Available online at www.sciencedirect.com

Electronic Notes in Theoretical Computer Science 201 (2008) 99–126

Electronic Notes in Theoretical Computer Science

www.elsevier.com/locate/entcs

Constructing and Reasoning About Security Protocols Using Invariants

Arjan J. Mooij[1]

School of Computer Science and Information Technology, The University of Nottingham, Jubilee Campus, Wollaton Road, Nottingham NG8 1BB, United Kingdom.

Abstract

In this work we explore the applicability of the programming method of Feijen and van Gasteren to the construction of security protocols. This method addresses the derivation of concurrent programs from a formal specification, and it is based on common notions like invariants and pre- and post-conditions. We show that fundamental security concepts like secrecy and authentication can nicely be specified in this way. Using some small extensions, the style of formal reasoning from this method can be applied to the security domain. To demonstrate our approach, we discuss an authentication protocol and a public-key distribution protocol, and we deal with their composition. Although this work does not contain any new protocols, it does offer a new view on describing, constructing and reasoning about security protocols.

Keywords: mathematical techniques, program construction, refinement, security

1 Introduction

The formal correctness of security protocols is generally considered to be important. The most common approach to establishing it is by means of verification after the protocols have been designed. The usual styles include automated state-space exploration [21] and interactive theorem proving [28]. However, actually designing and reasoning about security protocols is still considered to be complicated.

In this work we consider an approach in which, based on a formal specification, the protocols are constructed hand-in-hand with their formal correctness

[1] Email: Arjan.Mooij@cs.nott.ac.uk

doi:10.1016/j.entcs.2008.02.017

proof. Such formal derivation approaches are insightful, but so far they have received less attention [7,19]. Our goal is not the automatic synthesis of security protocols [29], but an effective and systematic way of reasoning about them.

The first step in a derivation is a proper specification. Such a specification must be formal and manageable, i.e., it must be precise yet comprehensible, and it must be easy to manipulate. In the case of security protocols, often dedicated belief logics and calculi are used, e.g., in [5,3]. Although this gives the desired formality, there are dangers of misinterpreting the new concepts, and often a large number of new calculational rules is introduced. In contrast, we propose the use of very familiar specification concepts, like pre- and post-conditions, and invariants. An overview of other uses of invariants for the analysis of security protocols can be found in [23]. In particular we show how notions like secrecy and authentication [22] can conveniently be modelled in this way.

For the reasoning about security protocols, also various dedicated approaches have been proposed, e.g., in [30]. In contrast, we study the use of a general method, which gives more flexibility, and enables the integration with techniques developed for other application domains. A similar approach has been explored in the context of data refinement [6], but its complexity seems to remain a problem [19].

Simplicity of exposition is at the core of the method of Feijen and van Gasteren [17] for constructing concurrent programs. It is based on the classical axiomatic theory of Owicki and Gries [27], where the reasoning about execution traces (like [28,30]) is replaced by the reasoning about assertions and invariants, i.e., predicate abstractions of states. This method has shown its merits in various application areas, e.g., [18,17,20,24]. In this work we explore its applicability to the formal derivation of security protocols.

The typical derivations in this style [12,17] start with a series of desired post-assertions. To establish their correctness, some program statements are inserted together with again some required pre-assertions. This is repeated until the required pre-assertions are suitable as the precondition of the program. Although this breaks with the more tempting forward style of reasoning, a similar backward style has roughly been sketched for security protocols in [3].

We show that the interference caused by the intruders has a limited influence on this method, as message communication in the presence of intruders is just a variation on communication using faulty channels. Hence we can largely reuse the corresponding reasoning techniques, which are based on common notions like invariants. These techniques are in particular very explicit in recording what needs to be derived from the message upon receipt, the importance of which has repeatedly been pointed out, e.g., in [2].

To demonstrate our approach we reconstruct the authentication protocol of Needham and Schroeder [26]. The well-known attack that was once detected by Lowe [21] is avoided in our derivations in a natural way. Apart from this classical authentication protocol, we also address a public-key distribution protocol, and we discuss their composition.

