Technical Report: Cryptographic Key Secrecy of the Strengthened Yahalom Protocol via a Symbolic Security Proof (Long Version)*

Michael Backes, Birgit Pfitzmann
IBM Zurich Research Laboratory, Switzerland
{mbc,bpf}@zurich.ibm.com
January 17, 2007

Abstract

Symbolic secrecy of exchanged keys is arguably one of the most important notions of secrecy shown with automated proof tools. It means that an adversary restricted to symbolic operations on terms can never get the entire key into its knowledge set. Cryptographic key secrecy essentially means computational indistinguishability between the real key and a random one, given the view of a much more general adversary.

We provide the first proof of cryptographic key secrecy for the strengthened Yahalom protocol, which constitutes one of the most prominent key exchange protocols analyzed by means of automated proof tools. The proof holds in the presence of arbitrary active attacks provided that the protocol is implemented using standard provably secure cryptographic primitives. We exploit recent results on linking symbolic and cryptographic key secrecy in order to perform a symbolic proof of secrecy for the Yahalom protocol in a specific setting that allows us to derive the desired cryptographic key secrecy from the symbolic proof.

1 Introduction

Cryptographic protocols for key establishment are an established technology. Nevertheless, most new networking and messaging stacks come with new protocols for such tasks. Since designing cryptographic protocols is known to be error-prone and, owing to the distributed-system aspects of multiple interleaved protocol runs, security proofs of such protocols are awkward to make for humans, automation of such proofs has been studied almost since cryptographic protocols first emerged. From the start, the actual cryptographic operations in such proofs were idealized into so-called Dolev-Yao models, following [35] with extensions in [36, 48], e.g., see [50, 47, 41, 57, 58, 1, 46, 53]. These models replace cryptography by term algebras, e.g., encrypting a message m twice does not yield a different message from the basic message space but the term $\mathsf{E}(\mathsf{E}(m))$. A typical cancellation rule is $\mathsf{D}(\mathsf{E}(m)) = m$ for all m. It is assumed that even an adversary can only operate on terms by the given operators and by exploiting the given cancellation rules. This assumption, in other words the use of initial models of the given equational specifications, makes it highly nontrivial to know whether results obtained over a Dolev-Yao model are also valid over real cryptography. One therefore calls properties and actions in Dolev-Yao models *symbolic* in contrast to *cryptographic*.

Arguably the most important and most common properties proved symbolically are secrecy properties, as initiated in [35], and in particular key secrecy properties. Symbolically, the secrecy of a key is represented by knowledge sets: The key is secret if the adversary can never get the corresponding symbolic term into

*An earlier version of this work appeared in [21].

its knowledge set. Cryptographically, key secrecy is defined by computational indistinguishability between the real key and a randomly chosen one, given the view of the adversary. Hence symbolic secrecy captures the absence of structural attacks that make the secret as a whole known to the adversary, and because of its simplicity it is accessible to formal proofs tools, while cryptographic secrecy constitutes a more fine-grained notion of secrecy that is much harder to establish.

The Yahalom protocol [33, 54] is one of the most prominent key exchange protocols. Paulson discovered that the original protocol from [33] is insecure and proposed a strengthened variant [54]. This was extensively investigated, e.g., in [54, 37, 32, 29]. However, all existing security proofs are restricted to the Dolev-Yao model. We provide the first security proof of the strengthened Yahalom protocol in the more comprehensive cryptographic sense, i.e., we show that keys exchanged between two honest users are secret in the strong sense of indistinguishability from random keys. This holds in the presence of arbitrary active attacks, provided that the Dolev-Yao abstraction of symmetric encryption is implemented by 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 [31, 30]. Efficient symmetric encryptions schemes provably secure in this sense exist under reasonable assumptions [30, 56].

We achieve this result by exploiting recent work on linking Dolev-Yao models to the standard model of cryptography. We analyze the Yahalom protocol based on the *cryptographic library* of Backes, Pfitzmann, and Waidner [24, 27, 18], which corresponds to a slightly extended Dolev-Yao model that can be faithfully realized using provably secure cryptographic primitives. In combination with a recent result on linking symbolic and cryptographic key secrecy [20], this allows us to perform a symbolic proof of secrecy for the Yahalom protocol and to derive the desired cryptographic key secrecy from that. This is the first symbolic proof of a cryptographic protocol that can be exploited to derive cryptographic secrecy for the exchanged keys. (Another such proof was conducted concurrently and independently by Canetti and Herzog, cf. below.)

Further Related Work. Cryptographic underpinnings of a Dolev-Yao model were first addressed by Abadi and Rogaway [3]. They only handled passive adversaries and symmetric encryption. The protocol language and security properties were extended in [2, 42, 38], but still only for passive adversaries. This excludes most of the typical ways of attacking protocols. A full cryptographic justification for a Dolev-Yao model, i.e., for arbitrary active attacks and within arbitrary surrounding interactive protocols, was first given in [24] with extensions in [27, 18]. Based on that Dolev-Yao model, the well-known Needham-Schroeder-Lowe and Otway-Rees protocols as well as a variant of the 3KP payment system were proved in [17, 5, 7]. The first and the third result are entirely authentication proofs and hence does not have to reason about secrecy aspects. The second one contains a key secrecy property but this was reformulated by hand into a (considerably weaker) integrity property so that the integrity preservation theorem could be used. Moreover, after the initial publication of the current work, a computationally sound security proof of basic and public-key Kerberos [6] and of a widely deployed Web Services protocol [14] was achieved in a similar manner. In general, our result can moreover be seen as a demonstration of the usefulness of the cryptographic library [24], their extensions [18, 27], and the corresponding general theorems for linking symbolic and cryptographic properties based on this library [11, 23, 15, 16, 20, 4] for establishing secrecy properties in a cryptographically sound manner. Moreover, automated proof support for the cryptographic library was recently achieved in [59].

Laud [43] 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. [39] and Micciancio and Warinschi [49] have recently also given a cryptographic underpinning under active attacks. Their results are narrower than that in [24] since they are specific for

public-key encryption, but consider slightly simpler real implementations; 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 [49] further points out several possible extensions of their work which all already exist in the earlier work of [24]. Since computational soundness has become a highly active line of research, we exemplarily list further recent results in this area without going into further details [12, 25, 26, 22, 10, 19, 8, 28, 13, 9].

Efforts are also under way to formulate syntactic calculi for dealing with probabilism and polynomial-time considerations, in particular [51, 44, 52, 40] and, as a second step, to encode them into proof tools. This approach can not yet handle protocols with any degree of automation. It is complementary to the approach of proving simple deterministic abstractions of cryptography and working with those wherever cryptography is only used in a blackbox way.

Concurrently and independently to our work, Canetti and Herzog [34] have linked ideal functionalities for mutual authentication and key exchange protocols to corresponding representations in a formal language. They apply their techniques to the Needham-Schroeder-Lowe protocol by considering the exchanged nonces as secret keys. Their work is restricted to the mentioned functionalities and in contrast to the universally composable library [24] hence does not address soundness of Dolev-Yao models in their usual generality. The considered language does not allow loops and offers public-key encryption as the only cryptographic operation (symmetric encryption is not considered although the language is used to reason about symmetric keys). Moreover, their approach to define a mapping between ideal and real traces following the ideas of [49] only captures trace-based properties (i.e., integrity properties); reasoning about secrecy properties additionally requires ad-hoc and functionality-specific arguments.

2 The Strengthened Yahalom Protocol

The Yahalom protocol [33] and its strengthened variant [54] are four-step protocols for establishing a shared secret encryption key between two users. The protocol relies on a distinguished trusted party T, and it is assumed that every user u initially shares a secret key K_{ut} with T. Expressed in the typical protocol notation as in, e.g., [45], the strengthened Yahalom works as follows.[1]

$$
\begin{array}{lrcl}
1. & u \to v & : & u, N_u \\
2. & v \to \mathsf{T} & : & v, N_v, (u, N_u)_{K_{vt}} \\
3. & \mathsf{T} \to u & : & N_v, (v, K_{uv}, N_u)_{K_{ut}}, (u, v, K_{uv}, N_v)_{K_{vt}} \\
4. & u \to v & : & (u, v, K_{uv}, N_v)_{K_{vt}}.
\end{array}
$$

User u seeks to share a new session key with user v. It generates a nonce N_u and sends it to v together with its identity (first message). Next, v generates a new nonce N_v, creates a new message containing the identity u and the nonce N_u, and encrypts it with the key it shares with T. Then v sends its identity, its nonce N_v, and the encryption to the trusted party (second message). Now T decrypts the encryption yielding the identity of u and the nonce N_u, generates a fresh key K_{uv} for u and v, generates a message according to the protocol description, and sends it to u (third message). Then u decrypts the first encryption and tests whether the contained nonce is the one it sent to v before, i.e., to the identity that is contained in this encryption. If so, it forwards the second encryption to v (fourth message) and terminates the protocol by outputting a handle to the shared secret key K_{uv} to its user. Finally v decrypts this message, obtains the shared key K_{uv}, and tests

[1] The strengthened protocol presented in [54] further contains an encryption of the nonce N_v with K_{uv} in the fourth term to guarantee entity authentication of u to v. We omitted this encryption to concentrate on the key secrecy property of the core key exchange functionality.

Algorithm 1 Evaluation of User Inputs in $\mathsf{M}_u^{\mathsf{Ya}}$ with $u \neq \mathsf{T}$ (Protocol Start)

Input: $(\mathsf{new\_prot}, \mathsf{Yahalom}, v)$ at $\mathsf{KE\_in}_u?$ with $v \in \{1, \ldots, n\} \setminus \{u\}$.

1: $n_u^{\mathsf{hnd}} \leftarrow \mathsf{gen\_nonce}()$.
2: $Nonce_u := Nonce_u \cup \{(n_u^{\mathsf{hnd}}, v, 1)\}$.
3: $u^{\mathsf{hnd}} \leftarrow \mathsf{store}(u)$.
4: $m_1^{\mathsf{hnd}} \leftarrow \mathsf{list}(u^{\mathsf{hnd}}, n_u^{\mathsf{hnd}})$.
5: $\mathsf{send\_i}(v, m_1^{\mathsf{hnd}})$.

if the message contains its own identity and the contained nonce was previously sent to T. If so, it outputs a handle to the shared key K_{uv} to its user and terminates the protocol.

2.1 Protocol Details with the Dolev-Yao-style Cryptographic Library

Almost all formal proof techniques for protocols first need a reformulation of the protocol into a more detailed version than the four steps above. These details include necessary tests on received messages, the types and generation rules for values like u and N_u, and a surrounding framework specifying the number of participants, the possibilities of multiple protocol runs, and the adversary capabilities. The same is true when using the Dolev-Yao-style cryptographic library from [24], i.e., it plays a similar role in our proof as "the CSP Dolev-Yao model" or "the inductive-approach Dolev-Yao model" in other proofs. We now present the protocol details in this framework, and explain general aspects of the framework in Section 2.2.

We write ":=" for deterministic assignment, and $\downarrow$ is an error element available as an addition to the domains and ranges of all functions and algorithms. The framework is automata-based, i.e., protocols are executed by interacting machines, and event-based, i.e., machines react on received inputs. By $\mathsf{M}_i^{\mathsf{Ya}}$ we denote the Yahalom machine for a participant i; it can act in the roles of both u and v above.

