.

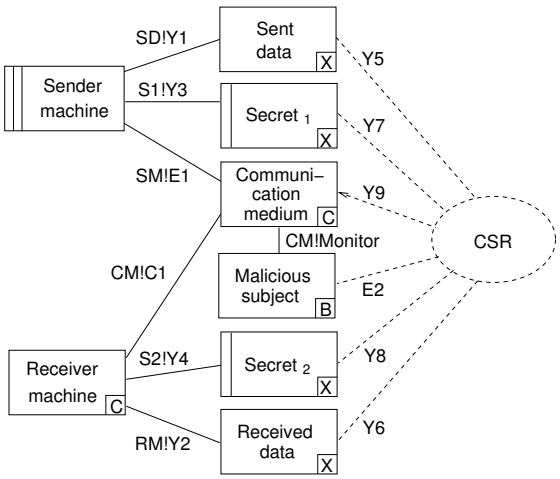

Fig. 2. Concretized security problem frame for confidential data transmission using encryption

Figure 1 shows the frame diagram of the SPF for confidential data transmission. The domain Sent data denotes the data that is sent by a sender, represented by the machine domain Sender machine. Analogously, the domain Received data denotes the data that is received by the domain Receiver machine. The data is transmitted over some network, which is represented by the domain Communication medium. Informally speaking, the sender machine generates the transmitted data from the sent data, and the receiver machine generates the received data from the communication medium. In this scenario, a potential attacker represented by the domain Malicious subject can eavesdrop on the Communication medium. The informal security requirement SR is described as follows:

> Malicious subject should not be able to infer Sent data and Received data except for their length by eavesdropping on Communication medium.

One of the CSPFs for confidential data transmission considers (symmetric and asymmetric) encryption. Its frame diagram is shown in Fig. 2. In transforming the security requirement for confidential data transmission into a concretized

security requirement CSR, the domains Secret_1 and Secret_2 are introduced for the encryption mechanism. The informal concretized security requirement CSR is described as follows:

> Malicious subject should not be able to infer Sent data and Received data except for their length without Secret_1 and Secret_2 by eavesdropping on Communication medium. Malicious subject should not be able to obtain Secret_1 and Secret_2.

In the subsequent Sects. 4 and 5, we first equip the frame diagrams of the (C)SPFs depicted in Figs. 1 and 2 with formal behavior descriptions, and second we analyze these formal descriptions with respect to confidentiality-preserving refinement.

4 Formal Foundation of (C)SPFs

The software development principle of *stepwise refinement* is popular in software engineering, and is also well supported by *formal methods*. When performing stepwise refinement, a software engineer develops software by creating intermediate levels of abstraction. Starting with the requirements, an abstract *specification* is constructed, which is refined by a more concrete *implementation*. Then, the implementation must be verified against the specification, and further refinement steps are accomplished until the desired level of abstraction is achieved.

Refinement is traditionally either data-refinement or behavior-refinement. Since the (C)SPFs deal with interfaces and communicating domains rather than with states, we decided to describe them using CSP (Communicating Sequential Processes) [9]. CSP is a model-based formal method to describe parallel processes that communicate synchronously via message passing. Furthermore, with the model-checker FDR2 (Failure-Divergence Refinement) [14] sophisticated tool support is available for CSP.

In Sect. 4.1, we present a general procedure to create a formal CSP model for a given (C)SPF. In Sect. 4.2, we apply the procedure described in Sect. 4.1 to create CSP models for the (C)SPFs shown in Figs. 1 and 2. Furthermore, we formalize the (concretized) security requirements of the (C)SPFs based on the functional CSP models in Sect. 4.3.

In Sect. 5, we show that the CSPF model in Fig. 7 is a refinement of the SPF model in Fig. 5, and we include the confidentiality requirements presented in Sect. 4.3 in our analysis in order to show a *confidentiality-preserving refinement* (CPR). Figure 3 describes that CPR is not only of interest on the pattern level, but also on the instance level. It is desirable that once we have shown the functional and confidentiality-preserving refinements on the pattern level, they also apply (conditionally or not) to the instance level. We call the latter *confidentiality-preserving instantiation*.