Overview

In Section 2 we summarise the theories and methods that we use. Our derivation approach is described in Section 3, followed by our specification of authentication in Section 4. We demonstrate them using an authentication protocol in Section 5, and using a public-key distribution protocol in Section 6. We use these results in Section 7 to explore the composition of security protocols. Finally we draw some conclusions and sketch some further work in Section 8.

2 Preliminaries

In this section we summarise some basic material that we use in the remainder of this work.

2.1 Concurrent programs

A concurrent program consists of a precondition *Pre* and a number of sequential programs, which are called its components. The components are to be executed in parallel by interleaving their atomic statements.

The programming language that we use for the components is based on the Guarded Command Language from [12]. The semantics of the atomic statements follows from the weakest liberal precondition, *wlp*, (and the weakest precondition, *wp*) predicate transformers. The weakest liberal precondition of a statement S and a predicate Q, to be denoted by $wlp.S.Q$, is the weakest predicate P that guarantees that each terminating execution of statement S starting from a state satisfying P establishes Q; in contrast to the *wp*, the *wlp* does not guarantee termination of the statement. In particular we use the following two kinds of atomic statements:

- assignment $v := E$, which assigns the value of expression E to variable v. Its *wlp* can be defined as

$$[\ wlp.(v := E).Q \quad \equiv \quad (v := E).Q\ ]$$

  where $(v := E).Q$ at the right-hand side denotes the substitution of expression E for variable v in predicate Q. The pair of square brackets $[\ldots]$ is a shorthand for "for all states", i.e., a universal quantifier binding all free program variables.

- non-deterministic assignment $v :\ P$, which assigns to variable v any value x such that the condition $(v := x).P$ holds. As long as there is no suitable value x, the execution of this statement is blocked. Its *wlp* can be defined as

$$[\ wlp.(v :\ P).Q \quad \equiv \quad (\forall x ::\ (v := x).P \ \Rightarrow\ (v := x).Q)\ ]$$

2.2 Security protocols

In the case of security protocols, the set of components consists of so-called honest components and intruder components. The honest components can be controlled (or programmed), while the intruders are outside our control.

The components can communicate using messages. The kinds of messages can be defined recursively. An *atomic* (plaintext) message is just a data value, where we assume that values from different data types are different. Another view on it is assuming that the values are explicitly typed. In particular the data values include component identifiers, nonces (introduced later on) and keys.

An *encrypted* (ciphertext) message $[k :\ y, z]$ for any key k consists of a series of messages, in this case the two messages y and z. We do not use the more-common notation $\{y, z\}_k$ to avoid confusion with the frequent use of $\{\ldots\}$ for assertions and to avoid the extra subscript for k. It is straightforward to generalise this kind of encrypted messages to containment of other numbers of messages, and to trivial encryption (i.e., just a tuple of messages). The usual models of encrypted messages distinguish explicitly between concatenating the messages y and z, and encrypting the result. We leave the intermediate concatenation step implicit, in order to emphasise the more-important encryption step, to simplify the descriptions of the intruders, and hence, to make the protocol derivations more effective.

All messages created in these two ways are assumed to be different. In particular we have the following freeness assumption [13], for any keys k and l, and messages v, w, y and z:

$$[k :\ v, w] = [l :\ y, z] \quad \equiv \quad k = l \ \wedge\ v = y \ \wedge\ w = z$$

Such a black-box encryption model provides a convenient layer of abstraction between cryptography and concurrency, but it ignores so-called type flaw attacks. Although this is a common abstraction, its main drawback is that it never holds in practice: the model contains *infinitely* many different messages, while practical messages are just *finite* bit-strings.

The intruders are assumed to behave according to the Dolev/Yao abstraction [13], i.e., the intruders can intercept, read and create messages. Based on the observed messages, the intruders can also derive new messages. If any key k and any two messages y and z can be derived, then the messages can be composed to derive the message $[k:\ y,z]$. If any encrypted message $[k:\ y,z]$ and the key $\overline{k}$ can be derived, then the encrypted message can be decomposed to derive the messages y and z. The key $\overline{k}$ denotes the key that undoes the encryption with key k; in asymmetric encryption models the keys k and $\overline{k}$ cannot be derived from each other.

2.3 Owicki/Gries theory

Partial correctness (or safety) is specified by annotating the program with assertions. An assertion is a predicate on the state of the system and it is located between brackets $\{\ldots\}$ at a control point. The control points are the locations between the atomic statements in the components.