The first type of input that $\mathsf{M}_i^{\mathsf{Ya}}$ can receive is a start message $(\mathsf{new\_prot}, \mathsf{Yahalom}, v)$ from its user denoting that it should start a protocol run with user v. The number of users is called n.[2] User inputs are distinguished from network inputs by arriving at a so-called port $\mathsf{KE\_in}_u?$. The "?" for input ports follows the CSP convention, and "KE" stands for key exchange because the user interface is the same for all key exchange protocols. The reaction on this input, i.e., the sending of the first message, is described in Algorithm 1. The command $\mathsf{gen\_nonce}$ generates the nonce. $\mathsf{M}_u^{\mathsf{Ya}}$ stores the resulting so-called *handle* n_u^{hnd} (a local name that this machine has for the corresponding term) in a set $Nonce_u$ for future comparison together with the identity v and an indicator that this nonce was generated and stored by u in the first step. The set $Nonce_u$ formally consists of triples (n^{hnd}, w, j) where n^{hnd} is a handle, $w \in \{1, \ldots, n\} \setminus \{u\}$, and $j \in \{1, 2, 3, 4\}$. A triple (n^{hnd}, w, j) means that $\mathsf{M}_u^{\mathsf{Ya}}$ stored the handle n^{hnd} in the j-th protocol step in a session with w. The command store inputs arbitrary application data into the cryptographic library, here the user identity u. The command list forms a list, and the final command $\mathsf{send\_i}$ means that $\mathsf{M}_u^{\mathsf{Ya}}$ 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 $\mathsf{M}_v^{\mathsf{Ya}}$ or performing Dolev-Yao-style algebraic operations on the term). The superscript $^{\mathsf{hnd}}$ on most parameters denotes that these are handles, i.e., the users obtain local names for the corresponding terms. This is an important aspect of [24] because it allows the same protocol description to be implemented once with Dolev-Yao-style idealized cryptography and once with real cryptography. The four commands we saw so far and their input and output domains belong to the interface (in the same sense as, e.g., a Java interface) of the underlying cryptographic library. This interface is implemented by both the idealized and the real version. In the first case, the handles are local names

[2] The set of users is $\{1, \ldots, n\}$ and the Yahalom protocol is designed such that $\mathsf{T} \notin \{1, \ldots, n\}$ where T denotes the trusted party.

of Dolev-Yao-style terms, in the second case of real cryptographic bitstrings, on which the adversary can perform arbitrary bit manipulations. We say more about these two implementations below.

We now define formally how protocol machines and the trusted party behave upon receiving an input from the network. To increase readability we augment the algorithm with 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. Ciphertexts are labeled c, the encrypted lists are labeled l, and terms whose type is still unknown are labelled t or x, all with suitable sub- and superscripts.

Network inputs arrive at port $\mathsf{out}_u?$ and are of the form $(v, u, \mathsf{i}, m^{\mathsf{hnd}})$ where v is the supposed sender, i denotes that the channel is insecure, and m^{hnd} is a handle to a list. The port $\mathsf{out}_u?$ is connected to the cryptographic library, whose two implementations represent the obtained Dolev-Yao-style term or real bitstring, respectively, to the protocol in a unified way by the handle m^{hnd}.

The behavior of $\mathsf{M}_u^{\mathsf{Ya}}$ for $u \in \{1, \ldots, n\}$ is given in Algorithm 2. Upon receiving a network input $\mathsf{M}_u^{\mathsf{Ya}}$ first decomposes the obtained message and checks if it 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 querying the type of the first component and by looking up the respective nonce in the set $Nonce_u$. After that, $\mathsf{M}_u^{\mathsf{Ya}}$ checks if the obtained message is indeed a suitably constructed message for the particular step by exploiting the content of $Nonce_u$. If so, $\mathsf{M}_u^{\mathsf{Ya}}$ 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. The behavior of $\mathsf{M}_{\mathsf{T}}^{\mathsf{Ya}}$ upon receiving an input $(v, \mathsf{T}, \mathsf{i}, m^{\mathsf{hnd}})$ from the cryptographic library at port $\mathsf{out}_{\mathsf{T}}?$ is defined similarly.

The formal definition of the behavior of the trusted party is given in Algorithm 3. We omit an information description.

A machine should immediately abort the handling of the current input if a cryptographic command does not yield the desired result, e.g., if a decryption fails. This is captured by the following convention.

Convention 1 *If* $\mathsf{M}_u^{\mathsf{Ya}}$ *receives* $\downarrow$ *as the answer of the cryptographic library to a command, then* $\mathsf{M}_u^{\mathsf{Ya}}$ *aborts the execution of the current algorithm, except for the command types* list_proj *or* send_i.

2.2 Overall Framework and Adversary Model

The framework that determines how machines such as our Yahalom machines and the machines of the idealized or real cryptographic library execute is taken from [55]. The basis is an asynchronous probabilistic execution model with distributed scheduling and with a well-defined Turing-machine refinement for complexity considerations. We already used implicitly above that for term construction and parsing commands to the cryptographic library, so-called local scheduling is defined, i.e., a result is returned immediately. The idealized or real network sending via this library, however, is scheduled by the adversary.

When protocol machines such as $\mathsf{M}_u^{\mathsf{Ya}}$ there is no guarantee that all these machines are correct. A trust model determines for what subsets $\mathcal{H}$ of $\{1, \ldots, n, \mathsf{T}\}$ we want to guarantee anything; here these are the subsets that contain at least the trusted party: We prove secrecy of keys shared by u and v whenever $u, v \in \mathcal{H}$ and thus whenever $\mathsf{M}_u^{\mathsf{Ya}}$ and $\mathsf{M}_v^{\mathsf{Ya}}$ are correct. Incorrect machines disappear and are replaced by the adversary. Each set of potential correct machines together with its user interface is called a structure, and the set of these structures is called the system. When considering the security of a structure, an arbitrary probabilistic machine H is connected to the user interface to represent all users, and an arbitrary probabilistic machine A is connected to the remaining free ports (typically the network) and to H to represent the adversary. In polynomial-time security proofs, H and A are polynomial-time. This setting implies that any number of

Algorithm 2 Evaluation of Network Inputs in $\mathsf{M}_u^{\mathsf{Ya}}$ with $u \neq \mathsf{T}$

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

$t_i^{\mathsf{hnd}} \leftarrow \mathsf{list\_proj}(m^{\mathsf{hnd}}, i)$ for $i = 1, 2, 3$.
$type_{t_1^{\mathsf{hnd}}} \leftarrow \mathsf{get\_type}(t_1^{\mathsf{hnd}})$.
if $type_{t_1^{\mathsf{hnd}}} = \mathsf{data} \wedge v \neq \mathsf{T} \wedge \forall j\colon (t_2^{\mathsf{hnd}}, v, j) \notin Nonce_u$ **then** {First Message is input}
  $type_{t_2^{\mathsf{hnd}}} \leftarrow \mathsf{get\_type}(t_2^{\mathsf{hnd}})$.
  $t_1 \leftarrow \mathsf{retrieve}(t_1^{\mathsf{hnd}})$.
  if $t_1 \neq v \vee type_{t_2^{\mathsf{hnd}}} \neq \mathsf{nonce}$ **then** Abort **end if**
  $n_u^{\mathsf{hnd}} \leftarrow \mathsf{gen\_nonce}()$.
  $Nonce_u := Nonce_u \cup \{(n_u^{\mathsf{hnd}}, v, 2)\}$.
  $u^{\mathsf{hnd}} \leftarrow \mathsf{store}(u)$.
  $l_2^{\mathsf{hnd}} \leftarrow \mathsf{list}(t_1^{\mathsf{hnd}}, t_2^{\mathsf{hnd}})$. $\{l_2^{\mathsf{hnd}} \approx (v, N_v)\}$
  $c_2^{\mathsf{hnd}} \leftarrow \mathsf{sym\_encrypt}(skse_{u,\mathsf{T}}^{\mathsf{hnd}}, l_2^{\mathsf{hnd}})$. $\{c_2^{\mathsf{hnd}} \approx (v, N_v)_{K_{ut}}\}$
  $m_2^{\mathsf{hnd}} \leftarrow \mathsf{list}(u^{\mathsf{hnd}}, n_u^{\mathsf{hnd}}, c_2^{\mathsf{hnd}})$. $\{m_2^{\mathsf{hnd}} \approx (u, N_u, (v, N_v)_{K_{ut}})\}$
  $\mathsf{send\_i}(\mathsf{T}, m_2^{\mathsf{hnd}})$.
else if $type_{t_1^{\mathsf{hnd}}} = \mathsf{nonce} \wedge v = \mathsf{T}$ **then** {Third Message is input}
  $l^{\mathsf{hnd}} \leftarrow \mathsf{sym\_decrypt}(skse_{u,\mathsf{T}}^{\mathsf{hnd}}, t_2^{\mathsf{hnd}})$. $\{l^{\mathsf{hnd}} \approx (v, K_{uv}, N_u)\}$
  $x_i^{\mathsf{hnd}} \leftarrow \mathsf{list\_proj}(l^{\mathsf{hnd}}, i)$ for $i = 1, 2, 3$.
  $x_1 \leftarrow \mathsf{retrieve}(x_1^{\mathsf{hnd}})$.
  $type_{x_2^{\mathsf{hnd}}} \leftarrow \mathsf{get\_type}(x_2^{\mathsf{hnd}})$.
  $type_{t_3^{\mathsf{hnd}}} \leftarrow \mathsf{get\_type}(t_3^{\mathsf{hnd}})$.
  if $(x_3^{\mathsf{hnd}}, x_1, 1) \notin Nonce_u \vee type_{x_2^{\mathsf{hnd}}} \neq \mathsf{skse} \vee type_{t_3^{\mathsf{hnd}}} \neq \mathsf{symenc}$ **then** Abort **end if**
  $Nonce_u := (Nonce_u \setminus \{(x_3^{\mathsf{hnd}}, x_1, 1)\}) \cup \{(x_3^{\mathsf{hnd}}, x_1, 3)\}$.
  $m_4^{\mathsf{hnd}} \leftarrow \mathsf{list}(t_3^{\mathsf{hnd}})$. $\{m_4^{\mathsf{hnd}} \approx (u, v, K_{uv}, N_v)_{K_{vt}}\}$
  Output $(\mathsf{ok\_initiator}, \mathsf{Yahalom}, x_1, x_2^{\mathsf{hnd}})$ at $\mathsf{KE\_out}_u$!.
  $\mathsf{send\_i}(x_1, m_4^{\mathsf{hnd}})$.
else if $type_{t_1^{\mathsf{hnd}}} = \mathsf{symenc} \wedge v \neq \mathsf{T}$ **then** {Fourth Message is input}
  $l^{\mathsf{hnd}} \leftarrow \mathsf{sym\_decrypt}(skse_{u,\mathsf{T}}^{\mathsf{hnd}}, t_1^{\mathsf{hnd}})$. $\{l^{\mathsf{hnd}} \approx (v, u, K_{uv}, N_u)\}$
  $x_i^{\mathsf{hnd}} \leftarrow \mathsf{list\_proj}(l^{\mathsf{hnd}}, i)$ for $i = 1, 2, 3, 4$.
  $x_i \leftarrow \mathsf{retrieve}(x_i^{\mathsf{hnd}})$ for $i = 1, 2$.
  $type_{x_3^{\mathsf{hnd}}} \leftarrow \mathsf{get\_type}(x_3^{\mathsf{hnd}})$.
  if $x_1 \neq v \vee x_2 \neq u \vee type_{x_3^{\mathsf{hnd}}} \neq \mathsf{skse} \vee (x_4^{\mathsf{hnd}}, x_1, 2) \notin Nonce_u$ **then** Abort **end if**
  $Nonce_u := (Nonce_u \setminus \{(x_4^{\mathsf{hnd}}, x_1, 2)\}) \cup \{(x_4^{\mathsf{hnd}}, x_1, 4)\}$.
  Output $(\mathsf{ok\_responder}, \mathsf{Yahalom}, x_1, x_3^{\mathsf{hnd}})$ at $\mathsf{KE\_out}_u$!.
else
  Abort
end if

Algorithm 3 Behavior of the Trusted Party $\mathsf{M}_{\mathsf{T}}^{\mathsf{Ya}}$

