

# On Simulatability Soundness and Mapping Soundness of Symbolic Cryptography

Michael Backes<sup>1</sup>, Markus Dürmuth<sup>1</sup>, and Ralf Küsters<sup>2</sup>

<sup>1</sup> Saarland University, Saarbrücken, Germany, {backes|duermuth}@cs.uni-sb.de

<sup>2</sup> ETH Zürich, Switzerland, ralf.kuesters@inf.ethz.ch

**Abstract.** The abstraction of cryptographic operations by term algebras, called Dolev-Yao models or symbolic cryptography, is essential in almost all tool-supported methods for proving security protocols. Recently significant progress was made – using two conceptually different approaches – in proving that Dolev-Yao models can be sound with respect to actual cryptographic realizations and security definitions. One such approach is grounded on the notion of simulatability, which constitutes a salient technique of Modern Cryptography with a long-standing history for a variety of different tasks. The other approach strives for the so-called mapping soundness – a more recent technique that is tailored to the soundness of specific security properties in Dolev-Yao models, and that can be established using more compact proofs. Typically, both notions of soundness for similar Dolev-Yao models are established separately in independent papers.

In this paper, the two approaches are related for the first time. Our main result is that simulatability soundness entails mapping soundness provided that both approaches use the same cryptographic implementation. Interestingly, this result does not depend on details of the simulator, which translates between cryptographic implementations and their Dolev-Yao abstractions in simulatability soundness. Hence, future research may well concentrate on simulatability soundness whenever applicable, and resort to mapping soundness in those cases where simulatability soundness is too strong a notion.

## 1 Introduction

Tool-supported verification of cryptographic protocols almost always relies on abstractions of cryptographic operations by term algebras with cancellation rules, called *symbolic cryptography* or *Dolev-Yao models* after the first authors [21]. An example term is  $D_{ske}(E_{pke}(E_{pke}(N)))$ , where  $E$  and  $D$  denote public-key encryption and decryption,  $ske$  and  $pke$  are corresponding private and public encryption keys, and  $N$  is a nonce (random string). The keys are written as indices for readability; formally they are normal operands in the term. A typical cancellation rule is  $D_{ske}(E_{pke}(t)) = t$  for all public/private key pairs  $(pke, ske)$  and terms  $t$ , thus the above term is equivalent to  $E_{pke}(N)$ . The proof tools handle these terms symbolically, i.e., they never evaluate them to bit strings. In other words, the tools perform abstract algebraic manipulations on trees consisting of operators and base messages, using only the cancellation rules, the message-construction rules of a particular protocol, and an abstract model of networks and adversaries.

It is not at all clear from the outset whether Dolev-Yao models are a sound abstraction from real cryptography with its computational security definitions, where messages are bit strings and the adversary is an arbitrary probabilistic polynomial-time (ppt) Turing machine. In particular, the tools assume that *only* the modeled operations and cancellation rules are possible manipulations on terms, and that terms that cannot be constructed with

these rules are completely secret. For instance, if an adversary (also called intruder) only saw the example term above and only the mentioned cancellation rule was given, then  $N$  would be considered secret.

Bridging this long-standing gap between Dolev-Yao models and real cryptographic definitions has recently received considerable attention, and remarkable progress has been made using two conceptually different approaches.

One such approach, which we call *simulatability soundness*, is grounded on the security notion of (black-box reactive) simulatability (BRSIM), which relates a real system (also called implementation or real protocol) with an ideal system (also called ideal functionality or ideal protocol). The real system is said to be as secure as the ideal system if every attack on the real system can be turned into an “equivalent” attack on the ideal system, where “equivalent” means indistinguishable by an environment (also called honest users). This security notion essentially means that the real system can be plugged into an arbitrary protocol instead of the ideal system without any noticeable difference [33, 34, 11]. Basically the same notion is also called UC (universal composability) for its universal composition properties [15].<sup>1</sup> In terms of the semantics community, BRSIM/UC could be called an implementation or refinement relation, with a particular emphasis on also retaining secrecy properties, in contrast to typical implementation relations. Now, results on simulatability soundness show that a (possibly augmented) Dolev-Yao model, specified as an ideal system, can be implemented in the sense of BRSIM/UC by a real system using standard cryptographic definitions. The first such result was presented in [9] and was extended to more cryptographic primitives in [10, 7]. The use of these results in protocol proofs was illustrated in [6, 4, 35, 3]. Simulatability soundness of a slightly simpler Dolev-Yao model and a restricted class of protocols using it was proven in [16].

The other approach, which we call *mapping soundness*, is grounded on a more recent technique that is tailored to the soundness of specific security properties in standard Dolev-Yao models. Mapping soundness for a given protocol is established by showing the existence of a mapping from bit strings to terms such that applying the mapping to an arbitrary trace of the real cryptographic execution of the protocol yields a trace of an ideal, Dolev-Yao style execution of the protocol. Compared to simulatability soundness, mapping soundness can often be established using more compact proofs and sometimes more relaxed cryptographic assumptions. Unlike simulatability soundness however, mapping soundness is restricted to specific classes of protocols, and it does not entail universal composition properties. This approach started with the seminal work of [2] which proved the mapping soundness of symmetric encryption under passive attacks. This result was extended in [1, 26] still for passive attacks. Various later papers extended this approach to active attacks and to different cryptographic primitives and security properties [30, 27, 19, 18, 16]. In the present work, we are concerned with mapping soundness for active attacks.

## 1.1 Our Results

In this paper, the two approaches are related for the first time. Our main result is that simulatability soundness entails mapping soundness provided that both approaches use

---

<sup>1</sup> The 2005 revision of the long version of [15] also contains an explicit blackbox version of UC, which is proven to be equivalent to UC. A similar equivalence was first shown in the long version of [33] for universal and blackbox synchronous reactive simulatability.

the same cryptographic implementation. Interestingly, this result does not depend on details of the simulator, which translates between cryptographic implementations and their Dolev-Yao abstractions in simulatability soundness. More precisely, our result is based on two assumptions on the Dolev-Yao model, specified as an ideal system  $M^{\text{ideal}}$ , and its cryptographic implementation  $M^{\text{real}}$ . The first assumption is that  $M^{\text{real}}$  is as secure as  $M^{\text{ideal}}$  in the sense of BRSIM/UC, which formalizes simulatability soundness for the Dolev-Yao model and the cryptographic implementation under consideration. The second assumption is that if protocols are executed based on  $M^{\text{ideal}}$  instead of  $M^{\text{real}}$ , then the resulting traces are so-called Dolev-Yao traces, i.e., these traces can be constructed while adhering to the rules of the term algebra and the protocol under consideration, which exactly reflects the intuition and purpose behind the Dolev-Yao model  $M^{\text{ideal}}$ . We furthermore demonstrate that our assumptions are met for the existing simulatability-sound Dolev-Yao model and its implementation from [9]. The argument of our proof only relies on these two assumptions and hence allows for easy extension to additional classes of cryptographic primitives and protocols beyond the ones considered in this paper.

We note that requiring the same cryptographic implementations for both simulatability soundness and mapping soundness means that existing results on simulatability soundness do not necessarily fully supersede existing results on mapping soundness: the former results may for instance require stronger assumptions on the security of cryptographic primitives, specific techniques from robust protocol design such as explicit type tags, additional randomization, etc. in order to establish simulatability between the cryptographic implementation and its Dolev-Yao abstraction. However, we believe that it is fair to say that future research may well concentrate on simulatability soundness whenever applicable, and resort to mapping soundness in those cases where simulatability soundness constitutes too strong a notion.