To prove partial correctness, we use the classical axiomatic theory from Owicki and Gries [27] using the nomenclature from [17]. An annotated program is correct if each assertion is correct. In turn, an assertion P in a component is correct if it is both

- locally correct, i.e., it is established in the component:
  - if P is an initial assertion in the component: $[\ \mathit{Pre} \Rightarrow P\ ]$ holds;
  - if P is preceded by an atomic statement $\{Q\}\ S$, where Q is a pre-assertion of statement S, then $[\ Q\ \Rightarrow\ \mathit{wlp}.S.P\ ]$ holds.
- globally correct, i.e., it is maintained by all other components:
  - for each atomic statement $\{Q\}\ S$ in any other component, where Q is a pre-assertion of statement S, $[\ P \wedge Q\ \Rightarrow\ \mathit{wlp}.S.P\ ]$ holds.

An invariant is an assertion that is located at every control point of the program. So an invariant is correct if it is both implied by the precondition, and maintained by each atomic statement in any component.

2.4 Method of Feijen/van Gasteren

The programming method from [17] deals with the formal construction of concurrent programs hand-in-hand with a suitable annotation and a correctness proof. The method being based on the style of [12], assertions play an important role. Multiple assertions can be placed at a single control point, which denotes their conjunction, and their correctness can be proved separately. A queried assertion is an assertion that is required to hold, but whose correctness has not yet been proved; it is marked with a query '**?**'.

Program construction starts with a specification in terms of a preliminary program and some queried assertions, such as post-conditions. Derivations consist of turning each queried assertion, one-by-one, into a correct assertion. The proof obligations for local correctness lead to a style in which programs are constructed from the required assertions towards the initial control point. When all assertions (which include those from the specification) are correct, the program is correct with respect to the specification.

If a queried assertion's correctness (in the current annotated program) cannot yet be proved, there are mainly two solutions (which can also be combined):

- introduce additional queried assertions in the current annotation;
- modify the program.

An important issue is whether these options endanger the correctness of the other assertions. Introducing additional assertions cannot endanger the correctness of the other assertions, and typically the weakest possible strengthening that serves the goal is calculated. However, modifying the program may transform all assertions into queried assertions again. The typically-used modification of the program is inserting a new statement (to establish local correctness).

Like its underlying theory [27], this method does not formally address progress. Since the role of progress in the derivations is often limited, progress is usually discussed in a pragmatic ad-hoc manner. Although in [15,16] we have shown how to integrate a full progress logic [14], in the current work we follow the original approach since progress plays no significant role.

2.5 *Proof obligations for faulty channels*

Our treatment of intruders will be based on a technique for faulty channels. Faulty channels are channels that can duplicate, reorder and lose messages, but in contrast to intruders (see Section 3.2) they cannot insert new messages or modify messages. Such channels have been studied in relation to, for example, alternating bit protocols [18,17] and sliding window protocols [20]. In this section we discuss an operational model, and an elegant rule for proving that an assertion is established by the receipt of a message.

2.5.1 Operational model

In terms of shared variables, an operational model of message communication over a single faulty channel may look as follows:

snd x	:	$C := C \cup \{x\}$
Faulty channel	:	**do** *true* $\rightarrow$
		$\langle \; y : \; y \in C$
		$; R := R \cup \{y\} \; \rangle$
		od
rcv $z \leftarrow m.z$	:	$\langle \; z : \; m.z \in R$
		$; R := R \backslash \{m.z\} \; \rangle$

The pairs of angular brackets $\langle \ldots \rangle$ denote larger atomic statements. The variable C denotes the set of transmitted messages, while the variable R denotes the receipt buffer of any of the components. The send statement **snd** x adds the message x to the set C, while the faulty-channel component repeatedly copies any message y from C into R. Function m maps any element of the type of variable z to a message, and it can be used to model the receipt of a data parameter z. The receive statement **rcv** $z \leftarrow m.z$ non-deterministically assigns to variable z some value such that the message $m.z$ can be removed from the set R; the receive statement is blocked as long as there is no suitable message in the set R.

