A Cryptographically Sound Dolev-Yao Style Security Proof of the Otway-Rees Protocol

Michael Backes

IBM Zurich Research Laboratory, Switzerland
mbc@zurich.ibm.com

Abstract. We present the first cryptographically sound security proof of the well-known Otway-Rees protocol. More precisely, we show that the protocol is secure against arbitrary active attacks including concurrent protocol runs if it is implemented using provably secure cryptographic primitives. Although we achieve security under cryptographic definitions, our proof does not have to deal with probabilistic aspects of cryptography and is hence in the scope of current proof tools. The reason is that we exploit a recently proposed ideal cryptographic library, which has a provably secure cryptographic implementation. Together with composition and preservation theorems of the underlying model, this allows us to perform the actual proof effort in a deterministic setting corresponding to a slightly extended Dolev-Yao model. Besides establishing the cryptographic security of the Otway-Rees protocol, our result also exemplifies the potential of this cryptographic library. We hope that it paves the way for cryptographically sound verification of security protocols by means of formal proof tools.

1 Introduction

Many practically relevant cryptographic protocols like SSL/TLS, IPSec, or SET use cryptographic primitives like signature schemes or encryption in a black-box way, while adding many non-cryptographic features. Vulnerabilities have accompanied the design of such protocols ever since early authentication protocols like Needham-Schroeder [34, 15], over carefully designed de-facto standards like SSL and PKCS [40, 13], up to current widely deployed products like Microsoft Passport [17]. However, proving the security of such protocols has been a very unsatisfactory task for a long time.

One way to conduct such proofs is the cryptographic approach, whose security definitions are based on complexity theory, e.g., [19, 18, 20, 10]. The security of a cryptographic protocol is proved by reduction, i.e., by showing that breaking the protocol implies breaking one of the underlying cryptographic primitives with respect to its cryptographic definition. This approach captures a very comprehensive adversary model and allows for mathematically rigorous and precise proofs. However, because of probabilism and complexity-theoretic restrictions, these proofs have to be done by hand so far, which yields proofs with faults and imperfections. Moreover, such proofs rapidly become too complex for larger protocols.

The alternative is the formal-methods approach, which is concerned with the automation of proofs using model checkers and theorem provers. As these tools currently cannot deal with cryptographic details like error probabilities and computational restrictions, abstractions of cryptography are used. They are almost always based on the

P. Samarati et al. (Eds.): ESORICS 2004, LNCS 3193, pp. 89–108, 2004.

so-called Dolev-Yao model [16]. This model simplifies proofs of larger protocols considerably and has given rise to a large body of literature on analyzing the security of protocols using various techniques for formal verification, e.g., [31, 29, 25, 14, 37, 1].

Among the protocols typically analyzed in the Dolev-Yao model, the Otway-Rees protocol [35], which aims at establishing a shared key between two users by means of a trusted third party, stands out as one of the most prominent protocols. It has been extensively studies in the past, e.g., in [36, 24, 37], and various new approaches and formal proof tools for the analysis of security protocols were validated by showing that they can prove the protocol in the Dolev-Yao model (respectively that they can find the well-known type-flaw attack if the underlying model does not provide sufficient typing itself; the model that our proof is based upon excludes this attack). However, all existing proofs of security of the Otway-Rees protocol are restricted to the Dolev-Yao model, i.e., no theorem exists which allows for carrying over the results of an existing proof to the cryptographic approach with its much more comprehensive adversary. Thus, despite of the tremendous amount of research dedicated to the Otway-Rees protocol, it is still an open question whether an actual implementation based on provably secure cryptographic primitives is secure under cryptographic security definitions. We close this gap by providing the first security proof of the Otway-Rees protocol in the cryptographic approach. We show that the protocol is secure against arbitrary active attacks if the Dolev-Yao-based abstraction of symmetric encryption is implemented using a symmetric encryption scheme that is secure against chosen-ciphertext attacks and additionally ensures integrity of ciphertexts. This is the standard security definition of authenticated symmetric encryption schemes [12, 11], and efficient symmetric encryptions schemes provably secure in this sense exist under reasonable assumptions [11, 39].

Obviously, establishing a proof in the cryptographic approach presupposes dealing with the mentioned cryptographic details, hence one naturally assumes that our proof heavily relies on complexity theory and is far out of scope of current proof tools. However, our proof is not performed from scratch in the cryptographic setting, but based on a recently proposed cryptographic library [8, 9, 7], which provides cryptographically faithful, deterministic abstractions of cryptographic primitives, i.e., the abstractions can be securely implemented using actual cryptography. Moreover, the library allows for nesting the abstractions in an arbitrary way, quite similar to the original Dolev-Yao model. While this was shown for public-key encryption and digital signatures in [8] and subsequently extended with message authentication codes in [9], the most recent extension of the library further incorporated symmetric encryption [7] which constitutes the most commonly used cryptographic primitive in the typical proofs with Dolev-Yao models, and also serves as the central primitive for expressing and analyzing the Otway-Rees protocol. However, as shown in [7], there are intrinsic difficulties in providing a sound abstraction from symmetric encryption in the strong sense of security used in [8]. Very roughly, a sound Dolev-Yao-style abstraction of symmetric encryption can only be established if a so-called *commitment problem* does not occur, which means that whenever a key that is not known to the adversary is used for encryption by an honest user then this key will never be revealed to the adversary. We will elaborate on the origin of this problem in more detail in the paper. While [7] discusses several solutions to this problem, the one actually taken is to leave it to the surrounding protocol to guarantee that the commitment problem does not occur, i.e., if a protocol that uses symmetric encryption should be faithfully analyzed, it additionally has to be shown that the protocol guarantees that keys are no longer sent in a form that might make them known to the adversary once an honest participant has started using them. Our proof shows that this is a manageable task that can easily be incorporated in the overall security proof without imposing a major additional burden on the prover.

Once we have shown that the Otway-Rees protocol does not raise the commitment problem, it is sufficient to prove the security of the Otway-Rees protocol based on the deterministic abstractions; then the result automatically carries over to the cryptographic setting. As the proof is deterministic and rigorous, it should be easily expressible in formal proof tools, in particular theorem provers. Even done by hand, our proof is much less prone to error than a reduction proof conducted from scratch in the cryptographic approach. We also want to point out that our result not only provides the up-to-now missing cryptographic security proof of the Otway-Rees protocol, but also exemplifies the usefulness of the cryptographic library [8] and their extensions [9, 7] for the cryptographically sound verification of cryptographic protocols.

Further Related Work. Cryptographic underpinnings of a Dolev-Yao model were first addressed by Abadi and Rogaway in [3]. However, they only handled passive adversaries and symmetric encryption. The protocol language and security properties handled were extended in [2, 26], but still only for passive adversaries. This excludes most of the typical ways of attacking protocols, e.g., man-in-the-middle attacks and attacks by reusing a message part in a different place or a concurrent protocol run. A full cryptographic justification for a Dolev-Yao model, i.e., for arbitrary active attacks and within arbitrary surrounding interactive protocols, was first given recently in [8] with extensions in [9, 7]. Based on the specific Dolev-Yao model whose soundness was proven in [8], the well-known Needham-Schroeder-Lowe protocol was proved in [6]. Besides the proof that we present in this paper, the proof in [6] is the only Dolev-Yao-style, computationally sound proof that we are aware of. However, it is considerably simpler than the one we present in this work since it only addresses integrity properties whereas our proof additionally establishes confidentiality properties; moreover, the Needham-Schroeder-Lowe protocols does not use symmetric encryption, hence the commitment problem does not occur there which greatly simplifies the proof. Another cryptographically sound proof of this protocol was concurrently developed by Warinschi [41]. The proof is conducted from scratch in the cryptographic approach which takes it out of the scope of formal proof tools.

Laud [27] has recently presented a cryptographic underpinning for a Dolev-Yao model of symmetric encryption under active attacks. His work enjoys a direct connection with a formal proof tool, but it is specific to certain confidentiality properties, restricts the surrounding protocols to straight-line programs in a specific language, and does not address a connection to the remaining primitives of the Dolev-Yao model. Herzog et al. [21, 22] and Micciancio and Warinschi [30] have recently also given a cryptographic underpinning under active attacks. Their results are considerably weaker than the one in [8] since they are specific for public-key encryption; moreover, the former relies on a stronger assumption whereas the latter severely restricts the classes of protocols and protocol properties that can be analyzed using this primitive. Section 6 of [30] further points out several possible extensions of their work which all already exist in the earlier work of [8].

Efforts are also under way to formulate syntactic calculi for dealing with probabilism and polynomial-time considerations, in particular [32, 28, 33, 23] and, as a second step, to encode them into proof tools. However, this approach can not yet handle protocols with any degree of automation. Generally it is complementary to, rather than competing with, the approach of proving simple deterministic abstractions of cryptography and working with those wherever cryptography is only used in a blackbox way.

Outline. Section 2 introduces the notation used in the paper and briefly reviews the aforementioned cryptographic library. Section 3 shows how to model the Otway-Rees protocol based on this library as well as how initially shared keys can be represented in the underlying model. Section 4 contains the security property of the Otway-Rees protocol in the ideal setting, and this property is proven in Section 5. Section 6 shows how to carry these results over to the cryptographic implementation of the protocol. Section 7 concludes.

2 Preliminaries

In this section, we give an overview of the ideal cryptographic library of [8, 9, 7] and briefly sketch its provably secure implementation. We start by introducing the notation used in this paper.