## 1.2 Further Related Work

Reactive simulatability was first defined generally in [33], based on simulatability definitions for secure (one-step) function evaluation [22, 23, 12, 29, 14]. On the side of formal methods, it is also highly related to the observational equivalence notions for a probabilistic  $\pi$ -calculus from [28]. Reactive definitions of simulatability for asynchronous systems were presented in [34, 15], called UC (universal composability) and with somewhat different details in the latter. Since then, these definitions have been used in many ways for proving individual cryptographic systems and general theorems. While the definitions of [34, 15] have not been rigorously mapped, we believe that for the results in this paper the differences do not matter, in particular if one thinks of the equivalent blackbox version of UC. Similarly, we believe that the results would hold in the formalism started in [28], and the recently proposed formalism put forward in [25].

In the wider field of linking formal methods and cryptography, there is also work on formulating syntactic calculi for dealing with probabilism and polynomial-time considerations directly and encoding them into proof tools, in particular [31, 32, 24, 20, 13]. This is orthogonal to the work of justifying Dolev-Yao models: In situations where Dolev-Yao models are applicable and sound, they are likely to remain important because of the strong simplification they offer to the tools, which enables the tools to treat larger overall systems automatically than with the more detailed models of cryptography.

### 1.3 Paper Outline

Section 2 reviews the basic terminology of symbolic cryptography, its deduction rules, and the definition of protocols. Section 3 reviews the notion of simulatability, and points out necessary requirements for a Dolev-Yao model to be sound in the sense of BRSIM/UC. Section 4 defines executions of protocols within the reactive simulatability framework, thus preparing a common ground for comparing both notions of soundness. Section 5 finally proves that simulatability soundness implies mapping soundness. Section 6 concludes.

## 2 Symbolic Cryptography

In this section we review basic terminology concerning Dolev-Yao models and the corresponding deduction rules for deriving new messages from a given set of messages. In addition, we give the definition of protocols along the lines of works on the mapping approach [30, 19, 18].

### 2.1 Basic Terminology, Dolev-Yao Terms, and Deduction Rules

We define  $\{0, 1\}^*$  to be the set of *payloads*. Payloads will typically be identifiers of agents, which is why we often refer to this set by ID. Similar to works on the mapping approach, see e.g., [19, 18], encryption and signature messages will be annotated with labels which represent random coins. This accounts for the fact that encrypting/signing the same message yields different ciphertexts/signatures when performed with different randomness. Therefore we define the set  $\text{Rand} := \text{ID} \cup \{\text{adv}\} \times \mathbb{N}$  of random coins where  $\text{adv} \notin \text{ID}$  denotes the adversary. A tuple  $(a, r)$  stands for the randomness  $r$  generated by agent  $a$ . By  $\text{ek}(A)$ ,  $\text{dk}(A)$ ,  $\text{sk}(A)$ , and  $\text{vk}(A)$  we denote the encryption, decryption, signing, and verification key of agent  $A \in \text{ID}$ , respectively.

Now, the set of (Dolev-Yao) messages  $M$  is defined by the following grammar:

$$M ::= \text{ID} \mid \langle M, M \rangle \mid \text{Nonce} \mid E_{\text{ek}(\text{ID})}^{\text{Rand}}(M) \mid \text{Sig}_{\text{vk}(\text{ID})}^{\text{Rand}}(M). \quad (1)$$

where  $\text{Nonce}$  is a set of nonces. We sometimes drop the labels for the random coins if they are not needed or clear from the context.

Given a set of messages  $\varphi$ , we now recall standard Dolev-Yao style rules for deriving new messages from  $\varphi$ . The derivation relation  $\vdash^A$  is parameterized by an agent  $A \in \text{ID} \cup \{\text{adv}\}$  since different agents use different random coins.

- *Initial knowledge*:  $\varphi \vdash^A m$  for all  $m \in \varphi$ ,
- *Pairing and unpairing*: If  $\varphi \vdash^A m_1$  and  $\varphi \vdash^A m_2$ , then  $\varphi \vdash^A \langle m_1, m_2 \rangle$ ; conversely, if  $\varphi \vdash^A \langle m_1, m_2 \rangle$ , then  $\varphi \vdash^A m_1$  and  $\varphi \vdash^A m_2$ .
- *Encryption and decryption*: If  $\varphi \vdash^A \text{ek}(b)$  and  $\varphi \vdash^A m$ , then  $\varphi \vdash^A E_{\text{ek}(b)}^r(m)$  for all  $b \in \text{ID}$ ,  $r \in \{A\} \times \mathbb{N}$ ; conversely, if  $\varphi \vdash^A E_{\text{ek}(b)}^r(m)$  and  $\varphi \vdash^A \text{dk}(b)$ , then  $\varphi \vdash^A m$  for all  $b \in \text{ID}$ ,  $r \in \text{Rand}$ .
- *Signature*: If  $\varphi \vdash^A \text{sk}(b)$  and  $\varphi \vdash^A m$ , then  $\varphi \vdash^A \text{Sig}_{\text{vk}(b)}^r(m)$  for all  $b \in \text{ID}$ ,  $r \in \{A\} \times \mathbb{N}$ .
- *Signature transformation*: If  $\varphi \vdash^A \text{Sig}_{\text{vk}(b)}^r(m)$ , then  $\varphi \vdash^A \text{Sig}_{\text{vk}(b)}^{r'}(m)$  for  $b \in \text{ID}$ ,  $r \in \text{Rand}$ ,  $r' \in \{A\} \times \mathbb{N}$ .<sup>2</sup>

<sup>2</sup> Note that the standard definition of signature security (unforgeability under chosen-message attack) does not exclude that the adversary, given a signature for a certain message, can create another signature for the same message.

- *Plaintext retrieval*: If  $\varphi \vdash^A \text{Sig}_{\text{vk}(b)}^r(m)$ , then  $\varphi \vdash^A m$  for all  $b \in \text{ID}, r \in \text{Rand}$ ,
- *Verification-key retrieval*: If  $\varphi \vdash^A \text{Sig}_{\text{vk}(b)}^r(m)$ , then  $\varphi \vdash^A \text{vk}(b)$  for all  $b \in \text{ID}, r \in \text{Rand}$ .
- *Encryption-key retrieval*: If  $\varphi \vdash^A \text{E}_{\text{ek}(b)}^r(m)$ , then  $\varphi \vdash^A \text{ek}(b)$  for all  $b \in \text{ID}, r \in \text{Rand}$ ,

Sometimes we will ignore labels for random coins in derivations of  $\text{adv}$ . In this case, we write  $\vdash$  instead of  $\vdash^{\text{adv}}$ . More precisely, if  $m$  is a message and  $m'$  is obtained from  $m$  by dropping the random coins in  $m$ , then  $\varphi \vdash^{\text{adv}} m$  implies  $\varphi \vdash m'$ .

## 2.2 Definition of Protocols

A  $k$ -party protocol is defined by  $k$  roles, where a role specifies the behavior of a party in a protocol run. Defining roles requires to first introduce variables. We assume disjoint sets of typed variables  $\text{X}.n$  for nonces and  $\text{X}.d$  for payloads.

Now, the  $i$ th role,  $i = 1, \dots, k$ , is defined to be a directed, edge-labeled finite tree where the edges originating in the same node are linearly ordered. Each edge is labeled with a rule  $(l, r)$  for terms  $l$  and  $r$ , where terms are messages which may contain variables (see below). We use certain distinguished variables  $A_1, \dots, A_k \in \text{X}.d$  and  $N_j \in \text{X}.n$  for  $j \geq 0$ . When the  $i$ th role is instantiated with parties  $a_1, \dots, a_k$ , then  $A_j$  is substituted by  $a_j$  for every  $j = 1, \dots, k$  and for the variables  $N_j$  occurring in the role, fresh nonces are generated. Note that such an instance of the  $i$ th role is carried out by party  $a_i$ . Similar to [18], we put syntactic restrictions on a role to make sure that it can actually be carried out when giving it a computational interpretation (see below for these restrictions).