The implementability of the send and receive statements, e.g., the accessibility of the required keys, must be checked separately, and we ignore this in our model. We do not mention the intended receiver in the send statement, even in the case of multiple possible receivers, so in fact we are modelling a faulty broadcast channel. Variables C and R can only occur in these operations, and hence the invariant $R \subseteq C$ is maintained, and we typically require this invariant as an implicit precondition.

2.5.2 Rule of import and export

To avoid explicitly reasoning in terms of this operational model, an elegant proof rule can be derived. We discuss this rule in detail in order to facilitate its reuse in Section 3.2. Moreover, it serves as a quick introduction to the style of program development, although we will refrain here from introducing any new statements.

Consider any series of message $m.x$ depending on parameter x, and any predicate P that does not contain C or R. Given a send and a receive statement in two different components, we want to derive a rule for establishing an assertion P after the receive statement. This specification can be summarised as follows:

rcv $v \leftarrow m.v$	
$\{\textbf{?}\ P\}$	**snd** $m.w$

Each of the two boxes in this figure denotes a single component, and v and w are variables. The assertions are printed in between brackets $\{\ldots\}$, and the remaining proof obligation is marked with a query '**?**'. Furthermore, we have deliberately omitted mentioning of the faulty-channel component.

For the global correctness of assertion P, we assume that the statements in the other components cannot violate it. For its local correctness we must consider the receipt of a message $m.x$, for any x, which corresponds to an assignment to variable v and removing the message $m.v$ from the receipt buffer R of the component. As a result, we require the invariant

$$(\forall x ::\ m.x \in R \ \Rightarrow\ (v := x).P)$$

Thus we obtain the following intermediate program:

rcv $v \leftarrow m.v$	
$\{P\}$	**snd** $m.w$

Inv: $\textbf{?}\ (\forall x ::\ m.x \in R \ \Rightarrow\ (v := x).P)$

For the maintenance of this queried invariant under the honest components, we assume that their statements cannot violate $(v := x).P$, for any x. For the maintenance under the faulty channel component, we must consider the copying of any transmitted message from C to the receipt buffer R. As a result, we require the invariant

$$(\forall x ::\ m.x \in C \ \Rightarrow\ (v := x).P)$$

Using $R \subseteq C$, this invariant is stronger than the previous one, and hence we only need the consider this one.

For the maintenance of this new invariant under the transmission of a message $m.w$, we must consider the addition of the message to the set C. As a result, we require the transmission to have a pre-assertion $(v := w).P$. For initial correctness of the invariant, we typically require the precondition $(\forall x ::\ m.x \notin C)$. Thus we obtain the following rule:

Pre: $(\forall x ::\ m.x \notin C)$

rcv $v \leftarrow m.v$	$\{\textbf{?}\ (v := w).P\}$
$\{P\}$	**snd** $m.w$

Inv: $(\forall x ::\ m.x \in C \ \Rightarrow\ (v := x).P)$

So, apart from any possible violation of the predicate $(v := x).P$, for any x, by the other statements in the system, the assertion P is established by this receipt of a message $m.v$ if before every transmission of any message $m.w$ the assertion $(v := w).P$ holds. This rule is called the rule of import (upon receipt) and export (upon transmission), and a similar rule exists for binary semaphores.

3 Techniques

In this section we develop our techniques for the formal derivation of security protocols. In particular we focus on the consequences of the presence of intruders.

3.1 Formalising the derivable messages

We start by formalising the set of messages D that can be derived by the intruders from (and including) the set of transmitted messages C, as described in Section 2.2. We assume that the initial knowledge of the intruders is part of the set C, and we will abbreviate conditions like $y \in D$ into the more compact $D.y$. For later use, we immediately generalise this set D with a parameter K denoting a set of keys.