Applying CSP and stepwise refinement to the (C)SPF approach has several benefits:

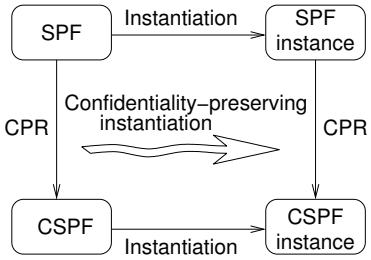

Fig. 3. Confidentiality-preserving instantiation

- Enable a developer to *unambiguously express security requirements* captured by (C)SPFs.
- Since problem frames and (C)SPFs as such only provide a static view of a system [1], we obtain an understanding of the dynamic aspects of (C)SPFs.
- Allow one to verify that the functional and the security requirements of a SPF are *correctly* implemented by an associated CSPF, i.e., that the functionality and the security requirement are preserved.
- Verification is tool-supported by the model-checker FDR2.
- The CSP models provide a point of contact to the formal probabilistic (and possibilistic) security requirement descriptions by Santen [18].

Note that the enhancements described in this paper are not restricted to (C)SPFs concerning confidentiality. In fact, also frames concerning integrity and authentication problems can be translated into CSP models to formally express the security requirements they capture.

4.1 Formal (C)SPF Models in CSP

We make use of the CSP ASCII notation named CSP_M since this is a prerequisite for formal verification using the model-checker FDR2.

Using CSP_M notation, we define *processes* that interact only by communicating. Communication takes the form of visible *events* or *actions*. A sequence of events produced by a process is called a *trace*. The set of all traces that can be produced by a process P are denoted $traces(P)$. Let a be an action and P be a process; then $a->P$ is the process that performs a and behaves like P afterwards. This is called *prefixing*. A process can have a name, e.g., $Q = a->P$. *Recursion* makes it possible to repeat processes and to construct processes that go on indefinitely, e.g., $Q = a->Q$.

We can make use of input and output data: the expression $in?x$ binds the identifier x to whatever value is chosen by the environment, where x ranges over the type of *channel in*. The expression $out!y$ binds an output action to the identifier y, where y ranges over the type of channel *out*. The variables x and y

[1] A formal metamodel covering the static nature of problem frames is already developed (cf. [8]).

can then be used in the process following the prefix. By convention, ? denotes input data and ! denotes output data.

A process acts in a *nondeterministic* way when its behavior is unpredictable because it is allowed to make internal decisions that affect its behavior as observed from outside. The *replicated internal choice operator* $|\sim|$ models these internal decisions: let P be a process and X a finite and non-empty data type, then $|\sim|\ a : X \bullet P(a)$ behaves according to the selected a. This operator gives the environment no control over which data item is chosen. In contrast, the *replicated external choice operator* $[]$ models external decisions: let P be a process and X a non-empty data type, then $[]\ a : X \bullet P(a)$ behaves according to the a selected by the environment.

To formalize a given (C)SPF, we describe each of its domains as a recursive CSP process. The interfaces and the control direction of the shared phenomena (control flow) of a domain are translated into CSP channels as well as input and output events. For lexical shared phenomena, we define data types and declare the corresponding channels to be of one of these data types.

Note that when using a model-checker such as FDR2 to analyze real-world problems, we have to address the state explosion problem. A common approach to keep the model-checking effort manageable is to simplify the system to be analyzed. For that reason, we usually must define simplified data types.

We describe a (C)SPF as a CSP process consisting of the CSP processes of all of its domains. The processes are combined using *synchronized parallel communication* denoted by $[|\ |]$. The synchronization is accomplished over the channels modelling the interfaces that connect the domains.

The described procedure can be applied to express any (C)SPF as a formal CSP model.

4.2 CSP Models of (C)SPFs Confidential Data Transmission (using Encryption)

As examples, we present in Figs. 5 and 7 the CSP models of the (C)SPFs confidential data transmission (using encryption) shown in Figs. 1 and 2.

```
-- Data type definitions
datatype Plaintext = p1 | p2 | p3 | p4
datatype Length = short | long

-- Channel declarations
channel SD_Y1, RM_Y2, SM_E1_S, CM_C1_S: Plaintext
channel CM_monitor_S : Length

-- Function definition
f(p) = if p==p1 or p==p2 then short else long
```

Fig. 4. Type and function definitions as well as channel declarations for the CSP model in Fig. 5

Figure 4 shows type and function definitions as well as channel declarations for the CSP model in Fig. 5. We define a simple data type named *Plaintext* with four values $p1, p2, p3, p4$. Then, we declare the channels SD_Y1, RM_Y2, SM_E1_S, and CM_C1_S (see Fig. 1) to be of this data type, i.e., all events communicated over these channels are $p1, p2, p3$, or $p4$.

