Technical Report: Security in Business Process Engineering[*]

Michael Backes, Birgit Pfitzmann, Michael Waidner

IBM Zurich Research Laboratory, Rüschlikon, Switzerland
{mbc,bpf,wmi}@zurich.ibm.com

Abstract. We present a general methodology for integrating arbitrary security requirements in the development of business processes in a both elegant and rigorous way. We show how trust relationships between different parties and their respective security goals can be reflected in a specification, which results in a realistic modeling of business processes in the presence of malicious adversaries. Special attention is given to the incorporation of cryptography in the development process with the main goal of achieving specifications that are sufficiently simple to be suited for formal verification, yet allow for a provably secure cryptographic implementation.

Keywords: Security in Business Process Modeling, Design, Verification and Validation

1 Introduction

For typical distributed business processes, especially those that run over the Internet, from supply chains over business federations to virtual enterprises, security plays a crucial role. For instance, analysts often see lack of security as a major impediment to the adoption of web services. However, the notion of security is often neglected in business-process models, which usually concentrate on modeling the process in a way that functional correctness can be shown, either manually or using formal proof tools like model checking. In contrast to features that are crucial for functional correctness, security features are typically integrated in an application in an ad-hoc manner, often during the actual implementation process. However, this approach brings about several problems.

First, the integration of security features into a development process is not well understood. In particular, a crucial ingredient for adequately modeling security issues – the incorporation of trust assumptions into a specification – is usually not considered, and hence needs further investigation.

Secondly, integrating security properties by hand is difficult and error-prone, and thus lack of experience of individual developers often leads to security leaks. As these developers usually do not have a strong background in security they need to be provided with concrete guidelines and suitable tools for the development of secure applications.

[*] An earlier version of this paper appeared in [16].

Thirdly, achieving security goals often relies on an appropriate use of cryptographic protocols, e.g., for achieving secrecy for a particular message, the underlying key exchange, or larger cryptographic protocols like payment systems or fair exchange. However, incorporating cryptography already in the specification is surely not desired since this would significantly complicate the use of formal methods in case a validation of security properties is desired. More precisely, a suitable tool would have to cope with probabilism, computational restrictions, error probabilities, and other essential details for reasoning about real cryptographic primitives in a meaningful way. No such tool exists yet. Hence abstractions for such protocols should be provided that are as simple as possible on the one hand, yet extensive enough to allow for being securely implemented on the other hand. One of the main problems in all prior work is the use of oversimplified abstractions, which are insufficient in the sense that they cannot be securely implemented, even if provably secure cryptographic primitives are used. Today, a large variety of applications presupposes the use of cryptographic protocols (but without actually specifying them), which exemplifies the demand for a common formal model that can deal with these issues on the one hand, but can still be conveniently used for expressing and analyzing a large variety of applications.

1.1 Our Contribution

This work presents a guideline how to integrate *arbitrary* security requirements in the specification of business process modeling in a both elegant and rigorous way. Before an application is specified, the developer should be aware of the security goals he wants to achieve. This is shown on the right-hand side of Figure 1. The security properties considered in this work are especially not restricted to commonly analyzed, message-specific properties like secrecy or authenticity of specific message, but comprise sophisticated properties like probabilistic non-interference (absence of probabilistic flow of information), fair exchange, or privacy guarantees based on privacy policies. Although these properties are essential for lots of business processes, their incorporation in the design process has not been well understood yet. Furthermore, the trust relationship between different parties has to be reflected in the design. We investigate the notion of trust models for this purpose, and we show how they can be used to allow for a convenient incorporation of trust into the development process. As shown in Figure 1, after reflecting these features in the specification, common approaches on step-wise refinement can be applied.

Our further work is based on a probabilistic model of reactive networks for expressing and analyzing cryptographic protocols from [41, 20, 19]. This model, commonly known as the BPW model by now, has ben widely used to the computationally accurate analysis of security protocols. In particular, it has been used to give the first full cryptographic justification for a Dolev-Yao model, i.e., for arbitrary active attacks and within arbitrary surrounding interactive protocols, was first given in [15, 18, 17, 21], which in turn allowed to analyse several well-known security protocols in a computationally sound manner [10, 2, 5, 13, 8, 4]. Especially interesting for our work is that the notion of simulatability that underlies the BPW model comes with several property preservation theorems: integrity, fairness, liveness, and non-interference [6, 14, 9, 11, 12, 3]. By exploiting the benefits of this approach, we show how business processes

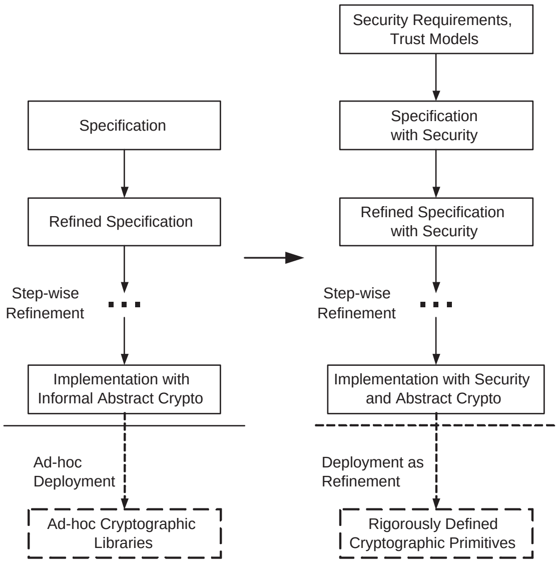
Security Requirements, Trust Models
Specification
Specification with Security
Refined Specification
Refined Specification with Security
Step-wise Refinement
Step-wise Refinement
Implementation with Informal Abstract Crypto
Implementation with Security and Abstract Crypto
Ad-hoc Deployment
Deployment as Refinement
Ad-hoc Cryptographic Libraries
Rigorously Defined Cryptographic Primitives

Fig. 1. A usual development approach based on step-wise refinement is shown on the left side. The right side shows the main extensions of our approach, i.e., including security requirements and trust models into specifications, and appropriate refinement of cryptographic abstractions.

including security issues can be specified in a way that is suited for formal verification as the use of probabilism that arises from the underlying cryptography can be avoided on this layer without destroying the link to the (necessarily probabilistic) cryptographic implementation. This means that the crypto-related parts of a specification can be refined automatically without any further assistance of the developer yielding a concrete implementation of the system (provided that the parts that are not related to security are sufficiently specified) such that this refinement preserves all security properties of the specification. This is also shown on the right-hand side of Figure 1. Here, a concrete implementation with still abstract representations of cryptography can be easily and securely refined, since the model offers an appropriate interface for a cryptographic library, and the abstract specification provided by the model ensures a correct deployment. As the final result of our whole approach, we obtain a concrete proposal for a secure implementation of the specification including an appropriate deployment of actual cryptography.

