

Cryptographically Sound and Machine-Assisted Verification of Security Protocols

Michael Backes¹ and Christian Jacobi²

¹ IBM Zurich Research Laboratory, Rüschlikon, Switzerland[†]

² IBM Deutschland Entwicklung GmbH, Processor Development 2, Böblingen, Germany[†]

¹ mbc@zurich.ibm.com, ² cjacobi@de.ibm.com

Abstract. We consider machine-aided verification of suitably constructed abstractions of security protocols, such that the verified properties are valid for the concrete implementation of the protocol with respect to cryptographic definitions. In order to link formal methods and cryptography, we show that integrity properties are preserved under step-wise refinement in asynchronous networks with respect to cryptographic definitions, so formal verifications of our abstractions carry over to the concrete counterparts. As an example, we use the theorem prover PVS to formally verify a system for ordered secure message transmission, which yields the first example ever of a formally verified but nevertheless cryptographically sound proof of a security protocol. We believe that a general methodology for verifying cryptographic protocols cryptographically sound can be derived by following the ideas of this example.

Keywords: cryptography, specification, verification, PVS, semantics, simulatability

1 Introduction

Nowadays, formal analysis and verification of security protocols is getting more and more attention in both theory and practice. One main goal of protocol verification is to consider the cryptographic aspects of protocols in order to obtain complete and mathematically rigorous proofs with respect to cryptographic definitions. We speak of (cryptographically) sound proofs in this case. Ideally, these proofs should be performed machine-aided in order to eliminate (or at least minimize) human inaccuracies. As formally verifying cryptographic protocols presupposes abstractions of them, which are suitable for formal methods, it has to be ensured that properties proved for these abstract specifications carry over to the concrete implementations.

Both formal verification and cryptographically sound proofs have been investigated very well from their respective communities during the last years. Especially the formal verification has been subject of lots of papers in the literature, e.g., [20, 13, 21, 1, 16]. The underlying abstraction is almost always based on the Dolev-Yao model [7]. Here cryptographic operations, e.g., E for encryption and D for decryption, are considered as operators in a free algebra where only certain predefined cancellation rules hold, i.e., twofold encryption of a message m does not yield another message from the basic message space but the term $E(E(m))$. A typical cancellation rule is $D(E(m)) = m$.

[†] Work was done while both authors were affiliated with Saarland University.

This abstraction simplifies proofs of larger protocols considerably. Unfortunately, these formal proofs lack a link to the rigorous security definitions in cryptography. The main problem is that the abstraction requires that *no* equations hold except those that can be derived within the algebra. Cryptographic definitions do not aim at such statements. For example, encryption is only required to keep cleartexts secret, but there is no restriction on structure in the ciphertexts. Hence a protocol that is secure in the Dolev-Yao framework is not necessarily secure in the real world even if implemented with provably secure cryptographic primitives, cf. [17] for a concrete counterexample. Thus, the appropriateness of this approach is at least debatable.

On the other hand, we have the computational view whose definitions are based on complexity theory, e.g., [8, 9, 5]. Here, protocols can be rigorously proven with respect to cryptographic definitions, but these proofs are not feasible for formal verification because of the occurrence of probabilities and complexity-theoretic restrictions, so they are both prone to errors and simply too complex for larger systems.

Unfortunately, achieving both points at the same time seems to be a very difficult task. To the best of our knowledge, no cryptographic protocol has been formally verified such that this verification is valid for the concrete implementation with respect to the strongest possible cryptographic definitions, cf. the Related Literature for more details.

Our goal is to link both approaches to get the best overall result: proofs of cryptographic protocols that allow abstraction and the use of formal methods, but retain a sound cryptographic semantics. Moreover, these proofs should be valid for completely asynchronous networks and the strongest cryptographic definitions possible, e.g., security against adaptive chosen ciphertext attack [5] in case of asymmetric encryption.

In this paper, we address the verification of integrity properties such that these properties automatically carry over from the abstract specification to the concrete implementation. Our work is motivated by the work of Pfizmann and Waidner which have already shown in [18] that integrity properties are preserved under refinement for a synchronous timing model. However, a synchronous definition of time is difficult to justify in the real world since no notion of rounds is naturally given there and it seems to be very difficult to establish them for the Internet, for example. In contrast to that, asynchronous scenarios are attractive, because no assumptions are made about network delays and the relative execution speed of the parties. Moreover, [18] solely comprises the theoretical background, since they neither investigated the use of nor actually used formal proof tools for the verification of a concrete example.

Technically, the first part of our work can be seen as an extension of the results of [18] to asynchronous scenarios as presented in [19]. This extension is not trivial since synchronous time is much easier to handle; moreover, both models do not only differ in the definition of time but also in subtle, but important details. The second part of this paper is dedicated to the actual verification of a concrete cryptographic protocol: secure message transmission with ordered channels [4]. This yields the first example of a machine-aided proof of a cryptographic protocol in asynchronous networks such that the proven security is equivalent to the strongest cryptographic definition possible.