We represent the interface between the Malicious subject domain and the Communication medium domain by a channel $CM\_monitor\_S$ of the data type *Length*. The data items leaked over this channel are defined by a *leakage function* f. As an example, the leaked data items are *short* and *long*, and they correspond to the lengths of the plaintexts sent over the channels of the process $CM\_monitor\_S$ (see definition of function f in Fig. 4).

```
-- Process for domain Sent data
SD_S = |~| pt : Plaintext @ SD_Y1!pt -> SD_S

-- Process for domain Sender machine
SM_S = SD_Y1?pt -> SM_E1_S!pt -> SM_S

-- Process for domain Communication medium
CM_S = SM_E1_S?pt -> CM_monitor_S!f(pt) -> CM_C1_S!pt -> CM_S

-- Process for domain Malicious subject
MS_S = CM_monitor_S?l -> MS_S

-- Process for domain Receiver machine
RM_S = CM_C1_S?pt -> RM_Y2!pt -> RM_S

-- Process for domain Received data
RD_S = RM_Y2?pt -> RD_S

-- Process for SPF Confidential Data Transmission
SPF_CONF = ((((SD_S [| {|SD_Y1|} |] SM_S)
       [| {|SM_E1_S|} |] CM_S) [| {|CM_monitor_S|} |] MS_S)
            [| {|CM_C1_S|} |] RM_S) [| {|RM_Y2|} |] RD_S
```

Fig. 5. CSP model of SPF depicted in Fig. 1

We describe each domain of the SPF in Fig. 1 as a recursive CSP process, e.g., the process SM_S in Fig. 5 is a formal representation of the domain Sender machine of the SPF confidential data transmission. The process SD_S in Fig. 5 offers a data item pt that might be internally processed (expressed using the replicated internal choice operator) to the environment using the channel SD_Y1. Then, the process SM_S begins with reading in this data item pt over channel SD_Y1, and pt might be passed over to the environment using the channel SM_E1_S.

We specify the SPF in Fig. 1 as a process SPF_CONF in Fig. 5 that combines all formalized domains of the SPF confidential data transmission. For example, the process SM_S synchronizes over the channel SD_Y1 with the process SD_S,

or, informally speaking, the domain Sender machine reads data from the domain Sent data.

```
-- Data type definitions
datatype Ciphertext = c1 | c2 | c3 | c4
datatype Secret = s1 | s2 | s3 | s4

-- Channel declaration
channel S1_Y3, S2_Y4 : Secret
channel SM_E1_I, CM_C1_I : Ciphertext.Secret
channel CM_monitor_I : Ciphertext.Length

-- Function definition
encr(p1,s1) = c1   encr(p1,s2) = c2   encr(p1,s3) = c1   encr(p1,s4) = c2
encr(p2,s3) = c2   encr(p2,s4) = c1   encr(p2,s1) = c2   encr(p2,s2) = c1
encr(p3,s1) = c3   encr(p3,s2) = c4   encr(p3,s3) = c3   encr(p3,s4) = c4
encr(p4,s3) = c4   encr(p4,s4) = c3   encr(p4,s1) = c4   encr(p4,s2) = c3

decr(c1,s1) = p1   decr(c1,s2) = p2   decr(c1,s3) = p1   decr(c1,s4) = p2
decr(c2,s1) = p2   decr(c2,s2) = p1   decr(c2,s3) = p2   decr(c2,s4) = p1
decr(c3,s1) = p3   decr(c3,s2) = p4   decr(c3,s3) = p3   decr(c3,s4) = p4
decr(c4,s1) = p4   decr(c4,s2) = p3   decr(c4,s3) = p4   decr(c4,s4) = p3
```

Fig. 6. Type and function definitions as well as channel declarations for the CSP model in Fig. 7

Figure 6 shows type and function definitions as well as channel declarations for the CSP model in Fig. 7. We introduce data types *Secret* and *Ciphertext*, and the functions *encr* and *decr* in Fig. 6 to model that encryption is used in the CSPF confidential data transmission using encryption. The functions *encr* and *decr* model a length-preserving cryptographic mechanism. Furthermore, we declare the channels $S1\_Y3$ and $S1\_Y4$ to be of type *Secret* and the channels SM_E1_I and CM_C1_I to be of the composed type *Ciphertext.Secret*. The role of the Malicious subject's monitoring channel has changed: the channel $CM\_monitor\_I$ not only leaks the lengths of the transferred data items to the environment, but also the complete ciphertexts. For that reason, the channel $CM\_monitor\_I$ is of the composed type *Ciphertext.Length*.