This stands in blatant contrast to prior approaches that make implicit or explicit use of real cryptography, in which the development of an application stops to be elaborate if it comes to the implementation of real crypto.

1.2 Prior Work

The early days of business-process modeling were stamped by the use of pictures that described the interaction between different components of the system. This is still very common at a high level of abstraction, but nowadays merging different views on the process within suitable specification languages, usually workflow management systems or UML. However, if formal validation is desired, there is a need to equip such languages with a non-ambiguous semantics before, e.g., model checking techniques can be applied. Hence, research on business process modeling has recently started to encode business-process diagrams into a formal model that can be given a suitable semantics, usually based on interacting state machines [32, 26], petri nets [30], or graph grammars [31]. Based on the language ConGolog, a more action-oriented representation of business processes is presented in [33]. However, these works did not explicitly take security aspects into account.

Besides the mentioned models, well-known standards for business modeling like ebXML and WSFL emerged that included security issues by postulating the use of secure and reliable message transmission, but they do not define this notion beyond the informal description, nor do they address more complex security goals. For a nice overview, see [37].

Up to now, models that explicitly address the incorporation of security issues in the design process are typically extensions of a fragment of UML that can be given the desired semantics. They address more general notions of security than in the standards like multi-level security [29] or role-based access control [34]. Although these are already significant results in this area, several important features like probabilistic flow of information and faithful representation of cryptography as described above have not been addressed yet. However, as some of the above work is at least implicitly based on cryptographic techniques like ensuring authenticity in access control or secrecy for transmitting confidential information in multi-level security, abstraction versions of cryptographic primitives like secure channels are included there, following the approach of Dolev and Yao [25]. Using these abstractions allows for fairly simple proofs of security properties. From the view of cryptography however, these abstractions are unfaithful in the sense that they cannot be securely implemented even if provably secure cryptographic primitives are used. A concrete counterexample is given in [40]. Being more precise, there is no guarantee that properties of these abstractions are also valid for the concrete implementation.

2 Security Requirements and Trust Models

This section investigates the incorporation of security requirements and trust models into the specification of a business process. We further introduce a certifies mail protocol as a running example, which we will consecutively develop as the paper proceeds.

To incorporate security in the development of a business process, several aspects naturally have to be taken into account. First the addressed security properties have to be identified. Up to now, this task usually has been restricted to relatively simple security properties like secrecy of a specific message, but typical processes that are based on more sophisticated protocols often presuppose achieving more general security properties like fair exchange or absence of information flow. Secondly, the relationship of trust between the different parties has to be investigated and incorporated in the design, i.e., which users trust which other users for which purpose. Further comprised by this point is that a developer might be interested in the number of malicious participants that the considered application can successfully tolerate. Dealing with questions of this kind leads to the notion of *trust models*, which are common in the security community, but have not been properly addressed for business process modeling so far.

2.1 Security Goals

Before starting to develop a secure application, the desired security goals have to be determined. Typically, the addressed security goals are restricted to either single-sided security, e.g., by establishing security using a trusted computing base that can be evaluated according to Common Criteria, or to channel- or message-specific properties like secrecy of a particular channel, which can (hopefully) be achieved in a single step by a suitable deployment of encryption schemes. However, this is not sufficient for living up to the multilateral security demands of multiple sides that are running a complex protocol. A typical example in business modeling are fair exchange systems, where different "basic" security goals collide from different sides and have to be achieved at once, e.g., a buyer wants to receive the ordered goods if he properly pays for them whereas a seller wants to receive his money if he delivers the goods. Dealing with such general goals for multiple parties is much more difficult for the developer since it is not sufficient to, e.g., simply classify a channel as being secure, but the security property may depend on the continuous interaction of several parties, or may even change over time. Hence, incorporating such properties in formal specification gives rise to several questions, which are often circumvented by including the property in prose without defining it beyond the reach of an informal description. However, this mainly passes on the problem to the implementor who now has to derive a secure implementation without sufficient guidelines by the specification. Hence a formal model for business processes should allow for expressing *arbitrary* security goals, and it should further provide concrete guidelines how a secure implementation can be achieved from a secure specification.

2.2 Trust Models

Typically, a trust model refers to one specific property only, i.e., one system might use different trust models for different properties. For example, a user might trust another user of his own company to proof-read some unpublished work without using it for its own advantage, but not to use his company credit card. Many companies and governments even officially do not have a centralized trust model, i.e., different departments are often not allowed to share all their knowledge and delegate all their tasks to each other.

For a meaningful incorporation of trust in the development process, a trust model has to meet the demands of two different points of view.

First, it has to specify which parties trust each other for which purpose, respectively which parties are considered as being potentially dishonest. This represents the individual demands and relationships between the different users. Obviously, these trust relationships heavily depend on the addressed security requirements, so it should be avoided to lump together all kinds of security requirements, but each requirement may need an individual treatment. This is an important feature in our underlying model, where different requirements can be expressed as integrity, privacy, and liveness requirements. For dealing with security issues in the presence of malicious adversaries, this view usually captures the number of dishonest parties that a system can successfully tolerate without losing its promised security properties. The common way to capture this formally is to use *access structures*. An access structure $\mathcal{ACC}$ is a subset of the powerset of all participants, representing the honest participants or correct hardware components; for the set $\mathcal{M} = \{1, \ldots, n\}$ of participants, e.g., we have $\mathcal{ACC} \subseteq 2^{\mathcal{M}}$. $\mathcal{ACC}$ has to be closed under element insertion, i.e., with $A \in \mathcal{ACC}$ and $A \subseteq B$, we have $B \in \mathcal{ACC}$. For cryptographic protocols, typical examples are threshold structures, which state that at least t parties have to be honest to guarantee the desired property, i.e., $\mathcal{ACC} = \{A \in 2^{\mathcal{M}} \mid |A| \geq t\}$. However, such threshold structures do not always adequately model trust for business processes, since the amount of tolerable malicious parties does not only rely on the actual number of such participants, but it might additionally reflect the users' privileges, rights, and influence, and hence their respective power to attack the system. Obviously, it will usually be much more difficult to keep data secret from a corrupted CEO with all her privileges than from a usual employee. Similarly, corrupting a pivotal server storing sensitive data like secret keys for lots of employees will surely be more attractive than corrupting a stand-alone workstation of a single employee.