Intuitively, in a protocol run, a role is executed by a party as follows. The execution of the role starts in the root of the role. When receiving a message, the party tries to match the message against one of the left-hand sides of the rules that the outgoing edges of the current node (at the beginning this is the root) are labeled with. If no left-hand side matches, nothing happens; the party does not change its state. If a left-hand side of the rule of one of the edges matches with the message, then the first such edge is picked (according to the given linear ordering on outgoing edges) and the right-hand side of this rule is sent to the network (i.e., the adversary) as output. Matching a message against a term (the left-hand side of a rule), means that a party parses the message according to the term. For example, if the term is  $\text{E}_{\text{ek}(a)}(\langle a, y \rangle)$  for  $y \in \text{X}.n$ , then the party would first try to decrypt the message with  $\text{dk}(a)$ , then check whether the result is a pair, check whether the first component of this pair is the name  $a$ , check whether the second component is a nonce (i.e., is tagged as a nonce and has the expected length), and then store this nonce in  $y$ . When executing a role, some variables in left-hand sides of a rule might have occurred already in left-hand sides of rules of preceding edges. In this case, the value of the variable is not overwritten, but it is checked whether the old and the new value coincide. This is rigorously defined in Section 4.

As mentioned above, we need to put syntactic restrictions on roles to make sure that they can be executed. For this purpose, we restrict the kind of terms that can occur on the left-hand side and right-hand side in rules. For the  $i$ th role of a protocol, terms on the left-hand side of a rule have to be of the following form:

$$\text{T}_i^l ::= \text{ID} \mid \text{X}.n \mid \text{X}.d \mid \langle \text{T}_i^l, \text{T}_i^l \rangle \mid \text{E}_{\text{ek}(A_i)}(\text{T}_i^l) \mid \text{Sig}_{\text{vk}(\{A_1, \dots, A_k\})}(\text{T}_i^l).$$

As explained above  $E_{\text{ek}(A_i)}(t)$  means that the party  $A_i$  (recall that  $A_i$  carries out the  $i$ th role) decrypts the received message with  $\text{dk}(A_i)$  and then parses the plaintext according to  $t$ . Since  $A_i$  only knows its own decryption key  $\text{dk}(A_i)$ , terms of the form  $E_{\text{ek}(A_j)}(t)$  for  $j \neq i$  are not allowed. We allow  $A_i$  to check the validity of the signatures of all other principals, i.e.,  $A_i$  is assumed to know  $\text{vk}(A_j)$  for all  $j$ .

One could consider more general terms than those contained in  $T_i^l$ , e.g., terms that contain specific ciphertexts, variables for encryption/verification keys, or variables for ciphertexts in order to model ciphertext forwarding. While our results can be lifted to these cases, we concentrate on  $T_i^l$  as to not encumber our main ideas with details that are of only minor importance in this paper.

For the  $i$ th role of a protocol, terms on the right-hand side of a rule have to be of the following form:

$$T_i^r ::= \text{ID} \mid X.n \mid X.d \mid \langle T_i^r, T_i^r \rangle \mid E_{\text{ek}(\{A_1, \dots, A_k\})}(T_i^r) \mid \text{Sig}_{\text{vk}(A_i)}(T_i^r).$$

A term  $E_{\text{ek}(A_j)}(t)$  means that party  $A_i$  first computes a bit string  $b$  for  $t$  and then encrypts  $b$  with the public key of  $A_j$ ;  $\text{Sig}_{\text{vk}(A_i)}(t)$  has a similar meaning. We require that variables on the right-hand side of a rule belong to  $\{A_1, \dots, A_k\} \cup \{N_j \mid j \geq 0\}$ , or occur on the left-hand side of the rule, or occur on the left-hand side of a preceding rule in a role to ensure that values of these variables are determined by the time they are used to form output messages. As in the case of  $T_i^l$ , several extensions are possible but not considered here for reasons of clarity.

Let **Roles** describe the set of all roles. Then, a *k-party protocol* is a mapping  $\Pi: \{1, \dots, k\} \rightarrow \text{Roles}$ .

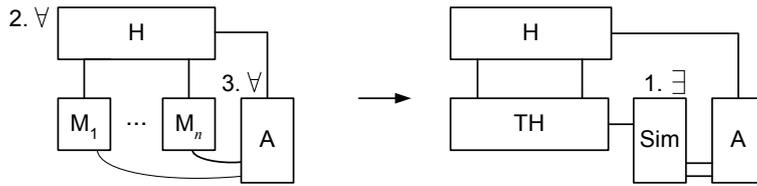
### 3 Simulatability and Requirements for Simulatability-sound Dolev-Yao Models

In this section, we review the notion of simulatability and point out necessary requirements for a Dolev-Yao model to be sound in the sense of BRSIM/UC.

#### 3.1 Review of Simulatability

Simulatability constitutes a general approach for comparing two systems, typically called real and ideal system. In terms of the semantics community one might speak of an implementation or refinement relation, specifically geared towards the preservation of what one might call secrecy properties compared with functional properties. We believe that all our following results are independent of the differences between the definition styles of the various recent papers on simulatability [28, 33, 15, 11, 25]. However, we have to fix a specific formalism, and we use that from [34, 11].

The ideal system in [34, 11] typically consists of a single machine TH, the *trusted host*, see Figure 1. In the context of simulatability soundness, TH represents a Dolev-Yao model. The real system consists of a set of machines  $M_u$ , one for every user  $u$ . In the context of simulatability soundness, the real system describes the cryptographic implementation. The ideal or real system interacts with arbitrary so-called *honest users*, collectively represented by a single machine H; this corresponds to potential protocols or human users interacting



**Figure 1.** Black-box reactive simulatability (BRSIM) between the real system  $M_1 \parallel \dots \parallel M_n$  and the ideal system TH, where  $M_u$  is the machine of user  $u \in \{1, \dots, n\}$ . The quantifiers are numbered to show their order.

with the ideal or real system. Furthermore, the ideal or real system interacts with an adversary A, who is often given more power than the honest users; in particular in real systems A typically controls the network and can manipulate messages on the bit string level. The adversary is also granted the ability to interact with the honest users H in order to influence their behavior, e.g., to suggest which messages are to be sent. Technically, the interaction with H models known-message and chosen-message attacks.

Now, *black-box reactive simulatability (BRSIM)* states that there exists a simulator Sim such that for all A, no H can distinguish (in the sense of computational indistinguishability of families of random variables [36]) whether it interacts with the real system and the real adversary, or with the ideal system and a combination of the real adversary and the simulator (which together form the ideal adversary) as depicted in Figure 1. Note that indistinguishability in particular entails that the ideal and real system offer identical interfaces to the honest users in order to prevent trivial distinguishability. We write  $M_1 \parallel \dots \parallel M_n \leq^{BRSIM} TH$  to say that the real system  $M_1 \parallel \dots \parallel M_n$  is *as secure as* the ideal system TH *in the sense of BRSIM/UC*.

The reader may regard the machines, i.e., the individual boxes in Figure 1, as probabilistic I/O automata, Turing machines, CSP or pi-calculus processes etc. The only requirement on the underlying system model is that the notion of an execution of a system when run together with an honest user and an adversary is well-defined. In [34, 11], the machines are a type of probabilistic I/O automata. We always assume that all parties are polynomial-time.