Define the set D_K, for any set K, as the smallest (but usually unbounded) set of messages such that the following three conditions hold, for any key k, and messages y and z:

- containment:
$$C \subseteq D_K$$
- composition:
$$D_K.k \;\wedge\; D_K.y \;\wedge\; D_K.z \quad\Rightarrow\quad D_K.[k : y, z]$$
- decomposition:
$$(\overline{k} \in K \;\vee\; D_K.\overline{k}) \;\wedge\; D_K.[k : y, z] \quad\Rightarrow\quad D_K.y \;\wedge\; D_K.z$$

The original set D corresponds to the instance $K = \emptyset$, and we will use the constant ω to denote the set of all keys. The set K enables decomposition to also decompose messages using a key from K that could not be derived from C. Furthermore D_K has the following monotonicity property, for any sets S and T:
$$S \;\subseteq\; T \quad\Rightarrow\quad D_S \;\subseteq\; D_T$$

3.2 Communication via intruders

The capabilities of the intruders are such that they can completely control the message flow in the system. Hence we would like to model the intruders as a special type of communication channel. Such a channel is similar to a faulty channel, as described in Section 2.5, but with the extra capability of inserting new messages and modifying messages, viz., via composition and decomposition.

To address these derived messages, in the operational model of a faulty-channel component, we only need to replace assignment $y:\ y \in C$ by assignment $y:\ y \in D$. Thus we obtain the following intruder component:

```
Intruder  :   do true →
                  ⟨  $y:\ y \in D$
                   ; $R := R \cup \{y\}$  ⟩
              od
```

The corresponding rule for establishing an assertion P using the receipt of a message $m.v$ must also be adapted. As before, the required invariant first becomes $(\forall x ::\ m.x \in R\ \Rightarrow\ (v := x).P)$, where R denotes the receipt buffer of the receiving component. For its maintenance under adding *derived* messages to the receipt buffer R by the intruder, it will be replaced by the stronger required invariant

$$(\forall x ::\ D.(m.x)\ \Rightarrow\ (v := x).P)$$

All invariants about the message communication are of this particular shape, and hence we will not explicitly mention the receipt buffers R anymore. As for faulty channels, its proof requires a pre-assertion of the send statement; but there are more proof obligations, as we will discuss in the next section.

rcv $v \leftarrow m.v$	$\{$**?** $(v := w).P\}$
$\{P\}$	**snd** $m.w$

Inv: **?** $(\forall x ::\ D.(m.x)\ \Rightarrow\ (v := x).P)$

This technique is related to several guidelines from [2]. In the first place, the invariant is very explicit about the meaning of the message $m.x$, viz., condition $(v := x).P$. The invariant also indicates how long the message $m.x$ needs to remain secret for the intruders, namely, until $(v := x).P$ holds. The special case that $[\ (v := x).P \equiv \mathit{false}\ ]$, for any x, denotes secrecy of the message $m.x$. It is very reassuring that these formal techniques correspond to important guidelines from the field.

3.3 Proving the correctness of the typical invariants

The general shape of the invariants is the conjunction of some terms like $D_K.x$ implies a predicate that does not depend on D. For simplicity reasons we discuss the correctness proof of such invariants using the following typical instance, for any set K, message x, and predicate P:

$$D_K.x\ \Rightarrow\ P$$

For its initialisation, a precondition $D_K.x \;\Rightarrow\; P$ would be sufficient, but often the stronger precondition $D_\omega.x \;\Rightarrow\; P$ is required. Regarding maintenance we will ignore in this section the statements that can violate P. What remains to be considered for maintenance is each send statement, i.e., each expansion of the set C. As a consequence the set D_K is atomically expanded with the new message, and by subsequent decomposition and composition.

To prove maintenance, we can directly apply the normal proof rules [27], but this may become quite complicated given the definition of D. In what follows we develop a simpler and sufficient proof approach, as far as the set D_K in the antecedent is concerned. We will treat such a compound expansion of D_K by showing that each of its expansions with a single message maintains the invariant.

Maintenance under any composition step only needs to be considered for a non-atomic message x, so assume that $x = [k :\; y, z]$ for any key k, and messages y and z. Independent from any specific send statement, composition can be addressed by just requiring the additional invariant

$$D_K.k \;\wedge\; D_K.y \;\wedge\; D_K.z \quad\Rightarrow\quad P$$

Usually we simplify and weaken the antecedent into only one of the conjuncts. In turn, term $D_K.y$ or $D_K.z$ in this new condition may need to be considered regarding composition, hence leading to a finite number of extra invariants. This number is usually very small, since the typical messages do not contain many nested layers of encryption.

In the case of decomposition, a similar approach would introduce an infinite number of explicit invariants, which is infeasible. If the message types are somehow limited, for example by restricting the amount of nested encryption layers, then the number of extra invariants may become finite, but it is still likely to be unmanageable (and certainly not scalable).