Secondly, it has to be specified which information may leak from a communication and how the adversary (respectively the dishonest parties) can interfere in the communication process, e.g., by modifying messages in transit. This corresponds to an external view of trust, which is captured using a *channel model*. Formally, a channel model is a function χ mapping each connection of a system model to an element of the set $\{\mathsf{s}, \mathsf{r}, \mathsf{i}\}$, representing (s)ecure, (r)eliable, or (i)nsecure channels. The set can be extended to other connection types. Both access structures and channel models will be further treated when we introduce our formal model.

2.3 A Certified Mail System as Running Example

In order to illustrate the need for the previously mentioned security goals, we continue with a simple example system: a certified mail system, which models a fair exchange of a message against a receipt, i.e., the message is delivered if and only if a valid receipt is issued. If this condition is violated, then the cheated person should be able to successfully complain at a trusted third party. Certified mail is an important primitive for electronic-commerce processes and other atomicity services.

Here and in the following, we let $\mathcal{M} := \{1, \ldots, n\}$ denote the set of the users of the certified mail system, and these users can communicate with each other over an insecure network like the Internet. In our certified mail system, this set has to comprise one distinguished trust third party v that has to be honest, i.e., it is trusted by both the sender and the recipient.

In order to implement such a system, each participant at first has to be able to check the authenticity of an incoming message. This obviously cannot be achieved that easily in the presence of malicious adversaries and an insecure network, since it allows for modifying a message and faking its origin without granting the recipient a possibility to detect this tampering. For successfully complaining at the third party in case of cheating, a party has to be forced to issue a commitment before it will actually receive the desired goods. The cheated user can use this commitment to convince the verifier in case of a cheating contractual partner. Cryptographic protocols are well-suited for these tasks.

3 General Model Description

This section contains a brief review of the asynchronous model of probabilistic reactive systems from [41], on which our subsequent work is based. The model itself is very rigorously defined, but we use an informal description here due to lack of space. The model is based on interacting state-machines as a common approach in business process modeling; actually, the whole model has many similarities with other commonly used models. The main difference is the incorporation of probabilism for dealing with the concrete versions of cryptography in a meaningful way.

The machine model is probabilistic state-transition machines, similar to probabilistic I/O automata as sketched by Lynch [36]. In prior formal models for business process modeling, the usage of probabilism is avoided by assuming either deterministic or non-deterministic machines. Although this is reasonable if the model does not intend to address security properties, which are either explicitly or – more often – implicitly based on cryptographic techniques, it does not allow for a meaningful analysis in the presence of cryptography.

Communication between different machines is done via *ports*. Inspired by the CSP-Notation [27], we write output and input ports as $\mathsf{p}!$ and $\mathsf{p}?$ respectively. *Connections* are defined implicitly by naming convention, that is port $\mathsf{p}!$ sends messages to $\mathsf{p}?$. To achieve asynchronous timing, a message is not directly sent to its recipient, but it is first stored in a special machine $\widetilde{\mathsf{p}}$ called a *buffer* and waits to be scheduled.

If a machine wants to schedule the i-th message of buffer $\widetilde{\mathsf{p}}$ (this machine must have the unique clock out-port $\mathsf{p}^{\triangleleft}!$) it simply sends i at $\mathsf{p}^{\triangleleft}!$. The buffer then schedules the i-th message and removes it from its internal list. In our case, most buffers are scheduled by the adversary, i.e., he has the clock out-port. The concept of buffers is essentially the same as in other state-based business-process models, which simply postulate the network to be asynchronous without mentioning where messages in transit are stored, but reasoning about cryptography in a meaningful way presupposes a higher level of rigorousness. Moreover, note that these buffers do not have to be explicitly taken care of when modeling a system. They are implicitly added in the model to allow for an asynchronous definition of time and a meaningful analysis of security properties.

If a machine is switched, it receives an input tuple at its input ports and performs its transition function yielding a new state and an output tuple in the deterministic case, or a finite distribution over the set of states and possible outputs in the probabilistic case. At each switching step of one particular machine, at most one value can arrive at every input port and the machine can at most produce one output per port.

A *collection* $\mathcal{C}$ of machines is a finite set of machines with pairwise different machine names and disjoint sets of ports. A port of a collection is called *free* if its connecting port is not in the collection. These port will be connected to the users and the adversary. A collection $\mathcal{C}$ is called *closed* if it has no free ports except a special master-clock in-port $\mathsf{clk}^{\triangleleft}?$. This port will be used to resolve situation where the execution cannot proceed.

For a closed collection, runs (sometimes called *traces* or *executions*) are defined as follows. Scheduling of machines is done sequentially, so we have exactly one active machine M at any time. If this machine has cloc out-ports, it is allowed to select the next message to be scheduled as explained above. If that message exists, it is delivered by the buffer and the unique receiving machine is the next active machine. If M tries to schedule multiple messages, only one is taken, and if it schedules none or the message does not exist, the special master scheduler is scheduled using the master-clock in-port $\mathsf{clk}^{\triangleleft}?$. For a formal definition of runs, see [41]. Due to the probabilistic transition functions of the individual machines, we further obtain a probability space over the runs, which will be useful for adequately expressing sophisticated security goals like probabilistic flow of information. We further define the restriction of a run to a set $\hat{M}$ of machines by restricting the run to those steps where a machine $\mathsf{M} \in \hat{M}$ is switched. This is called the *view* of $\hat{M}$.