Related Literature. An extended version of this work is available as an IBM Research Report [3]. The goal of retaining a sound cryptographic semantics and nevertheless provide abstract interfaces for formal methods is pursued by several researchers: our

approach is based on the model for reactive systems in asynchronous networks recently introduced in [19], which we believe to be really close to this goal. As we already mentioned above, Pfizmann and Waidner have shown in [18] that integrity properties are preserved for reactive systems, but only under a synchronous timing model and they have neither investigated the use of formal methods nor the verification of a concrete example. Other possible ways to achieve this goal have been presented in [20, 21, 13, 16, 12, 1], e.g., but these either do not provide abstractions for using formal methods, or they are based on unfaithful abstractions – following the approach of Dolev and Yao [7] – in the sense that no secure cryptographic implementation of them is known.

In [2], Abadi and Rogaway have shown that a slight variation of the standard Dolev-Yao abstraction is cryptographically faithful specifically for symmetric encryption. However, their results hold only for passive adversaries and for a synchronous timing model, but the authors already state that active adversaries and an asynchronous definition of time are important goals to strive for. Another interesting approach has been presented by Guttman et. al. [11], which starts adapting the strand space theory to concrete cryptographic definitions. However, their results are specific for the Wegman-Carter system so far. Moreover, as this system is information-theoretically secure, its security proof is much easier to handle than asymmetric primitives since no reduction proofs against underlying number-theoretic assumptions have to be made.

2 Reactive Systems in Asynchronous Networks

In this section we briefly sketch the model for reactive systems in asynchronous networks from [19]. All details not necessary for understanding are omitted. Machines are represented by probabilistic state-transition machines, similar to probabilistic I/O automata [14]. For complexity we consider every automaton to be implemented as a probabilistic Turing machine; complexity is measured in the length of its initial state, i.e., the initial worktape content (often a security parameter k in unary representation).

2.1 General System Model and Simulatability

A *system* consists of several possible *structures*. A structure is a pair (\hat{M}, S) of a set \hat{M} of connected correct machines and a subset S of the free ports¹, called *specified ports*. Roughly, specified ports provide certain services to the honest users. In a *standard cryptographic system*, the structures are derived from one intended structure and a trust model. The trust model consists of an access structure ACC and a channel model χ . Here ACC contains the possible sets \mathcal{H} of indices of uncorrupted machines (among the intended ones), and χ designates whether each channel is secure, reliable (authentic but not private) or insecure. Each structure can be completed to a *configuration* by adding an arbitrary *user* machine H and *adversary* machine A . H connects only to ports in S and A to the rest, and they may interact. The general scheduling model in [19] gives each connection c a buffer, and the machine with the corresponding clock port

¹ A port is called *free* if its corresponding port does not belong to a machine in \hat{M} . These ports are connected to the users and the adversary.

c^{\leq} can schedule a message when it makes a transition. In real asynchronous cryptographic systems, all connections are typically scheduled by A. Thus a configuration is a runnable system, i.e., one gets a probability space of *runs* and *views* of individual machines in these runs. For a configuration $conf$, we denote the random variables over this probability space by $run_{conf,k}$ and $view_{conf,k}$, respectively. For a polynomial l , we further obtain random variables for l -step prefixes of the runs, denoted by $run_{conf,k,l(k)}$. Moreover, a run r can be restricted to a set S of ports which is denoted by $r \upharpoonright_S$.

Simulatability essentially means that whatever can happen to certain users in the real system can also happen to the same users in the ideal (abstract) system: For every structure $struc_1 = (\hat{M}_1, S_1)$ of the real system, every user H, and every adversary A_1 there exists an adversary A_2 on a corresponding ideal structure $struc_2 = (\hat{M}_2, S_2)$, such that the view of H in the two configurations is indistinguishable, cf. Figure 1. Indistinguishability is a well-defined cryptographic notion from [22]. We write this $Sys_{real} \geq_{sec} Sys_{id}$

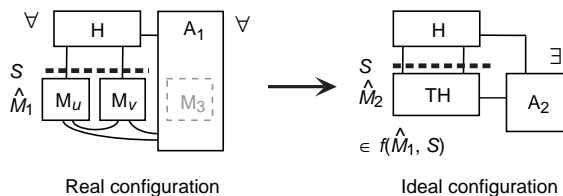


Fig. 1. Overview of the simulatability definition. The view of H must be indistinguishable in the two configurations. In this example, $\mathcal{H} = \{1, 2\}$.

and say that Sys_{real} is *at least as secure as* Sys_{id} . In general, a mapping f may denote the correspondence between ideal and real structures and one writes \geq_{sec}^f , but with the further restriction that f maps identical sets of specified ports. An important feature of the system model is transitivity of \geq_{sec} , i.e., the preconditions $Sys_1 \geq_{sec} Sys_2$ and $Sys_2 \geq_{sec} Sys_3$ together imply $Sys_1 \geq_{sec} Sys_3$ [19].