We introduce two new processes $S1\_I(s)$ and $S2\_I(t)$ in the CSP model in Fig. 7. They stand for the domains Secret_1 and Secret_2 of the CSPF confidential data transmission using encryption. Both processes are equipped with parameters that represent the secrets chosen by the environment. The process SM_I corresponds to the process SM_S of the CSP model in Fig. 5, and is extended by reading in a secret s over the channel $S1\_Y3$. Proceeding with the events of the process SM_I, the function *encr* is applied to a plaintext pt using a secret s, and the result as well as the secret s is passed over to CM_I using the channel SM_E1_I. Afterwards, the ciphertext ct as well as the secret s are passed over to the environment using the channel CM_C1_I. In a similar way, the function

```
-- Process for domain Sent data
SD_I = |~| pt : Plaintext @ SD_Y1!pt -> SD_I

-- Process for domain Secret_1
S1_I(s) = S1_Y3!s -> S1_I(s)

-- Process for domain Sender machine
SM_I = SD_Y1?pt -> S1_Y3?s -> SM_E1_I!encr(pt,s).s -> SM_I

-- Process for domain Communication medium
CM_I = SM_E1_I?ct.s -> CM_monitor_I!ct.f(decr(ct,s))
          -> CM_C1_I!ct.s -> CM_I

-- Process for domain Malicious subject
MS_I = CM_monitor_I?ct.l -> MS_I

-- Process for domain Secret_2
S2_I(t) = S2_Y4!t -> S2_I(t)

-- Process for domain Receiver machine
RM_I = CM_C1_I?ct.s -> S2_Y4?t -> RM_Y2!decr(ct,t) -> RM_I

-- Process for domain Received data
RD_I = RM_Y2?pt -> RD_I

-- Process for CSPF Confidential Data Transmission using Encryption
CSPF_CONF_ENCRYPTION(s, t) = ((((
       (SD_I [| {|SD_Y1|} |] SM_I) [| {|S1_Y3|} |] S1_I(s))
            [| {|SM_E1_I|} |] CM_I) [| {|CM_monitor_I|} |] MS_I)
                  [| {|CM_C1_I|} |] (RM_I [| {|S2_Y4|} |] S2_I(t)))
                     [| {|RM_Y2|} |] RD_I

-- initialization: choosing secrets
INIT_CSPF_CONF_ENCRYPTION =
       ([] x : Secret, y : Secret, x == y @ CSPF_CONF_ENCRYPTION(x, y))
```

Fig. 7. CSP model of CSPF depicted in Fig. 2

decr is used when receiving encrypted data (see process RM_I). The process MS_I corresponds to the process MS_S of the CSP model in Fig. 5, and is changed to be able to receive the ciphertexts and their lengths over channel $CM\_monitor\_I$.

We specify the CSPF in Fig. 2 as a process $CSPF\_CONF\_ENCRYPTION$ (s, t) in Fig. 7 that combines all formalized domains of the CSPF confidential data transmission using encryption.

The parameters of the processes $S1\_I(s)$ and $S2\_I(t)$ provide a point of contact to the *(C)SPFs Distributing Secrets* [5], which consider the problem to communicate *matching* secrets to those subjects who are privileged to receive

them (cf. [4]). Since no CSPF Distributing Secrets is considered in this paper, we simulate the mechanism using the replicated external choice operator to define the process *INIT_CSPF_CONF_ENCRYPTION*. Hence, the secrets are chosen by the environment, and as an example for modelling a *symmetric* encryption mechanism, the constraint $x == y$ requires both secrets to be equal.

Using FDR2, we successfully verified that the CSP models in Figs. 5 and 7 are deadlock-free and livelock-free.

4.3 Formal Description of Confidentiality Requirements

Confidentiality requirements can be expressed as *information flow properties* of two flavors:

possibilistic based on the fact that an IT system has a system behavior, which produces observations visible to the environment, there must exist at least one alternative possible system behavior that produces the same observation.

probabilistic stochastic system behavior is taken into account.

In this section, we consider *possibilistic* information flow properties, and we apply the framework for the specification of confidentiality requirements by Santen [18] to capture the confidentiality requirement of the (C)SPF confidential data transmission (using encryption).

In general, we call the formal description of a confidentiality requirement a *confidentiality property* (cf. Definition 9 in [18]). Since confidentiality properties are predicates on *sets* of traces, they cannot be modelled in CSP, and thus cannot be verified using FDR2. Nevertheless, we can specify a confidentiality property "on paper" and prove that a given machine and environment satisfy the property.

