Cryptographically Sound Theorem Proving[*]

Christoph Sprenger[1], Michael Backes[2], David Basin[1],
Birgit Pfitzmann[2], and Michael Waidner[2]

[1] ETH Zurich, Switzerland, {sprenger,basin}@inf.ethz.ch
[2] IBM Zurich Research Lab, Switzerland, backes@cs.uni-sb.de,
{bpf,wmi}@zurich.ibm.com

Abstract. We describe a faithful embedding of the Dolev-Yao model of Backes, Pfitzmann, and Waidner (CCS 2003) in the theorem prover Isabelle/HOL. This model is cryptographically sound in the strong sense of reactive simulatability/UC, which essentially entails the preservation of arbitrary security properties under active attacks and in arbitrary protocol environments. The main challenge in designing a practical formalization of this model is to cope with the complexity of providing such strong soundness guarantees. We reduce this complexity by abstracting the model into a sound, light-weight formalization that enables both concise property specifications and efficient application of our proof strategies and their supporting proof tools. This yields the first tool-supported framework for symbolically verifying security protocols that enjoys the strong cryptographic soundness guarantees provided by reactive simulatability/UC. As a proof of concept, we have proved the security of the Needham-Schroeder-Lowe protocol using our framework.

1 Introduction

Security proofs of cryptographic protocols are known to be difficult and the automation of such proofs has been studied soon after the first protocols were developed. From the start, the actual cryptographic operations in such proofs were idealized into so-called Dolev-Yao models, following [25, 26, 45], e.g., see [35, 56, 1, 41, 52, 13]. This idealization simplifies proof construction by freeing proofs from cryptographic details such as computational restrictions, probabilistic behavior, and error probabilities.

The first Dolev-Yao model with a cryptographic justification under arbitrary active attacks was introduced by Backes, Pfitzmann, and Waidner in [10] and extended in [11, 8]. This model, henceforth called the *BPW model*, can be implemented in the sense of reactive (blackbox) simulatability (BRSIM) [10] by real cryptographic systems that are secure according to standard cryptographic definitions. The security notion of BRSIM means that one system (here, the cryptographic realization) can be plugged into arbitrary protocols instead of another system (here, the BPW model), while retaining essentially arbitrary security properties [54, 19, 21, 12, 9]; it is also called UC for its universal composition properties. The BPW model currently constitutes the only Dolev-Yao model that is known to fulfill this strong security notion, as other soundness results are restricted to specific security properties or protocol classes.

The BPW model constitutes a deterministic, symbolic abstraction of a comprehensive set of cryptographic operations and allows one to prove the security of arbitrary protocols

---

built from these operations with respect to the cryptographic definitions by means of symbolic reasoning techniques, e.g., see the paper-and-pencil proofs in [7, 5, 6]. In order to relate the BPW model to a cryptographic realization in the sense of BRSIM/UC, the BPW model maintains certain *non-standard aspects* compared to other Dolev-Yao models. For example, abstract ciphertexts in the BPW model do not hide the length of their respective plaintexts, a signature over a specific message can be transformed by the adversary into another signature over the same message, and the protocols built on top of the BPW model do not directly manipulate messages, but use pointers (called *handles*) to refer to the messages being manipulated by the respective operations. While these aspects prevent the direct use of existing tools for symbolic protocol analysis, they are necessary to achieve the cryptographic soundness of the BPW model with respect to the strong soundness notion of BRSIM/UC.[1]

The complexity of the BPW model raises the following question, whose answer was initially unclear to us: Is it possible to reason efficiently about protocols based on this model using a theorem prover and without sacrificing the strong soundness guarantees? The main obstacle for an efficient mechanization is the complex state space, which includes message buffers, references to messages via handles, and the representation of messages themselves by a pointer-like data structure. Standard techniques for reasoning about state-based systems, such as Hoare logics and weakest precondition calculi, scale poorly to complex state spaces and pointer structures. It is helpful to distinguish two types of complexity in the BPW model: the inherent complexity required for BRSIM/UC cryptographic soundness, which cannot be eliminated, and the complexity due to particular modeling choices. Fortunately, we are able to reduce the latter kind of complexity, by employing a series of carefully chosen abstractions, to the point where we can positively answer the question raised above.

Our Contributions Our main contribution is a simplified and more abstract version of the BPW model and its formalization in the theorem prover Isabelle/HOL [51], the higher-order logic (HOL) [22, 4, 29] instance of the generic logical framework Isabelle. Our Isabelle/HOL theories are conservative extensions of HOL (i.e., the proofs rely only on the axioms of HOL) and constitute the first framework that combines machine-assisted symbolic reasoning about security protocols with the strong cryptographic soundness provided by the notion of BRSIM/UC.

This contribution has two parts. First, to support reasoning about state-based programs, we have embedded several *program logics* in Isabelle/HOL, including a weakest precondition calculus (WPC) and a Hoare logic for pre-/postcondition properties, and a linear-time temporal logic (LTL) for temporal properties. Using standard techniques, proofs of temporal properties are reduced to pre-/postcondition assertions in Hoare logic, which can in turn be reduced to the WPC. These are general-purpose reasoning tools, which can be reused in other contexts. Our general *proof strategy* is to employ the WPC, which uses rewriting to efficiently compute weakest preconditions, to automatically prove lemmas about the lower layers of our model (e.g., the functions of the BPW model). These lemmas are then combined in Hoare logic proofs at the higher layers (e.g., the protocol). Second, we have

---

[1] Weaker cryptographic soundness results that consider restricted security properties or restricted protocol classes might not have to maintain such aspects and thus be accessible to existing proof tools, cf. the paragraph on further related literature for more details.

produced two formalizations of the BPW model, each of which abstracts different features of the original model, while both faithfully represent its non-standard aspects.

In the first formalization, called the *indexed BPW model*, the component and communication model are abstracted into a light-weight, shallow embedding in Isabelle/HOL: machines and message buffers are simplified into state-manipulating components providing a set of interface functions and communicating by function invocation. However, the data representation closely follows the original BPW version: messages are represented by pointer-like structures with sharing of submessages between different protocol participants. Unfortunately, this abstraction step is insufficient in itself; our first attempt to prove the security of the Needham-Schroeder-Lowe protocol based on the indexed BPW model failed, essentially due to a lack of abstraction in both the model (complex pointer structures) and the specifications (complicated invariants). As a consequence, the WPC was either too slow to be useful or produced very large expressions that were difficult to understand. Moreover, they could not be further simplified, since an appropriate equational theory was not available. Thus, we had to resort to Hoare logic reasoning at low layers of the model, which required substantial user interaction and complicated intermediate preconditions.

In our second formalization, called the *term-based BPW model*, we address these problems by replacing the pointer-like messages with a simple inductive data type of messages. Since the new representation eliminates message sharing between users and, moreover, handles asymmetric key pairs and message lengths differently, its equivalence with the indexed model is non-trivial. To ensure the correctness of this step, we have proved in Isabelle/HOL that our two formalizations are strongly bisimilar. Since bisimilarity preserves BRSIM/UC, it is safe to replace the indexed model with the term-based model in protocol security proofs. The term-based model makes efficient automatic reasoning possible in two ways. First, it provides messages with a simple inductive structure that enables standard structural induction. This was not possible in the indexed model. Second, it enables concise property specifications using functional DY-like closure operators, such as Paulson's *analyze* and *parts* [52], which close a set of messages under cryptographically accessible submessages and all submessages, respectively. In fact, we were able to transfer Paulson's corresponding Isabelle/HOL theories to this term-based setting. The equational theories associated with these operators enable the efficient use of Isabelle's term rewriter for simplification. Overall, the combination of these two enhancements drastically improves the usability and performance of the WPC on the term-based BPW model when compared to the indexed version.

Our second contribution is the specification and verification of the security of the Needham-Schroeder-Lowe protocol in the term-based BPW model (and thus, by BRSIM/UC, also for the actual cryptographic implementation of the protocol). We consider this a proof of concept for our formalization and proof techniques. Note in this regard that [7, 57] have presented sound paper-and-pencil proofs of the NSL protocol and sound, tool-supported proofs have been given by [47] (exploiting a soundness result for restricted protocol classes and properties) and [20] (exploiting a recent soundness result with compositionality guarantees for specific protocol classes). However, our proof demonstrates that relatively efficient cryptographically sound proofs in the sense of BRSIM/UC are indeed possible and thereby provides evidence that this formalized framework can be successfully applied to reason about many commonly studied protocols.