In a typical ideal system, each structure contains only one machine TH called *trusted host*, whereas structures of real systems typically consist of several machines M_i , one for each honest user.

3 Integrity Properties

In this section, we show how the relation “at least as secure as” relates to integrity properties a system should fulfill, e.g., safety properties expressed in temporal logic. As a rather general version of integrity properties, independent of the concrete formal language, we consider those that have a linear-time semantics, i.e., that correspond to a set of allowed traces of in- and outputs. We allow different properties for different sets of specified ports, since different requirements of various parties in cryptography are often made for different trust assumptions.

Definition 1 (Integrity Properties). An integrity property Req for a system Sys is a function that assigns a set of valid traces at the ports in S to each set S with $(\hat{M}, S) \in$

Sys . More precisely such a trace is a sequence $(v_i)_{i \in \mathbb{N}}$ of values over port names and Σ^* , so that v_i is of the form $v_i := \bigcup_{p \in S} \{p : v_{p,i}\}$ and $v_{p,i} \in \Sigma^*$. We say that Sys fulfills Req

- a) *perfectly* (written $Sys \models^{\text{perf}} Req$) if for any configuration $conf = (\hat{M}, S, H, A) \in \text{Conf}(Sys)$, the restrictions $r \upharpoonright_S$ of all runs of this configuration to the specified ports S lie in $Req(S)$. In formulas, $[(run_{conf,k} \upharpoonright_S)] \subseteq Req(S)$ for all k , where $[\cdot]$ denotes the carrier set of a probability distribution.
- b) *statistically* for a class $SMALL$ ($Sys \models^{SMALL} Req$) if for any configuration $conf = (\hat{M}, S, H, A) \in \text{Conf}(Sys)$, the probability that $Req(S)$ is not fulfilled is small, i.e., for all polynomials l (and as a function of k),

$$P(run_{conf,k,l(k)} \upharpoonright_S \notin Req(S)) \in SMALL.$$

The class $SMALL$ must be closed under addition and contain any function g' less than or equal to any function $g \in SMALL$.

- c) *computationally* ($Sys \models^{\text{poly}} Req$) if for any polynomial configuration $conf = (\hat{M}, S, H, A) \in \text{Conf}_{\text{poly}}(Sys)$, the probability that $Req(S)$ is not fulfilled is negligible, i.e.,

$$P(run_{conf,k} \upharpoonright_S \notin Req(S)) \in NEGL.$$

For the computational and statistical case, the trace has to be finite. Note that a) is normal fulfillment. We write “ \models ” if we want to treat all three cases together. \diamond

Obviously, perfect fulfillment implies statistical fulfillment for every non-empty class $SMALL$ and statistical fulfillment for a class $SMALL$ implies fulfillment in the computational case if $SMALL \subseteq NEGL$.

We now prove that integrity properties of abstract specifications carry over to their concrete counterparts in the sense of simulatability, i.e., if the properties are valid for a specification, the concrete implementation also fulfills concrete versions of these goals. As specifications are usually built by only one idealized, deterministic machine TH, they are quite easy to verify using formal proof systems, e.g., PVS. Now, our result implies that these verified properties automatically carry over to the (usually probabilistic) implementation without any further work.

The actual proof will be done by contradiction, i.e., we will show that if the real system does not fulfill its goals, the two systems can be distinguished. However, in order to exploit simulatability, we have to consider an honest user that connects to *all* specified ports. Otherwise, the contradiction might stem from those specified ports which are connected to the adversary, but those ports are not considered by simulatability. The following lemma will help us to circumvent this problem:

Lemma 1. *Let a system Sys be given. For every configuration $conf = (\hat{M}, S, H, A) \in \text{Conf}(Sys)$, there is a configuration $conf_s = (\hat{M}, S, H_s, A_s) \in \text{Conf}(Sys)$ with $S \subseteq \text{ports}(H_s)$, such that $run_{conf} \upharpoonright_S = run_{conf_s} \upharpoonright_S$, i.e., the probability of the runs restricted to the set S of specified ports is identical in both configurations. If $conf$ is polynomial-time, then $conf_s$ is also polynomial-time. \square*

We omit the proof due to space constraints and refer to [3].

Lemma 2. *The statistical distance $\Delta(\phi(\text{var}_k), \phi(\text{var}'_k))$ between a function ϕ of two random variables is at most $\Delta(\text{var}_k, \text{var}'_k)$. \square*

This is a well-known fact, hence we omit the easy proof.