2.1 Notation

We write "$:=$" for deterministic and "$\leftarrow$" for probabilistic assignment. Let $\downarrow$ denote an error element available as an addition to the domains and ranges of all functions and algorithms. The list operation is denoted as $l := (x_1, \ldots, x_j)$, and the arguments are unambiguously retrievable as $l[i]$, with $l[i] = \downarrow$ if $i > j$. A database D is a set of functions, called entries, each over a finite domain called attributes. For an entry $x \in D$, the value at an attribute att is written $x.att$. For a predicate $pred$ involving attributes, $D[pred]$ means the subset of entries whose attributes fulfill $pred$. If $D[pred]$ contains only one element, we use the same notation for this element.

2.2 Overview of the Ideal and Real Cryptographic Library

The ideal (abstract) cryptographic library of [8, 9, 7] offers its users abstract cryptographic operations, such as commands to encrypt or decrypt a message, to make or test a signature, and to generate a nonce. All these commands have a simple, deterministic semantics. To allow a reactive scenario, this semantics is based on state, e.g., of who already knows which terms; the state is represented as a database. Each entry has a type (e.g., "ciphertext"), and pointers to its arguments (e.g., a key and a message). Further, each entry contains handles for those participants who already know it. A send operation makes an entry known to other participants, i.e., it adds handles to the entry. The ideal cryptographic library does not allow cheating. For instance, if it receives a command to encrypt a message m with a certain key, it simply makes an abstract database entry for the ciphertext. Another user can only ask for decryption of this ciphertext if he has obtained handles to both the ciphertext and the secret key. To allow for the proof of cryptographic faithfulness, the library is based on a detailed model of asynchronous reactive systems introduced in [38] and represented as a deterministic machine $\mathsf{TH}_{\mathcal{H}}$, called *trusted host*. The parameter $\mathcal{H} \subseteq \{1 \ldots, n\}$ denotes the honest participants, where n is a parameter of the library denoting the overall number of participants. Depending on the considered set $\mathcal{H}$, the trusted host offers slightly extended capabilities for the adversary. However, for current purposes, the trusted host can be seen as a slightly modified Dolev-Yao model together with a network and intruder model, similar to "the CSP Dolev-Yao model" or "the inductive-approach Dolev-Yao model".

The real cryptographic library offers its users the same commands as the ideal one, i.e., honest users operate on cryptographic objects via handles. The objects are now real cryptographic keys, ciphertexts, etc., handled by real distributed machines. Sending a term on an insecure channel releases the actual bitstring to the adversary, who can do with it what he likes. The adversary can also insert arbitrary bitstrings on non-authentic channels. The implementation of the commands is based on arbitrary secure encryption and signature systems according to standard cryptographic definitions, with certain additions like type tagging and additional randomizations.

The security proof of [8] states that the real library is *at least as secure* as the ideal library. This is captured using the notion of *reactive simulatability* [38], which states that whatever an adversary can achieve in the real implementation, another adversary can achieve given the ideal library, or otherwise the underlying cryptography can be broken [38]. This is the strongest possible cryptographic relationship between a real and an ideal system. In particular it covers arbitrary active attacks. Moreover, a composition theorem exists in the underlying model [38], which states that one can securely replace the ideal library in larger systems with the real library, i.e., without destroying the already established simulatability relation.

2.3 Detailed Description of the State of the Cryptographic Library

We conclude this section with the rigorous definition of the state of the ideal cryptographic library. A rigorous definition of the commands of the ideal library used for modeling the Otway-Rees protocol and for capturing the slightly extended adversary capabilities can be found in the long version of this paper [4].

The machine $\mathsf{TH}_{\mathcal{H}}$ has ports $\mathsf{in}_u?$ and $\mathsf{out}_u!$ for inputs from and outputs to each user $u \in \mathcal{H}$ and for $u = \mathsf{a}$, denoting the adversary. The notation follows the CSP convention, e.g., the cryptographic library obtains messages at $\mathsf{in}_u?$ that have been output at $\mathsf{in}_u!$. Besides the number n of users, the ideal cryptographic library is parameterized by a tuple L of length functions which are used to calculate the "length" of an abstract entry, corresponding to the length of the corresponding bitstring in the real implementation. Moreover, L contains bounds on the message lengths and the number of accepted inputs at each port. These bounds can be arbitrarily large, but have to be polynomially bounded in the security parameter. Using the notation of [8], the ideal cryptographic library is a *system* $Sys_{n,L}^{\mathsf{cry,id}}$ that consists of several *structures* $(\{\mathsf{TH}_{\mathcal{H}}\}, S_{\mathcal{H}})$, one for each value of the parameter $\mathcal{H}$. Each structure consists of a set of machines, here only containing the single machine $\mathsf{TH}_\mathcal{H}$, and a set $S_\mathcal{H} := \{\mathsf{in}_u?, \mathsf{out}_u! \mid u \in \mathcal{H}\}$ denoting those ports of $\mathsf{TH}_\mathcal{H}$ that the honest users connect to. Formally, we obtain $Sys^{\mathsf{cry,id}}_{n,L} := \{(\{\mathsf{TH}_\mathcal{H}\}, S_\mathcal{H}) \mid \mathcal{H} \subseteq \{1,\ldots,n\}\}$. In the following, we omit the parameters n and L for simplicity[1].

The main data structure of $\mathsf{TH}_\mathcal{H}$ is a database D. The entries of D are abstract representations of the data produced during a system run, together with the information on who knows these data. Each entry in D is of the form (recall the notation in Section 2.1)

$$(ind, type, arg, hnd_{u_1}, \ldots, hnd_{u_m}, hnd_\mathsf{a}, len)$$

where $\mathcal{H} = \{u_1, \ldots, u_m\}$. For each entry $x \in D$:

- $x.ind \in \mathcal{INDS}$, called index, consecutively numbers all entries in D. The set $\mathcal{INDS}$ is isomorphic to $\mathbb{N}$ and is used to distinguish index arguments from others. The index is used as a primary key attribute of the database, i.e., we write $D[i]$ for the selection $D[ind = i]$.
- $x.type \in typeset$ identifies the *type* of x.
- $x.arg = (a_1, a_2, \ldots, a_j)$ is a possibly empty list of arguments. Many values a_i are indices of other entries in D and thus in $\mathcal{INDS}$. We sometimes distinguish them by a superscript "ind".
- $x.hnd_u \in \mathcal{HNDS} \cup \{\downarrow\}$ for $u \in \mathcal{H} \cup \{\mathsf{a}\}$ are handles by which a user or adversary u knows this entry. $x.hnd_u = \downarrow$ means that u does not know this entry. The set $\mathcal{HNDS}$ is yet another set isomorphic to $\mathbb{N}$. We always use a superscript "hnd" for handles.
- $x.len \in \mathbb{N}_0$ denotes the "length" of the entry; it is computed by applying the functions from L.

Initially, D is empty. $\mathsf{TH}_\mathcal{H}$ has a counter $size \in \mathcal{INDS}$ for the current size of D. For the handle attributes, it has counters $curhnd_u$ (current handle) initialized with 0.

3 The Otway-Rees Protocol

The Otway-Rees protocol [35] is a four-step protocol for establishing a shared secret encryption key between two users. The protocol relies on a distinguished trusted third party T, i.e., $\mathsf{T} \notin \{1,\ldots,n\}$, and it is assumed that every user u initially shares a secret key K_{ut} with T. Expressed in the typical protocol notation, the Otway-Rees protocol works as follows[2].

$$\begin{aligned}
&1. \quad u \rightarrow v : M, (N_u, M, u, v)_{K_{ut}}\\
&2. \quad v \rightarrow \mathsf{T} : M, (N_u, M, u, v)_{K_{ut}}, (N_v, M, u, v)_{K_{vt}}\\
&3. \quad \mathsf{T} \rightarrow v : M, (N_u, K_{uv})_{K_{ut}}, (N_v, K_{uv})_{K_{vt}}\\
&4. \quad v \rightarrow u : M, (N_u, K_{uv})_{K_{ut}}.
\end{aligned}$$

[1] Formally, these parameters are thus also parameters of the ideal Otway-Rees system $Sys^{\mathsf{OR,id}}$ that we introduce in Section 3.2.

[2] For simplicity, we omit the explicit inclusion of u and v in the unencrypted part of the first and second message since the cryptographic library already provides the identity of the (claimed) sender of a message, which is sufficient for our purpose.

3.1 Capturing Distributed Keys in the Abstract Library

In order to capture that keys shared between users and the trusted third party have already been generated and distributed, we assume that suitable entries for the keys already exist in the database. We denote the handle of u to the secret key shared with v, where either $u \in \{1, \ldots, n\}$ and $v = \mathsf{T}$ or vice versa, as $skse_{u,v}^{\mathsf{hnd}}$. More formally, we start with an initially empty database D, and for each user $u \in \mathcal{H}$ two entries of the following form are added (the first one being a public-key identifier for the actual secret key as described below in more detail):

$$(ind := pkse_u, type := \mathsf{pkse}, arg := (), len := 0);^3$$

$$(ind := skse_u, type := \mathsf{skse}, arg := (ind - 1),$$
$$hnd_u := skse_{u,\mathsf{T}}^{\mathsf{hnd}}, hnd_{\mathsf{T}} := skse_{\mathsf{T},u}^{\mathsf{hnd}}, len := \mathsf{skse\_len}^*(k)).$$

Here $pkse_u$ and $skse_u$ are two consecutive natural numbers; $\mathsf{skse\_len}^*(k)$ denotes the abstract length of the secret key which will not matter in the following.