Further Related Work Early work on linking Dolev-Yao-style symbolic models and cryptography [3, 2, 30, 36] only considered passive attacks, and therefore cannot make general statements about protocols. The same holds for [31].

The security notion of BRSIM was first defined generally in [53], based on simulatability definitions for secure (one-step) function evaluation [27, 28, 15, 46, 18]. It was extended in [54, 19], called UC (universal composability) in the latter, and has been widely applied to prove individual cryptographic systems secure and to derive general theoretical results.

A cryptographic justification of a Dolev-Yao model in the sense of BRSIM/UC was given in [10] with extensions in [11, 8]. Later papers [47, 37, 20] considered to what extent restrictions to weaker security properties or less general protocol classes allow simplifications compared with [10]: Laud [37] has presented cryptographic foundations for a Dolev-Yao model of symmetric encryption but specific to certain confidentiality properties where the surrounding protocols are restricted to straight-line programs. Herzog et al. [31] and Micciancio and Warinschi [47] have presented cryptographic underpinnings for a Dolev-Yao model of public-key encryption, where the former result relies on a stronger assumption than [10] and the latter restricts the classes of protocols and protocol properties that can be analyzed using this primitive. Cortier and Warinschi [23] have considered secrecy aspects by showing that symbolically secret nonces are also computationally secret, i.e., indistinguishable from a fresh random value given the view of a cryptographic adversary. Baudet, Cortier, and Kremer [14] have established the soundness of equational theories in a Dolev-Yao model under passive attacks. We stress that the imposed restrictions on protocol classes or protocol properties in the aforementioned works eliminated at least some of the complications that are necessary if soundness in the stronger sense of BRSIM/UC is desired, and that these Dolev-Yao models might thus be accessible to existing verification tools without major adaptations.

Canetti and Herzog [20] have recently shown that a Dolev-Yao-style symbolic analysis can be conducted using the framework of universal composability for a restricted class of protocols, namely mutual authentication and key exchange protocols with the additional constraint that the protocols must be expressible as loop-free programs using public-key encryption as their only cryptographic operation. Concentrating on this specific protocol class permitted the direct use of the automatic verification tool ProVerif [16] to symbolically analyze secrecy aspects of the Needham-Schroeder-Lowe protocol by considering the exchanged nonces as secret keys. This work is the closest to ours since it achieves universal composition guarantees (for the case where this protocol class is composed into larger protocols), in contrast to all of the aforementioned results. However, the results are restricted to the functionalities noted above and hence do not provide soundness guarantees of a Dolev-Yao model in the sense of BRSIM/UC (which guarantees soundness for composing arbitrary protocols). Extending their work to achieve this stronger notion would require augmenting their model with at least some of the non-standard aspects of the BPW model, thus raising the need for a tailored verification framework as well.

Efforts are also under way to formulate syntactic calculi for dealing with probabilism and polynomial-time considerations, in particular [48, 40, 49, 33] and, as a second step, to encode them into proof tools. Datta, Derek, Mitchell, Shmatikov, and Turuani [24] have proposed a promising, comprehensive logic that enables them to prove computational security properties using a logical deduction system. Laud [38] has designed a type system for proving security protocols based on the BPW model. We are however not aware of any mechanized implementations of these frameworks.

Blanchet has recently presented an automated tool for proving secrecy properties of cryptographic protocols that relies directly on the cryptographic approach by transforming cryptographic games expressed in a probabilistic polynomial-time calculus [17]. This approach appears to be highly promising not only due to its ability to analyze security protocols without relying on abstractions of cryptography, but also because of its potential to complement the line of work on proving soundness of Dolev-Yao models by formally validating the existing paper-and-pencil proofs of soundness.

Organization In Sect. 2, we briefly review the BPW model and describe the encoding of the component and communication model underlying the BPW model in Isabelle/HOL. We formally introduce the indexed and the term-based BPW models in Sect. 3 and sketch the proof of their strong bisimilarity. In Sect. 4, we formally define the composition of a protocol with the BPW model in Isabelle. This yields a flexible template that can be instantiated with arbitrary protocol specifications. We specify the Needham-Schroeder-Lowe protocol and sketch its proof of security in Sect. 5. Finally, in Sect. 6, we draw conclusions and discuss future work.

2 Overview of the BPW Model and its Formalization

In this section, we review the BPW model and discuss the principal abstractions and design choices that we made in its formalization. The cryptographic realization of the BPW model and details from the proof of cryptographic soundness are not necessary for understanding the contributions of this paper and can be found in the original papers.

2.1 BPW Model

The BPW model constitutes a library of cryptographic operations, which keeps track of, and controls access to, the terms known by each party. The BPW model provides *local functions* for operating on terms and *send functions* for exchanging terms between an arbitrary, but fixed, number N of users and the adversary. Some of these functions reflect distinguished attack capabilities and are only offered to the adversary. At the interface, terms are referred to indirectly by *handles* (also called *pointers* or *local names* in other terminologies). This indirection is necessary for the cryptographic soundness proof of the BPW model in the strong sense of BRSIM/UC, since the BPW model and its cryptographic realization work with vastly different objects: abstract terms and bitstrings, respectively. Handles present these syntactically different objects in a uniform manner to the users and hence avoid that the BPW model can be trivially distinguished from its realization because of different interfaces.

To analyze a security protocol based on the BPW model, one reasons about a system where each user u runs its own protocol component P_u, which is implemented by invoking the respective functions of the BPW model (Fig. 1). Each protocol component maintains its own local state (e.g., to store the nonces it has generated) and provides interfaces for communicating with its user and with the BPW model. Fig. 1 depicts two typical control flows through the system: First, a user may give input to initiate the protocol, which then constructs a term corresponding to the first protocol message through a series of local interactions with the BPW model. Local means that term construction does not involve
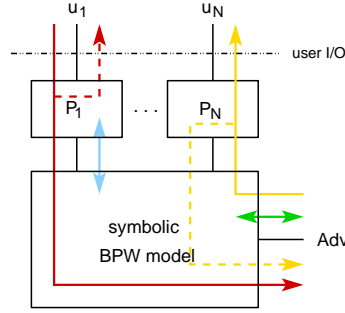
Fig. 1. System components and control flow

any interaction with the adversary; hence terms can be arbitrarily nested without revealing their structure or the contents of subterms to the adversary during construction. The term constructed may then be sent to the network (i.e., the adversary). Second, the adversary may decompose terms and construct new ones by local interactions with the BPW model, and send terms to users. The BPW model delivers terms sent by the adversary to the protocol component of the respective user, where they are then processed according to the protocol description.

2.2 Isabelle/HOL Preliminaries

Isabelle is a generic theorem prover, in which a variety of logics have been implemented. We use an implementation of higher-order logic (HOL), which can roughly be seen as logic on top of functional programming. We will assume that the reader has basic familiarity with both logic and typed functional programming. Proof automation in Isabelle is supported by a powerful simplifier that performs term rewriting and a tableau reasoner. These are invoked in isolation or in combination using different proof tactics.