There does not exist *the* confidentiality property that allows us to express every (informal) confidentiality requirement. Instead, an adequate confidentiality property depends on the confidentiality requirement that it formalizes (cf. [15] for a comprehensive overview of possibilistic information flow properties).

The concept of indistinguishable traces (cf. [18, p. 223]) is the foundation for defining confidentiality properties. Given a set of channels W, two traces s, $t \in traces(P)$ of a process P are *indistinguishable* by W (denoted $s \equiv_W t$) if their projections to W are equal: $s \equiv_W t \Leftrightarrow s \upharpoonright W = t \upharpoonright W$, where $s \upharpoonright W$ is the projection of the trace s to the sequence of events on W. The *indistinguishability class* $J_W^{P,k}(o)$ contains the traces of P with a length of at most k that produce the observation o on W.

Applied to the CSP models presented in Sect. 4.2 this means that any distinction (e.g., data item length is *short* or *long*) the malicious subject can make about the internal communication of the system (e.g., sending different plaintexts and ciphertexts) based on the observations on *CM_monitor_S* and *CM_monitor_I* is information revealed by the system. Conversely, any communication that cannot be distinguished by observing *CM_monitor_S* and *CM_monitor_I* is concealed by the system. We can determine two indistinguishability classes, one that contains those traces that produce the observation *short* on the monitoring

channel, and another one that contains those traces that produce the observation *long* on the monitoring channel.

An *adversary model* (cf. [18, p. 222]) is a system model that consists of the machine to be developed, the honest user environment, the adversary enviroment, and their interfaces. The CSP models presented in Sect. 4.2 constitute valid adversary models.

As defined by Santen (cf. Definition 11 in [18]), a *mask* $\mathcal{M}$ for an adversary model is a set of subsets of the traces over the alphabets of the processes modelling the machine to be developed, the honest user environment, and the adversary enviroment such that the members of each set are indistinguishable by observing the monitoring channel of the adversary enviroment W: $\forall\, M : \mathcal{M};\ t_1, t_2 : M \bullet t_1 \equiv_W t_2$.

If Malicious subject observes the single event $CM\_monitor\_S.l$ (where $l \in Length$), then s/he knows that exactly one data transfer has taken place. All traces of the form

$$t_0(pt) = \begin{cases} \langle SM\_E1\_S.pt, CM\_monitor\_S.short\rangle & \text{if } pt \in \{p1, p2\}, \\ \langle SM\_E1\_S.pt, CM\_monitor\_S.long\rangle & \text{else,} \end{cases}$$

where $pt \in Plaintext$, produce the observation $CM\_monitor\_S.l$ for Malicious subject. According to the informal confidentiality requirement as it has been stated in Sect. 3, this observation should not allow Malicious subject to infer the transferred plaintext.

Note: the leakage function f must not be injective. If the function f were injective, i.e., f assigns exactly one plaintext to each length, the confidentiality requirement could not be achieved.

A mask $\mathcal{M}_0$ supporting the confidentiality requirement needs to require that for a given length l all variations of plaintexts pt in the parameter list of the trace t_0 are possible causes of the observation $CM\_monitor\_S.l$. Therefore, the sets $M_0 = \{t_0(p1), t_0(p2)\}$ and $M_1 = \{t_0(p3), t_0(p4)\}$ should be members of $\mathcal{M}_0$.

If the traces in a set $M \in \mathcal{M}$ are indistinguishable by observing the monitoring channel, then the differences between these traces are kept confidential. This confidentiality property is named *concealed behavior* (cf. [18, p. 228]). It is formalized based on a set inclusion $M \subseteq J_W^{QE,k}(o)$, where the process QE is a *variant* (i.e., a purely deterministic process, cf. [18, p. 228]) of the adversary model. It is required that members of $\mathcal{M}$ are either completely contained in an indistinguishability class, or not at all. One says that the set of indistinguishability classes $\mathcal{I}$ *covers* $\mathcal{M}$.

In general, a given adversary model satisfies a confidentiality property, which is defined based on a *basic confidentiality property* (cf. [18, p. 225]), if there exists a probabilistic deterministic realization of a machine that satisfies the basic confidentiality property in all admissible environments. In the case of concealed behavior, the question is if there is an adversary model that covers a given mask.

To show that the adversary model represented by the CSP model in Fig. 5 conceals the mask $\mathcal{M}_0$, a deterministic machine realization must be found such that its composition with all realizations of the environment covers $\mathcal{M}_0$.