Theorem 1 (Conservation of Integrity Properties). Let a system Sys_2 be given that fulfills an integrity property Req , i.e., $Sys_2 \models Req$, and let $Sys_1 \geq_{\text{sec}}^f Sys_2$ for an arbitrary mapping f . Then also $Sys_1 \models Req$. This holds in the perfect and statistical sense, and in the computational sense if membership in the set $Req(S)$ is decidable in polynomial time for all S . \square

Proof. Req is well-defined on Sys_1 , since simulatability implies that for each $(\hat{M}_1, S_1) \in Sys_1$ there exists $(\hat{M}_2, S_2) \in f(\hat{M}_1, S_1)$ with $S_1 = S_2$. We will now prove that if Sys_1 does not fulfill the property, the two systems can be distinguished yielding a contradiction.

Assume that a configuration $conf_1 = (\hat{M}_1, S_1, H, A_1)$ of Sys_1 contradicts the theorem. As already described above, we need an honest user that connects to all specified ports. This is precisely what Lemma 1 does, i.e., there is a configuration $conf_{s,1}$ in which the user connects to all specified ports, with $run_{conf_{s,1}} \upharpoonright_{S_1} = run_{conf_1} \upharpoonright_{S_1}$, so $conf_{s,1}$ also contradicts the theorem. Note that all specified ports are now connected to the honest user; thus, we can exploit simulatability.² Because of our precondition $Sys_1 \geq_{\text{sec}}^f Sys_2$, there exists an indistinguishable configuration $conf_{s,2} = (\hat{M}, S, H_s, A_2)$ of Sys_2 , i.e., $view_{conf_{s,1}}(H_s) \approx view_{conf_{s,2}}(H_s)$. By assumption, the property is fulfilled for this configuration (perfectly, statistically, or computationally). Furthermore, the view of H_s in both configurations contains the trace at $S := S_1 = S_2$, i.e., the trace is a function \upharpoonright_S of the view.

In the perfect case, the distribution of the views is identical. This contradicts the assumption that $[(run_{conf_{s,1,k}} \upharpoonright_S)] \not\subseteq Req(S)$ while $[(run_{conf_{s,2,k}} \upharpoonright_S)] \subseteq Req(S)$.

In the statistical case, let any polynomial l be given. The statistical distance $\Delta(view_{conf_{s,1,k,l(k)}}(H_s), view_{conf_{s,2,k,l(k)}}(H_s))$ is a function $g(k) \in SMALL$. We apply Lemma 2 to the characteristic function $1_{v \upharpoonright_S \notin Req(S)}$ on such views v . This gives $|P(run_{conf_{s,1,k,l(k)}} \upharpoonright_S \notin Req(S)) - P(run_{conf_{s,2,k,l(k)}} \upharpoonright_S \notin Req(S))| \leq g(k)$. As $SMALL$ is closed under addition and under making functions smaller, this gives the desired contradiction.

In the computational case, we define a distinguisher Dis : Given the view of machine H_s , it extracts the run restricted to S and verifies whether the result lies in $Req(S)$. If yes, it outputs 0, otherwise 1. This distinguisher is polynomial-time (in the security parameter k) because the view of H_s is of polynomial length, and membership in $Req(S)$ was required to be polynomial-time decidable. Its advantage in distinguishing is $|P(Dis(1^k, view_{conf_{s,1,k}}) = 1) - P(Dis(1^k, view_{conf_{s,2,k}}) = 1)| = |P(run_{conf_{s,1,k}} \upharpoonright_S \notin Req(S)) - P(run_{conf_{s,2,k}} \upharpoonright_S \notin Req(S))|$. Since the second term is negligible by assumption, and $NEGL$ is closed under addition, the first term also has to be negligible, yielding the desired contradiction. \blacksquare

² In the proof for the synchronous timing model, this problem was avoided by combining the honest user and the adversary to the new honest user. However, this combination would yield an invalid configuration in the asynchronous model.

In order to apply this theorem to integrity properties formulated in a logic, e.g., temporal logic, we have to show that abstract derivations in the logic are valid with respect to the cryptographic sense. This can be proven similar to the version with synchronous time, we only include it for reasons of completeness (without proof).

Theorem 2.

- a) If $Sys \models Req_1$ and $Req_1 \subseteq Req_2$, then also $Sys \models Req_2$.
- b) If $Sys \models Req_1$ and $Sys \models Req_2$, then also $Sys \models Req_1 \cap Req_2$.