In Isabelle notation, t::T denotes a term t of type T. The expression $c\ x \equiv t$ defines the constant c with the parameter x as the term t. Definitions constitute the principal mechanism for producing conservative extensions of HOL. Type variables are identified by a leading apostrophe, as in $'a$. Given types $'a$ and $'b$, $'a \Rightarrow 'b$ is the type of (total) functions from $'a$ to $'b$, $'a \times 'b$ is the product type, and $'a\ set$ is the type of sets of elements of type $'a$. The type *unit* contains a single element. There are several mechanisms to define new types. A **datatype** declaration introduces an inductive data type. For example, the option type is defined by **datatype** $'a\ option = None \mid Some\ 'a$, which is polymorphic in the type variable $'a$. Functions of type $'a \Rightarrow 'b\ option$ are used to model partial functions from $'a$ to $'b$. The declaration **types** $T1 = T2$ merely introduces a new name for the type $T2$, possibly with parameters, as in **types** $'a \rightharpoonup 'b = 'a \Rightarrow 'b\ option$. Isabelle/HOL includes a package supporting extensible record types. For example, **record** *point* $= x$::*nat* y::*nat* defines a record type for points, of which the record $(\!|x=1,\ y=2|\!)$ is an element. Records are extensible: **record** *cpoint* $= point + c$::*color* extends points with a color field. Behind the scenes, the definition of a record type r creates a record scheme $'a\ r\_scheme$, which extends the declared type r with a polymorphic field *more* of type $'a$. The type r is derived as *unit r_scheme*. Record extensibility is based on the instantiation of the record scheme parameter $'a$ with one or more additional fields. Im-

6

portant for our formalization is that all extensions of *r* are compatible with the scheme *r_scheme*. For example, *cpoint* is compatible with *'a point_scheme* (but not with *point*).

2.3 Overview of the Formalization

We now summarize the abstraction steps and design choices that we have employed to simplify the component and communication model as well as the operational semantics underlying the BPW model. These simplifications enable a sound, light-weight formalization of the BPW model, the protocols, and their properties.

Component and Communication Model From a typed perspective, the BPW model and the protocol components can be represented by deterministic machines with transition functions of type $\Sigma \times I \Rightarrow (O\ option) \times \Sigma$, where Σ is the machine's state space and *I* and *O* are inputs and outputs, respectively. The types *I* and *O* as can be seen as (non-recursive) inductive data types, where each constructor corresponds to a port name and its arguments correspond to the values communicated over that port. Output is optional; usually, the absence of output indicates that an error occurred. The general communication framework underlying the BPW model stores messages in transit in so-called buffers, until the messages are scheduled by the designated scheduler for the respective connection. The most important special case is that machines are in charge of their outgoing connections themselves and schedule outgoing messages immediately, i.e., messages are passed on directly from sender to recipient. Since this is the case for the communication between the BPW model and the protocol components, these buffers can be safely omitted in the formalization leading to a substantial simplification.

Essentially, each machine transition can be seen as a function call (with parameters passed at some input port) and producing either a return value (at some output port) or an exception. Therefore, our formalization replaces the machine description of the BPW model by components consisting of a set of interface functions manipulating a common state, where communication over ports is replaced by function calls. Since state and exceptions play a crucial role in the BPW model, they are handled by appropriate abstractions in our formalization. In a purely functional context, such as Isabelle/HOL, such abstractions are provided by monads [50, 39]. Generally speaking, a monad *M* is a type constructor equipped with unit and composition operations, enjoying unit and associativity properties, respectively. A monadic interpretation of an operation with input type *A* and output type *B* has the function type $A \Rightarrow B\ M$. Different monads can represent a wide range of computational phenomena including state, exceptions, and non-determinism. Here, we use the deterministic state-exception monad *S*:

```
datatype 'a result  = Exception | Value 'a          −− result  type
types ('a,'s) S = 's ⇒ 'a  result  × 's             −− monad type

return :: 'a ⇒ ('a,  's) S                          −− monad unit
return a ≡ λs. (Value a,  s)

bind :: ('a,'s) S ⇒ ('a ⇒ ('b,'s) S) ⇒ ('b,'s) S    −− monad composition
bind m k ≡ λs. let (a,  t) = m s in
                    case a of Exception ⇒ (Exception,  t) | Value x ⇒ k x t
```

Note that the monad S is polymorphic in both the type of values and the type of states. The unit (called **return**) embeds a value in the monad and *bind* is a sequential composition, which passes results (values or exceptions) between function calls. We write **do** $x \leftarrow m;$ $k\,x$ instead of *bind m k*. There are also monad-specific operations for state assignment and for throwing and catching exceptions. The set of all these monad operations forms a simple imperative language that we use to formulate our models.

Runs and observations In the communication framework underlying the BPW model, a system run is defined as a sequence of local transitions of the form $(M,\,s,\,i,\,o,\,t)$, where M names the machine making the transition from the local state s and the input i to the local state t, and producing the output o (if any). This corresponds to a *small step semantics*, where the transitions of all individual machines are considered. The *user view* is derived by projecting runs on the transitions performed by the honest users. Our formalization uses a *big step semantics*, where internal transitions and communication are hidden. A transition consists of a pair of states $(s,\,t)$, where t is reached from s by calling an interface function. Formally, the transition relation for a monadic function $f :: A \Rightarrow (B,\,S)\,M$ is defined by $tr\ f \equiv \{(s,t) \mid \exists\,r\,a.\ f\ a\ s = (r,\,t)\}$. The transition relation of a component is the union of the relations derived from the components' interface functions. A run is a sequence of states arising from component transitions, triggered by external input. The big step semantics arises naturally given our procedural view of communication, and it clearly preserves BRSIM/UC since system-internal transitions do not affect the user view. Most importantly, a big step semantics facilitates proofs, since it supports the top-down case analysis of the system interface functions in invariant proofs, without the need to show that the invariant is preserved by each internal transition. Another design choice leading to simpler proofs is that we do not model user I/O events as part of each transition; instead we record I/O traces in a global history variable, which is extended with every I/O event. This can be seen as an observer component that logs all communication with the users. The advantage of having the entire trace available in each state is that precedence properties with reference to the past can be expressed as simple invariants (sets of states).

In the definition of reactive simulatability, users and the adversary constitute probabilistic, polynomially-bounded machines. In our formalization, we model them by universal quantification over all possible inputs, i.e., a single unbounded machine which non-deterministically produces arbitrary input to the system in each transition. This safely over-approximates the original setting, since the unbounded machine can (weakly) simulate any set of probabilistic, polynomially-bounded users and adversary.

Finally, the BPW model includes polynomial bounds on the length of handled messages and on the number of steps that each machine can perform. We have formalized the enforcement of the message-length bound using an uninterpreted function of the security parameter as the bound. This comprises, in particular, all polynomial functions and thus constitutes a safe over-approximation. Step bounds are dealt with similarly.

Program Logics and Verification Tools We conclude this section with a brief overview of the specification and proof machinery that we have constructed for verifying protocol properties. While the present paper concentrates on the *modeling* of the BPW model in Isabelle/HOL, a companion paper will be devoted to proof tools and techniques. We use several program logics and proof systems to specify and verify security properties: first, a weakest precondition calculus (WPC) based on Pitts' evaluation logic [55] and a Hoare

logic [32] on top of it, both tailored to our state-exception monad and, second, a linear-time temporal logic (LTL) to specify temporal behavior such as invariants or precedence properties [43]. We have derived a set of proof rules, similar to those of [42, 44], to reduce LTL properties to pre-/postcondition statements in Hoare logic, i.e., Hoare triples. We prove these Hoare triples by using the rules of Hoare logic or by unfolding them to statements of the WPC. The WPC allows us to automate proofs to a large extent, whereas the Hoare logic gives us manual control, when automation fails. These logics and tools are problem-independent and can be reused in different contexts.

3 Formalization of the BPW Model

Building on the simplified modeling framework outlined in Sect. 2.3, we present two formalizations of the BPW model in Isabelle/HOL. The first one, called the *indexed BPW model*, closely adheres to the original data representation of the BPW model. The second one, called the *term-based BPW model*, abstracts the representation of messages to inductively defined terms. Finally, we describe the bisimulation relation used in the proof of their equivalence. Both versions share the types of *parties* and *knowledge maps*:

datatype *party = User user* │ *Adv*
types *'a kmap = party ⇒ hnd ⇀ 'a*

Here, *user* denotes the type of honest users, which is isomorphic to the set $\{1..N\}$, and *hnd* is the type of handles, which is isomorphic to the set of natural numbers. Knowledge maps keep track of who knows what. They also serve as an access control mechanism by mediating between the handles at the interface and the internal representation of messages (of generic type *'a*).