3.1 Security-specific System Parts based on Trust Models

For security purposes, special collections are needed, because an adversary may have taken over parts of the initially intended system. This is handled by considering a *system* to be a set of possible *structures*, i.e., we have one structure for each element of the trust model. First, the actual system part is defined and then the environment, consisting of *users* and the *adversary*. Each structure furthermore separates the set of free ports into *specified ports* S and others. The specified ports are those where a certain service is guaranteed. Typical examples of inputs at specified ports are "send message m to id" for a message transmission system or "pay amount x to id" for a payment system. In the upcoming security definition, only the events at the specified ports have to be taken care of. This allows *abstract specifications* with *tolerable imperfections*, which can be explicitly granted to the adversary at the remaining ports of the system. A structure is completed to a *configuration* by adding machines H and A, modeling the joint honest users and the adversary, respectively. The machine H is restricted to the specified ports S, A connects to the remaining free ports of the structure and both machines can interact, e.g., in order to model active attacks.

Configurations are always closed, i.e., no inputs or outputs are related to some external participant. This is suitable since users and the adversary are explicitly contained in the configuration, so there is no need to model them externally as often done in other formal models. We will see that including the users in the model allows for using the concept of *simulatability* that, roughly speaking, reflects the notion of a cryptographically secure implementation.

3.2 Standard Cryptographic Systems

In a *standard cryptographic system*, the structures are derived from one *intended structure* and a *trust model*. The intended structure typically consists of n machines M_u, one for each possible user. This is shown on the left-hand side of Figure 2. The trust model consists of the already mentioned access structure $\mathcal{ACC}$ and channel model χ, cf. Section 2.2. Here, the access structure contains the possible sets of uncorrupted machines (among the intended ones), and the channel model classifies each channel as secure, reliable (authentic but not private) or insecure. Now, we have one structure for each element $\mathcal{H}$ of $\mathcal{ACC}$, i.e.,

$$Sys = \{(\hat{M}_{\mathcal{H}}, S_{\mathcal{H}}) \mid \mathcal{H} \in \mathcal{ACC}\}$$

in which only the machines contained in $\mathcal{H}$ are considered, i.e., we have $\hat{M}_{\mathcal{H}} = \{\mathsf{M}_u \mid u \in \mathcal{H}\}$. The remaining machines are absorbed into the adversary. Now we modify the channels between two correct machines according to the channel model in the following way. If a connection c is considered as secure ($\chi(c) = \mathsf{s}$), it remains unchanged, i.e., both parties are directly connected, and the adversary does not learn anything about communications on that channel. If a connection is reliable ($\chi(c) = \mathsf{r}$), every output is additionally sent to the adversary corresponding to listening on a channel without the ability of changing the content. This corresponds to an authenticated channel. Finally, messages sent on insecure connections $(\chi(c) = \mathsf{i})$ have to pass through the adversary, i.e., he can read and modify them in transit. A configuration of such a structure, after taking the trust model into account, is shown on the right-hand side of Figure 2. After these modifications, each derivation represents a realistic scenario for the particular trust model, where information may leak or authenticity may not be guaranteed because of an insecure network like the Internet, or where exchange of public keys can be precisely modeled using authenticated channels.

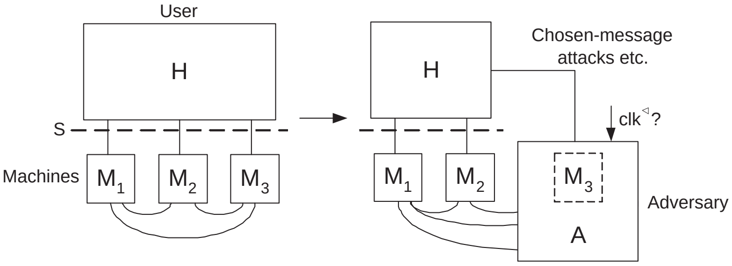

Fig. 2. A configuration of an intended structure is shown on the left-hand side (for $n = 3$). The right-hand side shows a derivation induced by the trust model. In this case, we have $\mathcal{H} = \{1, 2\}$, so the third machine is absorbed into the adversary, and the channel model classifies the connection between the correct machine as authenticated.

3.3 Simulatability

Simulatability is an important cryptographic concept, which allows for securely refining the cryptographic parts of a specification. Simulatability essentially means that whatever might happen to an honest user H in a real system Sys_{real} can also happen to the same honest user in an ideal System Sys_{id}. Formally speaking, for every configuration $conf_1$ of Sys_{real} there is a configuration $conf_2$ of Sys_{id} with the same users yielding *indistinguishable views* of H in both systems. We abbreviate this by $Sys_{\mathsf{real}} \geq_{\mathsf{sec}} Sys_{\mathsf{id}}$. Indistinguishability is a well-defined cryptographic notion from [44].

Several nice results on simulatability exist. At first, there exists a composition theorem stating that a step-wise refinement maintains the simulatability property [41].

Moreover, integrity and privacy properties (more precisely, integrity properties formulated in a linear-time logic and non-interference properties) are preserved under simulatability[9, 6], which allows for proving properties on a high-level abstract layer that can be refined later down to the cryptographic layer without destroying the already proven properties. These theorems are essential for modular proofs.

Hence, the analysis of security properties only has to be done for the abstract specification, and the results magically carry over to the actual cryptographic implementation. In the upcoming section, this approach is used to define abstractions of commonly used cryptographic primitives, which can be implemented in the sense of simulatability. The benefit is that these abstractions can be conveniently used within a specification, and serve as a construction kit for designing large processes out of these small building blocks. Refining the crypto-related parts can then be easily, but securely achieved without any additional work. Hence designers or process analysts are not forced to have a strong background in security.

4 Towards a Secure Implementation: Refinement using Common Cryptographic Primitives

In this section, we investigate the notion of refinement of a process within our model, distinguishing between refinement on the component level, on the action level, or on the trust level. As the refinement of abstract cryptographic specifications by real cryptographic protocols is one of the main benefits of our model, we continue with a brief review on abstract and concrete versions of the most important cryptographic primitives comprising secure channels, certified mail, and a simulatable cryptographic library.

4.1 Security in Different Layers of Refinement

Refinement is an important aspect of a design process, as shown in Figure 1. The part where our model is most beneficial is that it enables sound refinement of abstract specifications of cryptographic protocols by real cryptographic protocols. However, the inclusion of trust and security requirements may influence all layers.