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

1: $t_i^{\mathsf{hnd}} \leftarrow \mathsf{list\_proj}(m^{\mathsf{hnd}}, i)$ for $i = 1, 2, 3$.
2: $t_1 \leftarrow \mathsf{retrieve}(t_1^{\mathsf{hnd}})$. $\{t_1 \approx v\}$
3: $type_{t_2^{\mathsf{hnd}}} \leftarrow \mathsf{get\_type}(t_2^{\mathsf{hnd}})$.
4: $l^{\mathsf{hnd}} \leftarrow \mathsf{sym\_decrypt}(skse_{\mathsf{T},v}^{\mathsf{hnd}}, t_3^{\mathsf{hnd}})$. $\{l^{\mathsf{hnd}} \approx (u, N_u)\}$
5: $x_i^{\mathsf{hnd}} \leftarrow \mathsf{list\_proj}(l^{\mathsf{hnd}}, i)$ for $i = 1, 2$.
6: $x_1 \leftarrow \mathsf{retrieve}(x_1^{\mathsf{hnd}})$. $\{x_1 \approx u\}$
7: $type_{x_2^{\mathsf{hnd}}} \leftarrow \mathsf{get\_type}(x_2^{\mathsf{hnd}})$.
8: **if** $type_{t_2^{\mathsf{hnd}}} \neq \mathsf{nonce} \vee type_{x_2^{\mathsf{hnd}}} \neq \mathsf{nonce} \vee t_1 \neq v \vee x_1 \notin \{1, \ldots, n\} \setminus \{v\}$ **then** Abort **end if**
9: $skse^{\mathsf{hnd}} \leftarrow \mathsf{gen\_symenc\_key}()$. $\{skse^{\mathsf{hnd}} \approx K_{uv}\}$
10: $l_3^{(1)^{\mathsf{hnd}}} \leftarrow \mathsf{list}(t_1^{\mathsf{hnd}}, skse^{\mathsf{hnd}}, x_2^{\mathsf{hnd}})$. $\{l_3^{(1)^{\mathsf{hnd}}} \approx (v, K_{uv}, N_u)\}$
11: $c_3^{(1)^{\mathsf{hnd}}} \leftarrow \mathsf{sym\_encrypt}(skse_{\mathsf{T},x_1}^{\mathsf{hnd}}, l_3^{(1)^{\mathsf{hnd}}})$. $\{c_3^{(1)^{\mathsf{hnd}}} \approx (v, K_{uv}, N_u)_{K_{ut}}\}$
12: $l_3^{(2)^{\mathsf{hnd}}} \leftarrow \mathsf{list}(x_1^{\mathsf{hnd}}, t_1^{\mathsf{hnd}}, skse^{\mathsf{hnd}}, t_2^{\mathsf{hnd}})$. $\{l_3^{(2)^{\mathsf{hnd}}} \approx (u, v, K_{uv}, N_v)\}$
13: $c_3^{(2)^{\mathsf{hnd}}} \leftarrow \mathsf{sym\_encrypt}(skse_{\mathsf{T},v}^{\mathsf{hnd}}, l_3^{(2)^{\mathsf{hnd}}})$. $\{c_3^{(2)^{\mathsf{hnd}}} \approx (u, v, K_{uv}, N_v)_{K_{vt}}\}$
14: $m_3^{\mathsf{hnd}} \leftarrow \mathsf{list}(t_2^{\mathsf{hnd}}, c_3^{(1)^{\mathsf{hnd}}}, c_3^{(2)^{\mathsf{hnd}}})$. $\{m_3^{\mathsf{hnd}} \approx (N_v, (v, K_{uv}, N_u)_{K_{ut}}, (u, v, K_{uv}, N_v)_{K_{vt}})\}$
15: $\mathsf{send\_i}(x_1, m_3^{\mathsf{hnd}})$.

concurrent protocol runs with the honest participants and the adversary are considered because H and A can arbitrarily interleave protocol start inputs $(\mathsf{new\_prot}, \mathsf{Yahalom}, v)$ with the delivery of network messages.

For a set $\mathcal{H}$ of honest participants, the user interface of the Yahalom protocol machines is $S_{\mathcal{H}}^{\mathsf{KE}} := \{\mathsf{KE\_in}_u?, \mathsf{KE\_out}_u! \mid u \in \mathcal{H} \setminus \{\mathsf{T}\}\}$. The ideal and real Yahalom protocol serving this interface differ only in the cryptographic library, i.e., the Yahalom machines either rely on a set $\hat{M}_{\mathcal{H}}^{\mathsf{cry}} := \{\mathsf{M}_{u,\mathcal{H}}^{\mathsf{cry}} \mid u \in \mathcal{H}\}$ of real cryptographic machines or an ideal machine $\mathsf{TH}_{\mathcal{H}}^{\mathsf{cry}}$ called *trusted host*. With $\hat{M}_{\mathcal{H}}^{\mathsf{Ya}} := \{\mathsf{M}_u^{\mathsf{Ya}} \mid u \in \mathcal{H}\}$, the ideal system is $Sys^{\mathsf{Ya,id}} := \{(\hat{M}_{\mathcal{H}}^{\mathsf{Ya}} \cup \{\mathsf{TH}_{\mathcal{H}}^{\mathsf{cry}}\}, S_{\mathcal{H}}^{\mathsf{KE}}) \mid \{\mathsf{T}\} \subseteq \mathcal{H} \subseteq \{1, \ldots, n, \mathsf{T}\}\}$, and the real system is $Sys_{\mathcal{SE}}^{\mathsf{Ya,real}} := \{(\hat{M}_{\mathcal{H}}^{\mathsf{Ya}} \cup \hat{M}_{\mathcal{H}}^{\mathsf{cry}}, S_{\mathcal{H}}^{\mathsf{KE}}) \mid \{\mathsf{T}\} \subseteq \mathcal{H} \subseteq \{1, \ldots, n, \mathsf{T}\}\}$, where $\mathcal{SE}$ denotes the symmetric encryption scheme used.

On Polynomial Runtime. In order to be valid users of the real cryptographic library, the machines $\mathsf{M}_u^{\mathsf{Ya}}$ have to be polynomial-time. We therefore define that each machine $\mathsf{M}_u^{\mathsf{Ya}}$ maintains explicit polynomial bounds on the accepted message lengths and the number of inputs accepted at each port. As this is done exactly as in the cryptographic library, we omit the rigorous write-up.

3 The Key Secrecy Property

In the following, we formalize the key secrecy property of the ideal and real Yahalom protocols. The property is an instantiation of a general key secrecy definition for arbitrary protocols based on the ideal cryptographic library. It was introduced in [20] and is symbolic, based on the typical notion that a term is not an element of the adversary's knowledge set. In the given Dolev-Yao-style library, the adversary's knowledge set is the set of all terms to which the adversary has a handle.

We start this section by defining the possible states of the ideal and real cryptographic library as needed for formulating the property, and then define the property.

3.1 Overview and States of the Ideal Cryptographic Library

The ideal cryptographic library administrates Dolev-Yao-style terms and allows each user to operate on them via handles, i.e., via local names specific to this user. The handles also contain the information that knowledge sets give in other Dolev-Yao formalizations: The set of terms that a participant u knows, including $u = \mathsf{a}$ for the adversary, is the set of terms with a handle for u. The terms are typed; for instance, decryption only succeeds on ciphertexts and projection only on lists. Moreover, the terms are globally numbered by a so-called index. Each term is represented by its type (i.e., root node) and its first-level arguments, which can be indices of earlier terms. This enables easy distinction of, e.g., which of many nonces is encrypted in a larger term. These global indices are never visible at the user interface. The indices and the handles for each participant are generated by one counter each.

The data structure storing the terms in [24] is a database D. Generally, 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. Adding an entry x to D is abbreviated $D :\Leftarrow x$. Moreover, we write the list operation as $l := (x_1, \ldots, x_j)$, and argument retrieval as $l[i]$ with $l[i] = \downarrow$ if $i > j$. In the specific term database D, each entry x can have arguments $(ind, type, arg, hnd_{u_1}, \ldots, hnd_{u_m}, hnd_{\mathsf{a}}, len)$, for $\{u_1, \ldots, u_m\} = \mathcal{H}$ and the arguments have the following types and meaning:

- $x.ind$ is the global index of an entry. Its type $\mathcal{INDS}$ is isomorphic to $\mathbb{N}$ and distinguishes 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 $D[ind = i]$.
- $x.type \in typeset$ is the *type* of x. We use the types nonce, list, data (for payload data), skse and pkse (for symmetric encryption keys and corresponding "public-key identifiers", see below), and symenc (for symmetric encryptions).
- $x.arg = (a_1, a_2, \ldots, a_j)$ is a possibly empty list of arguments. Arguments of type $\mathcal{INDS}$ are indices of other entries (subterms); 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, where $x.hnd_u = \downarrow$ means that u does not know this entry and $\mathcal{HNDS}$ is 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.

The machine $\mathsf{TH}_{\mathcal{H}}$ has a counter $size \in \mathcal{INDS}$ for the current size of D and counters $curhnd_u$ (current handle) for the handles, all initialized with 0.

In order to capture that keys shared between users and the trusted 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, by $skse^{\mathsf{hnd}}_{u,v}$.

3.2 The Real Cryptographic Library

In the real implementation of the cryptographic library, each user has its own machine. This machine contains a database D_u with only three main attributes: the handle hnd_u for this user u, the real cryptographic bitstring $word$, and the type $type$. The users can use exactly the same commands as with the ideal library, e.g., en- or decrypt a message etc. These commands now trigger real cryptographic operations. The operations essentially use standard cryptographically secure primitives, but with certain additional tagging,

randomization etc. Send commands now trigger the actual sending of bitstrings between machines and/or to the adversary.

3.3 Definition of the Key Secrecy Property

The first step towards defining symbolic key secrecy is to consider one state of the ideal Dolev-Yao-style library and to define that a handle points to a symmetric key, that the key is symbolically unknown to the adversary, and that it has not been used for encryption or authentication. These are the symbolic conditions under which we can hope to prove that the corresponding real key is indistinguishable from a fresh random key for the adversary. Note that the operations that the Yahalom protocol performs on new keys are allowed in this sense. For Condition (3) in the definition, note that the arguments of a ciphertext term are (l, pk) where l is the plaintext index and pk the index of the public tag of the secret key, with $pk = sk - 1$ for the secret key index.

Definition 3.1 *(Symbolically Secret Encryption Keys [20])* Let $\{\mathsf{T}\} \subseteq \mathcal{H} \subseteq \{1, \ldots, n, \mathsf{T}\}$, a database state D of $\mathsf{TH}^{\mathsf{cry}}_{\mathcal{H}}$, and a pair $(u, l^{\mathsf{hnd}}) \in \mathcal{H} \times \mathcal{HNDS}$ of a user and a handle be given. Let $i := D[hnd_u = l^{\mathsf{hnd}}].ind$ be the corresponding database index. The *term under* (u, l^{hnd}) (1) *is a symmetric encryption key* iff $D[i].type = \mathsf{skse}$, (2) *is symbolically unknown (to the adversary)* iff $D[i].hnd_{\mathsf{a}} = \downarrow$, (3) *has not been used for encryption*, or short *is unused*, iff for all indices $j \in \mathbb{N}$ we have $D[j].type = \mathsf{symenc} \Rightarrow D[j].arg[2] \neq i - 1$, and (4) *is a symbolically secret key* iff it has the three previous properties. $\diamond$

A secret-key belief function is a general way to designate the keys whose secrecy should be proved. The underlying theory from [20] is based on such functions. We instantiate them for the Yahalom protocol and thus essentially for all individual key exchange protocols. A secret key belief function maps the user view to a set of triples (u, l^{hnd}, t) of a user, a handle, and a type, pointing to the supposedly secret keys. For the Yahalom protocol, we define secret-key belief functions $\mathsf{seckeys\_initiator\_Ya}$ for the initiator and $\mathsf{seckeys\_responder\_Ya}$ for the responder that designate the exchanged keys.