3.1 Shared Messages: the Indexed BPW Model

Our first formalization of the BPW model remains close to the original BPW model by using a pointer-like structure to represent messages. The state consists of a database storing messages, which are referred to by indices (of type *ind*, isomorphic to the natural numbers), together with a knowledge map instantiated to indices.

record *'d iLibState =*
  db :: ind ⇒ 'd entry   −− the database
  knowsI :: ind kmap   −− knowledge map

The database can be seen as a heap where entries are allocated. The knowledge map records which entries are known by which parties. We say that a database index is *defined*, if it is known by some party. Database entries have a content and a length field. Our presentation covers public-key encryption, but omits signatures for brevity.

datatype *'d content =*
    iNonce   −− nonce
  │ *iGarbage*   −− garbage
  │ *iPke ind*   −− public encryption key
  │ *iSke*   −− private encryption key
  │ *iData 'd*   −− payload data
  │ *iPair ind ind*   −− pair
  │ *iEncv ind ind*   −− valid ciphertext

9

```
      |  iEnci  ind                          −− invalid  ciphertext

  record 'd  entry  =
    cont :: 'd  content                       −− content
    len :: nat                                −− length  of  entry
```

Elements of the data type *'d content* correspond to message constructors, polymorphic in the type *'d* of payload messages, which depends on the application. Constructor arguments of type *ind* point to other entries in the database corresponding to submessages. For example, in the term *iEncv pki mi*, which represents a valid encryption, the first argument points to the public key used and the second to the message being encrypted. Also, each public key points to the matching secret key of the key pair. In contrast to commonly used Dolev-Yao models, our adversary may create garbage entries (constructor *iGarbage*) or invalid ciphertexts (constructor *iEnci*). In a well-formed database, each defined index determines a directed acyclic graph, the indexed BPW model representation of a *message*. We call payload data and pairs *non-cryptographic* messages and all others *cryptographic* messages. The length field in each entry is used to enforce a bound on the message length.

The BPW model interface functions manipulate the knowledge map and the database. As examples of local interface functions, the main operations for public key cryptography have the following types:

$$gen\_enc\_keypairI :: party \Rightarrow (hnd \times hnd, ('d, 's) \ iLibState\_scheme) \ S$$
$$encryptI , \ decryptI :: party \Rightarrow hnd \Rightarrow hnd \Rightarrow (hnd, ('d, 's) \ iLibState\_scheme) \ S$$

The function *gen_enc_keypairI* returns a public/secret key pair, *encryptI* takes a public key and a cleartext and returns the ciphertext, and *decryptI* takes a secret key and a ciphertext and returns a cleartext. Message arguments and results are referred to by handles. If some argument is invalid, an exception is raised. Note that these functions operate on the record scheme *('d, 's) iLibState_scheme* instead of the plain state record *'d iLibState* . Here, *'s* stands for future extensions of the state, for example, with the protocol state (Sect. 4). By using extensible records, all invariants proved about the BPW model automatically carry over to all future extensions of the state without any explicit lifting. We apply the same technique to the term-based BPW model.

One of the main differences between this model and other Dolev-Yao models is that each encryption of a given message with the same public key (both referred to by handles) results in a different ciphertext, that is, a new database entry pointed to by a fresh handle. This reflects the fact that secure encryption is necessarily probabilistic and shows the role of indices in modeling idealized randomness. In fact, all functions constructing cryptographic messages produce fresh database entries with each invocation. The situation is different for non-cryptographic messages, which are allocated only once and are shared between users. In other Dolev-Yao models, freshness is often introduced by side conditions on cryptographic messages, for example, requiring that a nonce has not occurred so far in a message on the network. Another important difference with other Dolev-Yao models is that the adversary (but not honest users) can learn the length of the cleartext underlying a ciphertext (via a separate function *adv_parse*, not shown here), thus modeling a length-revealing crypto system.

We have proved three basic invariants of the indexed BPW model, which are needed in the bisimulation proof (Sect. 3.3) and express well-definedness conditions: the knowledge map has a finite domain for each user and it is injective on that domain, and the arguments of entries at defined indices are themselves defined.

With respect to the original BPW model, we have made a number of simple abstractions in our formalization. First, we have factored out the access control lists in the entries of the original version into our isomorphic representation using knowledge maps, thus isolating a common element of our two formalizations. Second, we have replaced lists by pairs, without loss of generality. Pairs are sufficient for modeling concrete protocols and, because they are not recursively defined, simplify reasoning by obviating the need for certain inductive arguments. Third, we have abstracted the allocation of new objects such as indices and handles from a counting scheme to an arbitrary (but still deterministic) allocation scheme[2]. As a consequence, public key pairs are linked via an explicit pointer from the public to the secret key, instead of allocating them at successive indices. Again, this abstraction pays off by simplifying reasoning: an extra invariant making the link between key pairs explicit becomes obsolete.

As explained in the introduction, these abstractions turned out to be insufficient for a practically useful verification framework. The main problems arose from the lack of an inductive message structure supported by standard structural induction and from complicated ad hoc property specifications expressed without knowledge and subterm derivation operators (such as Paulson's *analyze* and *parts* [52]). Even though such operators could be defined in the indexed model, the fact that messages in the indexed BPW model do not exists independently of the state would complicate their definition and the derivation and application of the associated equational theories. These problems are addressed by our second, term-based formalization of the BPW model.

3.2 Inductively Defined Messages: the Term-Based BPW Model

Fortunately, the sharing of messages between different users in the indexed BPW model is inessential and can be eliminated. A more abstract representation of messages can be obtained using an inductive data type of messages. Isabelle automatically generates an induction scheme for each inductive data type. (Signatures are omitted for brevity.)

```
datatype 'd msg =
    mNonce tag                        −− nonce
  | mGarbage tag len                  −− adversary garbage
  | mPke key                          −− public encryption key
  | mSke key                          −− private decryption key
  | mData 'd                          −− data item
  | mPair ('d msg) ('d msg)           −− pair of messages
  | mEncv tag key ('d msg)            −− valid ciphertext
  | mEnci tag key len                 −− invalid ciphertext
```

This data-type definition replaces the previous index arguments in the content fields of database entries by recursive message arguments. Moreover, there are two other notable changes in moving to this representation. First, the role played by indices in allocating fresh database entries for cryptographic messages is taken by the elements of a new, but isomorphic, type *tag*, which can be thought of as an (abstraction of) random coins. The type *key* is just another name for *tag*. Matching key pairs are then simply those of the form (*mPke k, mSke k*). Instead of replacing the first argument of the encryption constructors by a recursive message argument, we directly record the corresponding key, thus avoiding unnecessary well-formedness conditions on messages. Second, we now determine the

---

[2] We use Hilbert's ε-operator, where $\varepsilon x.\ x \notin A$ picks some fresh x not in A, if there is one.

length of messages by a partially interpreted recursive function *len_ofM* :: *'d msg* ⇒ *len*, which allows us to remove redundant length information from the state. Length fields are still required for garbage and invalid ciphertexts, as the adversary can choose an arbitrary length for these two atomic message types.

This abstraction step substantially simplifies the structure of states by eliminating the need for the database and (largely) for length fields: a state of the term-based BPW model simply consists of a knowledge map storing messages:

record *'d mLibState* = *knowsM* ::*'d msg kmap*

This economy of state variables, together with our ability to reason inductively about messages, leads to a quite dramatic improvement in proof automation.

The second substantial improvement, which leads to more concise specifications and improved proof automation, stems from adapting to our setting the closure operators *parts* and *analyze* and their equational theories developed by Paulson [52]. The term *parts H* denotes the closure of the set of messages *H* under all submessages, whereas *analyze H* closes *H* under all cryptographically accessible submessages. Hence, the expression *analyze* (*ran* (*knowsM s u*)) denotes the set of messages that the party *u* can derive from his knowledge in state *s* (*ran f* denotes the range of the partial function *f*). Using *analyze* and *parts*, we define secrecy as follows:

secret :: (*'a*, *'b*) *mLibState_scheme* ⇒ *'a msg* ⇒ *party set* ⇒ *bool*
secret s m U ≡ *parts* {*m*} = {*m*} ∧
  (∀ *u*. *m* ∈ *analyze* (*ran* (*knowsM s u*) ∪ *ran* (*knowsM s Adv*)) ⟶ *u* ∈ *U*)

The proposition (*secret s m U*) means that message *m* is a secret shared by the set of parties *U* in state *s*. Note that we require secrets to be atomic. For the definition of non-atomic secrets we would need a *synthesize* operation corresponding to message construction on top of *analyze*, since secrets could possibly be built from already known messages. The inclusion of the adversary knowledge strengthens the definition and is exploited in invariant proofs, as we will see in Sect. 5.3.