Another important argument for having a joint model of business processes and cryptographic protocols is that typical business-process modeling may occur again *under* a layer with cryptographic protocols.

All this is illustrated in Figure 3. Part 1 of this figure shows an abstract cryptographic primitive, here fair exchange as sketched in Section 2.3. In an overall business-modeling process, this may be one element after previous refinements.

Part 2 of the figure shows refinement by a real cryptographic protocol. This is primarily component refinement because the behavior of the abstract primitive is modeled by one state machine, while the real protocol shows the real three parties of such a protocol. Further, it contains action refinement: On the abstract layer, the fair exchange is essentially one step (exchange or not, depending on certain preconditions; only asynchrony and giving the adversary a choice to disrupt the protocol breaks this step up a bit). The real protocol contains several real state transitions corresponding to an input of a protocol message, an internal action of one party, and output of the next protocol message.

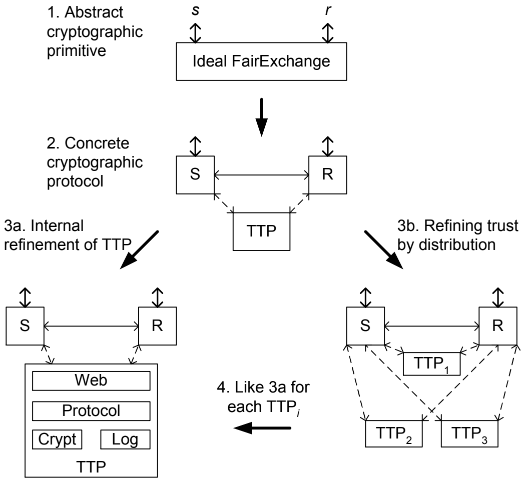
1. Abstract cryptographic primitive
s
r
Ideal FairExchange
2. Concrete cryptographic protocol
S
R
TTP
3a. Internal refinement of TTP
3b. Refining trust by distribution
S
R
Web
Protocol
Crypt
Log
TTP
4. Like 3a for each TTP$_i$
S
R
TTP$_1$
TTP$_2$
TTP$_3$

Fig. 3. Different types of refinement.

We show two possible next refinement steps: Part 3a of the figure shows business-process modeling of the party TTP. While in the cryptographic protocol its action is only a few monolithic state transitions, for a real such party quite complicated internal workflows must be designed. The figure just shows a web-service frontend, a protocol engine, a crypto engine and a logging component that interact to implement the state transitions. Further, one needs management components for these functional components and approval workflows for use of the management components. This may be treated as a standard business-process modeling and implementation issue without further specific security considerations, because the trust model in Layer 2 was that users s and r have to trust TTP. However, it is better to refine the trust model on this layer, so that nobody needs to trust all components of TTP. For instance, web-service frontends are less trusted than crypto engines. "Refinement" for trust means that no new trust may be introduced. More precisely, the system with the trust model inherited from the higher layer must fulfill the higher-layer properties, but the more we can restrict the necessary trust in sub-components the better.

Part 3b shows another way of refining the trust: Requiring s and r to trust TTP, even with respect to the organization as such and not its components, is quite a strong restriction. We may want to distribute this power. The figure shows refinement of TTP by a cryptographic protocol for secure state-machine distribution with 2-out-of-3 trust, i.e., as long as two of the three organizations TTP$_1$, TTP$_2$, and TTP$_3$ are trustworthy, the overall protocol is as secure as that on the higher layer. This goes along with a refinement of the messaging interface of components S and R to interact with the distributed TTP. (See [23] for real protocols achieving this.) After this step, for each individual TTP$_i$, business-process modeling as in Step 3a may be applied.

4.2 Secure Channels

In the following, we review the fundamental primitive of secure channels, which will probably serve as the foundation of upcoming modeling examples due to its simplicity but generality. We start with the concrete implementation, and continue with a deterministic, but faithful abstraction, i.e., properties proved for this specification carry over to the concrete implementation. It was a longstanding open question whether such "benign" abstractions exist, which was answered in the affirmative for some of the most important cryptographic primitives so far. The solution to the problem was to augment naive abstractions with tolerable imperfections to obtain idealized systems for which practical protocols exist.

Concrete Implementation. Our real system is a standard cryptographic system of the form where any subset of participants may be dishonest. It uses asymmetric encryption and digital signatures as cryptographic primitives. A user u can let his machine create signature and encryption keys that are sent to other users over authenticated channels. Messages sent from user u to user v are signed and encrypted by M_u and sent to M_v over an insecure channel, representing a real network. The adversary can schedule the communication between correct machines and send arbitrary messages to arbitrary users (but only using the identity of the dishonest users).

Abstract Specification. The abstract specification is a system of the form $Sys_{\mathsf{ideal}}^{\mathsf{SecMess}} = (\mathsf{TH}_{\mathcal{H}}, S_{\mathcal{H}}) \mid \mathcal{H} \in \mathcal{ACC}\}$ where $\mathsf{TH}_{\mathcal{H}}$ is a deterministic machine modeling an idealized behavior of the real system. A user can again let $\mathsf{TH}_{\mathcal{H}}$ create signature and encryption keys, or send messages to arbitrary users. In contrast to the real system, no keys are actually generated, but $\mathsf{TH}_{\mathcal{H}}$ only stores that a key generation of the particular user has happened and informs the adversary of this fact. If a message should be sent from user u to user v then this message is not sent over the network, but simply stored in an internal array of $\mathsf{TH}_{\mathcal{H}}$. Now $\mathsf{TH}_{\mathcal{H}}$ tells the adversary that a message has been sent from u to v, and gives it a handle to the particular position in the array that contains the message, along with the actual length of the plaintext message. The adversary can use this information to schedule the message, which is then delivered by the trusted host.

Intuitively, the machine $\mathsf{TH}_{\mathcal{H}}$ models the key essence of sending of encrypted messages and authenticated key exchange. The adversary does neither learn anything on the content of a sent message nor does he has the possibility to alter it, but he is explicitly granted certain information like the length of the plaintext, and that a message has indeed been sent. These are the already mentioned tolerable imperfections, which stem from the fact that a cryptographic implementation cannot avoid the adversary from learning this particular information on his own, at least not with reasonable loss of complexity. Hence, modeling these imperfections also within the specification is the key element for achieving secure implementations.