We choose the implementation of the CSP model in Fig. 5 that resolves the nondeterministic choice of the process SM_S in Fig. 5 by a probabilistic choice with equal probabilities for all alternatives.

The admissible environments consist of realizations that deterministically produce traces according to the pattern $t_0(pt)$, where $pt \in Plaintext$, i.e., data transmissions from Sender machine to Receiver machine.

The members M_0 and M_1 of $\mathcal{M}_0$ are covered by the indistinguishability classes of all resulting variants of SPF_CONF, because the chosen machine realization does not exclude any of the traces $t_0(pt)$, where $pt \in Plaintext$.

In summary, we formally described the (C)SPFs confidential data transmission (using encryption) by CSP models. Furthermore, we introduced a formal description of a sample confidentiality requirement. The presented approach is not limited to express an informal confidentiality requirement only by the confidentiality property concealed behavior. In contrast, other confidentiality properties (e.g., *ensured entropy* [18, p. 229]) can be used. In the next sections, we verify that the CSPF model is a correct refinement of the SPF model, and that the confidentiality requirement is preserved under refinement.

5 Confidentiality-Preserving Refinement

Refinement is the transformation of an abstract specification into a concrete specification (implementation). CSP supports three types of process refinements:

Trace refinement A process Q trace-refines a process P, if all the possible sequences of communications, which Q can perform, are also possible in P.
Failure refinement Trace refinement extended by consideration of deadlocks.
Failure-divergence refinement Failure refinement extended by consideration of livelocks.

We first prove on a functional level that the CSPF confidential data transmission using encryption failure-divergence refines the SPF confidential data transmission (Sect. 5.1). Second, we show that the confidentiality requirement is preserved in the CSPF confidential data transmission using encryption (Sect. 5.2).

5.1 (C)SPF Functional Refinement

To show that a CSPF refines a SPF, we make use of the failure-divergence refinement. Since all structural elements of a SPF are preserved in an associated CSPF, we can show a failure-divergence refinement after we reduce the structural additions of the CSPF to the SPF structure:

- We hide events that can only be communicated in the CSPF model, i.e., all events communicated over $S1\_Y3$ and $S2\_Y4$.
- We map those events that have a more concrete structure in the CSPF model to events that are compatible with events of the SPF model, e.g., events passed over SM_E1_I are substituted by events passed over SM_E1_S. This mapping constitutes a data refinement.

– The data refinement is characterized by the fact that a plaintext is refined by a pair consisting of a ciphertext and a secret. In the implementation, two such pairs are indistinguishable if the ciphertexts are equal, because the malicious subject can observe the ciphertexts and their lengths using channel *CM_monitor_I*.

We construct a CSP process *ABS_CSPF_CONF_ENCRYPTION* using *relational renaming* $[[< -]]$ and *hiding* $\setminus$ (the corresponding CSP artifacts are not shown in this paper because of space limitations). This process failure-divergence refines the CSP process *SPF_CONF*, which we successfully verified using FDR2[2]. This kind of refinement is called *behavior refinement of adversary models* (cf. [18, p. 232]).

5.2 (C)SPF Refinement of Confidentiality Requirements

After we have shown that the CSPF model in Fig. 7 is a functionally correct refinement of the SPF model in Fig. 5, we include the confidentiality requirement presented in Sect. 4.3 in our analysis in order to show that it is a confidentiality-preserving refinement.

In the CSP model of the CSPF confidential data transmission using encryption depicted in Fig. 7, the monitoring channel *CM_monitor_I* has changed compared to its specification *CM_monitor_S*. This harbors the danger that leaks are introduced in the implementation through stepwise refinement.

To show that the confidentiality property concealed behavior, i.e., that the CSP model of the SPF confidential data transmission in Fig. 5 conceals $\mathcal{M}_0$, is preserved (and no further leaks are introduced), we must show that a similar property applies for the CSP model of the CSPF confidential data transmission using encryption in Fig. 7. Figure 8 informally describes the approach.