### 3.2 On Simulatability-based Dolev-Yao Models and their Cryptographic Implementations

We now outline necessary requirements a Dolev-Yao model  $M^{\text{ideal}}$  offering the capabilities described in Section 2 and an implementation  $M^{\text{real}} = M_1^{\text{real}} \parallel \dots \parallel M_n^{\text{real}}$  realized by actual cryptographic primitives have to fulfill for being simulatability-sound. Solely fixing minimal requirements typically expected from Dolev-Yao models instead of considering a specific Dolev-Yao model frees our results from specific details and idiosyncrasies of existing models.

For achieving simulatability, the Dolev-Yao model  $M^{\text{ideal}}$  and its cryptographic implementation  $M^{\text{real}}$  have to offer an identical I/O interface which the honest users connect to. We hence assume that the interaction at the I/O interface is based on handles (pointers) to objects stored in the system, i.e., the user never obtains real bit strings (nonces, ciphertexts, etc.) from the cryptographic implementation but only handles to such objects. The

only exception are payloads which obviously have to be retrievable in their bit string representation in some way. Note that we do not fix any specific instantiation of these handles but we only assume that they can be operated on in the expected manner as discussed below.

The I/O interface has to permit suitable commands for constructing terms according to the Dolev-Yao style deduction rules given in Section 2, and for sending them to other principals. This in particular comprises the generation of nonces, pairs of messages (i.e., concatenations of messages), pairs of public and private keys, to perform public-key encryption/decryption, to generate and verify signatures, to retrieve payloads from their handles, and to send and receive messages to/from the network. Moreover, there have to exist commands for parsing handles, in particular for testing handles for equality (for simplicity, we assume that each user  $u$  is deterministically given the same handle again if a term is reused), and for querying the types of handles.

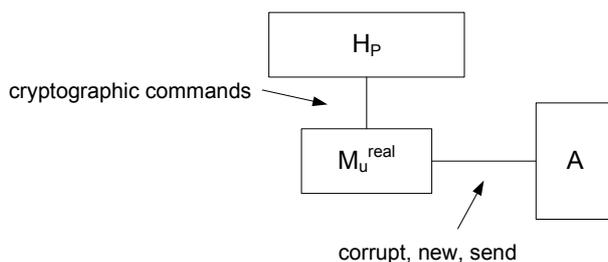
Concerning the network interface,  $M^{\text{ideal}}$  and  $M^{\text{real}}$  differ. The network interface of  $M^{\text{ideal}}$  offers the adversary commands for constructing and parsing terms according to the Dolev-Yao style deduction rules, and for sending the terms to users. The machines  $M_u^{\text{real}}$  output messages to and receive bit string messages from the adversary at their network interfaces.

Note that we did not describe the internal behavior of the Dolev-Yao model  $M^{\text{ideal}}$ . It turns out not to be relevant for achieving our results, but we later only have to require two properties of  $M^{\text{ideal}}$ : First,  $M^{\text{real}}$  is as secure as  $M^{\text{ideal}}$  in the sense of BRSIM/UC; second, the behavior of  $M^{\text{ideal}}$  in fact ensures that the adversary can only manipulate messages according to the Dolev-Yao rules presented in Section 2. More formally, the second property requires that when  $M^{\text{ideal}}$  is run with arbitrary honest users and an arbitrary adversary, the resulting protocol traces are so-called Dolev-Yao traces, which are defined in Section 5.

## 4 Reactive Execution of Protocols

We now describe the execution of a protocol  $\Pi$  along with an adversary who controls the network. More precisely, we describe the *concrete* execution of  $\Pi$ , i.e., the execution in which actual cryptographic algorithms are used, rather than their Dolev-Yao abstractions. Our definition corresponds to the one for mapping approaches [30, 19, 18]. However, we present the definition in the reactive simulatability framework using  $M^{\text{real}}$  in order to facilitate the presentation of our main result (Section 5).

We use an honest user machine  $H_\Pi$  to emulate the execution of  $\Pi$ . This machine makes use of  $M^{\text{real}}$  to carry out the necessary cryptographic operations. Recall that  $M^{\text{real}}$  uses actual cryptographic algorithms to perform the cryptographic operations and that at its I/O interface handles are used to point to the bit strings (payloads, ciphertexts, nonces etc.) stored in  $M^{\text{real}}$ . While  $M^{\text{real}}$  is a composition of machines  $M_u^{\text{real}}$ ,  $u \in \{1, \dots, n\}$ ,  $H_\Pi$  can emulate the execution of instances of  $\Pi$  by only using one  $M_u^{\text{real}}$  since within this machine key pairs for every party can be generated. This is even more general than using a separate machine for each party since it allows to model that the adversary dynamically generates new parties. We emphasize that the communication between the parties is still carried out over the network, so by using just one machine  $M_u^{\text{real}}$  we do not introduce any idealization. As usual, the network is controlled by the adversary  $A$ . The adversary can instruct  $H_\Pi$  to generate a new instance of a role  $\Pi(i)$  of  $\Pi$  and at the beginning of the execution  $A$



**Figure 2.** Setup of  $H_{II}$

can corrupt parties, which corresponds to the prevalent static corruption model for Dolev-Yao models. Altogether, the run of the system  $H_{II} \parallel M^{\text{real}} \parallel A$  corresponds to a concrete execution of instances of  $II$ . The setting is depicted in Figure 2.

It remains to describe  $H_{II}$ , i.e., the way  $H_{II}$  emulates instances of  $II$ . Similar to the definition of concrete executions in mapping approaches [19, 18], the machine  $H_{II}$  keeps a global state to remember what instances of  $II$  are running and in what local state these instances are. The *global state* is a tuple  $(\text{SId}, f, \varphi)$ , where

- $\text{SId}$  is a finite set of session IDs,
- $\varphi$  denotes the knowledge of the adversary at the current point in time, and
- $f$  maps every session identifier  $i$  in  $\text{SId}$  to the current (local) state  $f(i) = (i, \nu, p, (a_1, \dots, a_k))$  of that session, see below, where a session is an instance of one role of the protocol.

A *local state* is a tuple  $(i, \nu, p, (a_1, \dots, a_k))$  with the following components:  $i \in \{1, \dots, k\}$  is the index of the role  $II(i)$  that is executed in this session,  $\nu$  is a substitution that maps those variables in  $II(i)$  which were bound in the matching processes so far to handles (pointing to bit strings stored in  $M^{\text{real}}$ ),  $p$  is a node in the role  $II(i)$  marking the current point in the execution of  $II(i)$ , and  $(a_1, \dots, a_k)$  are the agents participating in this session. Recall that the session is carried out by  $a_i$  with the agents  $a_j$ ,  $j \in \{1, \dots, k\} \setminus \{i\}$ . The *initial global state* is  $(\emptyset, \emptyset, \emptyset)$ .

The machine  $H_{II}$  also keeps a table in which it remembers handles of the names of honest and dishonest principals along with their encryption/decryption/signing/verification keys (in case of honest principals) and encryption/verification keys (in case of dishonest principals). It also keeps a set of known handles to payloads and nonces. The table and the set are updated in the obvious way; we hence will not further describe it but simply assume that  $H_{II}$  knows the names and keys of all honest and dishonest principals as well as the (handles of) payloads and nonces which occurred so far in the protocol run.

We now describe how global states evolve in  $H_{II}$  in terms of transitions. In what follows, we often do not distinguish between payloads and their handles since  $H_{II}$  can retrieve the payloads from  $M^{\text{real}}$  by supplying the handle. For example, we do not distinguish between the name of an agent (represented as payload data) and the handle to this name.