The first entry has to be incorporated in order to reflect special capabilities that the adversary may have with respect to symmetric encryption schemes in the real world. For instance it must be possible for an adversary against the ideal library to check whether encryptions have been created with the same secret key since the definition of symmetric encryption schemes does not exclude this and it can hence happen in the real system. For public-key encryption, this was achieved in [8] by tagging ciphertexts with the corresponding public key so that the public keys can be compared. For symmetric encryption, this is not possible as no public key exists, hence this problem is solved by tagging abstract ciphertexts with an otherwise meaningless “public key” solely used as an identifier for the secret key. Note that the argument of a secret key points to its key identifier. In the following, public-key identifiers will not matter any further.

We omit the details of how these entries for user u are added by a command gen_symenc_key, followed by a command send_s for sending the secret key over a secure channel.

3.2 The Otway-Rees Protocol Using the Abstract Library

We now model the Otway-Rees protocol in the framework of [38] and using the ideal cryptographic library.

For each user $u \in \{1, \ldots, n\}$ we define a machine $\mathsf{M}_u^{\mathsf{OR}}$, called a *protocol machine*, which executes the protocol sketched above for participant identity u. It is connected to its user via ports $\mathsf{KS\_out}_u!$, $\mathsf{KS\_in}_u?$ (“KS” for “Key Sharing”) and to the cryptographic library via ports $\mathsf{in}_u!$, $\mathsf{out}_u?$. We further model the trusted third party as a machine $\mathsf{M}_{\mathsf{T}}^{\mathsf{OR}}$. It does not connect to any users and is connected to the cryptographic library via ports $\mathsf{in}_{\mathsf{T}}!$, $\mathsf{out}_{\mathsf{T}}?$. The combination of the protocol machines $\mathsf{M}_u^{\mathsf{OR}}$, the trusted third party $\mathsf{M}_{\mathsf{T}}^{\mathsf{OR}}$, and the trusted host $\mathsf{TH}_{\mathcal{H}}$ is the *ideal Otway-Rees system* $Sys^{\mathsf{OR,id}}$. It is shown in Figure 1; H and A model the arbitrary joint honest users and the adversary, respectively.

[3] Treating public-key identifiers as being of length 0 is a technicality in the proof of [7] and will not matter in the sequel.

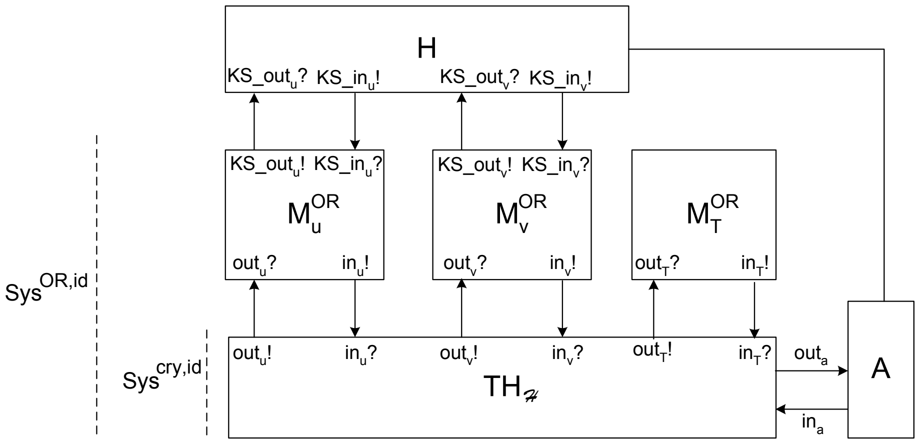

Fig. 1. Overview of the Otway-Rees Ideal System.

Using the notation of [8], we have $Sys^{\mathsf{OR,id}} := \{(\hat{M}_{\mathcal{H}}, S_{\mathcal{H}}) \mid \mathcal{H} \subseteq \{1, \ldots, n\}\}$, cf. the definition of the ideal cryptographic library in Section 2.3, where $\hat{M}_{\mathcal{H}} := \{\mathsf{TH}_{\mathcal{H}}\} \cup \{\mathsf{M}_u^{\mathsf{OR}} \mid u \in \mathcal{H} \cup \{\mathsf{T}\}\}$ and $S_{\mathcal{H}} := \{\mathsf{KS\_in}_u?, \mathsf{KS\_out}_u! \mid u \in \mathcal{H}\}$, i.e., for a given set $\mathcal{H}$ of honest users, only the protocol machines $\mathsf{M}_u^{\mathsf{OR}}$ with $u \in \mathcal{H}$ are actually present in a protocol run. The others are subsumed in the adversary.

The state of the protocol machine $\mathsf{M}_u^{\mathsf{OR}}$ consists of the bitstring u and a set $Nonce_u$ of pairs of the form $(n^{\mathsf{hnd}}, m^{\mathsf{hnd}}, v, j)$, where $n^{\mathsf{hnd}}, m^{\mathsf{hnd}}$ are handles, $v \in \{1, \ldots, n\}$, and $j \in \{1, 2, 3, 4\}$. Intuitively, a pair $(n^{\mathsf{hnd}}, m^{\mathsf{hnd}}, v, j)$ states that $\mathsf{M}_u^{\mathsf{OR}}$ generated the handle n^{hnd} in the j-th step of the protocol in a session run with v and session identifier m^{hnd}. The set $Nonce_u$ is initially empty. The trusted third party $\mathsf{M}_\mathsf{T}^{\mathsf{OR}}$ maintains an initially empty set SID_T to store already processed session IDs.

We now define how the protocol machine $\mathsf{M}_u^{\mathsf{OR}}$ evaluates inputs. They either come from user u at port $\mathsf{KS\_in}_u?$ or from $\mathsf{TH}_{\mathcal{H}}$ at port $\mathsf{out}_u?$. The behavior of $\mathsf{M}_u^{\mathsf{OR}}$ in both cases is described in Algorithm 1 and 3 respectively, which we will describe below. The trusted third party $\mathsf{M}_\mathsf{T}^{\mathsf{OR}}$ only receives inputs from the cryptographic library, and its behavior is described in Algorithm 2. We refer to Step i of Algorithm j as Step $j.i$. All three algorithms should immediately abort if a command to the cryptographic library does not yield the desired result, e.g., if a decryption requests fails. For readability we omit these abort checks in the algorithm descriptions; instead we impose the following convention on all three algorithms.

Convention 1 *For all $w \in \{1, \ldots, n\} \cup \{\mathsf{T}\}$ the following holds. If $\mathsf{M}_w^{\mathsf{OR}}$ enters a command at port $\mathsf{in}_w!$ and receives $\downarrow$ at port $\mathsf{out}_w?$ as the immediate answer of the cryptographic library, then $\mathsf{M}_w^{\mathsf{OR}}$ aborts the execution of the current algorithm, except if the command was of the form* list_proj *or* send_i.

Protocol start. The user of the protocol machine $\mathsf{M}_u^{\mathsf{OR}}$ can start a new protocol with user $v \in \{1, \ldots, n\} \setminus \{u\}$ by inputting $(\mathsf{new\_prot}, \mathsf{Otway\_Rees}, v)$ at port $\mathsf{KS\_in}_u?$. Our security proof holds for all adversaries and all honest users, i.e., especially those that start

Algorithm 1 Evaluation of Inputs from the User (Protocol Start).

Input: $(\mathsf{new\_prot}, \mathsf{Otway\_Rees}, v)$ at $\mathsf{KS\_in}_u?$ with $v \in \{1, \dots, n\} \setminus \{u\}$.
1: $n_u^{\mathsf{hnd}} \leftarrow \mathsf{gen\_nonce}()$.
2: $ID^{\mathsf{hnd}} \leftarrow \mathsf{gen\_nonce}()$.
3: $Nonce_u := Nonce_u \cup \{(n_u^{\mathsf{hnd}}, ID^{\mathsf{hnd}}, v, 1)\}$.
4: $u^{\mathsf{hnd}} \leftarrow \mathsf{store}(u)$.
5: $v^{\mathsf{hnd}} \leftarrow \mathsf{store}(v)$.
6: $l_1^{\mathsf{hnd}} \leftarrow \mathsf{list}(n_u^{\mathsf{hnd}}, ID^{\mathsf{hnd}}, u^{\mathsf{hnd}}, v^{\mathsf{hnd}})$.
7: $c_1^{\mathsf{hnd}} \leftarrow \mathsf{sym\_encrypt}(skse_{u,\mathsf{T}}^{\mathsf{hnd}}, l_1^{\mathsf{hnd}})$.
8: $m_1^{\mathsf{hnd}} \leftarrow \mathsf{list}(ID^{\mathsf{hnd}}, c_1^{\mathsf{hnd}})$.
9: $\mathsf{send\_i}(v, m_1^{\mathsf{hnd}})$.

protocols with the adversary (respectively a malicious user) in parallel with protocols with honest users. Upon such an input, $\mathsf{M}_u^{\mathsf{OR}}$ builds up the term corresponding to the first protocol message using the ideal cryptographic library according to Algorithm 1. The command gen_nonce generates the ideal nonce as well as the session identifier. $\mathsf{M}_u^{\mathsf{OR}}$ stores the resulting handles n_u^{hnd} and m^{hnd} in $Nonce_u$ for future comparison together with the identity of v and an indicator that these handles were generated in the first step of the protocol. The command store inputs arbitrary application data into the cryptographic library, here the user identities u and v. The command list forms a list and sym_encrypt is symmetric encryption. The final command send_i means that $\mathsf{M}_u^{\mathsf{OR}}$ sends the resulting term to v over an insecure channel. The effect is that the adversary obtains a handle to the term and can decide what to do with it (such as forwarding it to $\mathsf{M}_v^{\mathsf{OR}}$).