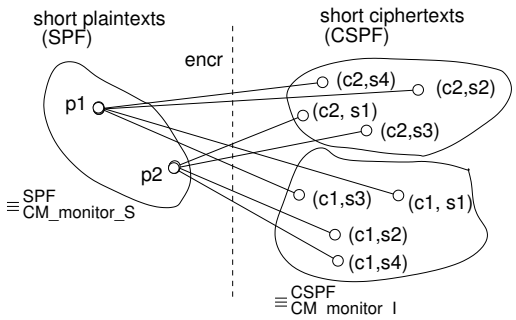

Fig. 8. Concretization and indistinguishability

Let $p1$ and $p2$ be indistinguishable plaintexts with respect to the channel *CM_monitor_S* and the SPF model, i.e., $p1 \equiv^{SPF}_{CM\_monitor\_S} p2$. Accord-

[2] FDR2 has to check 3.732.737 states with 10.323.200 transitions, which takes $\sim$ 3 minutes on a dual-core machine with 2 $\times$ 2 GHz and 2GB RAM.

ing to the function *encr*, the plaintext $p1$ is represented by the pairs $(c1, s1)$, $(c1, s3)$, $(c2, s2)$, and $(c2, s4)$, and the plaintext $p2$ is represented by the pairs $(c1, s2)$, $(c1, s4)$, $(c2, s1)$, and $(c2, s3)$. Thereby, the pairs containing $c1$ as well as the pairs containing $c2$ are indistinguishable with respect to the channel $CM\_monitor\_I$ and the CSPF model. Similar facts apply to the *long* plaintexts and ciphertexts.

In the concrete CSPF model, all traces of the form

$$\langle SM\_E1\_I.ct.s, CM\_monitor\_I.ct.l, CM\_C1\_I.ct.s \rangle$$

where $ct \in Ciphertext$, and $s \in Secret$ produce the observation $CM\_monitor\_I.ct$ for Malicious subject. In Sect. 5.1, the behavior refinement changed the monitoring channel and refined the data communicated by the processes. Since the confidentiality property concealed behavior refers to both, the monitoring channel and the data, we must relate the concrete monitoring channel and the data back to the abstract ones originally referred to by the confidentiality property. Applied to concealed behavior, this general concept provides a basis for defining *refined concealed behavior* (cf. [18, p. 239]).

After the re-abstraction, we must check if the re-abstracted traces are members of M_0: the re-abstracted traces are the same traces as the abstract ones. For that reason, the CSP model of the CSPF confidential data transmission using encryption in Fig. 7 conceals $\mathcal{M}_0$, and the confidentiality property concealed behavior is preserved in the CSPF confidential data transmission using encryption.

6 Related Work

In this section, we discuss our work in connection with other formal approaches to security requirements engineering, as well as with other approaches to formalize problem frames. Note that not all relevant publications are mentioned because of space limitations.

Li et al. [13] use an extended CSP version [12] to systematically derive a specification from requirements. Their work does not consider non-functional requirements such as security requirements. Furthermore, biddable domains are not formalized. Since biddable domains are used to model unpredictable parts of the environment (such as honest and malicious users), we believe that this is a key feature to security requirements engineering.

Nelson et al. [17] describe problem frames as well as requirements using Alloy [10]. Compared to our work, security requirements and their refinement to specifications are not considered. Additionally, Alloy does not allow to express security requirements in terms of information flow properties.

KAOS – Keep All Objectives Satisfied[3] is a goal-driven requirements engineering approach that can also be used to address security requirements by means of anti-goals [19]. A linear real-time temporal logic is used to formalize goals. The goals and further ingredients such as domain properties as well as

[3] c.f. `http://www.info.ucl.ac.be/~avl/ReqEng.html`

pre- and postconditions form patterns that can be instantiated and negated to describe anti-goals. This formal approach is also adopted by Secure Tropos [16].

SREF – Security Requirements Engineering Framework [2] is a framework that defines the notion of security requirements, considers security requirements in an application context, and helps answering the question whether the system can satisfy the security requirements. Haley et al. [1] introduce the notion of a *trust assumption*, which is "an assumption by an analyst that the specification of a domain can depend on certain properties of some other domain in order to satisfy a security requirement". To decide whether a system can satisfy the security requirements, Haley et al. make use of structured informal and formal argumentation [3]. A two-part argument structure for security requirement satisfaction arguments consisting of an informal and a formal argument is proposed. In combination with trust assumptions, satisfaction arguments facilitate showing that a system can meet its security requirements.

In contrast to our work, the approaches by van Lamsweerde (including Secure Tropos) and SREF do not allow to express security requirements in terms of information flow properties. Moreover, the refinement of security requirements to specifications is not covered.

7 Conclusion and Future Work

The paper at hand constitutes an extension of the (C)SPF approach by a formal foundation and a pattern-based refinement analysis, which is heavily based on Santen's work on the preservation of security requirements under refinement [18]. In fact, we combined his proposals for a formal security requirements analysis approach with our hitherto informal (C)SPF approach.

The main benefits of the extension are the clear and unambiguous security requirements descriptions and the support of verifiably correct and confidentiality-preserving refinement steps.