Following the prevalent static corruption model for Dolev-Yao models, the adversary can corrupt parties only at the beginning of the execution. This is captured by the adversary sending a message (a bit string) of the form  $(\text{corrupt}, a_1, \dots, a_l, g_1, \dots, g_l, h_1, \dots, h_l)$  to  $M^{\text{real}}$  where  $a_i$  are names of principals, and  $g_i$  and  $h_i$  are their encryption and verifica-

tion keys, respectively, provided by A. This corruption message is forwarded by  $M^{\text{real}}$  in terms of a handle to  $H_{\Pi}$  such that  $H_{\Pi}$  can go through this list (using the local commands of  $M^{\text{real}}$ ). As a result,  $H_{\Pi}$  will change the global state as follows:

*Corrupt message (from A via  $M_u^{\text{real}}$ ):*  $(\emptyset, \emptyset, \emptyset) \xrightarrow{(\text{corrupt}, a_1, \dots, a_l, g_1, \dots, g_l, h_1, \dots, h_l)} (\emptyset, \emptyset, \varphi')$  where  $l \geq 0$ . When  $H_{\Pi}$  receives a handle on the corruption message,  $H_{\Pi}$  tests if all  $a_i$  are payloads (interpreted as names of agents), all  $g_i$  are handle to encryption keys and all  $h_i$  are handles to verification keys. Otherwise, the execution is aborted. Now, the knowledge of the adversary recorded by  $H_{\Pi}$  is

$$\varphi' := \{a_i, \text{ek}(a_i), \text{dk}(a_i), \text{sk}(a_i), \text{vk}(a_i) \mid 1 \leq i \leq l\}.$$

Strictly speaking, for the definition of concrete executions,  $H_{\Pi}$  does not have to keep track of the knowledge of the adversary. However, its track record will be sufficient for our results.

The following two commands for creating new sessions and sending messages from A (via  $M^{\text{real}}$ ) may occur in arbitrary number and order.

*Initiate new session (from A via  $M_u^{\text{real}}$ ):*  $(\text{Sld}, f, \varphi) \xrightarrow{(\text{new}, i, a_1, \dots, a_k)} (\text{Sld}', f', \varphi')$  where  $\text{Sld}'$  and  $f'$  are constructed as follows: Let  $\text{sid} := |\text{Sld}| + 1$  be the new session identifier and  $\text{Sld}' := \text{Sld} \cup \{\text{sid}\}$ . Using  $M^{\text{real}}$ , create new encryption and signature pairs  $\text{ek}(a_i), \text{dk}(a_i), \text{sk}(a_i), \text{vk}(a_i)$  for all honest participants  $a_i$  that do not yet have one. Also, using  $M^{\text{real}}$ , create new nonces for all variables  $N_j$  occurring in  $\Pi(i)$  and create a handle to the payload  $\text{sid}$ . Define  $f'(\text{sid}') := f(\text{sid}')$  for each  $\text{sid}' \in \text{Sld}$ , and  $f(\text{sid}) := (i, \nu, \varepsilon, (a_1, \dots, a_k))$ , where  $\varepsilon$  is the root of the role tree, and  $\nu$  maps every  $A_j$  in  $\Pi(i)$  to  $a_j$  and every  $N_j$  occurring in  $\Pi(i)$  to the (handle of the) corresponding nonce. Let  $\varphi' := \varphi \cup \{\text{sid}\} \cup \{\text{ek}(a_j), \text{vk}(a_j) \mid j = 1, \dots, k\}$ . Using  $M^{\text{real}}$ , create a list containing  $\text{sid}$  and the created encryption and verification keys, and send this list (using the send command) via  $M^{\text{real}}$  to the adversary.

*Send message (from A via  $M_u^{\text{real}}$ ):*  $(\text{Sld}, f, \varphi) \xrightarrow{(\text{send}, \text{sid}, m)} (\text{Sld}, f', \varphi')$ : Suppose  $f(\text{sid}) = (i, \nu, p, (a_1, \dots, a_k))$  and let  $(l_1, r_1), \dots, (l_h, r_h)$  be the labels of edges leaving node  $p$ , in the order given. Machine  $H_{\Pi}$  now tries to parse  $m$  according to  $\nu(l_j)$  starting with  $\nu(l_1)$ , then continuing with  $\nu(l_2)$ , and so on, until the parsing is successful. If the parsing is not successful for every  $\nu(l_j)$ , the local state remains unchanged, and hence, the global state does not change either.

The parsing of  $m$  according to  $l := \nu(l_j)$  is performed by  $H_{\Pi}$  inductively on the structure of  $l$ . The parsing updates  $\nu$  since some variables not in the domain of the current  $\nu$ , may now be assigned some values. Also,  $H_{\Pi}$  keeps track of new payloads and nonces created by the adversary. For this, a set  $\varphi_{\text{new}}$  is maintained which at the beginning of the parsing is set to the empty set. Now, the parsing is performed by  $H_{\Pi}$  as follows: First, check whether  $m$  and  $l$  have the same type (by asking  $M^{\text{real}}$  for the type of  $m$  and then checking whether it corresponds to  $l$ ). If the types differ, then stop. Otherwise continue as follows:

- If  $l$  is a handle to a payload or a nonce, then check whether  $l = m$ . (Note that the same payloads/nonces get the same handles in  $M^{\text{real}}$ . Here we use that  $H_{\Pi}$  only employs one

machine  $M_u^{\text{real}}$  for some  $u \in \{1, \dots, n\}$ . We emphasize that this is not an idealization. Whether it is checked if bit strings or handles coincide does not make a difference.)

- If  $l \in \{0, 1\}^*$  is a payload, then retrieve the payload of  $m$  and check whether it coincides with  $l$ .
- If  $l \in X.n$ , then check whether  $m$  is a handle to a nonce. If not, then stop. Otherwise, extend  $\nu$  by mapping  $l$  to  $m$ . If  $m$  has not occurred before (i.e.,  $m$  is a handle to a new payload or nonce that the adversary generated), then add  $m$  to  $\varphi_{\text{new}}$ .
- If  $l \in X.d$ , then check whether  $m$  is a handle to a payload, and continue similarly to the previous case.
- If  $l = \langle t_1, t_2 \rangle$ , then recursively parse the first component of  $m$  according to  $t_1$  and  $\nu$ , and then the second component according to  $t_2$  and (the possibly updated)  $\nu$ .
- If  $l = E_{\text{ek}(a_i)}(t)$ , then decrypt  $m$  with  $\text{dk}(a_i)$ . If this fails, then stop. Otherwise, parse the resulting plaintext (given as a handle), according to  $t$ .
- If  $l$  is a signature term, then proceed analogously to the case of encryption.

If the parsing of  $m$  according to  $l$  is successful, we say that  $m$  and  $l$  match and call the resulting substitution (the updated  $\nu$ ), the matching function. Now, let  $h$  be minimal such that  $m$  matches with  $\nu(l_h)$  and let  $\theta$  be the resulting matching function. (If such an  $i$  does not exist, then, as mentioned, the global state remains unchanged.)

Next,  $H_{II}$  constructs, using  $M^{\text{real}}$ , the output message according to  $r := \theta(r_h)$ . The result is a handle to this message in  $M^{\text{real}}$ . The construction is carried out inductively on the structure of  $r$  as follows. Note that  $r$  does not contain variables since all variables are substituted with handles by  $\theta$ .