Evaluation of network inputs for protocol machines. The behavior of the protocol machine $\mathsf{M}_u^{\mathsf{OR}}$ upon receiving an input from the cryptographic library at port $\mathsf{out}_u?$ (corresponding to a message that arrives over the network) is defined similarly in Algorithm 3. By construction of $\mathsf{TH}_{\mathcal{H}}$, such an input is always of the form $(v, u, \mathsf{i}, m^{\mathsf{hnd}})$ where m^{hnd} is a handle to a list. To increase readability, and to clarify the connection between the algorithmic description and the usual protocol notation, we augment the algorithm with explanatory comments at its right-hand side to depict which handle corresponds to which Dolev-Yao term. We further use the naming convention that ingoing and outgoing messages are labeled m, where outgoing messages have an additional subscript corresponding to the protocol step. Encryptions are labeled c, the encrypted lists are labeled l, both with suitable sub- and superscripts.

$\mathsf{M}_u^{\mathsf{OR}}$ first determines the session identifier and aborts if it is not of type nonce. $\mathsf{M}_u^{\mathsf{OR}}$ then checks if the obtained message could correspond to the first, third, or fourth step of the protocol. (Recall that the second step is only performed by T.) This is implemented by looking up the session identifier in the set $Nonce_u$. After that, $\mathsf{M}_u^{\mathsf{OR}}$ checks if the obtained message is indeed a suitably constructed message for the particular step and the particular session ID by exploiting the contents of $Nonce_u$. If so, $\mathsf{M}_u^{\mathsf{OR}}$ constructs a message according to the protocol description, sends it to the intended recipient, updates the set $Nonce_u$, and possibly signals to its user that a key has been successfully shared with another user.

Algorithm 2 Behavior of the Trusted Third Party.

Input: $(v, \mathsf{T}, \mathsf{i}, m^{\mathsf{hnd}})$ at $\mathsf{out_T?}$ with $v \in \{1, \ldots, n\}$.

$ID^{\mathsf{hnd}} \leftarrow \mathsf{list\_proj}(m^{\mathsf{hnd}}, 1)$. $\quad \{ID^{\mathsf{hnd}} \approx M\}$

$type_1 \leftarrow \mathsf{get\_type}(ID^{\mathsf{hnd}})$.

$c^{(3)^{\mathsf{hnd}}} \leftarrow \mathsf{list\_proj}(m^{\mathsf{hnd}}, 3)$. $\quad \{c^{(3)^{\mathsf{hnd}}} \approx \{N_v, M, u, v\}_{K_{vt}}\}$

$l^{(3)^{\mathsf{hnd}}} \leftarrow \mathsf{sym\_decrypt}(skse^{\mathsf{hnd}}_{\mathsf{T},v}, c^{(3)^{\mathsf{hnd}}})$. $\quad \{l^{(3)^{\mathsf{hnd}}} \approx \{N_v, M, u, v\}\}$

$y_i^{\mathsf{hnd}} \leftarrow \mathsf{list\_proj}(l^{(3)^{\mathsf{hnd}}}, i)$ for $i = 1, 2, 3, 4$.

$y_i \leftarrow \mathsf{retrieve}(y_i^{\mathsf{hnd}})$ for $i = 3, 4$.

if $(ID^{\mathsf{hnd}} \in SID_{\mathsf{T}}) \vee (type_1 \neq \mathsf{nonce}) \vee (y_2^{\mathsf{hnd}} \neq ID^{\mathsf{hnd}}) \vee (y_3 \notin \{1, \ldots, n\} \setminus \{v\}) \vee (y_4 \neq v)$ **then**

  Abort

end if

$SID_{\mathsf{T}} := SID_{\mathsf{T}} \cup \{ID^{\mathsf{hnd}}\}$.

$c^{(2)^{\mathsf{hnd}}} \leftarrow \mathsf{list\_proj}(m^{\mathsf{hnd}}, 2)$. $\quad \{c^{(2)^{\mathsf{hnd}}} \approx \{N_u, M, u, v\}_{K_{ut}}\}$

$l^{(2)^{\mathsf{hnd}}} \leftarrow \mathsf{sym\_decrypt}(skse^{\mathsf{hnd}}_{\mathsf{T},y_3}, c^{(2)^{\mathsf{hnd}}})$. $\quad \{l^{(2)^{\mathsf{hnd}}} \approx \{N_u, M, u, v\}\}$

$x_i^{\mathsf{hnd}} \leftarrow \mathsf{list\_proj}(l^{(2)^{\mathsf{hnd}}}, i)$ for $i = 1, 2, 3, 4$.

$type_2 \leftarrow \mathsf{get\_type}(x_1^{\mathsf{hnd}})$.

$x_i \leftarrow \mathsf{retrieve}(x_i^{\mathsf{hnd}})$ for $i = 3, 4$.

if $(type_2 \neq \mathsf{nonce}) \vee (x_2^{\mathsf{hnd}} \neq y_2^{\mathsf{hnd}}) \vee (x_3 \neq y_3) \vee (x_4 \neq y_4)$ **then**

  Abort

end if

$skse^{\mathsf{hnd}} \leftarrow \mathsf{gen\_symenc\_key}()$. $\quad \{skse^{\mathsf{hnd}} \approx K_{uv}\}$

$l_3^{(2)^{\mathsf{hnd}}} \leftarrow \mathsf{list}(x_1^{\mathsf{hnd}}, skse^{\mathsf{hnd}})$. $\quad \{l_3^{(2)^{\mathsf{hnd}}} \approx \{N_u, K_{uv}\}\}$

$c_3^{(2)^{\mathsf{hnd}}} \leftarrow \mathsf{sym\_encrypt}(skse^{\mathsf{hnd}}_{\mathsf{T},y_3}, l_3^{(2)^{\mathsf{hnd}}})$. $\quad \{c_3^{(2)^{\mathsf{hnd}}} \approx \{N_u, K_{uv}\}_{K_{ut}}\}$

$l_3^{(3)^{\mathsf{hnd}}} \leftarrow \mathsf{list}(y_1^{\mathsf{hnd}}, skse^{\mathsf{hnd}})$. $\quad \{l_3^{(3)^{\mathsf{hnd}}} \approx \{N_v, K_{uv}\}\}$

$c_3^{(3)^{\mathsf{hnd}}} \leftarrow \mathsf{sym\_encrypt}(skse^{\mathsf{hnd}}_{\mathsf{T},v}, l_3^{(3)^{\mathsf{hnd}}})$. $\quad \{c_3^{(3)^{\mathsf{hnd}}} \approx \{N_v, K_{uv}\}_{K_{vt}}\}$

$m_3^{\mathsf{hnd}} \leftarrow \mathsf{list}(ID^{\mathsf{hnd}}, c_3^{(2)^{\mathsf{hnd}}}, c_3^{(3)^{\mathsf{hnd}}})$. $\quad \{m_3^{\mathsf{hnd}} \approx M, \{N_u, K_{uv}\}_{K_{ut}}, \{N_v, K_{uv}\}_{K_{vt}}\}$

$\mathsf{send\_i}(v, m_3^{\mathsf{hnd}})$.

Behavior of the trusted third party. The behavior of $\mathsf{M}^{\mathsf{OR}}_{\mathsf{T}}$ upon receiving an input $(v, \mathsf{T}, \mathsf{i}, m^{\mathsf{hnd}})$ from the cryptographic library at port $\mathsf{out_T?}$ is defined similarly in Algorithm 2. We omit an informal description.

3.3 On Polynomial Runtime

In order to use existing composition results of the underlying model, the protocol machines $\mathsf{M}^{\mathsf{OR}}_{w}$ and $\mathsf{M}^{\mathsf{OR}}_{\mathsf{T}}$ must be polynomial-time. Similar to the cryptographic library, we define that each of these machines maintains explicit polynomial bounds on the message lengths and the number of inputs accepted at each port.

4 The Security Property

In the following, we formalize the security property of the ideal Otway-Rees protocol. The property consists of a *secrecy property* and a *consistency property*. The secrecy

Algorithm 3 Evaluation of Inputs from $\mathsf{TH}_{\mathcal{H}}$ (Network Inputs).

Input: $(v, u, \mathsf{i}, m^{\mathsf{hnd}})$ at $\mathsf{out}_u?$ with $v \in \{1, \ldots, n\} \setminus \{u\}$.