Definition 3.2 *(Secret-key Belief Functions for the Yahalom Protocol)* A *secret-key belief function* for a set $\mathcal{H}$ is a function $\mathsf{seckeys}$ that maps each view $view$ of the user to an element of $(\mathcal{H} \times \mathcal{HNDS} \times \{\mathsf{skse}\})^*$.

The *secret-key belief functions* $\mathsf{seckeys\_initiator\_Ya}$ and $\mathsf{seckeys\_responder\_Ya}$ of the Yahalom protocol map each element $(\mathsf{ok\_initiator}, \mathsf{Yahalom}, v, skse^{\mathsf{hnd}})$ respectively $(\mathsf{ok\_responder}, \mathsf{Yahalom}, v, skse^{\mathsf{hnd}})$ of $view$ arriving at port $\mathsf{KE\_out}_u?$ in the users view to $(u, skse^{\mathsf{hnd}}, \mathsf{skse})$ if $u \in \mathcal{H}$, and to ϵ otherwise. Elements of $view$ that are not of this form are also mapped to ϵ. $\diamond$

We now define symbolic key secrecy for such a function. In addition to the conditions for individual keys, we require that all elements point to different terms, so that we can expect the corresponding list of cryptographic keys to be entirely random.

Definition 3.3 *(Symbolic Key Secrecy Generally and for the Yahalom Protocol)* Let a user H^* suitable for a structure $(\{\mathsf{TH}^{\mathsf{cry}}_{\mathcal{H}}\}, S^{\mathsf{cry}}_{\mathcal{H}})$ of the cryptographic library $Sys^{\mathsf{cry,id}}$ and a secret-key belief function $\mathsf{seckeys}$ for $\mathcal{H}$ be given. The ideal cryptographic library with this user *keeps the keys in* $\mathsf{seckeys}$ *strictly symbolically secret* iff for all configurations $conf = (\{\mathsf{TH}^{\mathsf{cry}}_{\mathcal{H}}\}, S^{\mathsf{cry}}_{\mathcal{H}}, \mathsf{H}, \mathsf{A})$ of this structure, every $v \in view_{conf}(\mathsf{H})$, and every element $(u_i, l_i^{\mathsf{hnd}}, t_i)$ of the set $\mathsf{seckeys}(v)$, the term under $(u_i, l_i^{\mathsf{hnd}})$ is a symbolically secret key of type t_i, and $D[hnd_{u_i} = l_i^{\mathsf{hnd}}].ind \neq D[hnd_{u_j} = l_j^{\mathsf{hnd}}].ind$ for all $i \neq j$.

The ideal Yahalom protocol *keeps the exchanged keys of honest users strictly symbolically secret* iff the ideal cryptographic library keeps the keys in $\mathsf{seckeys\_initiator\_Ya}$ and $\mathsf{seckeys\_responder\_Ya}$ strictly symbolically secret with all users H^* that are the combination of the machines $\mathsf{M}^{\mathsf{Ya}}_u$ for $u \in \mathcal{H}$ and a user H of those machines. $\diamond$

General cryptographic key secrecy requires that no polynomial-time adversary can distinguish the keys designated by the function seckeys from fresh keys. The cryptographic key secrecy of the Yahalom protocol is the instantiation for seckeys_initiator_Ya and seckeys_responder_Ya and the configurations of the Yahalom protocol.

Definition 3.4 *(Cryptographic Key Secrecy Generally and for the Yahalom Protocol)* Let a polynomial-time configuration $conf = (\hat{M}_{\mathcal{H}}^{\mathsf{cry}}, S_{\mathcal{H}}^{\mathsf{cry}}, \mathsf{H}, \mathsf{A})$ of the real cryptographic library $Sys_{\mathcal{SE}}^{\mathsf{cry,real}}$ and a secret-key belief function seckeys for $\mathcal{H}$ be given. Let $\mathsf{gen}_{\mathsf{SE}}$ denote the key generation algorithm. This configuration *keeps the keys in* seckeys *cryptographically secret* iff for all probabilistic-polynomial time algorithms Dis (the distinguisher), we have

$$|\mathsf{Pr}[\mathsf{Dis}(1^k, va, keys_{real}) = 1] - \mathsf{Pr}[\mathsf{Dis}(1^k, va, keys_{fresh}) = 1]| \in NEGL$$

where $NEGL$ denotes the negligible function of the security parameter k and the used random variables are defined as follows: For $r \in run_{conf}$, let $va := view_{conf}(\mathsf{A})(r)$ be the view of the adversary, let $(u_i, l_i^{\mathsf{hnd}}, t_i)_{i=1,\ldots,n} := \mathsf{seckeys}(view_{conf}(\mathsf{H})(r))$ be the user-handle-type triples of presumably secret keys, and let the keys be $keys_{real} := (sk_i)_{i=1,\ldots,n}$ with

$$sk_i \quad := \quad D_{u_i}[hnd_{u_i} = l_i^{\mathsf{hnd}}].word \text{ if } D_{u_i}[hnd_{u_i} = l_i^{\mathsf{hnd}}].type = t_i, \text{ else } \epsilon;$$

and $keys_{fresh} := (sk_i')_{i=1,\ldots,n}$ with $sk_i' \leftarrow \mathsf{gen}_{\mathsf{A}}(1^k)$ if $t_i = \mathsf{ska}$, else $sk_i' \leftarrow \epsilon$.

A polynomial-time configuration $(\hat{M}_{\mathcal{H}}^{\mathsf{cry}} \cup \hat{M}_{\mathcal{H}}^{\mathsf{Ya}}, S_{\mathcal{H}}^{\mathsf{KE}}, \mathsf{H}, \mathsf{A})$ of the real Yahalom protocol $Sys^{\mathsf{Ya,real}}$ *keeps the exchanged keys of honest users cryptographically secret* iff the configuration $(\hat{M}_{\mathcal{H}}^{\mathsf{cry}}, S_{\mathcal{H}}^{\mathsf{cry}}, \{\mathsf{H}\} \cup \hat{M}_{\mathcal{H}}^{\mathsf{Ya}}, \mathsf{A})$ keeps the keys in seckeys_initiator_Ya and seckeys_responder_Ya cryptographically secret. ◇

The following theorem captures the security of the Yahalom protocol.

Theorem 3.1 *(Security of the Yahalom Protocol)* The ideal Yahalom system $Sys^{\mathsf{Ya,id}}$ from Section 2.2 keeps the exchanged keys of honest users strictly symbolically secret, and all polynomial-time configurations of the real system $Sys^{\mathsf{Ya,real}}$ keep the exchanged keys of honest users cryptographically secret. □

4 Proof of the Cryptographic Realization from the Idealization

As discussed in the introduction, the idea of our approach is to prove Theorem 3.1 for the protocol using the ideal Dolev-Yao-style cryptographic library. Then the result for the real system follows automatically. The notion that a system Sys_1 securely implements another system Sys_2 in the sense of reactive simulatability (recall the introduction), is written $Sys_1 \geq_{\mathsf{sec}}^{\mathsf{poly}} Sys_2$ (in the computational case). The main result of [24, 27, 18] is therefore

$$Sys^{\mathsf{cry,real}} \geq_{\mathsf{sec}}^{\mathsf{poly}} Sys^{\mathsf{cry,id}}. \tag{1}$$

If symmetric encryption is present, this result is additionally subject to the condition that the surrounding protocol, in our case the Yahalom protocol, does not raise a so-called commitment problem for symmetric encryption. It is a nice obseration that this condition can immediately concluded from the overall proof; we give the formal argument in Appendix A. For technical reasons, one further has to ensure that the protocol does not create encryption cycles (such as encrypting a key with itself); this is needed even for much weaker properties than simulatability, see [3]. This property clearly holds for the Yahalom protocol.

Once we have shown that the considered keys are symbolically secret and that the commitment problem does not occur for the Yahalom protocol, we can exploit the key-secrecy preservation theorem of [20]: If for certain honest users H and a secret-key belief function seckeys the ideal cryptographic library keeps the keys in seckeys strictly symbolically secret, then every configuration of H with the real cryptographic library keeps the keys in seckeys cryptographically secret.

5 Proof in the Ideal Setting

This section finally contains the proof of the ideal part of Theorem 3.1: We prove that the Yahalom protocol with the ideal, Dolev-Yao-style cryptographic library keeps the exchanged keys of honest users strictly symbolically secret. The proof idea is the following: If an honest user u successfully terminates a session run with another honest user v, we first show that the established key has been created by the trusted party. Then we exploit that the trusted party and the honest users 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 never gets a handle to the key. The main challenge is to find suitable invariants on the state of the ideal Yahalom system. This is similar to formal proofs using other Dolev-Yao model, and the similarity supports our hope that the new, sound cryptographic library can be used like other Dolev-Yao models in automated tools. We now present the invariants of the system $Sys^{\mathsf{Ya,id}}$. Their proof is postponed to Appendix B.

The first invariants, *correct nonce owner* and *unique nonce use*, are easily proved. They essentially state that handles n^{hnd} where $(n^{\mathsf{hnd}}, \cdot, \cdot)$ is contained in a set $Nonce_u$ point to entries of type nonce and that no nonce is in two such sets.

Invariant 1 *(Correct Nonce Owner)* For all $u \in \mathcal{H} \setminus \{\mathsf{T}\}$, $v \in \{1, \ldots, n\}$, $j \in \{1, 2, 3, 4\}$ and $(n^{\mathsf{hnd}}, v, j) \in Nonce_u$, we have $D[hnd_u = n^{\mathsf{hnd}}] \neq \downarrow$ and $D[hnd_u = n^{\mathsf{hnd}}].type = \mathsf{nonce}$.

Invariant 2 *(Unique Nonce Use)* For all $u, v \in \mathcal{H} \setminus \{\mathsf{T}\}$, all $w, w' \in \{1, \ldots, n\}$, all $j, j' \in \{1, 2, 3, 4\}$, and all $i \leq size$: If $(D[i].hnd_u, w, j) \in Nonce_u$ and $(D[i].hnd_v, w', j') \in Nonce_v$, then $(u, w) = (v, w')$.

The invariant *encrypted-key secrecy* states that a key shared between honest u and v as well as all lists containing this key can only be known to u, v, and T. Moreover, it states that such lists only occur within symmetric encryptions created with the key shared between u and T respectively between v and T.

Invariant 3 *(Encrypted-Key Secrecy)* For all $u, v \in \mathcal{H} \setminus \{\mathsf{T}\}$ and all $i \leq size$ with $D[i].type = \mathsf{symenc}$: Let $l^{\mathsf{ind}} := D[i].arg[1]$, $pkse^{\mathsf{ind}} := D[i].arg[2]$, $x_t^{\mathsf{ind}} := D[l^{\mathsf{ind}}].arg[t]$, and $x_t := D[x_t^{\mathsf{ind}}].arg[1]$ for $t = 1, 2, 3$. If $D[l^{\mathsf{ind}}].type = \mathsf{list} \wedge pkse^{\mathsf{ind}} = pkse_u \wedge x_1 = v \wedge D[x_t^{\mathsf{ind}}].type = \mathsf{skse}$ for some $t \in \{1, 2, 3\}$ then

a) $D[x_t^{\mathsf{ind}}].hnd_w = \downarrow$ and $D[l'^{\mathsf{ind}}].hnd_w = \downarrow$ for $(\mathcal{H} \setminus \{u, v, \mathsf{T}\}) \cup \{\mathsf{a}\}$ and for all l'^{ind} with $x_t^{\mathsf{ind}} \in D[l'^{\mathsf{ind}}].arg$.

b) For all $l', k \leq size$ such that $D[l'].type = \mathsf{list} \wedge x_t^{\mathsf{ind}} \in D[l'].arg$, we have that $l' \in D[k]$ only if $D[k].type = \mathsf{symenc}$ and $D[k].arg[2] \in \{pkse_u, pkse_v\}$.