3.3 Bisimulation with Indexed BPW Model

We now establish the bisimilarity of our two formalizations of the BPW model. By this result, both versions yield identical views to the honest users which trivially preserves BRSIM/UC. Due to the close correspondence described by the bisimulation, even state-based properties can be easily translated from the term-based to the indexed version.

The bisimulation proof shows that all pairs of interface functions transform bisimilar states into bisimilar states with identical output on all possible inputs. Since the interface functions are deterministic, this is sufficient to establish a bisimulation between the two versions[3]. We are thus using a shallow embedding of bisimulation: the notion of bisimulation itself is not formalized explicitly. The message abstraction relation

message s i2t :: (*ind* × *'d msg*) *set*

is the central element of our bisimulation: it associates database indices to messages and is parametrized by a state *s* of the indexed BPW model and a function *i2t* mapping indices to tags. The latter witnesses the fact that tags assume the role of indices for message

---

[3] Formally, this can be explained as an instance of coalgebraic bisimulation [34].

freshness. Note that the relation is defined independently of states of the term-based BPW model. The inductive definition of *message* contains a rule for each constructor of the type *'d content*. For example, the rule for valid ciphertexts reads:

$\llbracket$ *s* $\in$ *contains i* (*iEncv pki mi*); *tg* = *i2t i*;
  (*pki*, *mPke k*) $\in$ *message s i2t*; (*mi*, *m*) $\in$ *message s i2t* $\rrbracket$
$\Longrightarrow$ (*i*, *mEncv tg k m*) $\in$ *message s i2t*

This rule states that, at some fixed state *s*, an index *i* abstracts to the ciphertext message (*mEncv tg k m)* if the index *i* contains (*iEncv pki mi*), the index *pki* abstracts to the public key message (*mPke k*), the index *mi* abstracts to message *m*, and the tag *tg* is the image of *i* under *i2t*. The main property proved for *message* is its functionality.

The bisimulation relation essentially consists of pairs of states for which the domains of the knowledge maps are identical and the message at *knowsM s u h* (if defined) is an abstraction of the index at *knowsI s u h*.

I2M :: (*ind* $\Rightarrow$ *tag*) $\Rightarrow$ ((*'d*, *'s*) *iLibState_scheme* $\times$ (*'d*, *'s*) *mLibState_scheme*) *set*
I2M i2t $\equiv$ {(*s*, *t*). *bij i2t* $\wedge$
  ($\forall u$. *dom* (*knowsI s u*) = *dom* (*knowsM t u*)) $\wedge$
  ($\forall u$. $\forall h \in$ *dom* (*knowsI s u*).
    knowsI s u h = *Some i* $\wedge$ *knowsM t u h* = *Some m* $\longrightarrow$ (*i*, *m*) $\in$ *message i2t s*) }

We have actually defined a family of relations parametrized by a function *i2t* of type *ind* $\Rightarrow$ *tag*, which is required to be a bijection in order to map different database entries to different messages. The proper bisimulation relation is the union over all family members, i.e., the second-order property $R = \bigcup i2t. \; I2M \; i2t$. Since both indices and tags are freely allocated, but not all indices are associated with a tag (e.g. payload data is untagged), the parameter *i2t* cannot be determined statically. Defining *R* as the union over all parameters allows us to update *i2t* with mappings (*i*, *tg*), where *i* is a fresh index and *tg* is a fresh tag. Since the resulting map must again be a bijection, we achieve this update by swapping the values of *i2t* at *i* and $i2t^{-1}(tg)$.

The proof of bisimulation uses a set of derived proof rules similar to those of Hoare logic, but involving two components instead of just one as for invariant proofs. These rules rely on basic invariants proved for the indexed and the term-based BPW model.

4 Generic Protocol Modeling and Verification Framework

Based on the term-based BPW model, we model a generic framework for the specification and cryptographically sound verification of security protocols. Afterwards, we instantiate this framework to the concrete protocols under study.

4.1 Protocols and Observer

The global state extends the BPW model state with the local state for each protocol component and the trace observed at the user interface.

record (*'i*, *'o*, *'d*, *'s*) *globState* = *'d mLibState* +
  loc :: *user* $\Rightarrow$ *'s*   −− local state for each user
  trace :: (*'i*, *'o*) *trace*   −− observed user i/o trace

13

Our setup is polymorphic in four types: the type *'d* of payload data (from the BPW model), the type *'s* of local states, as well as *'i* and *'o*, the types of user input and output, respectively. Concrete protocols later instantiate these type parameters to concrete types. The observer trace is a history variable, where all user I/O events are recorded. Its type is a lists of pairs of a user name and an input or output event:

datatype (*'i, 'o*) *uio = uIn 'i | uOut 'o* —— user input / output
types (*'i, 'o*) *trace = (user × ('i, 'o) uio) list*

Next, we define the interface of protocol components. Each protocol component provides a user and a network input handler and may produce output either for the user or the network (Fig. 1).

datatype *'o proto_out = pToUser 'o | pToNet netmsg* —— protocol output

record (*'i, 'o, 'd, 's*) *proto_comp =*
  proto_user_handler :: 'i ⇒ ('o proto_out, ('i, 'o, 'd, 's) globState) S
  proto_net_handler :: user ⇒ hnd ⇒ ('o proto_out, ('i, 'o, 'd, 's) globState) S

types (*'i, 'o, 'd, 's*) *protocol = user ⇒ ('i, 'o, 'd, 's) proto_comp*

A protocol is then defined as a function from users to protocol components. The observer has a single interface function *log*, which simply adds an I/O event to the trace.

Standard Alice-and-Bob notation and Paulson's Isabelle protocol specifications [52] are centered around the protocol messages that are transmitted between the different roles. In the BPW model (and its formalization), we take a more process-oriented view by specifying the reaction of the protocol to user and network input. In particular, a protocol run is always initiated and terminated by explicit, observable, user I/O events, possibly with other user interaction in between. This user interaction enables the formulation of cryptographically meaningful properties about user I/O traces.

4.2 The Complete System

We compose the BPW model with the protocol and the observer, yielding the complete system. This system has two types of interface functions: the system user and network handlers and the local functions provided by the BPW model to the adversary. We restrict our presentation to the system user and network handlers, whose types are:

sys_user_handler :: ('i, 'o, 'd, 's) protocol ⇒ user ⇒ 'i ⇒
                    ('o sys_out, ('i, 'o, 'd, 's) globState) S
sys_net_handler :: ('i, 'o, 'd, 's) protocol ⇒ netmsg ⇒
                    ('o sys_out, ('i, 'o, 'd, 's) globState) S

Both handlers produce a system output of type *'o sys_out*, which is just the system-level version of type *'o proto_out*. The user handler takes an input from the user (of type *'i*), while the network handler takes a network message as an argument. Network messages are triples (u, v, h), where u is the supposed sender, v is the receiver, and h is a message handle. The BPW model provides two send functions, one for users and one for the adversary:

send_i, adv_send_i :: netmsg ⇒ (netmsg, ('d, 's) mLibState_scheme) S

By invoking *send_i* (*u*, *v*, *uh*), the user *u* sends the message denoted by his handle *uh* to the adversary (intended for user *v*). The result is a network message *send_i* (*u*, *v*, *ah*), where *ah* is the adversary's handle for the same message. Such a handle is created if it does not exist yet. The call *adv_send_i* (*u*, *v*, *ah*) has a similar effect, but this time the message is sent from the adversary to user *v*. Note that the adversary is free to falsify the name *u* of the originator. This gives the adversary complete control over the network, as in other Dolev-Yao models.