$ID^{\mathsf{hnd}} \leftarrow \mathsf{list\_proj}(m^{\mathsf{hnd}}, 1)$. $\{ID^{\mathsf{hnd}} \approx M\}$
$type_1 \leftarrow \mathsf{get\_type}(ID^{\mathsf{hnd}})$.
if $type_1 \neq \mathsf{nonce}$ **then**
  Abort
end if
if $v \neq \mathsf{T} \land \forall j, n^{\mathsf{hnd}}: (n^{\mathsf{hnd}}, ID^{\mathsf{hnd}}, v, j) \notin Nonce_u$ **then** {First Message is input}
  $c^{(2)^{\mathsf{hnd}}} \leftarrow \mathsf{list\_proj}(m^{\mathsf{hnd}}, 2)$. $\{c^{(2)^{\mathsf{hnd}}} \approx (N_v, M, v, u)_{K_{vt}}\}$
  $n_u^{\mathsf{hnd}} \leftarrow \mathsf{gen\_nonce}()$.
  $Nonce_u := Nonce_u \cup \{(n_u^{\mathsf{hnd}}, ID^{\mathsf{hnd}}, v, 2)\}$.
  $u^{\mathsf{hnd}} \leftarrow \mathsf{store}(u)$.
  $v^{\mathsf{hnd}} \leftarrow \mathsf{store}(v)$.
  $l_2^{(3)^{\mathsf{hnd}}} \leftarrow \mathsf{list}(n_u^{\mathsf{hnd}}, ID^{\mathsf{hnd}}, v^{\mathsf{hnd}}, u^{\mathsf{hnd}})$. $\{l_2^{(3)^{\mathsf{hnd}}} \approx N_u, M, v, u\}$
  $c_2^{(3)^{\mathsf{hnd}}} \leftarrow \mathsf{sym\_encrypt}(skse_{u,\mathsf{T}}^{\mathsf{hnd}}, l_2^{(3)^{\mathsf{hnd}}})$. $\{c_2^{(3)^{\mathsf{hnd}}} \approx (N_u, M, v, u)_{K_{ut}}\}$
  $m_2^{\mathsf{hnd}} \leftarrow \mathsf{list}(ID^{\mathsf{hnd}}, c^{(2)^{\mathsf{hnd}}}, c_2^{(3)^{\mathsf{hnd}}})$. $\{m_2^{\mathsf{hnd}} \approx M, (N_v, M, v, u)_{K_{vt}}, (N_u, M, v, u)_{K_{ut}}\}$
  $\mathsf{send\_i}(\mathsf{T}, m_2^{\mathsf{hnd}})$.
else if $v = \mathsf{T}$ **then** {Third Message is input}
  $c^{(2)^{\mathsf{hnd}}} \leftarrow \mathsf{list\_proj}(m^{\mathsf{hnd}}, 2)$. $\{c^{(2)^{\mathsf{hnd}}} \approx (N_v, K_{uv})_{K_{vt}}\}$
  $c^{(3)^{\mathsf{hnd}}} \leftarrow \mathsf{list\_proj}(m^{\mathsf{hnd}}, 3)$. $\{c^{(3)^{\mathsf{hnd}}} \approx (N_u, K_{uv})_{K_{ut}}\}$
  $l^{(3)^{\mathsf{hnd}}} \leftarrow \mathsf{sym\_decrypt}(skse_{u,\mathsf{T}}^{\mathsf{hnd}}, c^{(3)^{\mathsf{hnd}}})$. $\{l^{(3)^{\mathsf{hnd}}} \approx N_u, K_{uv}\}$
  $y_i^{\mathsf{hnd}} \leftarrow \mathsf{list\_proj}(l^{(3)^{\mathsf{hnd}}}, i)$ for $i = 1, 2$.
  $type_2 \leftarrow \mathsf{get\_type}(y_2^{\mathsf{hnd}})$.
  if $(\nexists! w \in \{1, \ldots, n\} \setminus \{u\}: (y_1^{\mathsf{hnd}}, ID^{\mathsf{hnd}}, w, 2) \in Nonce_u) \lor (type_2 \neq \mathsf{skse})$ **then**
    Abort
  end if
  $Nonce_u := (Nonce_u \setminus \{(y_1^{\mathsf{hnd}}, ID^{\mathsf{hnd}}, w, 2)\}) \cup \{(y_1^{\mathsf{hnd}}, ID^{\mathsf{hnd}}, w, 3)\}$.
  $m_4^{\mathsf{hnd}} \leftarrow \mathsf{list}(ID^{\mathsf{hnd}}, c^{(2)^{\mathsf{hnd}}})$. $\{m_4^{\mathsf{hnd}} \approx M, \{N_v, K_{uv}\}_{K_{vt}}\}$
  $\mathsf{send\_i}(w, m_4^{\mathsf{hnd}})$.
  Output $(\mathsf{ok}, \mathsf{Otway\_Rees}, w, ID^{\mathsf{hnd}}, y_2^{\mathsf{hnd}})$ at $\mathsf{KS\_out}_u!$.
else if $v \neq \mathsf{T} \land \exists! n^{\mathsf{hnd}}: (n^{\mathsf{hnd}}, ID^{\mathsf{hnd}}, v, 1)$ **then** {Fourth Message is input}
  $c^{(2)^{\mathsf{hnd}}} \leftarrow \mathsf{list\_proj}(m^{\mathsf{hnd}}, 2)$. $\{c^{(2)^{\mathsf{hnd}}} \approx \{N_u, K_{uv}\}_{K_{ut}}\}$
  $l^{(2)^{\mathsf{hnd}}} \leftarrow \mathsf{sym\_decrypt}(skse_{u,\mathsf{T}}^{\mathsf{hnd}}, c^{(2)^{\mathsf{hnd}}})$. $\{l^{(2)^{\mathsf{hnd}}} \approx \{N_u, K_{uv}\}\}$
  $x_i^{\mathsf{hnd}} \leftarrow \mathsf{list\_proj}(l^{(2)^{\mathsf{hnd}}}, i)$ for $i = 1, 2$.
  $type_3 \leftarrow \mathsf{get\_type}(x_2^{\mathsf{hnd}})$.
  if $x_1^{\mathsf{hnd}} \neq n^{\mathsf{hnd}} \lor type_3 \neq \mathsf{skse}$ **then**
    Abort
  end if
  $Nonce_u := (Nonce_u \setminus \{(x_1^{\mathsf{hnd}}, ID^{\mathsf{hnd}}, v, 1)\}) \cup \{(x_1^{\mathsf{hnd}}, ID^{\mathsf{hnd}}, v, 4)\}$.
  Output $(\mathsf{ok}, \mathsf{Otway\_Rees}, v, ID^{\mathsf{hnd}}, x_2^{\mathsf{hnd}})$ at $\mathsf{KS\_out}_u!$.
else
  Abort
end if

$\forall u, v \in \mathcal{H}, \forall t_1, t_2 \in \mathbb{N}$:	*# For all honest users u and v,*
$(t_1 : \mathsf{KS\_out}_u!(\mathsf{ok}, \mathsf{Otway\_Rees}, v, ID^{\mathsf{hnd}}, skse_u^{\mathsf{hnd}})$	*# if u has established a shared key with v*
$\Rightarrow$	*# then*
$t_2 : D[hnd_u = skse_u^{\mathsf{hnd}}].hnd_{\mathsf{a}} = \downarrow)$	*# the adversary never learns this key*

Fig. 2. The Secrecy Property Req^{Sec}.

$\forall u, v \in \mathcal{H}, \forall t_1, t_2 \in \mathbb{N}$:	*# For all honest users u and v,*
$t_1 : \mathsf{KS\_out}_u!(\mathsf{ok}, \mathsf{Otway\_Rees}, v, ID_u^{\mathsf{hnd}}, skse_u^{\mathsf{hnd}}) \wedge$	*# if u has established a key with v*
$t_2 : \mathsf{KS\_out}_v!(\mathsf{ok}, \mathsf{Otway\_Rees}, w, ID_v^{\mathsf{hnd}}, skse_v^{\mathsf{hnd}}) \wedge$	*# and v has established a key with w*
$t_1 : D[hnd_u = ID_u^{\mathsf{hnd}}] = t_2 : D[hnd_v = ID_v^{\mathsf{hnd}}]$	*# and the sessions are equal*
$\Rightarrow (u = w \Leftrightarrow$	*# then u is equal to w if and only if*
$t_1 : D[hnd_u = skse_u^{\mathsf{hnd}}] = t_2 : D[hnd_v = skse_v^{\mathsf{hnd}}])$	*# both keys are equal.*

Fig. 3. The Consistency Property Req^{Cons}.

property states that if two honest users successfully terminate a protocol session and then share a key, the adversary will never learn this key, which captures the confidentiality aspects of the protocol. The consistency property states that if two honest users establish a session key then both need to have a consistent view of who the peers to the session are, i.e., if an honest user u establishes a key with v, and v establishes the same key with user w, then u has to equal w. Moreover, we incorporate the correctness of the protocol into the consistency property, i.e., if the aforementioned outputs occur and $u = w$ holds, then both parties have obtained the same key[4]. In the following definitions, we write $t : D$ to denote the contents of database D at time t, i.e., at the t-th step of the considered trace, and $t : p?m$ and $t : p!m$ to denote that message m occurs at input port respectively output port p at time t.

The secrecy property Req^{Sec} is formally captured as follows: If an output $(\mathsf{ok}, \mathsf{Otway\_Rees}, v, ID^{\mathsf{hnd}}, skse_u^{\mathsf{hnd}})$ occurs at $\mathsf{KS\_out}_v!$ at an arbitrary time t_1, then the key corresponding to $skse_u^{\mathsf{hnd}}$ never gets an adversary handle, i.e., $t_2 : D[hnd_u = skse_u^{\mathsf{hnd}}].hnd_{\mathsf{a}} = \downarrow$ for all t_2. Figure 2 contains the formal definition of Req^{Sec}.

The *consistency property* Req^{Cons} is formally captured as follows: Assume that outputs $(\mathsf{ok}, \mathsf{Otway\_Rees}, v, ID_u^{\mathsf{hnd}}, skse_u^{\mathsf{hnd}})$ and $(\mathsf{ok}, \mathsf{Otway\_Rees}, w, ID_v^{\mathsf{hnd}}, skse_v^{\mathsf{hnd}})$ occur at $\mathsf{KS\_out}_u!$ respectively at $\mathsf{KS\_out}_v!$ at arbitrary times t_1 and t_2 for honest users u and v such that the session identifiers are the same, i.e., $t_1 : D[hnd_u = ID_u^{\mathsf{hnd}}] = t_2 : D[hnd_v = ID_v^{\mathsf{hnd}}]$. Then the handles $skse_u^{\mathsf{hnd}}$ and $skse_v^{\mathsf{hnd}}$ point to the same entry in the database, i.e., $t_1 : D[hnd_u = skse_u^{\mathsf{hnd}}] = t_2 : D[hnd_v = skse_v^{\mathsf{hnd}}]$ if and only if $u = w$. The formal definition of Req^{Cons} is given in Figure 3.