The invariant *correct encryption owner* finally states that certain protocol messages can only be constructed by the "intended" users or by the trusted party, respectively. We refer to Step i of Algorithm j as Step $j.i$.

Invariant 4 *(Correct Encryption Owner)* For all $u \in \mathcal{H} \setminus \{\mathsf{T}\}$ and all $i \leq size$ with $D[i].type = \mathsf{symenc}$: Let $l_k^{\mathsf{ind}} := D[i].arg[2k-1]$ and $pkse_k^{\mathsf{ind}} := D[i].arg[2k]$ for $1 \leq k \leq \frac{|D[i].arg|}{2}$ (entries of type symenc have an even number of arguments by construction). Let further $x_{k,t}^{\mathsf{ind}} := D[l_k^{\mathsf{ind}}].arg[t]$ and $x_{k,t,u}^{\mathsf{hnd}} := D[x_{k,t}^{\mathsf{ind}}].hnd_u$ for $t = 1, 2, 3, 4$, and $x_{k,t} := D[x_{k,t}^{\mathsf{ind}}].arg[1]$ for $t = 1, 2$.

a) If $pkse_k^{\mathsf{ind}} = pkse_u$, $x_{k,1} \in \mathcal{H}$, $D[x_{k,2}^{\mathsf{ind}}].type = \mathsf{skse}$, and $(x_{k,3,u}^{\mathsf{hnd}}, x_{k,1}, j) \in Nonce_u$ for some $j \in \{1, 3\}$ and some $k \in \{1, \ldots, \frac{|D[i].arg|}{2}\}$, then $D[i]$ was created by $\mathsf{M}_{\mathsf{T}}^{\mathsf{Ya}}$ in Step 3.11.

b) If $pkse_k^{\mathsf{ind}} = pkse_u$, $x_{k,1} \in \mathcal{H}$, $x_{k,2} = u$, $D[x_{k,3}^{\mathsf{ind}}].type = \mathsf{skse}$, and $(x_{k,4,u}^{\mathsf{hnd}}, x_{k,1}, j) \in Nonce_u$ for some $j \in \{2, 4\}$ and some $k \in \{1, \ldots, \frac{|D[i].arg|}{2}\}$, then $D[i]$ was created by $\mathsf{M}_{\mathsf{T}}^{\mathsf{Ya}}$ in Step 3.13.

5.1 Proof of the Key Secrecy Property

To increase readability, we partition the proof into several steps with explanations in between. Assume that $u, v \in \mathcal{H}$ and that $\mathsf{M}_u^{\mathsf{Ya}}$ outputs $(\mathsf{ok\_initiator}, \mathsf{Yahalom}, v, skse_u^{\mathsf{hnd}})$ or $(\mathsf{ok\_responder}, \mathsf{Yahalom}, v, skse_u^{\mathsf{hnd}})$ to its user. We first show that this implies that $\mathsf{M}_u^{\mathsf{Ya}}$ has received a message corresponding to the third or fourth protocol step so that the contained encryptions are of the form that allows us to apply *correct encryption owner* to show that they were created by $\mathsf{M}_{\mathsf{T}}^{\mathsf{Ya}}$. After that we will exploit *encrypted-key secrecy* to show that keys created by $\mathsf{M}_{\mathsf{T}}^{\mathsf{Ya}}$ and to be shared amongst u and v will remain secret to the adversary. The following property of $\mathsf{TH}_{\mathcal{H}}$ proven in [24] will be useful in this proof to show that properties proven for one time also hold at another time.

Lemma 5.1 The ideal cryptographic library $Sys^{\mathsf{cry,id}}$ has the following property: The only modifications to existing entries x in D are assignments to previously undefined attributes $x.hnd_u$ (except for counter updates in entries for signature keys, which we do not have to consider here), and appending new elements to the list of arguments of symmetric encryptions. □

Proof. (Ideal part of Theorem 3.1) Assume that $\mathsf{M}_u^{\mathsf{Ya}}$ outputs $(\mathsf{ok\_initiator}, \mathsf{Yahalom}, v, skse_u^{\mathsf{hnd}})$ or $(\mathsf{ok\_responder}, \mathsf{Yahalom}, v, skse_u^{\mathsf{hnd}})$ at $\mathsf{KE\_out}_u!$ for $u, v \in \mathcal{H}$ at time t_3, and let $skse^{\mathsf{ind}} := D[hnd_u = skse_u^{\mathsf{hnd}}].ind$. By definition of Algorithms 1 and 2, this output can only happen in Step 2.23 respectively 2.32, and only after there was an input $(w, u, \mathsf{i}, m_u^{3^{\mathsf{hnd}}})$ respective $(w, u, \mathsf{i}, m_u^{4^{\mathsf{hnd}}})$ at a time $t_2 < t_3$. Here and in the sequel we use the notation of Algorithm 1, 2, and 3 but we distinguish the variables from its different executions by an additional superscript indicating the number of the (claimed) received protocol message, here ^{3} and ^{4}, and give handles an additional subscript for their owner, here u.

Case 1: Output in Step 2.23. Assume that $\mathsf{M}_u^{\mathsf{Ya}}$ outputs $(\mathsf{ok\_initiator}, \mathsf{Yahalom}, v, skse_u^{3^{\mathsf{hnd}}})$ at $\mathsf{KE\_out}_u!$ for $u, v \in \mathcal{H}$ in Step 2.23 at time t_3. Hence the execution of Algorithm 2 for this input must have given $l_u^{3^{\mathsf{hnd}}} \neq \downarrow$ in Step 2.15, since the algorithm would otherwise abort by Convention 1 without creating an output. Let $t_2^{3^{\mathsf{ind}}} := D[hnd_u = t_{2,u}^{3^{\mathsf{hnd}}}].ind$, $l^{3^{\mathsf{ind}}} := D[hnd_u = l_u^{3^{\mathsf{hnd}}}].ind$ and $(l_1, pkse_1, l_2, pkse_2, \ldots, l_j, pkse_j) := D[t_2^{3^{\mathsf{ind}}}].arg$ (which is the general argument format for symmetric encryption entries). The definition of $\mathsf{sym\_decrypt}$ then implies $D[l^{3^{\mathsf{ind}}}].type = \mathsf{list}$ and $D[t_2^{3^{\mathsf{ind}}}].type = \mathsf{symenc}$, and further that there exists some unique k with $1 \leq k \leq j$ such that $D[t_2^{3^{\mathsf{ind}}}].arg[2k-1] = l^{3^{\mathsf{ind}}}$, and

$$D[t_2^{3^{\mathsf{ind}}}].arg[2k] = pkse_u. \tag{2}$$

Let $x_i^{3^{\mathsf{ind}}} := D[l^{3^{\mathsf{ind}}}].arg[i]$ for $i = 1, 2, 3$ at the time of Step 2.16. By Step 2.17 and the definition of $\mathsf{retrieve}$ we have

$$D[x_1^{3^{\mathsf{ind}}}].arg[1] = x_1^3 = v. \tag{3}$$

By definition of $\mathsf{list\_proj}$ and $\mathsf{get\_type}$, and since the condition of Step 2.20 is false, we finally have

$$x_{1,u}^{3^{\mathsf{hnd}}} = D[x_1^{3^{\mathsf{ind}}}].hnd_u \text{ at time } t_3, \tag{4}$$

$$(x_{3,u}^{3^{\mathsf{hnd}}}, x_1^3, 3) \in Nonce_u \wedge D[x_2^{3^{\mathsf{ind}}}].type = \mathsf{skse} \text{ at time } t_3, \tag{5}$$

and

$$(x_{3,u}^{3^{\mathsf{hnd}}}, x_1^3, 1) \in Nonce_u \wedge D[x_2^{3^{\mathsf{ind}}}].type = \mathsf{skse} \text{ at time } t_2. \tag{6}$$

Case 2: Output in Step 2.32. This case is similar to the first one: Assume that $\mathsf{M}_u^{\mathsf{Ya}}$ outputs $(\mathsf{ok\_responder}, \mathsf{Yahalom}, v, skse_u^{4\mathsf{hnd}})$ at $\mathsf{KE\_out}_u!$ for $u, v \in \mathcal{H}$ in Step 2.32 at time t_3. The execution of Algorithm 2 for this input must have given $l_u^{4\mathsf{hnd}} \neq \downarrow$ in Step 2.26, since the algorithm would otherwise abort by Convention 1 without creating an output. Let $t_1^{4\mathsf{ind}} := D[hnd_u = t_{1,u}^{4\mathsf{hnd}}].ind$, $l^{4\mathsf{ind}} := D[hnd_u = l_u^{4\mathsf{hnd}}].ind$ and $(l_1, pkse_1, l_2, pkse_2, \ldots, l_j, pkse_j) := D[t_1^{4\mathsf{ind}}].arg$. The definition of sym_decrypt then implies $D[l^{4\mathsf{ind}}].type = \mathsf{list}$ and $D[t_1^{4\mathsf{ind}}].type = \mathsf{symenc}$, and further that there exists a unique k with $1 \leq k \leq j$ such that $D[t_1^{4\mathsf{ind}}].arg[2k-1] = l^{4\mathsf{ind}}$, and

$$D[t_1^{4\mathsf{ind}}].arg[2k] = pkse_u. \tag{7}$$

Let $x_i^{4\mathsf{ind}} := D[l^{4\mathsf{ind}}].arg[i]$ for $i = 1, 2, 3, 4$ at the time of Step 2.27. By Step 2.28 and the definition of retrieve we have $x_i^4 = D[x_i^{4\mathsf{ind}}].arg[1]$ for $i = 1, 2$ and

$$D[x_1^{4\mathsf{ind}}].arg[1] = x_1^4 = v. \tag{8}$$

By definition of list_proj and get_type, and because of the conditions of Step 2.25 and 2.30, we have

$$x_2^4 = u, \tag{9}$$

$$x_{1,u}^{4\mathsf{hnd}} = D[x_1^{4\mathsf{ind}}].hnd_u \text{ at time } t_3, \tag{10}$$

$$(x_{4,u}^{4\mathsf{hnd}}, x_1^4, 4) \in Nonce_u \wedge D[x_3^{4\mathsf{ind}}].type = \mathsf{skse} \text{ at time } t_3, \tag{11}$$

and

$$(x_{4,u}^{4\mathsf{hnd}}, x_1^4, 2) \in Nonce_u \wedge D[x_3^{4\mathsf{ind}}].type = \mathsf{skse} \text{ at time } t_2. \tag{12}$$

■

This first part of the proof shows that $\mathsf{M}_u^{\mathsf{Ya}}$ has received an encryption of a specific form as part of a third or fourth protocol message. Now we apply *correct encryption owner* to the encryption entry $D[t_2^{3\mathsf{ind}}]$ for the first case respectively to $D[l_1^{4\mathsf{ind}}]$ for the second case to show that this entry was created by $\mathsf{M}_{\mathsf{T}}^{\mathsf{Ya}}$.

Proof. (cont'd) Equations (2), (4) and (5) respectively (7), (10) and (11) are the preconditions for Part a) respectively Part b) of *correct encryption owner*. Hence the entry $D[t_2^{3\mathsf{ind}}]$ was created by $\mathsf{M}_{\mathsf{T}}^{\mathsf{Ya}}$ in Step 3.11 respectively the entry $D[t_1^{4\mathsf{ind}}]$ was created by $\mathsf{M}_{\mathsf{T}}^{\mathsf{Ya}}$ in Step 3.13.