- If  $r$  is a handle, then return this handle.
- If  $r \in \{0, 1\}^*$  is a payload, then create a handle to this payload, and then return the handle.
- If  $r$  is a pair, then recursively, construct messages for the two components. With the resulting handles, retrieve a handle to the pair from  $M^{\text{real}}$ .
- If  $r = E_{\text{ek}(a_j)}(t)$ , then recursively construct a message for  $t$ . With the resulting handle and the handle to  $\text{ek}(a_j)$ , retrieve a handle from  $M^{\text{real}}$  to the corresponding ciphertext and return this handle.
- If  $r = \text{Sig}_{\text{vk}(a_i)}(t)$ , then proceed analogously to the previous case; however, to generate the signature the handle to  $\text{sk}(a_i)$  is used.

Let  $m^{\text{hnd}}$  denote the handle to the output message. Also, let  $r'$  be obtained from  $r$  by adding as the label for the random coins in encryption and signature subterms of  $r$ , the handle (together with  $a_i$ ) obtained when creating the corresponding messages. The motivation for this is that different messages stored in  $M^{\text{real}}$  get different handles, and hence, these handles can be used as labels for random coins. For example, if  $r = \langle a_i, E_{\text{ek}(a_j)}(a_i) \rangle$ , then  $r = \langle a_i, E_{\text{ek}(a_j)}^{(a_i, e^{\text{hnd}})}(a_i) \rangle$  where  $e^{\text{hnd}}$  is the handle corresponding to the message created according to  $E_{\text{ek}(a_j)}(a_i)$ .

Now,  $H_{II}$  first updates the global state and then sends the message corresponding to  $m^{\text{hnd}}$  to the adversary.

The global state is updated as follows:  $\text{Sld}$  remains unchanged;  $f'$  coincides with  $f$  on all  $\text{Sld} \setminus \{\text{sid}\}$  and  $f'(\text{sid}) = (i, \theta, ph, (a_1, \dots, a_k))$  where  $ph$  is the  $h$ th successor of  $p$  in  $\Pi(i)$ ;  $\varphi' = \varphi \cup \varphi_{\text{new}} \cup \{r'\}$ .

Later we will modify the definition of  $H_{\Pi}$  as follows, where the modified  $H_{\Pi}$  will be referred to by  $H'_{\Pi}$ :  $H'_{\Pi}$  tests whether  $t_m = \theta(l)$ , the term corresponding to  $m$ , is deducible from the adversary's current knowledge, i.e., if  $\varphi \cup \varphi_{new} \vdash t_m$ . If not,  $H'_{\Pi}$  stops the execution before updating the global state and sending  $\text{send}(m^{\text{hnd}})$ , and outputs failure.

We now define traces of  $\Pi$  when executed with the adversary  $A$ . Our definition is based on executions of the system  $H_{\Pi} \parallel M^{\text{real}} \parallel A$ .

**Definition 1 (Traces).** *A trace of  $\Pi$  when executed with the adversary  $A$  is a sequence  $g_0 \xrightarrow{C_1} g_1 \xrightarrow{C_2} g_2 \xrightarrow{C_3} \dots \xrightarrow{C_n} g_n$  of transitions  $g_i \xrightarrow{C_{i+1}} g_{i+1}$  as defined above for  $H_{\Pi}$  obtained by executing the system  $H_{\Pi} \parallel M^{\text{real}} \parallel A$ . The  $C_i$  are the corrupt, new, and send commands and  $g_0 = (\emptyset, \emptyset, \emptyset)$  is the initial global state. A send transition only belongs to the trace if  $H_{\Pi}$  successfully parsed the input message.*

## 5 Simulatability Soundness implies Mapping Soundness

Mapping soundness is established in the following style: One defines *concrete protocol traces* where several instances of the protocol run along with an adversary, a polynomial-time machine, who controls the network. Messages are bit strings and the cryptographic operations are carried out by cryptographic algorithms. This corresponds to runs of the system  $H_{\Pi} \parallel M^{\text{real}} \parallel A$ . In addition, one defines *symbolic protocol traces* where messages are Dolev-Yao terms. If the adversary is restricted to only derive new messages from a given set of messages by applying Dolev-Yao derivation rules (see Section 2), then these symbolic protocol traces are called *Dolev-Yao traces*. Now one constructs a mapping from bit strings to terms such that applying the mapping to an arbitrary trace of the concrete cryptographic execution yields a Dolev-Yao trace: Different payloads and nonces are mapped to different constants, encryption/decryption/verification/signing keys are represented by  $\text{ek}(a)$ ,  $\text{dk}(a)$ ,  $\text{vk}(a)$ , and  $\text{sk}(a)$  where  $a$  is the constant representing the name of an agent. Pairings, ciphertexts, and signatures are represented by the corresponding Dolev-Yao terms. Given such a mapping, it is then shown that the resulting symbolic protocol trace constitutes a Dolev-Yao trace with overwhelming probability. In other words, this shows that in the setting considered the polynomial-time adversary is not more powerful than the Dolev-Yao intruder.

In this section we show that mapping soundness is implied by simulatability soundness, i.e., by results that prove cryptographic implementations as secure as Dolev-Yao style abstractions in the sense of BRSIM/UC.

Before we can state and prove our result, let us make the following observation about  $H_{\Pi} \parallel M^{\text{real}} \parallel A$ . On the one hand, as explained in Section 4, this system describes concrete protocol executions: The different instances of the protocol only communicate over the network and this network is controlled by the adversary, who is a ppt Turing machine. The messages exchanged are bit strings and are computed using cryptographic algorithms. On the other hand,  $M^{\text{real}}$  provides an abstract interface to  $H_{\Pi}$  in the sense that  $H_{\Pi}$  does not obtain bit strings from  $M^{\text{real}}$  (except for payloads), but only handles to bit strings stored in  $M^{\text{real}}$ . In fact,  $M^{\text{real}}$  realizes the mapping from bit strings to handles, and these handles can be interpreted as Dolev-Yao terms: A handle to a payload/nonce

can be interpreted as a constant representing this payload/nonce. A handle to an encryption/decryption/verification/signing key of an agent  $a$  can be interpreted as the ground term  $\text{ek}(a)$ ,  $\text{dk}(a)$ ,  $\text{vk}(a)$ , and  $\text{sk}(a)$ , respectively. Similarly, handles to pairs, ciphertexts, and signatures, can be interpreted as Dolev-Yao terms representing these objects. We note that since all handles are maintained in one machine  $M_u^{\text{real}}$  for some  $u \in \{1, \dots, n\}$ , different payloads/nonces/etc. are referred to by different handles. Hence, the mapping from bit strings to Dolev-Yao terms, and with this, the translation of concrete protocol traces to symbolic traces is performed by  $M^{\text{real}}$ . In other words, the interface interactions  $H_\Pi$  conducts with honest users already constitute a symbolic trace corresponding to a concrete trace. Hence it turns out not to be necessary anymore to define this translation. This might be surprising, since a natural intuition suggests that in simulatability soundness results this translation is performed by the simulator. It is important to note that while  $M^{\text{real}}$  implicitly provides a mapping from concrete traces to symbolic traces, this does not necessarily mean that the latter trace is a Dolev-Yao trace. Our main result essentially shows that from simulatability soundness it follows that the symbolic trace derived from the mapping constitutes a Dolev-Yao trace with overwhelming probability, which is exactly what mapping soundness intends to establish. The following definition captures the notion of a Dolev-Yao trace.