Here “ \subseteq ” and “ \cap ” are interpreted pointwise, i.e., for each S . This holds in the perfect and statistical sense, and in the computational sense if for a) membership in $Req_2(S)$ is decidable in polynomial time for all S . \square

If we now want to apply this theorem to concrete logics, we have to show that the common deduction rules hold. For modus ponens, e.g., if one has derived that a and $a \rightarrow b$ are valid in a given model, then b is also valid in this model. If Req_a etc. denote the semantics of the formulas, i.e., the trace sets they represent, we have to show that

$$(Sys \models Req_a \text{ and } Sys \models Req_{a \rightarrow b}) \text{ implies } Sys \models Req_b.$$

From Theorem 2b we conclude $Sys \models Req_a \cap Req_{a \rightarrow b}$. Obviously, $Req_a \cap Req_{a \rightarrow b} = Req_{a \wedge b} \subseteq Req_b$ holds, so the claim follows from Theorem 2a.

4 Verification of the Ordered Channel Specification

In this section we review the specification for secure message transmission with ordered channels [4], and we formally verify that message reordering is in fact prevented.

4.1 Secure Message Transmission with Ordered Channels

Let n and $\mathcal{M} := \{1, \dots, n\}$ denote the number of participants and the set of indices respectively. The specification is of the typical form $Sys^{spec} = \{(\{TH_{\mathcal{H}}\}, S_{\mathcal{H}}) | \mathcal{H} \subseteq \mathcal{M}\}$, i.e., there is one structure for every subset of the machines, denoting the honest users. The remaining machines are corrupted, i.e., they are absorbed into the adversary.

The ideal machine $TH_{\mathcal{H}}$ models initialization, sending and receiving of messages. A user u can initialize communications with other users by inputting a command of the form `(snd_init)` to the port in_u of $TH_{\mathcal{H}}$. In real systems, initialization corresponds to key generation and authenticated key exchange. Sending of messages to a user v is triggered by a command `(send, m, v)`. If v is honest, the message is stored in an internal array $deliver_{u,v}^{spec}$ of $TH_{\mathcal{H}}$ together with a counter indicating the number of the message. After that, a command `(send_blindly, i, l, v)` is output to the adversary, l and i denote the length of the message m and its position in the array, respectively. This models that the adversary will notice in the real world that a message has been sent and he might also be able to know the length of that message. Because of the underlying asynchronous timing model, $TH_{\mathcal{H}}$ has to wait for a special term `(receive_blindly, u, i)` or `(rec_init, u)` sent by the adversary, signaling, that the message stored at the i th position

of $deliver_{u,v}^{\text{spec}}$ should be delivered to v , or that a connection between u and v should be initialized. In the first case, $\text{TH}_{\mathcal{H}}$ reads $(m, j) := deliver_{u,v}^{\text{spec}}[i]$ and checks whether $msg_out_{u,v}^{\text{spec}} \leq j$ holds for a message counter $msg_out_{u,v}^{\text{spec}}$. If the test is successful the message is delivered at $out_v!$ and the counter is set to $j + 1$, otherwise $\text{TH}_{\mathcal{H}}$ outputs nothing. The condition $msg_out_{u,v}^{\text{spec}} \leq j$ ensures that messages can only be delivered in the order they have been received by $\text{TH}_{\mathcal{H}}$, i.e., neither replay attacks nor reordering messages is possible for the adversary; cf. [4] for details. The user will receive inputs of the form $(receive, u, m)$ and (rec_init, u) , respectively. If v is dishonest, $\text{TH}_{\mathcal{H}}$ will simply output $(send, m, v)$ to the adversary. Finally, the adversary can send a message m to a user u by sending a command $(receive, v, m)$ to the port $from_adv_u?$ of $\text{TH}_{\mathcal{H}}$ for a corrupted user v , and he can also stop the machine of any user by sending a command $(stop)$ to a corresponding port of $\text{TH}_{\mathcal{H}}$, which corresponds to exceeding the machine's runtime bound in the real world.

In contrast to the concrete implementation, which we will review later on, the machine $\text{TH}_{\mathcal{H}}$ is completely deterministic, hence it can be expressed very well within a formal proof system, which supports the required data structures.

4.2 The Integrity Property

The considered specification has been designed to fulfill the property that the order of messages is maintained during every trace of the configuration. Thus, for arbitrary traces, arbitrary users $u, v \in \mathcal{H}$, $u \neq v$, and any point in time, the messages from v received so far by u via $\text{TH}_{\mathcal{H}}$ should be a sublist of the messages sent from v to $\text{TH}_{\mathcal{H}}$ aimed for forwarding to u . The former list is called *receive-list*, the latter *send-list*.