Some Variants. For secure channels, several variants have already been proven with a simulatability definition like ordered channels [7], which guarantee maintaining the order of sent messages, or reliable channels [14], which guarantee eventual delivery of a message. One of the most important extension is the cryptographic firewall [9] which uses digital signatures to establish a virtual private network, i.e., it builds up a firewall around users which are only connected via an insecure network like the Internet. The firewall is mainly achieved be augmenting the secure channel primitive with a filtering system that sorts out message, which are sent by senders outside of the firewall. The secure channels already provide authenticity of messages and their origins, hence the filtering procedure precisely sorts out the desired message. However, certain complications have to be taken into account like avoiding denial-of-service attacks etc.

4.3 A Primitive for Certified Mail

Now we review the primitive for our running example of certified mail.

Concrete Implementation. The real system is again a standard cryptographic system; however, now a specific party TTP must be honest, i.e., all sets in $\mathcal{ACC}$ contain it.

The system starts with a key-exchange phase. Later, for each exchange, the sender s and recipient r run a subprotocol. As an example, we sketch the efficient asynchronous protocol from [1].

It consists of a four-message standard flow, and two subprotocols where one of the original parties complains to TTP, and TTP fairly aborts or finalizes this exchange. In Message 1, the sender gives a commitment to the mail. A commitment is a lower cryptographic primitive that fixes a message, but, like encryption, does not allow the message to be read yet. In Message 2, the recipient signs that he is willing to provide a receipt if this commitment is opened to him. In Message 3, the sender opens the commitment, i.e., sends the message and a verification value, and in Message 4, the recipient sends a receipt.

If the recipient does not send the receipt (or due to the asynchronous network or an outside adversary it is delayed for too long), the sender shows Messages 1 to 3 to TTP. If the recipient has sent the promise (Message 2) and does not receive the opened commitment, he also complains to TTP with Messages 1 and 2. If a sender-complaint arrives first, TTP provides a replacement receipt. If a recipient-complaint arrives first, TTP signs for the recipient that the exchange was aborted.

Abstract Specification. The abstract specification again essentially consists of one deterministic trusted host $\mathsf{TH}_{\mathcal{H}}^{\mathsf{CertMail}}$ in each structure. The main inputs are of the form (send, r, m) for sending a mail m to a recipient r, and $(\mathsf{receive}, s)$ for a recipient to indicate willingness to get a mail from s and provide a receipt. Essentially, the trusted host looks whether there are two matching inputs; if yes, it outputs the message m to r and a success indicator to s. A detailed specification is already subtle and interesting for the higher processes. First, we need transaction identifiers as additional parameters to define which inputs match. Secondly, we have to choose between the simple inputs just defined and a so-called labeled variant where the recipient also agrees to a subject for the mail to be received, i.e., the above inputs have yet additional parameters. This may be of more use in many applications. Thirdly, there are inputs to show and verify a receipt, because that is the purpose of receipts in a higher protocol. Finally, we have to model the power of an adversary in realistic protocols like the one above: He may learn of exchanges even between honest participants, but not the message and subject involved, and he can force protocols to end unsuccessfully.

4.4 A Foundation for Sophisticated Cryptographic Protocols: A Cryptographic Library

When using more sophisticated protocols, we cannot only rely on simple secure channels, since several messages might be encrypted twofold, nonces might have to be included etc. Moreover, plugging in cryptographic primitives in larger protocols in a naive way may give rise to man-in-the-middle attacks, type confusion attacks, and so on. Hence, we need a more sophisticated system that takes care of all these subtleties, mainly by following the rules of robust protocol design. Moreover, it should allow for composing messages, for including important design principles like nonces etc.

Recently, a simulatable cryptographic library has been introduced in [15, 18], which provides these important design principles, and serves as a powerful tool for integrating security in the analysis (and the design) of business processes. We omit a more detailed description due to lack of space.

4.5 Modeling the Certified Mail System

In the following, we express the certified mail system of Section 2.3 in our model. The system offers each user one port for sending and receiving messages from the system, respectively. Using the conventions of [41], these ports are named $\mathsf{out}_u!$ and $\mathsf{in}_u?$ for user u. (The corresponding ports $\mathsf{out}_u?$ and $\mathsf{in}_u!$ are then ports of the system.) These ports are also the specified ports of the system. Using the abstract primitive $\mathsf{TH}^{\mathsf{CertMail}}$ for certified mail from Section 4.3, we can now easily derive an abstract version of our system that has the desired functionality. This is shown at the left-hand side of Figure 4. The derivation of a concrete implementation is also very simple. We just have to

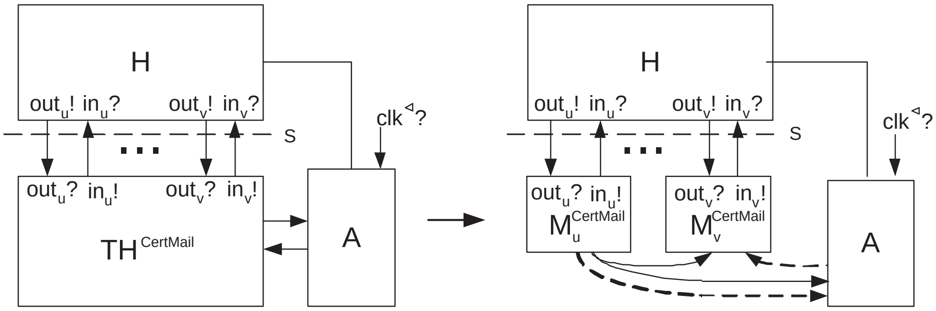

Fig. 4. Refinement of the Certified Mail Specification. The abstract primitive is replaced by its concrete cryptographic counterpart.

replace the abstract system $\mathsf{TH}^{\mathsf{CertMail}}$ with its predefined secure implementation, consisting of the actual protocol machines $\mathsf{M}_u^{\mathsf{CertMail}}$. Since this implementation has been derived according to the simulatability paradigm (cf. Section 3.3), the desired integrity property of certified mail carries over to the implementation without any further work, i.e., this implementation is provably secure if the specification can be proven to be secure. Hence, the final step is the actual validation of the specification, preferably using formal methods. The feasibility of formal methods for this task is the topic of the next section.