**Definition 2 (Dolev-Yao Traces).** *A trace*

$$(\text{Sld}_0, f_0, \varphi_0) \xrightarrow{C_1} (\text{Sld}_1, f_1, \varphi_1) \xrightarrow{C_2} \dots \xrightarrow{C_r} (\text{Sld}_r, f_r, \varphi_r)$$

is called a Dolev-Yao trace if and only if the following holds: For all  $i$  such that  $C_i$  is of the form  $(\text{send}, \text{sid}_i, m_i)$  we have that  $\varphi_{i-1} \cup (\varphi_{i-1})_{\text{new}} \vdash^{\text{adv}} t_{m_i}$  where  $t_{m_i}$  is the Dolev-Yao term corresponding to  $m_i$  and  $(\varphi_{i-1})_{\text{new}}$  contains the new constants in  $t_{m_i}$  generated by the adversary (see the definition of  $H_\Pi$  in Section 4).

Now, we are ready to prove our main result, namely that simulatability soundness implies mapping soundness. This result is based on two assumptions. The first assumption is that  $M^{\text{real}} \leq^{\text{BRSIM}} M^{\text{ideal}}$ , i.e., the cryptographic implementation is as secure as the Dolev-Yao abstraction in the sense of BRSIM/UC. This exactly formalizes simulatability soundness. The second assumption is that if protocols are executed based on  $M^{\text{ideal}}$  instead of  $M^{\text{real}}$ , then the resulting traces are Dolev-Yao traces. More precisely, for every ideal adversary  $A'$  (which may be a composition of a simulator and a real adversary) all traces of  $H_\Pi \parallel M^{\text{ideal}} \parallel A'$  are Dolev-Yao traces, which exactly reflects the intuition and purpose behind the Dolev-Yao abstraction  $M^{\text{ideal}}$ .

**Theorem 1 (Simulatability Soundness implies Mapping Soundness).** *Let  $\Pi$  be a protocol. Assume the following two properties about  $M^{\text{real}}$  and  $M^{\text{ideal}}$ :*

1.  $M^{\text{real}} \leq^{\text{BRSIM}} M^{\text{ideal}}$ .
2. For every ideal adversary  $A'$ , all traces of  $H_\Pi \parallel M^{\text{ideal}} \parallel A'$  are Dolev-Yao traces.

*Then, for all (real) adversaries  $A$ , the probability that a trace of  $H_\Pi \parallel M^{\text{real}} \parallel A$  is a Dolev-Yao trace is overwhelming.*

*Proof.* First, it follows from existing results that the relation  $\varphi \vdash m$  given a set  $\varphi$  of Dolev-Yao messages and a Dolev-Yao message  $m$  can be decided in polynomial time in the size of  $\varphi$  and  $m$  (see, e.g., [17]).

Now, consider  $H'_H$  as defined in Section 4. By construction,  $H'_H$  behaves exactly as  $H_H$  except that it checks whether the Dolev-Yao term corresponding to the received message can be deduced by the current intruder knowledge plus the new handles (corresponding to payloads and nonces generated by the adversary) in the received message. If this is not the case, then  $H'_H$  outputs **failure**. Since the relation  $\vdash$  can be decided in polynomial time and  $H_H$  is a polynomial time machine,  $H'_H$  runs in polynomial time as well. Furthermore, by definition of Dolev-Yao traces it follows that the probability that a trace of  $H_H \parallel M^{\text{real}} \parallel A$  is a non-Dolev-Yao trace is exactly the probability that  $H'_H$  outputs **failure** in a run of  $H'_H \parallel M^{\text{real}} \parallel A$ . For proving the theorem, it hence remains to show that the latter probability is negligible for every  $A$ .

By the first assumption in the theorem, we know that  $M^{\text{real}} \leq^{BRSIM} M^{\text{ideal}}$ . Thus, there exists a simulator  $S$  such that for every  $A$  the view of  $H'_H$  in  $H'_H \parallel M^{\text{real}} \parallel A$  is indistinguishable from the view of  $H'_H$  in  $H'_H \parallel M^{\text{ideal}} \parallel S \parallel A$ . We can consider  $S \parallel A$  to be an ideal adversary  $A'$ , i.e.,  $A' = S \parallel A$ . Now, by the second assumption in the theorem and since  $H'_H$  exactly behaves as  $H_H$  on Dolev-Yao traces, we can conclude that  $H'_H$  never outputs **failure** in a run of  $H'_H \parallel M^{\text{ideal}} \parallel A'$ . Finally, it follows that the probability that  $H'_H$  outputs **failure** in a run of  $H'_H \parallel M^{\text{real}} \parallel A$  is negligible as otherwise the views of  $H'_H$  in  $H'_H \parallel M^{\text{real}} \parallel A$  and  $H'_H \parallel M^{\text{ideal}} \parallel A'$  could be distinguished.  $\square$

We emphasize that the argument in the proof of Theorem 1 is quite generic. As mentioned, the two assumptions stated in the theorem are what simulatability soundness should provide in any case. In addition to these two assumptions, we only used the fact that  $H_H$  can be extended in such a way that it stops whenever it encounters a non-Dolev-Yao trace. Only the definition of  $H_H$  and the extension of  $H_H$  depend on the specific cryptographic primitives and the class of protocols considered. The rest of the argument is completely independent of these details and it resembles property preservation theorems for simulatability [5, 8]. Therefore, the above theorem should also hold for larger classes of cryptographic primitives and protocols.

We conclude this section by showing that the two assumptions in Theorem 1 are met by a concrete cryptographic implementation and its Dolev-Yao abstraction. More precisely, we consider the cryptographic implementation and its Dolev-Yao abstraction presented in [9]. Let us refer to the cryptographic implementation by  $M_{\text{BPW}}^{\text{real}}$  and the Dolev-Yao abstraction by  $M_{\text{BPW}}^{\text{ideal}}$ , called the *BPW model* henceforth. The I/O interface of  $M_{\text{BPW}}^{\text{real}}$  (and hence, that of  $M_{\text{BPW}}^{\text{ideal}}$ ) offers all commands also offered by  $M^{\text{real}}$ , except that in [9] lists instead of pairs are considered. The first assumption of Theorem 1, now with respect to  $M_{\text{BPW}}^{\text{real}}$  and  $M_{\text{BPW}}^{\text{ideal}}$ , is satisfied by the main result proved in [9]. The second assumption of Theorem 1 easily follows from the definition of the BPW model  $M_{\text{BPW}}^{\text{ideal}}$ : The machine  $M_{\text{BPW}}^{\text{ideal}}$  internally does not store bit strings but Dolev-Yao terms. If a (handle to a) term is sent by an honest user to the network (and hence, the ideal adversary), then at the network interface only a (potentially new) handle to the term is given to the ideal adversary. The ideal adversary can now only parse and manipulate this and other terms to which he has obtained a handle via the network interface of  $M_{\text{BPW}}^{\text{ideal}}$  and this interface only allows for

commands that correspond to the Dolev-Yao derivation rules given in Section 2. From this observation the second assumption of Theorem 1 easily follows. We summarize this in the following proposition.

**Proposition 1.** *The cryptographic implementation and Dolev-Yao abstraction presented in [9] satisfy the assumptions made in Theorem 1.*

## 6 Conclusion

In this paper, the two notions of *simulatability soundness* and *mapping soundness* for bridging the long-standing gap between Dolev-Yao models and cryptographic realizations with their computational security definitions are related for the first time. Our main result is that simulatability soundness entails mapping soundness provided that both approaches use the same cryptographic implementation. Interestingly, this result does not depend on details of the simulator, which translates between cryptographic implementations and their Dolev-Yao abstractions in simulatability soundness. The argument of our proof is generic in the sense that it can easily be extended to additional classes of cryptographic primitives and protocols beyond the ones considered in this paper.