In order to obtain trustworthy proofs, we formally verify the integrity property in the theorem proving system PVS [15]. This will be described in the following. For reasons of readability and brevity, we use standard mathematical notation instead of PVS syntax. The PVS sources are available online.³

The formalization of the machine $\text{TH}_{\mathcal{H}}$ in PVS is described in [4]. We assume that the machine operates on an input set $\mathcal{I}_{\text{TH}_{\mathcal{H}}}$ (short \mathcal{I}), a state set $States_{\text{TH}_{\mathcal{H}}}$ (short S), and an output set $\mathcal{O}_{\text{TH}_{\mathcal{H}}}$ (short \mathcal{O}). For convenience, the transition function $\delta_{\text{TH}_{\mathcal{H}}} : \mathcal{I} \times S \rightarrow S \times \mathcal{O}$ is split into $\delta : \mathcal{I} \times S \rightarrow S$ and $\omega : \mathcal{I} \times S \rightarrow \mathcal{O}$, which denote the next-state and output part of $\delta_{\text{TH}_{\mathcal{H}}}$, respectively. The function $\delta_{\text{TH}_{\mathcal{H}}}$ is defined in PVS's specification language, which contains a complete functional programming language. PVS provides natural, rational, and real numbers, arithmetic, lists, arrays, etc. Furthermore, custom datatypes (including algebraic abstract datatypes) are supported.

In order to formulate the property, we need a PVS-suited, formal notation of (infinite) runs of a machine, of lists, of what it means that a list l_1 is a sublist of a list l_2 , and we need formalizations of the *receive-list* and *send-list*.

Definition 2 (Input sequence, state trace, output sequence). Let M be a machine with input set \mathcal{I}_M , state set $States_M$, output set \mathcal{O}_M , state transition function δ , and output transition function ω . Call $s_{init} \in States_M$ the initial state. An input sequence $i : \mathbb{N} \rightarrow \mathcal{I}_M$ for machine M is a function mapping the time (modeled as the set \mathbb{N})

³ <http://www.zurich.ibm.com/~mbc/OrdSecMess.tgz>

to inputs $i(t) \in \mathcal{I}_M$. A given input sequence i defines a sequence of states $s^i: \mathbb{N} \rightarrow \text{States}_M$ of the machine M by the following recursive construction:

$$\begin{aligned} s^i(0) &:= s_{init}, \\ s^i(t+1) &:= \delta(i(t), s^i(t)). \end{aligned}$$

The sequence s^i is called state-trace of M under i . The output sequence $o^i: \mathbb{N} \rightarrow O$ of the run is defined as

$$o^i(t) := \omega(i(t), s^i(t)).$$

We omit the index i if the input sequence is clear from the context. For components x of the state type, we write $x(t)$ for the content of x in $s(t)$, e.g., we write $deliver_{u,v}^{\text{spec}}(t)$ to denote the content at time t of the list $deliver_{u,v}^{\text{spec}}$, which is part of the state of $\text{TH}_{\mathcal{H}}$. \diamond

These definitions precisely match the model-intern definition of views, cf. [19]. In the context of $\text{TH}_{\mathcal{H}}$, the input sequence i consists of the messages that the honest users and the adversary send to $\text{TH}_{\mathcal{H}}$. In the following, a list l_1 being a sublist of a list l_2 is expressed by $l_1 \subseteq l_2$, l_1 being a sublist of the k -prefix of l_2 by $l_1 \subseteq^k l_2$.

Lemma 3. *Let l_1, l_2 be lists over some type T , let $k \in \mathbb{N}_0$. It holds:*

$$k < \text{length}(l_2) \wedge l_1 \subseteq^k l_2 \implies \text{append}(\text{nth}(l_2, k), l_1) \subseteq^{k+1} l_2,$$

that is, one may append the k^{th} element (counted from 0) of l_2 to l_1 while preserving the prefix-sublist property. \square

Definition 3 (Receive- and send-list). *Let i be an input sequence for machine $\text{TH}_{\mathcal{H}}$, and let s and o be the corresponding state-trace of $\text{TH}_{\mathcal{H}}$ and the output sequence, respectively. Let $u, v \in \mathcal{H}$. The receive-list is obtained by appending a new element m whenever v receives a message (receive, m, u) from $\text{TH}_{\mathcal{H}}$. The send-list is obtained by appending m whenever u sends a message (send, m, v) to $\text{TH}_{\mathcal{H}}$. Formally, this is captured in the following recursive definitions:*

$$\begin{aligned} \text{recvlist}_{u,v}^i(t) &:= \begin{cases} \text{null} & \text{if } t = -1, \\ \text{append}(m, \text{recvlist}_{u,v}^i(t-1)) & \text{if } t \geq 0 \wedge o^i(t) = (\text{receive}, m, u) \\ & \text{at } \text{out}_v!. \\ \text{recvlist}_{u,v}^i(t-1) & \text{otherwise} \end{cases} \\ \text{sendlist}_{u,v}^i(t) &:= \begin{cases} \text{null} & \text{if } t = -1, \\ \text{append}(m, \text{sendlist}_{u,v}^i(t-1)) & \text{if } t \geq 0 \wedge i(t) = (\text{send}, m, v) \\ & \text{at } \text{in}_u?. \\ \text{sendlist}_{u,v}^i(t-1) & \text{otherwise} \end{cases} \end{aligned}$$