5 On the Applicability of Formal Methods

The formal verification of security issues for arbitrary (non business-related) processes has been subject to lots of papers in the literature (a very partial list includes [42, 35, 39]). Just as we did in the previous section, the use of actual cryptography was avoided by considering abstractions. However, the used abstractions are oversimplified in the sense that no secure implementation of them is known. Thus the above concepts for formal verification cannot be easily transfered to the demands of business process modeling, which was one of the main reasons why we decided to base our work on a more cryptographic approach. The disadvantage is that applying formal methods to the abstractions of the previous section is more time-consuming, and also not investigated that well.

As already mentioned in the previous section, all our abstractions are at least deterministic hence we do not have to consider tools for dealing with probabilism like probabilistic model checking. Moreover, no such tool exists for dealing with the cryptographic details like error probabilities. Because of avoiding these problems, our abstractions are in scope of formal proof systems, at least of formal theorem provers, for which significant results on proving medium-size examples of this kind already exist, e.g., in [7, 6] using the theorem prover PVS [38]. All of our abstractions presented above are surely in scope of theorem proving, and further work in this area may additionally benefit from specialized tools such as SAL [22] – an extension of PVS – that provides an environment for the analysis of systems specified as state machines.

However, the use of automated model checking instead of theorem proving would surely be very promising as it has become a very popular method to verify the properties of finite-state concurrent systems [24]. It is also the more common approach in the business process community, mainly because it has been successfully used in other models to verify medium-sized examples [28, 32]. Since interacting finite-state machines are the foundation of these techniques, they are as well applicable to our underlying model. More precisely, if we avoid using our cryptographic abstractions from the previous section, and concentrate on modeling common examples without security just as prior work does, using our model for this task does not impose any disadvantages.

When using our abstractions, the main problem could be that they are simply too complex for being model-checked, which hence needs further investigation. However, the basic primitives like secure channels or the cryptographic firewall are surely feasible for current model checkers. In contrast to that, the more sophisticated abstractions like certified mail or even the cryptographic library have very complex transition functions that are additionally based on complex and unbounded datastructure like a database with different types of entries. An ad-hoc application of a model checker will fail almost surely, but we are confident that the complexity can be further reduced by developing or adapting commonly data-independence techniques, e.g., [43].

References

1. N. Asokan, V. Shoup, and M. Waidner. Asynchronous protocols for optimistic fair exchange. In *Proc. 19th IEEE Symposium on Security & Privacy*, pages 86–99, 1998.
2. M. Backes. A cryptographically sound dolev-yao style security proof of the Otway-Rees protocol. In *Proceedings of 9th European Symposium on Research in Computer Security (ESORICS)*, volume 3193 of *Lecture Notes in Computer Science*, pages 89–108. Springer, 2004.

3. M. Backes. Quantifying probabilistic information flow in computational reactive systems. In *Proceedings of 10th European Symposium on Research in Computer Security (ESORICS)*, volume 3679 of *Lecture Notes in Computer Science*, pages 336–354. Springer, 2005.
4. M. Backes, I. Cervesato, A. D. Jaggard, A. Scedrov, and J.-K. Tsay. Cryptographically sound security proofs for basic and public-key kerberos. In *Proceedings of 11th European Symposium on Research in Computer Security(ESORICS)*, volume 4189 of *Lecture Notes in Computer Science*, pages 362–383. Springer, 2006. Preprint on IACR ePrint 2006/219.
5. M. Backes and M. Duermuth. A cryptographically sound Dolev-Yao style security proof of an electronic payment system. In *Proceedings of 18th IEEE Computer Security Foundations Workshop (CSFW)*, pages 78–93, 2005.
6. M. Backes and C. Jacobi. Cryptographically sound and machine-assisted verification of security protocols. In *Proc. 20th Annual Symposium on Theoretical Aspects of Computer Science (STACS)*, volume 2607 of *Lecture Notes in Computer Science*, pages 675–686. Springer, 2003.
7. M. Backes, C. Jacobi, and B. Pfitzmann. Deriving cryptographically sound implementations using composition and formally verified bisimulation. In *Proc. 11th Symposium on Formal Methods Europe (FME 2002)*, volume 2391 of *Lecture Notes in Computer Science*, pages 310–329. Springer, 2002.
8. M. Backes, S. Moedersheim, B. Pfitzmann, and L. Vigano. Symbolic and cryptographic analysis of the secure WS-ReliableMessaging Scenario. In *Proceedings of Foundations of Software Science and Computational Structures (FOSSACS)*, volume 3921 of *Lecture Notes in Computer Science*, pages 428–445. Springer, 2006.
9. M. Backes and B. Pfitzmann. Computational probabilistic non-interference. In *Proceedings of 7th European Symposium on Research in Computer Security (ESORICS)*, volume 2502 of *Lecture Notes in Computer Science*, pages 1–23. Springer, 2002.
10. M. Backes and B. Pfitzmann. A cryptographically sound security proof of the Needham-Schroeder-Lowe public-key protocol. In *Proceedings of 23rd Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS)*, volume 2914 of *Lecture Notes in Computer Science*, pages 1–12. Springer, 2003. Preprint on IACR ePrint 2003/121.
11. M. Backes and B. Pfitzmann. Intransitive non-interference for cryptographic purposes. In *Proceedings of 24th IEEE Symposium on Security and Privacy*, pages 140–152, 2003.
12. M. Backes and B. Pfitzmann. Relating symbolic and cryptographic secrecy. *IEEE Transactions on Dependable and Secure Computing (TDSC)*, 2(2):109–123, 2005.
13. M. Backes and B. Pfitzmann. On the cryptographic key secrecy of the strengthened Yahalom protocol. In *Proceedings of 21st IFIP International Information Security Conference (SEC)*, pages 233–245, 2006.
14. M. Backes, B. Pfitzmann, M. Steiner, and M. Waidner. Polynomial fairness and liveness. In *Proceedings of 15th IEEE Computer Security Foundations Workshop (CSFW)*, pages 160–174, 2002.
15. M. Backes, B. Pfitzmann, and M. Waidner. A composable cryptographic library with nested operations. In *Proceedings of 10th ACM Conference on Computer and Communications Security (CCS)*, pages 220–230, 2003.
16. M. Backes, B. Pfitzmann, and M. Waidner. Security in business process engineering. In *Proceedings of 2003 International Conference on Business Process Management*, volume 2678 of *Lecture Notes in Computer Science*, pages 168–183. Springer, 2003.
17. M. Backes, B. Pfitzmann, and M. Waidner. Symmetric authentication within a simulatable cryptographic library. In *Proceedings of 8th European Symposium on Research in Computer Security (ESORICS)*, volume 2808 of *Lecture Notes in Computer Science*, pages 271–290. Springer, 2003. Preprint on IACR ePrint 2003/145.
18. M. Backes, B. Pfitzmann, and M. Waidner. A universally composable cryptographic library. *IACR Cryptology ePrint Archive*, 2003:15, 2003.