[4] A violation of the consistency property has been pointed out in [24] which arises since in their modeling the trusted third party creates multiple keys if it is repeatedly triggered with the same message. We explicitly excluded this in our definition of the trusted third party by storing the session IDs processed so far, cf. Step 7 and 10 in Algorithm 2.

The notion of a system *Sys* fulfilling a property *Req* essentially comes in two flavors [5]. *Perfect fulfillment*, $Sys \models^{\mathsf{perf}} Req$, means that the property holds with probability one (over the probability spaces of runs, a well-defined notion from the underlying model [38]) for all honest users and for all adversaries. *Computational fulfillment*, $Sys \models^{\mathsf{poly}} Req$, means that the property only holds for polynomially bounded users and adversaries, and only with negligible error probability. Perfect fulfillment implies computational fulfillment. The following theorem captures the security of the ideal Otway-Rees protocol.

Theorem 1. *(Security of the Otway-Rees Protocol based on the Ideal Cryptographic Library) Let* $Sys^{\mathsf{OR,id}}$ *be the ideal Otway-Rees system defined in Section 3.2, and* Req^{Sec} *and* Req^{Cons} *the secrecy and consistency property of Figure 2 and 3. Then* $Sys^{\mathsf{OR,id}} \models^{\mathsf{perf}} Req^{\mathsf{Sec}} \wedge Req^{\mathsf{Cons}}$. □

5 Proof in the Ideal Setting

This section sketches the proof of Theorem 1, i.e., the proof of the Otway-Rees protocol using the ideal, deterministic cryptographic library. The complete proof can be found in the long version of this paper [4]. The proof idea is the following: If an honest user u successfully terminates a session run with another honest user v, then we first show that the established key has been created before by the trusted third party. After that, we exploit that the trusted third party as well as all honest users may only send this key within an encryption generated with a key shared between u and T respectively v and T, and we conclude that the adversary hence never gets a handle to the key. This shows the secrecy property, and the consistency property can also be easily derived from this. The main challenge was to find suitable invariants on the state of the ideal Otway-Rees system. This is somewhat similar to formal proofs using the Dolev-Yao model, and the similarity supports our hope that the new, sound cryptographic library can be used in the place of the Dolev-Yao models in automated tools.

The first invariants, *correct nonce owner* and *unique nonce use*, are easily proved and essentially state that handles x^{hnd} where $(x^{\mathsf{hnd}}, \cdot, \cdot, \cdot)$ is contained in a set $Nonce_u$ indeed point to entries of type nonce, and that no nonce is in two such sets. The next two invariants, *nonce secrecy* and *nonce-list secrecy*, deal with the secrecy of certain terms. They are mainly needed to prove the invariant *correct list generation*, which establishes who created certain terms. The last invariant, *key secrecy*, states that the adversary never learns keys created by the trusted third party for use between honest users.

- *Correct Nonce Owner.* For all $u \in \mathcal{H}$, and for all $(x^{\mathsf{hnd}}, \cdot, \cdot, \cdot) \in Nonce_u$, it holds $D[hnd_u = x^{\mathsf{hnd}}] \neq \downarrow$ and $D[hnd_u = x^{\mathsf{hnd}}].type = \mathsf{nonce}$.
- *Unique Nonce Use.* For all $u, v \in \mathcal{H}$, all $w, w' \in \{1, \ldots, n\}$, and all $j \leq size$: If $(D[j].hnd_u, \cdot, w, \cdot) \in Nonce_u$ and $(D[j].hnd_v, \cdot, w', \cdot) \in Nonce_v$, then $(u, w) = (v, w')$.

Nonce secrecy states that the nonces exchanged between honest users u and v remain secret from all other users and from the adversary. For the formalization, note that the handles x^{hnd} to these nonces are contained as elements $(x^{\mathsf{hnd}}, \cdot, v, \cdot)$ in the set $Nonce_u$.

The claim is that the other users and the adversary have no handles to such a nonce in the database D of $\mathsf{TH}_{\mathcal{H}}$:

- *Nonce Secrecy.* For all $u, v \in \mathcal{H}$ and for all $j \leq \mathit{size}$: If $(D[j].\mathit{hnd}_u, \cdot, v, \cdot) \in \mathit{Nonce}_u$ then $D[j].\mathit{hnd}_w \neq \downarrow$ implies $w \in \{u, v, \mathsf{T}\}$. In particular, this means $D[j].\mathit{hnd}_{\mathsf{a}} = \downarrow$.

Similarly, the invariant *nonce-list secrecy* states that a list containing such a handle can only be known to u, v, and T. Further, it states that the identity fields in such lists are correct. Moreover, if such a list is an argument of another entry, then this entry is an encryption created with the secret key that either u or v share with T. (Formally this means that this entry is tagged with the corresponding public-key identifier as an abstract argument, cf. Section 3.1.)

- *Nonce-List Secrecy.* For all $u, v \in \mathcal{H}$ and for all $j \leq \mathit{size}$ with $D[j].\mathit{type} = \mathsf{list}$: Let $x_i^{\mathsf{ind}} := D[j].\mathit{arg}[i]$ for $i = 1, 2, 3, 4$. If $(D[x_1^{\mathsf{ind}}].\mathit{hnd}_u, \cdot, v, l) \in \mathit{Nonce}_u$ then
  a) $D[j].\mathit{hnd}_w \neq \downarrow$ implies $w \in \{u, v, \mathsf{T}\}$ for $l \in \{1, 2, 3, 4\}$.
  b) If $l \in \{1, 4\}$ and $D[x_3^{\mathsf{ind}}].\mathit{type} = \mathsf{data}$, then $D[x_3^{\mathsf{ind}}].\mathit{arg} = (u)$ and $D[x_4^{\mathsf{ind}}].\mathit{arg} = (v)$.
  c) If $l \in \{2, 3\}$ and $D[x_3^{\mathsf{ind}}].\mathit{type} = \mathsf{data}$, then $D[x_3^{\mathsf{ind}}].\mathit{arg} = (v)$ and $D[x_4^{\mathsf{ind}}].\mathit{arg} = (u)$.
  d) for $l \in \{1, 2, 3, 4\}$ and for all $k \leq \mathit{size}$ it holds $j \in D[k].\mathit{arg}$ only if $D[k].\mathit{type} = \mathsf{symenc}$ and $D[k].\mathit{arg}[1] \in \{\mathit{pkse}_u, \mathit{pkse}_v\}$.

The invariant *correct list owner* states that certain protocol messages can only be constructed by the "intended" users respectively by the trusted third party.

- *Correct List Owner.* For all $u, v \in \mathcal{H}$ and for all $j \leq \mathit{size}$ with $D[j].\mathit{type} = \mathsf{list}$: Let $x_i^{\mathsf{ind}} := D[j].\mathit{arg}[i]$ for $i = 1, 2$ and $x_{1,u}^{\mathsf{hnd}} := D[x_1^{\mathsf{ind}}].\mathit{hnd}_u$.
  a) If $(x_{1,u}^{\mathsf{hnd}}, \cdot, v, l) \in \mathit{Nonce}_u$ and $D[x_2^{\mathsf{ind}}].\mathit{type} \neq \mathsf{skse}$, then $D[j]$ was created by $\mathsf{M}_u^{\mathsf{OR}}$ in Step 1.6 if $l = 1$ and in Step 3.12 if $l = 2$.
  b) If $(x_{1,u}^{\mathsf{hnd}}, \mathit{ID}_u^{\mathsf{hnd}}, v, l) \in \mathit{Nonce}_u$ and $D[x_2^{\mathsf{ind}}].\mathit{type} = \mathsf{skse}$, then $D[j]$ was created by $\mathsf{M}_{\mathsf{T}}^{\mathsf{OR}}$ in Step 2.22 if $l = 3$ and in Step 2.20 if $l = 4$. Moreover, we have $D[\mathit{hnd}_u = \mathit{ID}_u^{\mathsf{hnd}}] = D[\mathit{hnd}_{\mathsf{T}} = \mathit{ID}_{\mathsf{T}}^{\mathsf{hnd}}]$, where $\mathit{ID}_{\mathsf{T}}^{\mathsf{hnd}}$ denotes the handle that T obtained in Step 2.1 in the same execution.

Finally, the invariant *key secrecy* states that a secret key entry that has been generated by the trusted third party to be shared between honest users u and v can only be known to u, v, and T. In particular, the adversary will never get a handle to it. This invariant is key for proving the secrecy and the consistency property of the Otway-Rees protocol.

- *Key Secrecy.* For all $u, v \in \mathcal{H}$ and for all $j \leq \mathit{size}$ with $D[j].\mathit{type} = \mathsf{skse}$: If $D[j]$ was created by $\mathsf{M}_{\mathsf{T}}^{\mathsf{OR}}$ in Step 2.19 and, with the notation of Algorithm 2, we have that $y_3 = u$ and $y_4 = v$ in the current execution of $\mathsf{M}_{\mathsf{T}}^{\mathsf{OR}}$, then $D[j].\mathit{hnd}_w \neq \downarrow$ implies $w \in \{u, v, \mathsf{T}\}$.