\diamond

We now are ready to give a precise, PVS-suited formulation of the integrity property we are aiming to prove:

Theorem 3. For any $\text{TH}_{\mathcal{H}}$ input sequence i , for any $u, v \in \mathcal{H}$, $u \neq v$, and any point in time $t \in \mathbb{N}$, it holds

$$\text{rcvlist}_{u,v}^i(t) \subseteq \text{sendlist}_{u,v}^i(t). \quad (1)$$

In the following, we omit the index i . □

Proof (sketch). The proof is split into two parts: we prove $\text{rcvlist}_{u,v}(t-1) \subseteq \text{deliver}_{u,v}^{\text{spec}}(t)$ and $\text{deliver}_{u,v}^{\text{spec}}(t) \subseteq \text{sendlist}_{u,v}(t-1)$. The claim of the theorem then follows from transitivity of sublists.

The claim $\text{deliver}_{u,v}^{\text{spec}}(t) \subseteq \text{sendlist}_{u,v}(t-1)$ is proved by induction on t . Both induction base and step are proved in PVS by the built-in strategy (`grind`), which performs automatic definition expanding and rewriting with sublist-related lemmas.

The claim $\text{rcvlist}_{u,v}(t-1) \subseteq \text{deliver}_{u,v}^{\text{spec}}(t)$ is more complicated. The claim is also proved by induction on t . However, it is easy to see that the claim is not inductive: in case of a `(receive_blindly, u, i)` at `from_adv_v?`, $\text{TH}_{\mathcal{H}}$ outputs `(receive, m, u)` to `out_v!`, where $(m, j) := \text{deliver}_{u,v}^{\text{spec}}[i]$, i.e., m is the i th message of the $\text{deliver}_{u,v}^{\text{spec}}$ list (cf. Section 4.1, or [4] for more details). By the definition of the receive-list, the message m is appended to $\text{rcvlist}_{u,v}$. In order to prove that $\text{rcvlist}_{u,v} \subseteq \text{deliver}_{u,v}^{\text{spec}}$ is preserved during this transition, it is necessary to know that the receive list was a sublist of the prefix of the $\text{deliver}_{u,v}^{\text{spec}}$ list that does not reach to m . It would suffice to know that

$$\text{rcvlist}_{u,v}(t-1) \subseteq^i \text{deliver}_{u,v}^{\text{spec}}(t).$$

Then the claim follows from Lemma 3.

We therefore strengthen the invariant to comprise the prefix-sublist property. However, the value i in the above prefix-sublist relation stems from the input `(receive_blindly, u, i)`, and hence is not suited to state the invariant. To circumvent this problem, we recursively construct a sequence $\text{last_rcv_blindly}_{u,v}(t)$ which holds the parameter i of the last valid `(receive_blindly, u, i)` received by $\text{TH}_{\mathcal{H}}$ on `from_adv_v?`; then

$$\text{rcvlist}_{u,v}(t-1) \subseteq^l \text{deliver}_{u,v}^{\text{spec}}(t) \text{ with } l = \text{last_rcv_blindly}_{u,v}(t)$$

is an invariant of the system. We further strengthen this invariant by asserting that $\text{last_rcv_blindly}_{u,v}(t)$ and the j 's stored in the $\text{deliver}_{u,v}^{\text{spec}}$ list grow monotonically. Together this yields the inductive invariant. We omit the details and again refer the to the PVS files available online.³ ■

Applying Definition 1 of integrity properties, we can now define that the property Req holds for an arbitrary trace tr if and only if Equation 1 holds for all $u, v \in \mathcal{H}$, $u \neq v$, and the input sequence i of the given trace tr . Thus, Theorem 3 can be rewritten in the notation of Definition 1 as $[(\text{run}_{\text{conf}, k}[S]) \subseteq \text{Req}(S)] \subseteq \text{Req}(S)$ for all k , i.e., we have shown that the specification Sys^{spec} perfectly fulfills the integrity property Req .

4.3 The Concrete Implementation

For understanding it is sufficient to give a brief review of the concrete implementation Sys^{impl} , a detailed description can be found in [4]. Sys^{impl} is of the typical form $\text{Sys}^{\text{impl}} = \{(\hat{M}_{\mathcal{H}}, S_{\mathcal{H}}) \mid \mathcal{H} \subseteq \mathcal{M}\}$, where $\hat{M}_{\mathcal{H}} = \{M_u \mid u \in \mathcal{H}\}$, i.e., there is one

machine for each honest participant. It uses asymmetric encryption and digital signatures as cryptographic primitives, which satisfy the strongest cryptographic definition possible, i.e., security against adaptive chosen-ciphertext attack in case of encryption (e.g., [6]) and security against existential forgery under adaptive chosen-message attacks in case of digital signatures (e.g., [10]).