19. M. Backes, B. Pfitzmann, and M. Waidner. A general composition theorem for secure reactive system. In *Proceedings of 1st Theory of Cryptography Conference (TCC)*, volume 2951 of *Lecture Notes in Computer Science*, pages 336–354. Springer, 2004.
20. M. Backes, B. Pfitzmann, and M. Waidner. Secure asynchronous reactive systems. *IACR Cryptology ePrint Archive*, 2004:82, 2004.
21. M. Backes, B. Pfitzmann, and M. Waidner. Symmetric authentication within a simulatable cryptographic library. *International Journal of Information Security (IJIS)*, 4(3):135–154, 2005.
22. S. Bensalem, V. Ganesh, Y. Lakhnech, C. Muñoz, S. Owre, H. Rueß, J. Rushby, V. Rusu, H. Saïdi, N. Shankar, E. Singerman, and A. Tiwari. An overview of SAL. In *LFM 2000: Fifth NASA Langley Formal Methods Workshop*, pages 187–196, 2000.
23. C. Cachin and J. A. Poritz. Secure intrusion-tolerant replication on the Internet. In *Proc. International Conference on Dependable Systems and Networks (DSN)*, pages 167–176, 2002.
24. E. Clark, O. Grumberg, and D. Peled. *Model Checking*. The MIT Press, 1999.
25. D. Dolev and A. C. Yao. On the security of public key protocols. *IEEE Transactions on Information Theory*, 29(2):198–208, 1983.
26. D. Harel and M. Politi. *Modeling Reactive Systems with Statecharts: The Statemate Approach*. McGraw-Hill, 1998.
27. C. A. R. Hoare. *Communicating Sequential Processes*. International Series in Computer Science, Prentice Hall, Hemel Hempstead, 1985.
28. W. Janssen, R. Mateescu, S. Mauw, P. Fennema, and P. van der Stappen. Model checking for managers. In *Proc. Theoretical and Practical Aspects of SPIN Model Checking*, volume 1680 of *Lecture Notes in Computer Science*, pages 92–107. Springer, 1999.
29. J. Jürjens. Towards development of secure systems using UMLsec. In *Proc. Fundamental Approaches for Software Engineering (FASE)*, pages 187–200, 2001.
30. E. Kindler and T. Vesper. A temporal logic for events and states. In *Proc. 19th International Conference on Application and Theory of Petri Nets*, volume 1420 of *Lecture Notes in Computer Science*, pages 365–384. Springer, 1998.
31. C. Klauck and H.-J. Mueller. Formal business process engineering based on grammer graphs. *International Journal on Production Economics*, 50:129–140, 1997.
32. J. Koehler, G. Tirenni, and S. Kumaran. From business process model to consistent implementation: A case for formal verification methods. In *Proc. 6th IEEE International Enterprise Distributed Object Computing Conference (EDOC)*, pages 96–106, 2002.
33. M. Koubarakis and D. Plexousakis. A formal model for business process modelling and design. In *Proc. Conference on Advanced Information System Engineering*, pages 142–156, 2000.
34. T. Lodderstedt, D. Basin, and J. Doser. SecureUML: A UML-based modeling language for model-driven security. In *Proc. 5th International Conference on the Unified Modeling Language*, volume 2460 of *Lecture Notes in Computer Science*, pages 425–441. Springer, 2002.
35. G. Lowe. Breaking and fixing the Needham-Schroeder public-key protocol using FDR. In *Proc. 2nd International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, volume 1055 of *Lecture Notes in Computer Science*, pages 147–166. Springer, 1996.
36. N. Lynch. *Distributed Algorithms*. Morgan Kaufmann Publishers, San Francisco, 1996.
37. D. O'Riordan. Business process standards for web services. available at http://www.webservicesarchitect.com/content/articles/BPSFWSBDO.pdf.
38. S. Owre, N. Shankar, and J. M. Rushby. PVS: A prototype verification system. In *Proc. 11th International Conference on Automated Deduction (CADE)*, volume 607 of *Lecture Notes in Computer Science*, pages 748–752. springer, 1992.

39. L. Paulson. The inductive approach to verifying cryptographic protocols. *Journal of Cryptology*, 6(1):85–128, 1998.
40. B. Pfitzmann, M. Schunter, and M. Waidner. Cryptographic security of reactive systems. Presented at the DERA/RHUL Workshop on Secure Architectures and Information Flow, Electronic Notes in Theoretical Computer Science (ENTCS), March 2000. `http://www.elsevier.nl/cas/tree/store/tcs/free/noncas/pc/menu.htm`.
41. B. Pfitzmann and M. Waidner. A model for asynchronous reactive systems and its application to secure message transmission. In *Proc. 22nd IEEE Symposium on Security & Privacy*, pages 184–200, 2001.
42. A. W. Roscoe. Modelling and verifying key-exchange protocols using CSP and FDR. In *Proc. 8th IEEE Computer Security Foundations Workshop (CSFW)*, pages 98–107, 1995.
43. A. W. Roscoe and P. Broadfoot. Proving security protocols with model checkers by data independence techniques. *Journal of Computer Security*, 7(2,3):147–190, 1998.
44. A. C. Yao. Theory and applications of trapdoor functions. In *Proc. 23rd IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 80–91, 1982.