We stress that requiring the same cryptographic implementations for both simulatability soundness and mapping soundness means that existing results on simulatability soundness do not necessarily fully supersede existing results on mapping soundness: the former results may for instance require stronger assumptions on the security of cryptographic primitives, specific techniques from robust protocol design such as explicit type tags, additional randomization, etc. in order to establish simulatability between the cryptographic implementation and its Dolev-Yao abstraction. However, we believe that it is fair to say that future research may well concentrate on simulatability soundness whenever applicable, and resort to mapping soundness in those cases where simulatability soundness constitutes too strong a notion.

## References

1. M. Abadi and J. Jürjens. Formal eavesdropping and its computational interpretation. In *Proc. 4th International Symposium on Theoretical Aspects of Computer Software (TACS)*, pages 82–94, 2001.
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, I. Cervesato, A. D. Jaggard, A. Scedrov, and J.-K. Tsay. Cryptographically sound security proofs for basic and public-key Kerberos. In *Proc. 11th European Symposium on Research in Computer Security (ESORICS)*. Springer, 2006.
4. M. Backes and M. Dürmuth. A cryptographically sound Dolev-Yao style security proof of an electronic payment system. In *Proc. 18th IEEE Computer Security Foundations Workshop (CSFW)*, pages 78–93, 2005.
5. 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.
6. M. Backes and B. Pfitzmann. A cryptographically sound security proof of the Needham-Schroeder-Lowe public-key protocol. *IEEE Journal on Selected Areas in Communications*, 22(10):2075–2086, 2004.

7. M. Backes and B. Pfitzmann. Symmetric encryption in a simulatable Dolev-Yao style cryptographic library. In *Proc. 17th IEEE Computer Security Foundations Workshop (CSFW)*, pages 204–218, 2004.
8. M. Backes and B. Pfitzmann. Relating symbolic and cryptographic secrecy. *IEEE Transactions on Dependable and Secure Computing*, 2(2):109–123, 2005.
9. M. Backes, B. Pfitzmann, and M. Waidner. A composable cryptographic library with nested operations (extended abstract). In *Proc. 10th ACM Conference on Computer and Communications Security*, pages 220–230, 2003. Full version in IACR Cryptology ePrint Archive 2003/015, Jan. 2003, <http://eprint.iacr.org/>.
10. M. Backes, B. Pfitzmann, and M. Waidner. Symmetric authentication within a simulatable cryptographic library. In *Proc. 8th European Symposium on Research in Computer Security (ESORICS)*, volume 2808 of *Lecture Notes in Computer Science*, pages 271–290. Springer, 2003.
11. M. Backes, B. Pfitzmann, and M. Waidner. Secure asynchronous reactive systems. IACR Cryptology ePrint Archive 2004/082, Mar. 2004. To appear in *Information and Computation*.
12. D. Beaver. Secure multiparty protocols and zero knowledge proof systems tolerating a faulty minority. *Journal of Cryptology*, 4(2):75–122, 1991.
13. B. Blanchet. A computationally sound mechanized prover for security protocols. In *Proc. 27th IEEE Symposium on Security & Privacy*, pages 140–154, 2006.
14. R. Canetti. Security and composition of multiparty cryptographic protocols. *Journal of Cryptology*, 3(1):143–202, 2000.
15. R. Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *Proc. 42nd IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 136–145, 2001. Extended version in Cryptology ePrint Archive, Report 2000/67, <http://eprint.iacr.org/>.
16. R. Canetti and J. Herzog. Universally composable symbolic analysis of mutual authentication and key exchange protocols. In *Proc. 3rd Theory of Cryptography Conference (TCC)*, volume 3876 of *Lecture Notes in Computer Science*, pages 380–403. Springer, 2006.
17. Y. Chevalier, R. Küsters, M. Rusinowitch, and M. Turuani. An NP decision procedure for protocol insecurity with XOR. In *Proc. 18th Annual IEEE Symposium on Logic in Computer Science (LICS)*, pages 261–270, 2003.
18. V. Cortier, S. Kremer, R. Küsters, and B. Warinschi. Computationally Sound Symbolic Secrecy in the Presence of Hash Functions. In *Proc. of 26th Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2006)*, Lecture Notes in Computer Science. Springer, 2006. To appear.
19. V. Cortier and B. Warinschi. Computationally sound, automated proofs for security protocols. In *Proc. 14th European Symposium on Programming (ESOP)*, pages 157–171, 2005.
20. A. Datta, A. Derek, J. Mitchell, V. Shmatikov, and M. Turuani. Probabilistic polynomial-time semantics for a protocol security logic. In *Proc. 32nd International Colloquium on Automata, Languages and Programming (ICALP)*, volume 3580 of *Lecture Notes in Computer Science*, pages 16–29. Springer, 2005.
21. D. Dolev and A. C. Yao. On the security of public key protocols. *IEEE Transactions on Information Theory*, 29(2):198–208, 1983.
22. O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game – or – a completeness theorem for protocols with honest majority. In *Proc. 19th Annual ACM Symposium on Theory of Computing (STOC)*, pages 218–229, 1987.
23. S. Goldwasser and L. Levin. Fair computation of general functions in presence of immoral majority. In *Advances in Cryptology: CRYPTO '90*, volume 537 of *Lecture Notes in Computer Science*, pages 77–93. Springer, 1990.
24. R. Impagliazzo and B. M. Kapron. Logics for reasoning about cryptographic constructions. In *Proc. 44th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 372–381, 2003.
25. R. Küsters. Simulation-based security with inexhaustible interactive turing machines. In *Proc. of 19th IEEE Computer Security Foundations Workshop (CSFW)*, pages 309–320, 2006.
26. P. Laud. Semantics and program analysis of computationally secure information flow. In *Proc. 10th European Symposium on Programming (ESOP)*, pages 77–91, 2001.
27. P. Laud. Symmetric encryption in automatic analyses for confidentiality against active adversaries. In *Proc. 25th IEEE Symposium on Security & Privacy*, pages 71–85, 2004.

28. 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.
29. S. Micali and P. Rogaway. Secure computation. In *Advances in Cryptology: CRYPTO '91*, volume 576 of *Lecture Notes in Computer Science*, pages 392–404. Springer, 1991.
30. D. Micciancio and B. Warinschi. Soundness of formal encryption in the presence of active adversaries. In *Proc. 1st Theory of Cryptography Conference (TCC)*, volume 2951 of *Lecture Notes in Computer Science*, pages 133–151. Springer, 2004.
31. J. Mitchell, M. Mitchell, and A. Scedrov. A linguistic characterization of bounded oracle computation and probabilistic polynomial time. In *Proc. 39th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 725–733, 1998.
32. J. Mitchell, M. Mitchell, A. Scedrov, and V. Teague. A probabilistic polynomial-time process calculus for analysis of cryptographic protocols (preliminary report). *Electronic Notes in Theoretical Computer Science*, 47:1–31, 2001.
33. 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. Extended version (with Matthias Schunter) IBM Research Report RZ 3206, May 2000, [http://www.semper.org/sirene/publ/PfSW1\\_00ReactSimulIBM.ps.gz](http://www.semper.org/sirene/publ/PfSW1_00ReactSimulIBM.ps.gz).
34. 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. Extended version of the model (with Michael Backes) IACR Cryptology ePrint Archive 2004/082, <http://eprint.iacr.org/>.
35. C. Sprenger, M. Backes, D. Basin, B. Pfitzmann, and M. Waidner. Cryptographically sound theorem proving. In *Proc. 19th IEEE Computer Security Foundations Workshop (CSFW)*, pages 153–166, 2006.
36. A. C. Yao. Theory and applications of trapdoor functions. In *Proc. 23rd IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 80–91, 1982.