6 Proof of the Cryptographic Realization

If Theorem 1 has been proven, it remains to show that the Otway-Rees protocol based on the real cryptographic library computationally fulfills corresponding secrecy and consistency requirements. Actually, different corresponding requirements can easily be derived from the proof in the ideal setting. Obviously, carrying over properties from the ideal to the real system relies crucially on the fact that the real cryptographic library is at least as secure as the ideal one. This has been established in [8, 7], but only subject to the side condition that the surrounding protocol, i.e., the Otway-Rees protocol in our case, does not raise a so-called *commitment problem*. Establishing this side condition is crucial for using symmetric encryption in abstract, cryptographically sound proofs. We explain the commitment problem in the next section to illustrate the cryptographic issue underlying the commitment problem, and we exploit the invariants of Section 5 to show that the commitment problem does not occur for the Otway-Rees protocol. As our proof is the first Dolev-Yao-style, cryptographically sound proof of a protocol that uses symmetric encryption, our result also shows that the commitment problem, and hence also symmetric encryption, can be conveniently dealt with in cryptographically sound security proofs by means of the approach of [7].

For technical reasons, one further has to ensure that the surrounding protocol does not create “encryption cycles” (such as encrypting a key with itself), which had to be required even for acquiring properties weaker than simulatability, cf. [3] for further discussions. This property is only a technical subtlety and clearly holds for the Otway-Rees protocol.

6.1 Absence of the Commitment Problem for the Otway-Rees Protocol

As the name suggests, a “commitment problem” in simulatability proofs captures a situation where the simulator commits itself to a certain message and later has to change this commitment to allow for a correct simulation. In the case of symmetric encryption, the commitment problem occurs if the simulator learns in some abstract way that a ciphertext was sent and hence has to construct an indistinguishable ciphertext, knowing neither the secret key nor the plaintext used for the corresponding ciphertext in the real world. To simulate the missing key, the simulator will create a new secret key, or rely on an arbitrary, fixed key if the encryption systems guarantees indistinguishable keys, see [3]. Instead of the unknown plaintext, the simulator will encrypt an arbitrary message of the correct length, relying on the indistinguishability of ciphertexts of different messages. So far, the simulation is fine. It even stays fine if the message becomes known later because secure encryption still guarantees that it is indistinguishable that the simulator’s ciphertext contains a wrong message. However, if the secret key becomes known later, the simulator runs into trouble, because, learning abstractly about this fact, it has to produce a suitable key that decrypts its ciphertext into the correct message. It cannot cheat with the message because it has to produce the correct behavior towards the honest users. This is typically not possible.

The solution for this problem taken in [7] for the cryptographic library is to leave it to the surrounding protocol to guarantee that the commitment problem does not occur, i.e., the surrounding protocol must guarantee that keys are no longer sent in a form that might make them known to the adversary once an honest participant has started using them. To exploit the simulatability results of [7], we hence have to prove this condition for the Otway-Rees protocol. Formally, we have to show that the following property NoComm does not occur: "If there exists an input from an honest user that causes a symmetric encryption to be generated such that the corresponding key is not known to the adversary, then future inputs may only cause this key to be sent within an encryption that cannot be decrypted by the adversary". This event can be rigorously defined in the style of the secrecy and consistency property but we omit the rigorous definition due to space constraints and refer to [7]. The event NoComm is equivalent to the event "if there exists an input from an honest user that causes a symmetric encryption to be generated such that the corresponding key is not known to the adversary, the adversary never gets a handle to this key" but NoComm has the advantage that it can easily be inferred from the abstract protocol description without presupposing knowledge about handles of the cryptographic library. For the Otway-Rees protocol the event NoComm can easily be verified by inspection of the abstract protocol description, and a detailed proof based on Algorithms 1-3 can also easily be performed by exploiting the invariants of Section 5.

Lemma 1. *(Absence of the Commitment Problem for the Otway-Rees Protocol) The ideal Otway-Rees system* $Sys^{\mathsf{OR,id}}$ *perfectly fulfills the property* NoComm, *i.e.,* $Sys^{\mathsf{OR,id}} \models^{\mathsf{perf}} \mathsf{NoComm}$. □

Proof. Note first that the secret key shared initially between a user and the trusted third party will never be sent by definition in case the user is honest, and it is already known to the adversary when it is first used in case of a dishonest user. The interesting cases are thus the keys generated by the trusted third party in the protocol sessions.

Let $j \leq size$, $D[j].type = \mathsf{skse}$ such that $D[j]$ was created by $\mathsf{M}_\mathsf{T}^\mathsf{OR}$ in Step 2.19, where, with the notation of Algorithm 2, we have $y_3 = u$ and $y_4 = v$ for $y_3, y_4 \in \{1, \ldots, n\}$. If u or v were dishonest, then the adversary would get a handle for $D[j]$ after $\mathsf{M}_\mathsf{T}^\mathsf{OR}$ finishes its execution, i.e., in particular before $D[j]$ has been used for encryption for the first time, since the adversary knows the keys shared between the dishonest users and the trusted third party. If both u and v are honest, *key secrecy* then immediately implies that $t : D[j].hnd_\mathsf{a} = \downarrow$ for all $t \in \mathbb{N}$, which finishes the proof. ■

6.2 Proof of Secrecy and Consistency

As the final step in the overall security proof, we show how to derive corresponding secrecy and consistency properties from the proofs in the ideal setting and the simulatability result of the underlying library.

We show this only for secrecy and sketch the proof for consistency. Note that the secrecy property Req^{Sec} specifically relies on the state of $\mathsf{TH}_\mathcal{H}$, hence it cannot be used to capture the security of the real Otway-Rees system, where $\mathsf{TH}_\mathcal{H}$ is replaced with the secure implementation of the cryptographic library. The natural counterpart of Req^{Sec} in the real system is to demand that the adversary never learns the key (now as an actual bitstring), which can be captured in various ways. One possibility that allows for a very convenient proof is to capture the property as a so-called *integrity property* in the sense of [5]. Integrity properties correspond to sets of traces at the in- and output ports connecting the system to the honest users, i.e., properties that can be expressed solely via statements about events at the port set $S_{\mathcal{H}}$; in particular, integrity properties do not rely on the state of the underlying machine. Integrity properties are preserved under simulatability, i.e., they carry from the ideal to the real system without any additional work. Formally, the following *preservation theorem* has been established in [5].

Theorem 2. *(Preservation of Integrity Properties (Sketch)) Let two systems Sys_1, Sys_2 be given such that Sys_1 is at least as secure as Sys_2 (written $Sys_1 \geq_{\mathsf{sec}}^{\mathsf{poly}} Sys_2$). Let Req be an integrity property for both Sys_1 and Sys_2, and let $Sys_2 \models^{\mathsf{poly}} Req$. Then also $Sys_1 \models^{\mathsf{poly}} Req$.* □

We can now easily rephrase the secrecy property Req^{Sec} into an equivalent integrity property that is well-defined for both the ideal and the real Otway-Rees system by employing standard techniques, e.g., by assuming that once the adversary has learned the shared key, the adversary sends the key to an honest user. Formally, we may augment the behavior the protocol machine $\mathsf{M}_u^{\mathsf{OR}}$ so that if it receives a message $(\mathsf{broken}, skse_u^{\mathsf{hnd}})$ from a dishonest sender, it outputs this message to its user u at port $\mathsf{KS\_out}_u!$. The property Req^{Sec} can then be rewritten by replacing the statement $t_2 : D[hnd_u = skse_u^{\mathsf{hnd}}].hnd_{\mathsf{a}} = \downarrow$ with $t_2 : \mathsf{KS\_out}_u!m \implies m \neq (\mathsf{broken}, skse_u^{\mathsf{hnd}})$. We call the resulting integrity property $Req_{\mathsf{real}}^{\mathsf{Sec}}$. If we denote the ideal Otway-Rees system based on these augmented protocol machines by $Sys'^{\mathsf{OR,id}}$ then we clearly have $Sys^{\mathsf{OR,id}} \models^{\mathsf{perf}} Req^{\mathsf{Sec}}$ if and only if $Sys'^{\mathsf{OR,id}} \models^{\mathsf{perf}} Req_{\mathsf{real}}^{\mathsf{Sec}}$ since a user may only receive a message $(\mathsf{broken}, skse_u^{\mathsf{hnd}})$ if the adversary already has a handle to $skse_u^{\mathsf{hnd}}$, and conversely if an adversary has a handle to $skse_u^{\mathsf{hnd}}$ it can create and send the message $(\mathsf{broken}, skse_u^{\mathsf{hnd}})$. This can easily be turned into a formal proof by inspection of the commands list and send_i offered by the trusted host. The preservation theorem now immediately allows us to carry over the secrecy property to the real Otway-Rees system.