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. Similar to $\text{TH}_{\mathcal{H}}$ each machine maintains internal counters used for discarding messages that are out of order. The adversary schedules the communication between correct machines and it can send arbitrary messages m to arbitrary users.

Now the validity of the integrity property of the concrete implementation immediately follows from the Preservation Theorem and the verification of the abstract specification. More precisely, we have shown that the specification fulfills its integrity property of Theorem 3 perfectly, which especially implies computational fulfillment. As $\text{Sys}^{\text{impl}} \stackrel{\text{poly}}{\underset{\text{sec}}{\geq}} \text{Sys}^{\text{spec}}$ has already been shown in [4], our proof of integrity carries over to the concrete implementation for the computational case according to Theorem 1.

5 Conclusion

In this paper, we have addressed the problem how cryptographic protocols in asynchronous networks can be verified both machine-aided and sound with respect to the definitions of cryptography. We have shown that the verification of integrity properties of our abstract specifications automatically carries over to the cryptographic implementations, and that logic derivations among integrity properties are valid for the concrete systems in the cryptographic sense, which makes them accessible to theorem provers. As an example, we have formally verified the scheme for ordered secure message transmission [4] using the theorem proving system PVS [15]. This yields the first formal verification of an integrity property of a cryptographic protocol whose security is equivalent to the underlying cryptography with respect to the strongest cryptographic definitions possible.

References

1. M. Abadi and A. D. Gordon. A calculus for cryptographic protocols: The spi calculus. *Information and Computation*, 148(1):1–70, 1999.
2. M. Abadi and P. Rogaway. Reconciling two views of cryptography: The computational soundness of formal encryption. In *Proc. 1st IFIP International Conference on Theoretical Computer Science*, volume 1872 of *Lecture Notes in Computer Science*, pages 3–22. Springer, 2000.
3. M. Backes and C. Jacobi. Cryptographically sound and machine-assisted verification of security protocols. Research Report RZ 3468, IBM Research, 2002.
4. 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.

5. M. Bellare, A. Desai, D. Pointcheval, and P. Rogaway. Relations among notions of security for public-key encryption schemes. In *Advances in Cryptology: CRYPTO '98*, volume 1462 of *Lecture Notes in Computer Science*, pages 26–45. Springer, 1998.
6. R. Cramer and V. Shoup. Practical public key cryptosystem provably secure against adaptive chosen ciphertext attack. In *Advances in Cryptology: CRYPTO '98*, volume 1462 of *Lecture Notes in Computer Science*, pages 13–25. Springer, 1998.
7. D. Dolev and A. C. Yao. On the security of public key protocols. *IEEE Transactions on Information Theory*, 29(2):198–208, 1983.
8. S. Goldwasser and S. Micali. Probabilistic encryption. *Journal of Computer and System Sciences*, 28:270–299, 1984.
9. S. Goldwasser, S. Micali, and C. Rackoff. The knowledge complexity of interactive proof systems. *SIAM Journal on Computing*, 18(1):186–207, 1989.
10. S. Goldwasser, S. Micali, and R. L. Rivest. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM Journal on Computing*, 17(2):281–308, 1988.
11. J. D. Guttman, F. J. Thayer Fabrega, and L. Zuck. The faithfulness of abstract protocol analysis: Message authentication. In *Proc. 8th ACM Conference on Computer and Communications Security*, pages 186–195, 2001.
12. P. Lincoln, J. Mitchell, M. Mitchell, and A. Scedrov. A probabilistic poly-time framework for protocol analysis. In *Proc. 5th ACM Conference on Computer and Communications Security*, pages 112–121, 1998.
13. 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.
14. N. Lynch. *Distributed Algorithms*. Morgan Kaufmann Publishers, San Francisco, 1996.
15. 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.
16. L. Paulson. The inductive approach to verifying cryptographic protocols. *Journal of Cryptology*, 6(1):85–128, 1998.
17. 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>.
18. B. Pfitzmann and M. Waidner. Composition and integrity preservation of secure reactive systems. In *Proc. 7th ACM Conference on Computer and Communications Security*, pages 245–254, 2000.
19. 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.
20. 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.
21. S. Schneider. Security properties and CSP. In *Proc. 17th IEEE Symposium on Security & Privacy*, pages 174–187, 1996.
22. A. C. Yao. Theory and applications of trapdoor functions. In *Proc. 23rd IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 80–91, 1982.