In both cases, the algorithm execution must have started with an input $(w', \mathsf{T}, \mathsf{i}, m_{\mathsf{T}}^{2\mathsf{hnd}})$ at $\mathsf{out}_{\mathsf{T}}?$ at a time $t_1 < t_2$ with $w' \in \{1, \ldots, n\}$. We conclude $l_{\mathsf{T}}^{2\mathsf{hnd}} \neq \downarrow$ in Step 3.4 because of Convention 1 and set $l^{2\mathsf{ind}} := D[hnd_{\mathsf{T}} = l_{\mathsf{T}}^{2\mathsf{hnd}}].ind$. The definition of sym_decrypt implies $D[l^{2\mathsf{ind}}].type = \mathsf{list}$, $D[t_3^{2\mathsf{ind}}].type = \mathsf{symenc}$, $l^{2\mathsf{ind}} = D[t_3^{2\mathsf{ind}}].arg[2k'-1]$ and $pkse_{w'} = D[t_3^{2\mathsf{ind}}].arg[2k']$ for some unique $k' \in \{1, \ldots, |D[t_3^{2\mathsf{ind}}].arg|$, cf. the first part of the proof. Let $x_i^{2\mathsf{ind}} := D[l^{2\mathsf{ind}}].arg[i]$ for $i = 1, 2$ at the time of Step 3.5.

As the condition of Step 3.8 is false immediately afterwards, we obtain $x_{i,\mathsf{T}}^{2\mathsf{hnd}} \neq \downarrow$ for $i = 1, 2$. The definitions of list_proj and get_type together with Lemma 5.1 imply

$$x_{i,\mathsf{T}}^{2\mathsf{hnd}} = D[x_i^{2\mathsf{ind}}].hnd_{\mathsf{T}} \text{ for } i = 1, 2 \text{ at time } t_3. \tag{13}$$

Step 3.8 further ensures $t_1^2 = w'$ and $x_1^2 \in \{1, \ldots, n\} \setminus \{w'\}$. By definition gen_symenc_key we obtain $skse^{2\mathsf{hnd}} \neq \downarrow$ and

$$D[skse^{2\mathsf{ind}}].type = \mathsf{skse} \tag{14}$$

in Step 3.9 for $skse^{2^{\mathsf{ind}}} := D[hnd_{\mathsf{T}} = skse^{2^{\mathsf{hnd}}}].ind$. Now we exploit that $\mathsf{M}_{\mathsf{T}}^{\mathsf{Ya}}$ creates the entry $D[t_2^{3^{\mathsf{ind}}}]$ in Step 3.11 with the input $\mathsf{sym\_encrypt}(skse_{\mathsf{T},x_1}^{2^{\mathsf{hnd}}}, l_{3,\mathsf{T}}^{(1),2^{\mathsf{hnd}}})$ respectively the entry $D[t_1^{4^{\mathsf{ind}}}]$ in Step 3.13 with the input $\mathsf{sym\_encrypt}(skse_{\mathsf{T},w'}^{2^{\mathsf{hnd}}}, l_{3,\mathsf{T}}^{(2),2^{\mathsf{hnd}}})$. In particular, this implies $k = 1$ in Equtations (2) and (7) by the definition of $\mathsf{sym\_encrypt}$. Let $l_3^{(i),2^{\mathsf{ind}}} := D[hnd_{\mathsf{T}} = l_{3,\mathsf{T}}^{(1),2^{\mathsf{hnd}}}].ind$ for $i = 1, 2$. With the definitions of list and $\mathsf{list\_proj}$ this implies $l_3^{(1),2^{\mathsf{ind}}} = l^{3^{\mathsf{ind}}}$ respectively $l_3^{(2),2^{\mathsf{ind}}} = l^{4^{\mathsf{ind}}}$. Together with Equations (3), (8), (9) and (13), this implies

$$skse^{\mathsf{ind}} = x_2^{3^{\mathsf{ind}}} = x_2^{2^{\mathsf{ind}}} = skse^{2^{\mathsf{ind}}} \wedge x_2^{2^{\mathsf{ind}}} = x_3^{3^{\mathsf{ind}}} \wedge t_1^2 = x_1^3 = v \wedge x_1^2 = u, \tag{15}$$

where $x_1^2 = u$ follows from $pkse_{x_1^2} = D[l_3^{(1),2^{\mathsf{ind}}}].arg[2] = D[l^{3^{\mathsf{ind}}}].arg[2] = pkse_u$, respectively

$$skse^{\mathsf{ind}} = x_2^{4^{\mathsf{ind}}} = x_2^{2^{\mathsf{ind}}} = skse^{2^{\mathsf{ind}}} \wedge x_1^{2^{\mathsf{ind}}} = x_2^{4^{\mathsf{ind}}} \wedge t_1^2 = x_2^4 = u \wedge x_1^2 = x_1^4 = v, \tag{16}$$

where $t_1^2 := D[t_1^{2^{\mathsf{ind}}}].arg[1]$. Thus Equations (3), (4) and (6) imply

$$(x_{2,u}^{2^{\mathsf{hnd}}}, v, 1) \in Nonce_u \wedge D[x_2^{2^{\mathsf{ind}}}].type = \mathsf{skse} \text{ at time } t_2 \tag{17}$$

respectively Equations (8), (10) and (12) imply

$$(x_{1,u}^{2^{\mathsf{hnd}}}, v, 2) \in Nonce_u \wedge D[x_2^{2^{\mathsf{ind}}}].type = \mathsf{skse} \text{ at time } t_2. \tag{18}$$

■

After this second part of the proof, we will finally derive that the terms selected by $\mathsf{seckeys\_initiator\_Ya}$ and $\mathsf{seckeys\_responder\_Ya}$ are symbolically unused symmetric keys that have furthermore not been used for encryption yet.

Proof. (cont'd) Note that the entries $D[t_2^{3^{\mathsf{ind}}}]$ and $D[t_1^{4^{\mathsf{ind}}}]$ fulfill the requirements of *encrypted-key secrecy* with respect to key entry $D[skse^{2^{\mathsf{ind}}}]$, which implies $D[skse^{2^{\mathsf{ind}}}].hnd_{\mathsf{a}} = \downarrow$. Because of $skse^{2^{\mathsf{ind}}} = skse^{\mathsf{ind}}$ we have $D[skse^{\mathsf{ind}}].hnd_{\mathsf{a}} = D[skse^{2^{\mathsf{ind}}}].hnd_{\mathsf{a}} = \downarrow$, i.e., the term under $(u, skse_u^{\mathsf{hnd}})$ is symbolically unknown. Moreover $skse^{2^{\mathsf{ind}}} = skse^{\mathsf{ind}}$ together with Equation (14) imply $D[skse^{\mathsf{ind}}].type = D[skse^{2^{\mathsf{ind}}}].type = \mathsf{skse}$, i.e., the term under $(u, skse_u^{\mathsf{hnd}})$ is a symmetric key.

It remains to show that the key is unused at time t_3. The only way to create an entry $D[j]$ with $D[j].type = \mathsf{symenc}$ and $D[j].arg[2] = skse^{\mathsf{ind}} - 1$ is by inputting a command $\mathsf{encrypt}$ at port in_w? such that $D[skse^{\mathsf{ind}}].hnd_w \neq \downarrow$. Since we have shown that $D[skse^{\mathsf{ind}}].hnd_w \neq \downarrow$ only if $w \in \{u, v, \mathsf{T}\}$, hence we only have to show that neither of them enters such a command until time t_3. By inspection of Algorithm 3, this clearly holds for T, since this may only happen in Steps 3.11 or 3.13. In both cases, the key used is one of those that were iniatially distributed, i.e., $D[j].arg[2] = skse_w - 1$ for some $w \in \{1, \ldots, n\}$. Since we have shown that each key selected by $\mathsf{seckeys\_initiator\_Ya}$ or $\mathsf{seckeys\_responder\_Ya}$ is newly generated by $\mathsf{M}_{\mathsf{T}}^{\mathsf{Ya}}$, we in particular have $skse_w \neq skse^{\mathsf{ind}}$. Similar reasoning can be applied to Algorithm 1 and 2 of $\mathsf{M}_u^{\mathsf{Ya}}$ to show that the only used keys are the ones shared between u and T respectively between v and T. ■

References

[1] M. Abadi and A. D. Gordon. A calculus for cryptographic protocols: The spi calculus. In *Proc. 4th ACM Conference on Computer and Communications Security*, pages 36–47, 1997.

[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 *Lecture Notes in Computer Science*, pages 3–22. Springer, 2000.

[4] M. Backes. Quantifying probabilistic information flow in computational reactive systems. In *Proceedings of 10th European Symposium on Research in Computer Security (ESORICS)*, volume 3679 of *Lecture Notes in Computer Science*, pages 336–354. Springer, 2005.

[5] M. Backes. Real-or-random key secrecy of the Otway-Rees protocol via a symbolic security proof. *Electronic Notes in Theoretical Computer Science (ENTCS)*, 155:111–145, 2006.

[6] M. Backes, I. Cervesato, A. D. Jaggard, A. Scedrov, and J.-K. Tsay. Cryptographically sound security proofs for basic and public-key kerberos. In *Proceedings of 11th European Symposium on Research in Computer Security(ESORICS)*, volume 4189 of *Lecture Notes in Computer Science*, pages 362–383. Springer, 2006. Preprint on IACR ePrint 2006/219.

[7] M. Backes and M. Duermuth. A cryptographically sound Dolev-Yao style security proof of an electronic payment system. In *Proceedings of 18th IEEE Computer Security Foundations Workshop (CSFW)*, pages 78–93, 2005.

[8] M. Backes, M. Duermuth, D. Hofheinz, and R. Kuesters. Conditional reactive simulatability. In *Proceedings of 11th European Symposium on Research in Computer Security (ESORICS)*, volume 4189 of *Lecture Notes in Computer Science*, pages 424–443. Springer, 2006. Preprint on IACR ePrint 2006/132.

[9] M. Backes, M. Dürmuth, and R. Küsters. On simulatability soundness and mapping soundness of symbolic cryptography. *IACR Cryptology ePrint Archive*, 2007:233, 2007.

[10] M. Backes and T. Gross. Tailoring the Dolev-Yao abstraction to web services realities. In *Proceedings of 2005 ACM Secure Web Services Workshop (SWS)*, pages 65–74, 2005.

[11] 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.

[12] M. Backes, C. Jacobi, and B. Pfitzmann. Deriving cryptographically sound implementations using composition and formally verified bisimulation. In *Proceedings of 11th International Symposium on Formal Methods Europe (FME)*, volume 2391 of *Lecture Notes in Computer Science*, pages 310–329. Springer, 2002.

[13] M. Backes and P. Laud. Computationally sound secrecy proofs by mechanized flow analysis. In *Proceedings of 13th ACM Conference on Computer and Communications Security (CCS)*, pages 370–379, 2006.

[14] M. Backes, S. Moedersheim, B. Pfitzmann, and L. Vigano. Symbolic and cryptographic analysis of the secure WS-ReliableMessaging Scenario. In *Proceedings of Foundations of Software Science and Computational Structures (FOSSACS)*, volume 3921 of *Lecture Notes in Computer Science*, pages 428–445. Springer, 2006.

[15] M. Backes and B. Pfitzmann. Intransitive non-interference for cryptographic purposes. In *Proceedings of 24th IEEE Symposium on Security and Privacy*, pages 140–152, 2003.

[16] M. Backes and B. Pfitzmann. Computational probabilistic non-interference. *International Journal of Information Security (IJIS)*, 3(1):42–60, 2004.

[17] M. Backes and B. Pfitzmann. A cryptographically sound security proof of the Needham-Schroeder-Lowe public-key protocol. *IEEE Journal on Selected Areas of Computing (JSAC)*, 22(10):2075–2086, 2004.

[18] M. Backes and B. Pfitzmann. Symmetric encryption in a simulatable Dolev-Yao style cryptographic library. In *Proceedings of 17th IEEE Computer Security Foundations Workshop (CSFW)*, pages 204–218, 2004.