In the future, we would like to consider probabilistic confidentiality properties, applicability conditions and environment patterns for (C)SPFs, and the compositionality of confidentiality-preserving refinement.

Acknowledgments My sincere gratitude belongs to Thomas Santen for his great support on this work. Thomas gave feedback on the CSP artifacts, and helped to improve my understanding of confidentiality requirements as well as their formal specification. Furthermore, I thank Maritta Heisel for her extensive and valuable feedback on my work.

References

[1] C. Haley, R. Laney, J. Moffett, and B. Nuseibeh. Picking battles: The impact of trust assumptions on the elaboration of security requirements. In J. et.al., editor, *iTrust'04*, pages 347–354, 2004.

[2] C. B. Haley, R. Laney, J. Moffett, and B. Nuseibeh. Security requirements engineering: A framework for representation and analysis. *IEEE Transactions on Software Engineering*, 34(1):133 – 153, 2008.

[3] C. B. Haley, J. D. Moffett, R. Laney, and B. Nuseibeh. Arguing security: Validating security requirements using structured argumentation. In *Proceedings of the 3rd Symposium on Requirements Engineering for Information Security (SREIS'05) held in conjunction with the 13th International Requirements Engineering Conference (RE'05)*, 2005.

[4] D. Hatebur, M. Heisel, and H. Schmidt. Security engineering using problem frames. In G. Müller, editor, *Proceedings of the International Conference on Emerging Trends in Information and Communication Security (ETRICS)*, LNCS 3995, pages 238–253. Springer-Verlag, 2006.

[5] D. Hatebur, M. Heisel, and H. Schmidt. A pattern system for security requirements engineering. In *Proceedings of the International Conference on Availability, Reliability and Security (AReS)*, pages 356–365. IEEE, 2007.

[6] D. Hatebur, M. Heisel, and H. Schmidt. A security engineering process based on patterns. In *Proceedings of the International Workshop on Secure Systems Methodologies using Patterns (SPatterns)*, pages 734–738. IEEE, 2007.

[7] D. Hatebur, M. Heisel, and H. Schmidt. Analysis and component-based realization of security requirements. In *Proceedings of the International Conference on Availability, Reliability and Security (AReS)*, IEEE Transactions, pages 195–203. IEEE, 2008.

[8] D. Hatebur, M. Heisel, and H. Schmidt. A formal metamodel for problem frames. In *Proceedings of the International Conference on Model Driven Engineering Languages and Systems (MODELS)*, volume 5301, pages 68–82. Springer Berlin / Heidelberg, 2008.

[9] C. A. R. Hoare. *Communicating Sequential Processes*. Prentice Hall, 1986.

[10] D. Jackson. Micromodels of software: Lightweight modelling and analysis with Alloy. http://softwareabstractions.org/.

[11] M. Jackson. *Problem Frames. Analyzing and structuring software development problems*. Addison-Wesley, 2001.

[12] L. Lai, L. Lai, and J. W. Sanders. A refinement calculus for communicating processes with state. In *1st Irish Workshop on Formal Methods: Proceedings, Electronic Workshops in Computing*. Springer, 1997.

[13] Z. Li, J. G. Hall, and L. Rapanotti. From requirements to specifications: a formal approach. In *Proceedings of the International Workshop on Advances and Applications of Problem Frames (IWAAPF 2006)*, pages 65–70. ACM, 2006.

[14] F. S. E. Limited. Failures-divergence refinement (FDR2), 2008.

[15] H. Mantel. *A Uniform Framework for the Formal Specification and Verification of Information Flow Security*. PhD thesis, Universität des Saarlandes, Saarbrücken, Germany, Juli 2003.

[16] H. Mouratidis and P. Giorgini. Secure Tropos: A security-oriented extension of the Tropos methodology. *International Journal of Software Engineering and Knowledge Engineering*, 17(2):285 – 309, 2007.

[17] M. Nelson, T. Nelson, P. Alencar, and D. Cowan. Exploring problem-frame concerns using formal analysis. In *Proceedings of the International Workshop on Advances and Applications of Problem Frames (IWAAPF 2004)*, pages 61–68. IET, 2004.

[18] T. Santen. Preservation of probabilistic information flow under refinement. *Information and Computation*, 206(2-4):213–249, April 2008.

[19] A. van Lamsweerde. Elaborating security requirements by construction of intentional anti-models. In *Proceedings of the 26th International Conference on Software Engineering (ICSE)*, pages 148–157. IEEE Computer Society, 2004.