In order to illustrate the message flow through the system (cf. Fig. 1), let us consider the system network handler in more detail:

```
sys_net_handler  proto  anm ≡
    do (v,  u,  mh) ← adv_send_i anm;          −− get  message from network
    do pout ← proto_net_handler (proto  u)  v  mh;   −− handle  message
    case pout of
      pToUser uom ⇒
        do log (u,  uOut uom);                  −− log  output
        return (sToUser u uom)                  −− output  to  user
    | pToNet unm ⇒
        do anm ← send_i unm;                    −− output  to  network
        return (sToNet  anm)
```

Its input is a network message from the adversary, which he sends to the receiver *u* using the send function *adv_send_i*. The resulting network message contains a message for *u*, which is fed into the protocol network handler of the receiver's protocol component. The output of the handler is either intended for the user, in which case the output is logged by the observer and returned to the user, or it is a reply message that is sent back to the network (adversary) via the user send function *send_i*.

When specifying a concrete protocol in this framework, we need to provide the user and network handlers for our protocol. This determines concrete types for user I/O, payload data, and the local state of protocol components, instantiating the type variables *'i*, *'o*, *'d*, and *'s*. Once this is done, we are ready to specify and verify protocol properties.

5 A Cryptographically Sound Proof of the NSL Protocol

We model and verify the well-known three message version of the NSL protocol:

$$\text{NSL1.} \quad u \to v : \ \{N_u, u\}_{K_v}$$
$$\text{NSL2.} \quad v \to u : \ \{N_u, N_v, v\}_{K_u}$$
$$\text{NSL3.} \quad u \to v : \ \{N_v\}_{K_v}$$

Here, we assume that each user has generated an asymmetric key pair and that the authentic public keys of all users are known to every party. Below, we introduce our formal specification of the NSL protocol. Afterwards, we describe the invariants we have verified and sketch the proof of one such invariant. Finally, we discuss the benefits gained from the abstractions we have made.

5.1 Protocol specification

We specify the NSL protocol in our framework by defining a protocol component for each user. Each such component P_u records the set of nonces it generates in protocol sessions with user *v* in the local variable *nonces*, under the name of user *v*:

record *ustate* = *nonces* :: *user* $\Rightarrow$ *hnd set* —— set of nonce handles

We can initiate a protocol run by indicating the name of the responder. The protocol is terminated by returning the name of the initiator to the responder. Thus, both user input and output are of type *user*. Moreover, the only payload data used in the NSL protocol are user names. Therefore, we use an abbreviation for the states of the protocol:

types *NSLstate* = (*user*, *user*, *user*, *ustate*) *globState*

The NSL protocol is then specified by instantiating the user and network handlers:

```
NeedhamSchroederLowe ::(user, user, user, ustate) protocol
NeedhamSchroederLowe u ≡ (|
   proto_user_handler  = λv.                      —— initiate  protocol  with v
      do enforceb (u ≠ v);                        —— avoid  talking  to  self
      mk_msg1 u v,                                —— first   message

   proto_net_handler   = λv emh.                  —— respond to  protocol  messages
      do enforceb (u ≠ v);                        —— avoid  talking  to  self
      do pm ← parse_msg u v emh;
      case pm of
        msg1 vnh vid ⇒ mk_msg2 u v vnh            —— second message
      | msg2 unh vnh vid ⇒ mk_msg3 u v vnh        —— third  message
      | msg3 vnh ⇒ return (pToUser v)             —— terminate  protocol  with v
|)
```

The user handler for user *u* initiates a protocol run with user *v* by constructing the first protocol message. The network handler takes the name *v* of the sender and a message handle *emh*, parses the message and, depending on the result, replies by either producing a reply message or by terminating the protocol, indicating which user has (supposedly) been authenticated. The *enforceb* statements prevent a protocol component from talking to itself by throwing an exception if the stated condition is not satisfied.

As an example, we show the definition of *mk_msg1 u v*, which constructs the first protocol message (NSL1):

```
mk_msg1 ::user ⇒ user ⇒ (user proto_out, NSLstate) S
mk_msg1 u v ≡
      do nh ← gen_add_nonce u v;                  —— generate  and  register  nonce
      do uih ← store (User u) u;
      do mh ← pair (User u) (nh, uih)
      do emh ← encrypt (User u) (pke (User u) v) mh;
      return (pToNet (u, v, emh))                 —— send 1st message
```

In this definition, *pke (User u) v* denotes the handle by which user *u* refers to user *v*'s public key *mPke (ukey v)*. The statement *gen_add_nonce u v* generates a fresh nonce and adds it to *nonces (loc s u) v*, i.e. the nonces used by user *u* in sessions with user *v*. The subsequent calls incrementally construct the message.

5.2 Verified Properties

The main property we have proved is that the responder authenticates the initiator. This is formulated as a property of the observed user I/O trace and therefore transfers to the cryptographic level.
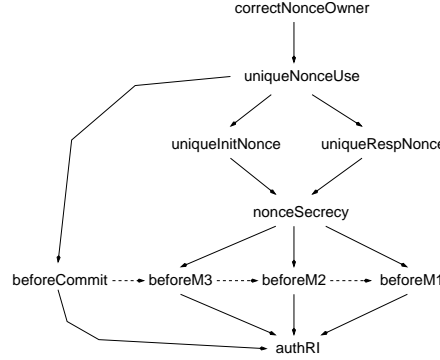
Fig. 2. Invariants of the NSL protocol

authRI :: *NSLstate set*
authRI ≡ {*s*. ∀ *I R. Commit R I* ∈ *set* (*trace s*) ∧ *I* ≠ *R* ⟶ *Init I R* ∈ *set* (*trace s*)}

Here *Init I R* is a nicer syntax for (*I, uIn R*) and *Commit R I* for (*R, uOut I*). We require that the initiator and responder are distinct. The observer history variable *trace* makes the entire trace of I/O events available in each state *s*. To express the authentication property is it sufficient to consider *set* (*trace s*), the unordered *set* of I/O events at state *s*. The use of a history variable to record I/O traces has the advantage of reducing temporal precedence properties with reference to the past to simple invariants (i.e. sets of states). The actual theorem states that *authRI* is an invariant. This is formulated in Isabelle as the LTL property:

theorem *authRI_invariant: NSLtrsys* ⊨ □(*Pred authRI*)

This theorem says that all states on all runs of the transition system *NSLtrsys* derived from the NSL protocol system satisfy *authRI*. The proof of this invariant is based on the auxiliary invariants listed in Fig. 2, along with their dependencies.

The basic invariants *correctNonceOwner* and *uniqueNonceUse* state properties of the local variable *nonces*: handles stored in this variable do indeed denote nonces and each nonce recorded in this variable is created by a unique user for a protocol session with a unique responder. The invariants *uniqueInitNonce* and *uniqueResponseNonce* express that the initiator nonce in message NSL1 and the responder nonce in NSL2 uniquely determine all the other fields of the respective message. Based on these invariants we can prove that the protocol nonces remain secret (invariant *nonceSecrecy*) between the protocol participants:

nonceSecrecy :: *NSLstate set*
nonceSecrecy ≡ {*s*. ∀ *u v n. n* ∈ *Nonces s u v* ⟶ *secret s* (*mNonce n*) {*User u, User v*}}

Here, *Nonces* is the set of nonces denoted by handles stored in the variable *nonces*:

Nonces :: *NSLstate* ⇒ *user* ⇒ *user* ⇒ *tag set*
Nonces s u v ≡ {*n*. ∃ *h. knowsM s* (*User u*) *h* = *Some* (*mNonce n*) ∧ *h* ∈ *nonces* (*loc s u*) *v*}

The notion of secrecy was already defined in Sect. 3.2. Finally, the authentication property *authRI* is derived from the conjunction of four auxiliary invariants, *beforeCommit*, *beforeM3*, *beforeM2* and *beforeM1*, each of these going one message back in the protocol (indicated by the dashed line in Fig. 2).

5.3 A Typical Invariant Proof

A protocol invariant is usually proved by showing that it is preserved by all system interface functions. As an example, let us consider the proof of *nonceSecrecy*. We point out some interesting cases and defer the discussion of our general proof strategy to Sect. 5.4.