Theorem 3. *(Security of the Real Otway-Rees Protocol) Let $Sys'^{\mathsf{OR,real}}$ denote the Otway-Rees system based on the real cryptographic library and the protocol machines augmented for capturing the integrity property $Req_{\mathsf{real}}^{\mathsf{Sec}}$. Then $Sys'^{\mathsf{OR,real}} \models^{\mathsf{poly}} Req_{\mathsf{real}}^{\mathsf{Sec}}$.* □

Proof. Let $Sys^{\mathsf{cry,id}}$ and $Sys^{\mathsf{cry,real}}$ denote the ideal and the real cryptographic library from [8] augmented with symmetric encryption as introduced in [7]. In [8, 7] it has already been shown that $Sys^{\mathsf{cry,real}} \geq_{\mathsf{sec}}^{\mathsf{poly}} Sys^{\mathsf{cry,id}}$ holds for suitable parameters in the ideal system, provided that neither the commitment problem nor encryption cycles occur. We have shown both conditions in the previous section. Let $Sys'^{\mathsf{OR,id}}$ denote the ideal Otway-Rees system based on the augmented protocol machines. Since $Sys'^{\mathsf{OR,real}}$ is derived from $Sys'^{\mathsf{OR,id}}$ by replacing the ideal with the real cryptographic library, $Sys'^{\mathsf{OR,real}} \geq_{\mathsf{sec}}^{\mathsf{poly}} Sys'^{\mathsf{OR,id}}$ follows from the composition theorem of [38]. We only have to show that the theorem's preconditions are in fact fulfilled. This is straightforward, since the machines $\mathsf{M}_u^{\mathsf{OR}}$ are polynomial-time (cf. Section 3.3). Now Theorem 1 implies $Sys^{\mathsf{OR,id}} \models^{\mathsf{poly}} Req^{\mathsf{Sec}}$ which yields $Sys'^{\mathsf{OR,id}} \models^{\mathsf{poly}} Req_{\mathsf{real}}^{\mathsf{Sec}}$. Since $Req_{\mathsf{real}}^{\mathsf{Sec}}$ is an integrity property Theorem 2 yields $Sys'^{\mathsf{OR,real}} \models^{\mathsf{poly}} Req_{\mathsf{real}}^{\mathsf{Sec}}$. ■

Similar to the secrecy property, the consistency property Req^{Cons} specifically relies on the state of $\mathsf{TH}_{\mathcal{H}}$. The corresponding consistency property for the real Otway-Rees system can be defined by requiring that both handles point to the same bitstring, i.e., by replacing $t_1 : D[hnd_u = skse_u^{\mathsf{hnd}}] = t_2 : D[hnd_v = skse_v^{\mathsf{hnd}}$ with $t_1 : D_u[hnd_u = skse_u^{\mathsf{hnd}}].word = t_2 : D_v[hnd_v = skse_v^{\mathsf{hnd}}].word$ for the databases D_u and D_v of the real library. We omit a formal proof that the real Otway-Rees system computationally fulfills this property; the proof can be established similar to the proof of the secrecy property where one additionally exploits that if the real Otway-Rees protocol is run with an arbitrary adversary and we have $t_1 : D[hnd_u = skse_u^{\mathsf{hnd}}] = t_2 : D[hnd_v = skse_v^{\mathsf{hnd}}]$ then there always exist an adversary against the ideal Otway-Rees protocol such that $t_1 : D_u[hnd_u = skse_u^{\mathsf{hnd}}].word = t_2 : D_v[hnd_u = skse_u^{\mathsf{hnd}}].word$, cf. [8, 7].

6.3 Towards Stronger Properties

To conclude, we sketch that also stronger properties can be derived for the real Otway-Rees protocol from Theorem 1 and the proof of the simulatability result of the cryptographic library, e.g., a stronger notion of secrecy: There does not exist a polynomial-time machine that is able to distinguish the adversary's view in a correct protocol execution from the adversary's view in a protocol execution where all keys shared between honest users are replaced with a fixed message of equal length (which means that the adversary does not learn anything about these keys except for their lengths). It is easy to show that the real Otway-Rees protocol fulfills this property because one could otherwise exploit Theorem 1 to distinguish the ideal cryptographic library from the real one using standard techniques, which would yield a contradiction to the results of [8, 7].

The proof idea is as follows: In the simulatability proof of the cryptographic library, the simulator simulates all keys for which no adversary handle exists with a fixed message since it does not know the appropriate key [7]. Moreover, when run with the ideal system and the simulator, the adversary does not learn any information in the Shannon sense about those symmetric keys for which it does not have a handle [8, 7]. Hence Theorem 1 implies that these statements in particular hold for the secret keys shared between honest users. Now if an adversary $\mathsf{A}_{\mathsf{Dis}}$ existed that violated the above property with not negligible advantage over pure guessing, we could define a distinguisher Dis for the ideal and real library by first triggering $\mathsf{A}_{\mathsf{Dis}}$ as a black-box submachine with the obtained view and by then outputting the guess of $\mathsf{A}_{\mathsf{Dis}}$ as a guess for distinguishing the ideal and the real library. It is easy to show that Dis provides a correct simulation for $\mathsf{A}_{\mathsf{Dis}}$ and hence succeeds in distinguishing the ideal and the real library with not negligible probability.

7 Conclusion

We have proven the Otway-Rees protocol in the real cryptographic setting via a deterministic, provably secure abstraction of a real cryptographic library. Together with composition and preservation theorems from the underlying model, this library allowed us to perform the actual proof effort in a deterministic setting corresponding to a slightly extended Dolev-Yao model. We hope that it paves the way for the actual use of automatic proof tools for this and many similar cryptographically faithful proofs of security protocols.

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 J. Jürjens. Formal eavesdropping and its computational interpretation. In *Proc. 4th International Symposium on Theoretical Aspects of Computer Software (TACS)*, pages 82–94, 2001.
3. 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 *LNCS*, pages 3–22. Springer, 2000.
4. M. Backes. A cryptographically sound Dolev-Yao style security proof of the Otway-Rees protocol. Research Report RZ 3539, IBM Research, 2004.
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 *LNCS*, pages 675–686. Springer, 2003.
6. M. Backes and B. Pfitzmann. A cryptographically sound security proof of the Needham-Schroeder-Lowe public-key protocol. In *Proc. 23rd Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS)*, pages 1–12, 2003.
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)*, 2004.
8. 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/`.
9. 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 *LNCS*, pages 271–290. Springer, 2003.
10. 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 *LNCS*, pages 26–45. Springer, 1998.
11. M. Bellare and C. Namprempre. Authenticated encryption: Relations among notions and analysis of the generic composition paradigm. In *Advances in Cryptology: ASIACRYPT 2000*, volume 1976 of *LNCS*, pages 531–545. Springer, 2000.
12. M. Bellare and P. Rogaway. Encode-then-encipher encryption: How to exploit nonces or redundancy in plaintexts for efficient constructions. In *Advances in Cryptology: ASIACRYPT 2000*, volume 1976 of *LNCS*, pages 317–330. Springer, 2000.
13. D. Bleichenbacher. Chosen ciphertext attacks against protocols based on the RSA encryption standard PKCS. In *Advances in Cryptology: CRYPTO '98*, volume 1462 of *LNCS*, pages 1–12. Springer, 1998.
14. M. Burrows, M. Abadi, and R. Needham. A logic for authentication. Technical Report 39, SRC DIGITAL, 1990.
15. D. E. Denning and G. M. Sacco. Timestamps in key distribution protocols. *Communications of the ACM*, 24(8):533–536, 1981.
16. D. Dolev and A. C. Yao. On the security of public key protocols. *IEEE Transactions on Information Theory*, 29(2):198–208, 1983.
17. D. Fisher. Millions of .Net Passport accounts put at risk. *eWeek*, May 2003. (Flaw detected by Muhammad Faisal Rauf Danka).
18. 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.

19. S. Goldwasser and S. Micali. Probabilistic encryption. *Journal of Computer and System Sciences*, 28:270–299, 1984.
20. S. Goldwasser, S. Micali, and C. Rackoff. The knowledge complexity of interactive proof systems. *SIAM Journal on Computing*, 18(1):186–207, 1989.
21. J. Herzog. *Computational Soundness of Formal Adversaries*. PhD thesis, MIT, 2002.
22. J. Herzog, M. Liskov, and S. Micali. Plaintext awareness via key registration. In *Advances in Cryptology: CRYPTO 2003*, volume 2729 of *LNCS*, pages 548–564. Springer, 2003.
23. R. Impagliazzo and B. M. Kapron. Logics for reasoning about cryptographic constructions. In *Proc. 44th IEEE Symposium on Foundations of Computer Science*, pages 372–381, 2003.
24. F. Javier Thayer, J. C. Herzog, and J. D. Guttman. Honest ideals on strand spaces. In *Proc. 11th IEEE Computer Security Foundations Workshop (CSFW)*, pages 66–77, 1998.
25. R. Kemmerer. Analyzing encryption protocols using formal verification techniques. *IEEE Journal on Selected Areas in Communications*, 7(4):448–457, 1989.
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. C. Meadows. Using narrowing in the analysis of key management protocols. In *Proc. 10th IEEE Symposium on Security & Privacy*, pages 138–147, 1989.
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 *LNCS*, pages 133–151. Springer, 2004.
31. J. K. Millen. The interrogator: A tool for cryptographic protocol security. In *Proc. 5th IEEE Symposium on Security & Privacy*, pages 134–141, 1984.
32. 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*, pages 725–733, 1998.
33. J. Mitchell, M. Mitchell, A. Scedrov, and V. Teague. A probabilistic polynominal-time process calculus for analysis of cryptographic protocols (preliminary report). *Electronic Notes in Theoretical Computer Science*, 47:1–31, 2001.
34. R. Needham and M. Schroeder. Using encryption for authentication in large networks of computers. *Communications of the ACM*, 12(21):993–999, 1978.
35. D. Otway and O. Rees. Efficient and timely mutual authentication. *Operation Systems Review*, 21(1):8–10, 1987.
36. L. Paulson. Mechanized proofs for a recursive authentication protocol. In *Proc. 10th IEEE Computer Security Foundations Workshop (CSFW)*, pages 84–95, 1997.
37. L. Paulson. The inductive approach to verifying cryptographic protocols. *Journal of Cryptology*, 6(1):85–128, 1998.
38. 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.
39. P. Rogaway, M. Bellare, J. Black, and T. Krovetz. OCB: A block-cipher mode of operation for efficient authenticated encryption. In *Proc. 8th ACM Conference on Computer and Communications Security*, pages 196–205, 2001.
40. D. Wagner and B. Schneier. Analysis of the SSL 3.0 protocol. In *Proc. 2nd USENIX Workshop on Electronic Commerce*, pages 29–40, 1996.
41. B. Warinschi. A computational analysis of the Needham-Schroeder-(Lowe) protocol. In *Proc. 16th IEEE Computer Security Foundations Workshop (CSFW)*, pages 248–262, 2003.