[19] M. Backes and B. Pfitzmann. Limits of the cryptographic realization of Dolev-Yao-style XOR. In *Proceedings of 10th European Symposium on Research in Computer Security (ESORICS)*, volume 3679 of *Lecture Notes in Computer Science*, pages 178–196. Springer, 2005.

[20] M. Backes and B. Pfitzmann. Relating symbolic and cryptographic secrecy. In *Proc. 26th IEEE Symposium on Security & Privacy*, pages 171–182, 2005. Extended version in IACR Cryptology ePrint Archive 2004/300.

[21] M. Backes and B. Pfitzmann. On the cryptographic key secrecy of the strengthened Yahalom protocol. In *Proceedings of 21st IFIP International Information Security Conference (SEC)*, pages 233–245, 2006.

[22] M. Backes, B. Pfitzmann, and A. Scedrov. Key-dependent message security under active attacks - BRSIM/UC-soundness of symbolic encryption with key cycles. *IACR Cryptology ePrint Archive*, 2005:421, 2005.

[23] M. Backes, B. Pfitzmann, M. Steiner, and M. Waidner. Polynomial liveness. *Journal of Computer Security*, 12(3-4):589–617, 2004.

[24] M. Backes, B. Pfitzmann, and M. Waidner. A composable cryptographic library with nested operations. In *Proc. 10th ACM Conference on Computer and Communications Security*, pages 220–230, 2003.

[25] M. Backes, B. Pfitzmann, and M. Waidner. Security in business process engineering. In *Proceedings of 2003 International Conference on Business Process Management*, volume 2678 of *Lecture Notes in Computer Science*, pages 168–183. Springer, 2003.

[26] M. Backes, B. Pfitzmann, and M. Waidner. Low-level ideal signatures and general integrity idealization. In *Proceedings of 7th Information Security Conference (ISC)*, volume 3225 of *Lecture Notes in Computer Science*, pages 39–51. Springer, 2004.

[27] M. Backes, B. Pfitzmann, and M. Waidner. Symmetric authentication within a simulatable cryptographic library. *International Journal of Information Security (IJIS)*, 4(3):135–154, 2005.

[28] M. Backes, B. Pfitzmann, and M. Waidner. Limits of the reactive simulatability/UC of Dolev-Yao models with hashes. In *Proceedings of 11th European Symposium on Research in Computer Security(ESORICS)*, volume 4189 of *Lecture Notes in Computer Science*, pages 404–423. Springer, 2006.

[29] D. Basin, S. Mödersheim, and L. Viganò. OFMC: A symbolic model checker for security protocols. *International Journal of Information Security*, 2004.

[30] 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 *Lecture Notes in Computer Science*, pages 531–545. Springer, 2000.

[31] 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 *Lecture Notes in Computer Science*, pages 317–330. Springer, 2000.

[32] C. Bodei, M. Buchholtz, P. Degano, F. Nielson, and H. Nielson. Automatic validation of protocol narration. In *Proc. 16th IEEE Computer Security Foundations Workshop (CSFW)*, pages 126–140, 2003.

[33] M. Burrows, M. Abadi, and R. Needham. A logic for authentication. Technical Report 39, SRC DIGITAL, 1990.

[34] R. Canetti and J. Herzog. Universally composable symbolic analysis of cryptographic protocols (the case of encryption-based mutual authentication and key exchange). Cryptology ePrint Archive, Report 2004/334, 2004. `http://eprint.iacr.org/`.

[35] D. Dolev and A. C. Yao. On the security of public key protocols. *IEEE Transactions on Information Theory*, 29(2):198–208, 1983.

[36] S. Even and O. Goldreich. On the security of multi-party ping-pong protocols. In *Proc. 24th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 34–39, 1983.

[37] J. Guttman. Key compromise and the authentication tests. In *Proc. Mathematical Foundations of Programming Semantics*, volume 17 of ENTCS, pages 1–21, 2001.

[38] J. Herzog. *Computational Soundness of Formal Adversaries*. PhD thesis, MIT, 2002.

[39] J. Herzog, M. Liskov, and S. Micali. Plaintext awareness via key registration. In *Advances in Cryptology: CRYPTO 2003*, volume 2729 of *Lecture Notes in Computer Science*, pages 548–564. Springer, 2003.

[40] 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.

[41] R. Kemmerer, C. Meadows, and J. Millen. Three systems for cryptographic protocol analysis. *Journal of Cryptology*, 7(2):79–130, 1994.

[42] P. Laud. Semantics and program analysis of computationally secure information flow. In *Proc. 10th European Symposium on Programming (ESOP)*, pages 77–91, 2001.

[43] P. Laud. Symmetric encryption in automatic analyses for confidentiality against active adversaries. In *Proc. 25th IEEE Symposium on Security & Privacy*, pages 71–85, 2004.

[44] 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.

[45] G. Lowe. An attack on the Needham-Schroeder public-key authentication protocol. *Information Processing Letters*, 56(3):131–135, 1995.

[46] G. Lowe. Breaking and fixing the Needham-Schroeder public-key protocol using FDR. In *Proc. 2nd International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, volume 1055 of *Lecture Notes in Computer Science*, pages 147–166. Springer, 1996.

[47] C. Meadows. Using narrowing in the analysis of key management protocols. In *Proc. 10th IEEE Symposium on Security & Privacy*, pages 138–147, 1989.

[48] M. Merritt. *Cryptographic Protocols*. PhD thesis, Georgia Institute of Technology, 1983.
[49] 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.
[50] J. K. Millen. The interrogator: A tool for cryptographic protocol security. In *Proc. 5th IEEE Symposium on Security & Privacy*, pages 134–141, 1984.
[51] 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.
[52] 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.
[53] L. Paulson. The inductive approach to verifying cryptographic protocols. *Journal of Cryptology*, 6(1):85–128, 1998.
[54] L. Paulson. Relations between secrets: Two formal analyses of the yahalom protocol. *Journal of Computer Security*, 9(3):197–216, 2001.
[55] 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/`.
[56] 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.
[57] A. W. Roscoe. Modelling and verifying key-exchange protocols using CSP and FDR. In *Proc. 8th IEEE Computer Security Foundations Workshop (CSFW)*, pages 98–107, 1995.
[58] S. Schneider. Security properties and CSP. In *Proc. 17th IEEE Symposium on Security & Privacy*, pages 174–187, 1996.
[59] C. Sprenger, M. Backes, D. Basin, B. Pfitzmann, and M. Waidner. Cryptographically sound theorem proving. In *Proceedings of 19th IEEE Computer Security Foundations Workshop (CSFW)*, pages 153–166, 2006.

A Absence of the Commitment Problem for the Yahalom 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 [18] 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 [18], we hence have to

prove this condition for the Yahalom 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 using temporal logics but we omit the rigorous definition due to space constraints and refer to [18]. 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 Yahalom 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 we will present in Section 5.

Technically, the event NoComm is an *integrity property*, and the notion that $Sys^{\mathsf{Ya,id}}$ *perfectly fulfills* NoComm, written $Sys^{\mathsf{Ya,id}} \models^{\mathsf{perf}} \mathsf{NoComm}$ means that the property holds with probability one (over the probability spaces of runs, a well-defined notion from the underlying model [55]) for all honest users and for all adversaries.

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

Proof. Note first that the secret key shared initially between a user and the trusted 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 party in the protocol sessions.

Let $i \leq size$, $D[i].type = \mathsf{skse}$ such that $D[i]$ was created by $\mathsf{M}_{\mathsf{T}}^{\mathsf{Ya}}$ in Step 3.9, where, with the notation of Algorithm 3, we have $x_1 = u$ and $t_1 = v$ for $x_1, t_1 \in \{1, \ldots, n\}$. If u or v were dishonest, then the adversary would get a handle for $D[i]$ after $\mathsf{M}_{\mathsf{T}}^{\mathsf{Ya}}$ finishes its execution, i.e., in particular before $D[i]$ has been used for encryption for the first time, since the adversary knows the keys shared between the dishonest users and the trusted party. If both u and v are honest, *encrypted-key secrecy* applied to the entry $D[{c_3^{(1)}}^{\mathsf{ind}}]$ created in Step 3.11 in the same execution of $\mathsf{M}_{\mathsf{T}}^{\mathsf{Ya}}$ then immediately implies $D[i].hnd_{\mathsf{a}} = \downarrow$ for all time t, which finishes the proof. ■

B Proof of the Invariants

In the following we prove *correct nonce owner*, *unique nonce use*, *correct encryption owner*, and *encrypted-key secrecy* by induction. Hence assume that all invariants hold at a particular time t in a run of the system, and we have to show that they still hold at time $t + 1$.

We start with the proof of *correct nonce owner*.

Proof. (*Correct nonce owner*) Let $(n^{\mathsf{hnd}}, v, j) \in Nonce_u$ for $u \in \mathcal{H} \setminus \{\mathsf{T}\}$, $v \in \{1, \ldots, n\}$, and $j \in \{1, 2, 3, 4\}$. By construction, this entry has been added to $Nonce_u$ by $\mathsf{M}_u^{\mathsf{Ya}}$ in Step 1.2, Step 2.8, Step 1.21, or Step 1.31. In the last two cases, the entry $(n^{\mathsf{hnd}}, v, j-2)$ was already contained in $Nonce_u$ at time t, hence the claim follows by induction hypothesis of *correct nonce owner*. Thus consider the first two cases. In both cases x^{hnd} has been generated by the command gen_nonce() at some time t, input at port $\mathsf{in}_u?$ of $\mathsf{TH}_{\mathcal{H}}^{\mathsf{cry}}$. Convention 1 implies $n^{\mathsf{hnd}} \neq \downarrow$, as $\mathsf{M}_u^{\mathsf{Ya}}$ would abort otherwise and not add the entry to the set $Nonce_u$. The definition of gen_nonce then implies $D[hnd_u = n^{\mathsf{hnd}}] \neq \downarrow$ and $D[hnd_u = x^{\mathsf{hnd}}].type = \mathsf{nonce}$ at time t. Because of Lemma 5.1 this also holds at all later times $t' > t$, which finishes the proof. ■

The following proof of *unique nonce use* is quite similar.

Proof. (*Unique Nonce Use*) Assume for contradiction that both $x_1 := (D[i].hnd_u, w, j) \in Nonce_u$ and $x_2 := (D[i].hnd_v, w', j') \in Nonce_v$ at some time t. Without loss of generality, let $t+1$ be the first such time and let $x_2 \notin Nonce_v$ at time t. By construction, x_2 is thus added to $Nonce_v$ at time $t+1$ by Step 1.2, Step 2.8, Step 1.21, or Step 1.31. In the last two cases, the entry $(x^{\mathsf{hnd}}, w', j-2)$ was already contained in $Nonce_u$ (for the same handle x^{hnd} and the same identity w') and the claim follows by induction hypothesis for *unique nonce use* again. In the first two cases, $D[i].hnd_v$ has been generated by the command $\mathsf{gen\_nonce}()$ at time t. The definition of $\mathsf{gen\_nonce}$ implies that $D[i]$ is a new entry and $D[i].hnd_v$ its only handle at time t, and thus also at time $t+1$. With *correct nonce owner* this implies $u = v$. Further, $x_2 = (D[i].hnd_v, w', j')$ is the only entry that is put into $Nonce_v$ at times t and $t+1$. Thus also $w = w'$. This is a contradiction. ∎

Proof. (*Correct encryption owner*) Let $u \in \mathcal{H} \setminus \{\mathsf{T}\}$, $i \leq size$ with $D[i].type = \mathsf{symenc}$. Let $l_k^{\mathsf{ind}} := D[i].arg[2k-1]$ and $pkse_k^{\mathsf{ind}} := D[i].arg[2k]$ for $k \in \{1, \ldots, \frac{|D[i].arg|}{2}\}$. Let further $x_{k,q}^{\mathsf{ind}} := D[l_k^{\mathsf{ind}}].arg[q]$ and $x_{k,q,u}^{\mathsf{hnd}} := D[x_{k,q}^{\mathsf{ind}}].hnd_u$ for $q = 1,2,3,4$, and $x_{k,1} := D[x_{k,1}^{\mathsf{ind}}].arg[1]$. Assume that for some k we have $pkse_k^{\mathsf{ind}} = pkse_u$, and assume further that $(x_{k,3,u}^{\mathsf{hnd}}, x_{k,1}, j) \in Nonce_u$ for some $j \in \{1,3\}$ or that $(x_{k,4,u}^{\mathsf{hnd}}, x_{k,1}, j) \in Nonce_u$ for some $j \in \{2,4\}$.