We proceed bottom-up by showing that the invariant is preserved by all BPW model interface functions. Unsurprisingly, the most interesting cases are the send functions, where messages are exchanged between parties. The lemma for the user send function *send_i* reads as follows:

> **lemma** *nonceSecrecy_send_iN:*
> {*nS_send_pre u h* ∩ *nonceSecrecy* ∩ *correctNonceOwner* ∩ *finiteKnowsM*}
>   *send_i* (*u*, *v*, *h*)
> {$>$ λ*x. nonceSecrecy*}

This Hoare triple states that if *send_i* is called in a state satisfying the precondition and terminates normally, then the resulting state again satisfies *nonceSecrecy*. Note that previously proved invariants are used to strengthen the precondition. The basic auxiliary invariants *correctNonceOwner* and *finiteKnowsM* are sufficient to establish the preservation of *nonceSecrecy* by all BPW model interface functions except *send_i*, where we need the additional precondition *nS_send_pre*:

> *nS_send_pre* :: *user* ⇒ *hnd* ⇒ *NSLstate set*
> *nS_send_pre u h* ≡ {*s.* ∀ *m n w ua va.*
>   *knowsM s* (*User u*) *h* = *Some m* ⟶ *n* ∈ *Nonces s ua va* ⟶
>   *mNonce n* ∈ *analyze* ({*m*} ∪ *ran* (*knowsM s w*) ∪ *ran* (*knowsM s Adv*)) ⟶
>   *w* ∈ {*User ua, User va*} }

Note the similarity with the definition of *nonceSecrecy*. Intuitively, this predicate states that the message *m* to be sent (and referred to by the handle *h*) can be added to the knowledge of the adversary without compromising the secrecy of any protocol nonces. This precondition is formulated in a largely protocol-independent manner. It remains to be shown that our concrete protocol messages satisfy this condition.

Interestingly, the strengthening of the definition of secrecy obtained by adding the adversary knowledge under the *analyze* operator is essential. This has the effect that nonce secrecy is trivially preserved by the adversary's send function *adv_send_i*. Without this strengthening, the predicate *nS_send_pre* would arise as a precondition of *adv_send_i* and make that case unprovable, since we do not have any control over what messages the adversary may send to users. The strengthening shifts the precondition to the user side, where the protocol determines which messages are sent.

Using invariants *keySecrecy*, *uniqueInitNonce*, and *uniqueRespNonce*, we can indeed show that the protocol messages satisfy the precondition *nS_send_pre* (the BPW-model invariant *keySecrecy* guarantees that secret keys do not leak to the adversary). For example, for the preservation of *nonceSecrecy* by the system user handler, we have proved that *proto_user_handler* establishes a postcondition stating that message NSL1 has been constructed with a fresh nonce. Together with the invariant *keySecrecy*, this fact implies *nS_send_pre* for message NSL1. The cases for messages NSL2 and NSL3 are similar, but require the additional use of *uniqueInitNonce* and *uniqueRespNonce*, respectively.

The preservation results on the BPW-model level are easily lifted to protocol functions not calling *send_i* (e.g. *mk_msg1*) by repeated application of the Hoare proof rule for sequential composition, "pulling" the invariant over the individual function calls.

5.4 Discussion and Evaluation

Reasoning in the BPW model is inherently stateful and, as originally proposed involves complex pointer-based data structures. As observed in the introduction, our main task in formalizing this model was to develop abstractions, proof strategies, and supporting proof tools to allow us to reduce this complexity and reason efficiently about the state and the state-transitions that result from calls to the interface functions. One of our strategies was to automate as much reasoning as possible using the WPC. The main enabling factors for this strategy were the model abstraction provided by moving from the indexed to the term-based model and the property abstraction introduced by using the operators *analyze* and *parts* along with their equational theories. We now routinely use the WPC up to the level of BPW-model interface functions and switch to Hoare logic only at the protocol and system levels.

More precisely, we have adopted the following proof strategy for systematically proving invariants, which we illustrate using the NSL protocol and a hypothetical invariant *I* as an example. We explain our three-step strategy in a top-down manner, although in practical work we often proceed bottom-up. First, we apply a LTL proof rule to reduce the temporal statement that *I* is an NSL invariant (expressed as *NSLtrsys* $\models \Box(Pred\ I)$) to a set of Hoare triples of the form:

$$\{I \cap J\}\ h\ x\ \{> \lambda z.\ I\}$$

There is one such triple for each system interface function *h*, stating that *h* preserves *I* on all inputs *x*. The LTL proof rule achieving this reduction embodies an induction over positions in system runs and uses auxiliary invariants (here represented by *J*) in order to strengthen the induction hypothesis. Second, we use the rules of Hoare logic to decompose these preservation statements into similar statements about the BPW-model interface functions. However, as illustrated in Sect. 5.3, the preservation of the invariant *I* by BPW-model interface functions *f* may require auxiliary preconditions (*pre_f x*):

$$\{(pre\_f\ x) \cap I \cap J\}\ f\ x\ \{> \lambda z.\ I\}$$

We must ensure that we can derive any such auxiliary precondition (*pre_f x*) of a BPW-model interface function *f* called in a protocol handler *h* from the postconditions of functions called in *h* before *f*[4]. In order to minimize the use of ad hoc lemmas, we prove characteristic Hoare triples for the auxiliary functions appearing in the protocol handlers (such as *parse_msg* and *mk_msg1* in the NSL protocol). These Hoare triples have only auxiliary invariants in the precondition and a strong postcondition characterizing the effect of the respective function. The idea is to collect all the information we need to prove (*pre_f x*) from such characteristic postconditions. The difficulty of this step depends on the number of BPW interface functions requiring auxiliary preconditions, which are generally few (often only *send_i*). In the third step, we prove the preservation lemmas for the BPW-model interface functions by unfolding them into the WPC and then applying the automatic proof tools including the simplifier and the tableau reasoner. The former makes heavy use of the equational theories of *analyze* and *parts*. The automatic tools may require additional lemmas about consequences of auxiliary invariants to complete the proof.

After abstracting most of the non-inherent complexity of the BPW model, we obtained a framework in which cryptographically sound protocol verification in the sense of

---

[4] Note that since the adversary's interface functions are also system-level interface functions, they are not allowed to have such auxiliary preconditions, if *I* is to be system invariant.

BRSIM/UC is possible. However, due to the pointer-like nature of handles we are constrained to the fine-grained BPW interface functions to handle messages in a constructor-wise manner. This is the main remaining intrinsic complexity in our model. In contrast, the complexity added by the non-standard aspects of the cryptographic operations (length-revealing ciphertexts, signature transformations) do not complicate proofs significantly.

Paulson's security protocol proofs in Isabelle/HOL provide a natural benchmark for our own proofs and for judging the cost of this remaining complexity. Ideally, the cost would be zero. That is, we could construct proofs using cryptographically sound abstractions with an effort comparable to that required when using the considerably simpler abstractions provided by the Dolev-Yao model. For the moment, we are still some distance from this ideal as our proofs are roughly two orders of magnitudes larger than Paulson's. Whereas he uses a few lines to prove an invariant, we need an entire Isabelle theory of several hundred lines. A similar picture arises at the global level: his NSL proof needs roughly 3 pages (counting the automatically generated Isabelle documentation), while ours occupies 140 pages. However, the length of a proof is a poor measure of its complexity. A substantial part of this difference can be attributed to the fact that we have to show preservation of every invariant by all 17 BPW-model interface functions before we can start reasoning at the protocol level. However, as explained above, most lemmas can be derived systematically and largely automatically using the WPC. A typical proof script for a preservation lemma in an invariant proof requires 2–6 lines of tactics and the variation between them is small. We think that the complexity of the property specifications and the proofs (e.g., the invention of invariants) is comparable to Paulson's and we are optimistic about being able to further reduce this gap in the future.

6   Conclusion

We have developed an abstraction of the BPW model, along with strategies and proof tools, that enables practical protocol security proofs with strong soundness guarantees. In doing so, we have substantially reduced the non-inherent complexity of the BPW model in a way that brings us closer to the considerably simpler abstractions provided by the standard Dolev-Yao model and inductive proof techniques used, e.g., by Paulson.

We see a number of directions for future work. First, we would like to develop methods to reduce the impact of the inherent complexity. One possibility is to investigate changes to the model, either by building a higher-level interface for protocols or even changing the model itself (which would, however, necessitate a new soundness proof). For example, it would simplify proofs to reduce the number of interface functions from 17 to 3, namely functions for building, parsing, and sending messages. This would enable more compact protocol specifications as well as shorter proofs based on general results about these functions. Second, we have built basic proof tools and have developed systematic strategies for constructing proofs using the general automated reasoning tools provided by Isabelle, mainly rewriting and tableau theorem proving. However we have not yet developed any specialized proof tactics tailored to our strategies. As mentioned in Sect. 5.4, we see considerable potential for improvement here. Finally, we intend to carry out further case studies in order to broaden our experience with our formalization and proof strategies. It would be useful here to incorporate more features into our formalization such as symmetric encryption and MACs. The technical details have been worked out [11, 8] and await implementation.

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. P. B. Andrews. *An Introduction to Mathematical Logic and Type Theory: To Truth Through Proof*. Academic Press, 1986.

5. M. Backes. A cryptographically sound Dolev-Yao style security proof of the Otway-Rees protocol. In *Proc. 9th European Symposium on Research in Computer Security (ESORICS)*, volume 3193 of *Lecture Notes in Computer Science*, pages 89–108. Springer, 2004.

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

7. M. Backes and B. Pfitzmann. A cryptographically sound security proof of the Needham-Schroeder-Lowe public-key protocol. *Journal on Selected Areas in Communications*, 22(10):2075–2086, 2004.

8. M. Backes and B. Pfitzmann. Symmetric encryption in a simulatable Dolev-Yao style cryptographic library. In *Proc. 17th IEEE Computer Security Foundations Workshop (CSFW)*, pages 204–218, 2004. Full version in IACR Cryptology ePrint Archive 2004/059, Feb. 2004, `http://eprint.iacr.org/`.

9. M. Backes and B. Pfitzmann. Relating symbolic and cryptographic secrecy. *Transactions on Dependable and Secure Computing*, 2(2):109–123, 2005.

10. 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/`.

11. M. Backes, B. Pfitzmann, and M. Waidner. Symmetric authentication within a simulatable cryptographic library. In *Proc. 8th European Symposium on Research in Computer Security (ESORICS)*, volume 2808 of *Lecture Notes in Computer Science*, pages 271–290. Springer, 2003. Extended version in IACR Cryptology ePrint Archive 2003/145, Jul. 2003, `http://eprint.iacr.org/`.

12. M. Backes, B. Pfitzmann, and M. Waidner. A general composition theorem for secure reactive systems. In *Proc. 1st Theory of Cryptography Conference (TCC)*, volume 2951 of *Lecture Notes in Computer Science*, pages 336–354. Springer, 2004.

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

14. M. Baudet, V. Cortier, and S. Kremer. Computationally sound implementations of equational theories against passive adversaries. In *Proc. 32nd International Colloquium on Automata, Languages and Programming (ICALP)*, volume 3580 of *Lecture Notes in Computer Science*, pages 652–663. Springer, 2005.

15. D. Beaver. Secure multiparty protocols and zero knowledge proof systems tolerating a faulty minority. *Journal of Cryptology*, 4(2):75–122, 1991.

16. B. Blanchet. Automatic proof of strong secrecy for security protocols. In *Proc. 25th IEEE Symposium on Security & Privacy*, pages 86–100, 2004.

17. B. Blanchet. A computationally sound mechanized prover for security protocols. In *Proc. 27th IEEE Symposium on Security & Privacy*, 2006.

18. R. Canetti. Security and composition of multiparty cryptographic protocols. *Journal of Cryptology*, 3(1):143–202, 2000.

19. R. Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *Proc. 42nd IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 136–145, 2001. Extended version in Cryptology ePrint Archive, Report 2000/67, http://eprint.iacr.org/.

20. 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/.

21. R. Canetti and T. Rabin. Universal composition with joint state. In *Advances in Cryptology: CRYPTO 2003*, volume 2729 of *Lecture Notes in Computer Science*, pages 265–281. Springer, 2003.

22. A. Church. A formulation of the simple theory of types. *Journal of Symbolic Logic*, pages 56–68, 1940.

23. V. Cortier and B. Warinschi. Computationally sound, automated proofs for security protocols. In *Proc. 14th European Symposium on Programming (ESOP)*, pages 157–171, 2005.

24. A. Datta, A. Derek, J. Mitchell, V. Shmatikov, and M. Turuani. Probabilistic polynomial-time semantics for a protocol security logic. In *Proc. 32nd International Colloquium on Automata, Languages and Programming (ICALP)*, volume 3580 of *Lecture Notes in Computer Science*, pages 16–29. Springer, 2005.

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

26. 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.

27. 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.

28. S. Goldwasser and L. Levin. Fair computation of general functions in presence of immoral majority. In *Advances in Cryptology: CRYPTO '90*, volume 537 of *Lecture Notes in Computer Science*, pages 77–93. Springer, 1990.

29. M. J. C. Gordon and T. F. Melham, editors. *Introduction to HOL: A Theorem Proving Environment for Higher Order Logic*. Cambridge University Press, 1993.

30. J. D. Guttman, F. J. Thayer Fabrega, and L. Zuck. The faithfulness of abstract protocol analysis: Message authentication. In *Proc. 8th ACM Conference on Computer and Communications Security*, pages 186–195, 2001.

31. 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.

32. C. A. R. Hoare. An axiomatic basis for computer programming. *Communications of the ACM*, 12(10):576–585, October 1969.

33. 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.

34. B. Jacobs and J. Rutten. A tutorial on (co)algebras and (co)induction. *EATCS Bulletin*, 6:222–259, 1997.

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

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

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

38. P. Laud. Secrecy types for a simulatable cryptographic library. In *Proc. 12th ACM Conference on Computer and Communications Security*, 2005.

39. S. Liang, P. Hudak, and M. Jones. Monad transformers and modular interpreters. In *22nd ACM Symposium on Principles of Programming Languages (POPL '95)*, pages 333–343, New York, NY, USA, jan 1995. ACM Press.

40. 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.

41. 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.

42. Z. Manna and A. Pnueli. Completing the temporal picture. *Theoretical Computer Science*, 83(1):97–139, 1991.

43. Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems*. Springer Verlag, 1992.

44. Z. Manna and A. Pnueli. *Temporal Verification of Reactive Systems – Safety*. Springer Verlag, 1995.

45. M. Merritt. *Cryptographic Protocols*. PhD thesis, Georgia Institute of Technology, 1983.

46. S. Micali and P. Rogaway. Secure computation. In *Advances in Cryptology: CRYPTO '91*, volume 576 of *Lecture Notes in Computer Science*, pages 392–404. Springer, 1991.

47. 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.

48. 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.

49. 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.

50. E. Moggi. Notions of computation and monads. *Information and Computation*, 1991.

51. T. Nipkow, L. C. Paulson, and M. Wenzel. *Isabelle/HOL – A Proof Assistant for Higher-Order Logic*, volume 2283 of *Lecture Notes in Computer Science*. Springer-Verlag, 2002.

52. L. Paulson. The inductive approach to verifying cryptographic protocols. *Journal of Cryptology*, 6(1):85–128, 1998.

53. B. Pfitzmann and M. Waidner. Composition and integrity preservation of secure reactive systems. In *Proc. 7th ACM Conference on Computer and Communications Security*, pages 245–254, 2000. Extended version (with Matthias Schunter) IBM Research Report RZ 3206, May 2000, http://www.semper.org/sirene/publ/PfSW1_00ReactSimulIBM.ps.gz.

54. 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/.

55. A. M. Pitts. Evaluation logic. In G. Birtwistle, editor, *IVth Higher Order Workshop, Banff 1990*, Workshops in Computing, pages 162–189. Springer-Verlag, Berlin, 1991.

56. S. Schneider. Security properties and CSP. In *Proc. 17th IEEE Symposium on Security & Privacy*, pages 174–187, 1996.

57. B. Warinschi. A computational analysis of the Needham-Schroeder-(Lowe) protocol. In *Proc. 16th IEEE Computer Security Foundations Workshop (CSFW)*, pages 248–262, 2003.