The only possibilities to violate the invariant *correct encryption owner* are that (1) the entry $D[i]$ is created at time $t+1$ or that (2) the handle $D[i].hnd_u$ or $D[x_{k,q}^{\mathsf{ind}}].hnd_u$ for $q = 3$ in part a) or $q = 4$ in part b) is created at time $t+1$ for an entry $D[i]$ that already exists at time t or that (3) the handle $(x_{k,q,u}^{\mathsf{hnd}}, x_{k,1}, j)$ for $q \in \{3,4\}$ is added to $Nonce_u$ at time $t+1$ such that $(x_{k,q,u}^{\mathsf{hnd}}, x_1, j-2)$ was not contained in $Nonce_u$ at time t (i.e., we have to consider $j \in \{1,2\}$). In all other cases the invariant holds by the induction hypothesis for *correct encryption owner* and Lemma 5.1.

We start with the third case. Assume that $(x_{k,q,u}^{\mathsf{hnd}}, x_{k,1}, j)$ for $q \in \{3,4\}$ and $j \in \{1,2\}$ is added to $Nonce_u$ at time $t+1$. By construction, this only happens in a transition of $\mathsf{M}_u^{\mathsf{Ya}}$ in Step 1.2 and Step 2.8. However, here the entry $D[x_{k,q}^{\mathsf{ind}}]$ has been generated by the command $\mathsf{gen\_nonce}$ input at $\mathsf{in}_u?$ at time t, hence $x_{k,q}^{\mathsf{ind}}$ cannot be contained as an argument of an entry $D[l_k^{\mathsf{ind}}]$ at time t. Formally, this corresponds to the fact that D is *well-formed*, i.e., index arguments of an entry are always smaller than the index of the entry itself; this has been shown in [24]. Since a transition of $\mathsf{M}_u^{\mathsf{Ya}}$ does not modify entries in $\mathsf{TH}_{\mathcal{H}}^{\mathsf{cry}}$, this also holds at time $t+1$.

For proving the remaining two cases, assume that $D[i].hnd_u$ or $D[x_{k,q}^{\mathsf{ind}}].hnd_u$ is created at time $t+1$ for an already existing entry $D[i]$ or $D[l_k^{\mathsf{ind}}]$ or that $D[i]$ is generated at time $t+1$. Because both can only happen in a transition of $\mathsf{TH}_{\mathcal{H}}^{\mathsf{cry}}$, this implies $(x_{k,q,u}^{\mathsf{hnd}}, x_{k,1}, j) \in Nonce_u$ already at time t, since transitions of $\mathsf{TH}_{\mathcal{H}}^{\mathsf{cry}}$ cannot modify the set $Nonce_u$. Now *correct nonce owner* implies $x_{k,q,u}^{\mathsf{hnd}} = D[x_{k,q}^{\mathsf{ind}}].hnd_u \neq \downarrow$ already at time t and thus also at time $t+1$ by Lemma 5.1. Symmetric encryptions can only be generated by the basic command $\mathsf{sym\_encrypt}$, which requires handles to all its elements. More precisely, if $w \in \mathcal{H} \cup \{\mathsf{a}\}$ creates an entry $D[i']$ with $D[i'].type = \mathsf{symenc}$ and $(x'_1, \ldots, x'_m) := D[i].arg$ at time $t+1$ then $D[x'_i].hnd_w \neq \downarrow$ for $i = 1, \ldots, m$ already at time t. In particular, we have that $D[x'_{2k}].ind = D[pkse_k^{\mathsf{ind}}].ind = pkse_u$. The definition of $\mathsf{sym\_encrypt}$ then implies $D[skse_u].hnd_w \neq \downarrow$ and hence $D[i]$ must have been created by either u or T.

We finally have to show that the entry $D[i]$ has been created by T in the claimed steps. This can easily be seen by inspection of Algorithms 1, 2, and 3. We only show it in detail for the first part of the invariant; it can be proven similarly for second part. Let $(x_{k,3,u}^{\mathsf{hnd}}, x_{k,1}, j) \in Nonce_u$ and $D[x_{k,2}^{\mathsf{ind}}].type = \mathsf{skse}$. By inspection of Algorithms 1, 2, and 3 and because $D[i].type = \mathsf{symenc}$, we see that the entry $D[i]$ must have been created by either $\mathsf{M}_u^{\mathsf{Ya}}$ in Step 2.11 or by $\mathsf{M}_{\mathsf{T}}^{\mathsf{Ya}}$ in Step 3.11 or 3.13. The list encrypted in Step 2.11 only has two elements, which implies $x_{k,3}^{\mathsf{ind}} = \downarrow$ and hence $x_{k,3,u}^{\mathsf{hnd}} = \downarrow$, and by *correct nonce owner* this gives a

contradiction to $(x^{\mathsf{hnd}}_{k,3,u}, x_{k,1}, j) \in Nonce_u$. Similarly, if the entry $D[i]$ were created in Step 3.13 then we had $D[x^{\mathsf{ind}}_{k,2}].type = \mathsf{data}$ as the algorithm would have aborted otherwise in Step 3.2 by the definition of retrieve and Convention 1. ∎

Finally, we prove *encrypted-key secrecy*.

Proof. (*Encrypted-key secrecy*) Let $u, v \in \mathcal{H}$, $i \leq size$ with $D[i].type = \mathsf{symenc}$. Let $l^{\mathsf{ind}} := D[i].arg[1]$, $pkse^{\mathsf{ind}} := D[i].arg[2]$, $x^{\mathsf{ind}}_j := D[l^{\mathsf{ind}}].arg[j]$, and $x_j := D[x^{\mathsf{ind}}_t].arg[1]$ for $t = 1, 2, 3$. Assume $D[l^{\mathsf{ind}}].type = \mathsf{list}$, $pkse^{\mathsf{ind}} = pkse_u$, $x_1 = v$, and $D[x^{\mathsf{ind}}_j].type = \mathsf{skse}$ for some j.

Part a) of the invariant can only be affected if a handle for w is added to an entry $D[l'^{\mathsf{ind}}]$ or $D[x^{\mathsf{ind}}_j]$ that already exist at time t. (Creation of $D[l'^{\mathsf{ind}}]$ at time t with a handle for w is impossible as above because that presupposes handles to all arguments of $D[l'^{\mathsf{ind}}]$, i.e., in particular to $D[x^{\mathsf{ind}}_j]$, which contradicts *encrypted-key secrecy* at time t; creation of $D[x^{\mathsf{ind}}_2]$ at time t with a handle for w would yield a contraction to x^{ind}_2 being an argument of l^{ind} at time $t+1$ since D is *well-formed*, cf. the proof of *correct encryption owner*. Thus we only have to consider those commands that add handles for w to entries of type list and skse that already existed at time t. The only commands that add handles for w to $D[l^{\mathsf{ind}}]$, i.e., a list entry, are list_proj, decrypt, adv_parse, send_i, and adv_send_i applied to an entry $D[k]$ with $l'^{\mathsf{ind}} \in D[k].arg$. *Encrypted-Key secrecy* for the entry $D[l']$ at time t then yields $D[k].type = \mathsf{symenc}$. Thus the commands list_proj, send_i, and adv_send_i do not have to be considered any further. Moreover, *encrypted-key secrecy* also yields $D[k].arg[2] \in \{pkse_u, pkse_v\}$. The keys shared between u and T respectively between v and T are not known to $w \notin \{u, v, \mathsf{T}\}$, formally $D[skse_u].hnd_w = D[skse_v].hnd_w = \downarrow$; Hence the command sym_decrypt input at $\mathsf{in}_w?$ does not violate the invariant. Finally, the command adv_parse applied to an entry of type symenc with unknown secret key also does not give a handle to the cleartext list, i.e., to $D[k].arg[1]$, but only outputs its length and the key identifier.

The only commands that add handles for w to $D[x^{\mathsf{ind}}_j]$, i.e., a symmetric key entry, are list_proj or adv_parse input at $\mathsf{in}_w?$, where adv_parse has to be applied to an entry of type list, since only entries of type list can have arguments which are indices to symmetric key entries. More precisely, if one of the commands violated the invariant there would exist an entry $D[i']$ at time t such that $D[l'^{\mathsf{ind}}].type = \mathsf{list}$, $D[l'^{\mathsf{ind}}].hnd_w \neq \downarrow$ and $x^{\mathsf{ind}}_j \in D[l'^{\mathsf{ind}}].arg$. *Encrypted-key secrecy* for the entry $D[l'^{\mathsf{ind}}]$ at time t implies $D[l'^{\mathsf{ind}}].hnd_w = \downarrow$, which yields a contradiction.

Part c) of the invariant can only be violated if a new entry $D[k]$ is created at time $t+1$ with $l' \in D[k].arg$ such that $x^{\mathsf{ind}}_j \in D[l'].arg$ (by Lemma 5.1 and well-formedness). As $D[l']$ already exists at time t, *encrypted-key secrecy* for $D[l']$ implies $D[l'].hnd_w = \downarrow$ for $w \notin \{u, v, \mathsf{T}\}$ at time t. We can easily see by inspection of the commands that the new entry $D[k]$ must have been created by one of the commands list and sym_encrypt (or by encrypt or sign, which create an asymmetric encryption or a signature, respectively), since entries newly created by other commands cannot have arguments that are indices of entries of type list. Since all these commands entered at a port $\mathsf{in}_z?$ presuppose $D[j].hnd_z \neq \downarrow$, the entry $D[k]$ is created by $w \in \{u, v, \mathsf{T}\}$ at time $t+1$. However, the only steps that can create an entry $D[k]$ with $l' \in D[k].arg$ (with the properties demanded for the entry $D[l']$) are Steps 3.11 and 3.13. In all these cases, we have $D[k].type = \mathsf{symenc}$. Further, we have $D[k].arg[2] = pkse_{w'}$ where w' denotes w's current believed partner. We have to show that $w' \in \{u, v\}$. Let $t^{\mathsf{ind}}_1 := D[hnd_{\mathsf{T}} = l^{(1)^{\mathsf{hnd}}}_3].arg[1]$ and $t_1 := D[t^{\mathsf{ind}}_1].arg[1]$ at the time of Step 3.10.

Since the entry $D[x^{\mathsf{ind}}_t]$ is created immediately before in Step 3.9, we have that the entry $D[k]$ has been created in Step 3.11 is the first database entry with the properties demanded for $D[k]$. If $i = k$, then we have $w' = u$ by construction and we are done. If $i \neq k$ then nothing has to be shown since no entry $D[i]$ exists yet for which we have to show something.

If $D[k]$ has been created in Step 3.13 we only have to show something if $D[i]$ has been created before in Step 3.11. In this case Step 3.2 and the check in Step 3.8, imply $w' = t_1 = D[t^{\mathsf{ind}}_1].arg[1] =$

$D[x_1^{\mathsf{ind}}].arg[1] = x_1 = v$ by definition of $D[i]$.

Hence in both cases we obtained $w' \in \{u, v\}$, i.e., the list containing the symmetric key was indeed encrypted with the key of either u or v. ■