Notice that it is not useful to decompose any message that was created via composition; see also [9]. Hence we only need to consider the recursive decomposition of the transmitted messages. Upon transmitting a new message, at least the new message must be decomposed recursively. However, in contrast to composition, decomposition may also derive new keys, in which case the earlier transmitted messages may become further decomposable. In what follows we discuss these two series of transmitted messages separately.

We first consider the maintenance under any (recursive) decomposition step of the newly transmitted message. For each occurrence of x in the transmitted message $[k :\; x, y]$, for any key k and message y, this results in the following required pre-assertion of this decomposition step:

$$\overline{k} \in K \;\vee\; D_K.\overline{k} \quad\Rightarrow\quad P$$

In particular this leads to a pre-assertion P if the message x itself is transmitted.

For the earlier transmitted messages, we consider the case that a (recursive) decomposition step of the newly transmitted message reveals a key $k : \; k \notin K$. In this case we use our generalisation of D and require the following pre-assertion:

$$D_{(K \,\cup\, \{k\})}.x \quad \Rightarrow \quad P$$

Effectively this means that the key k should already be considered for the recursive decomposition resulting from the earlier message transmissions. In contrast to the conditions that prohibit certain messages to be derived, the parameter K in the generalised construct D_K indicates which keys may be derived safely.

To convert the required pre-assertions of the individual decomposition steps into the required pre-assertion of the message transmission, we must strengthen them until they are maintained under the other steps resulting from this message transmission. In practice there are two simple ways to achieve this goal: either require such an individual pre-assertion as an invariant (whose proof obligations include these), or strengthen it such that it contains no occurrences of D anymore (which makes maintenance trivial).

3.4 Progress in the presence of intruders

Progress and termination of security protocols can mainly be hindered by the receive statements, which can be blocked. As the intruders may intercept and lose all sent messages, denial-of-service attacks cannot be prevented in our model, and hence in general progress cannot be guaranteed. In this work we will only consider progress in case the intruders behave like a proper communication channel. In particular we will rely on the following variation on the ground rule of progress from [17]:

> "For each receive statement of a message m in a component, it should hold that the rest of the system has the potential of ultimately sending the message m."

4 Specification of authentication

Suppose there are two disjoint sets of honest components A and B. The goal is to specify authentication between any pair of a component from A and a component from B. To this end, we introduce the constants m_a and m_b for each component $a : \; a \in A$ and $b : \; b \in B$ respectively to denote the (either honest or intruder) component with which each honest component decides to communicate. Constant m_a is only accessible by component a, and similarly, constant m_b is only accessible by component b.

After any component a's execution of the protocol, it must be guaranteed that if honest component a decided to communicate with honest component b, then b decided to communicate with a; and dually after any component b's execution. In other words, the intruders cannot fool any single honest component, for example by abusing another honest component.

$$\boxed{\begin{array}{ll} \text{Comp.}a,\ a \in A: & \ldots \\ & \{\mathbf{?}\ (\forall b : b \in B :\ m_a = b\ \Rightarrow\ m_b = a)\} \end{array}}$$

$$\boxed{\begin{array}{ll} \text{Comp.}b,\ b \in B: & \ldots \\ & \{\mathbf{?}\ (\forall a : a \in A :\ m_b = a\ \Rightarrow\ m_a = b)\} \end{array}}$$

This specification is less operational than the ones that are collected in [23], and it corresponds to an important notion of authentication given by Lowe in [22], viz., agreement. In terms of Lowe's CSP-based model, for each honest component c the initial choice of the m_c value corresponds to his extra "running" event, while reaching the end of the protocol for any honest component c corresponds to his extra "commit" event; the requirement that each "commit" event must be preceded by an appropriate "running" event is expressed directly by our required post-assertions.

Termination of the protocol needs to be guaranteed for each pair of components (a, b) from $A \times B$ for which $m_a = b\ \wedge\ m_b = a$ holds, provided that the intruders behave like a proper communication channel between a and b.

5 Derivation of an authentication protocol

To demonstrate our formal reasoning approach, we present a derivation of the authentication protocol of Needham, Schroeder and Lowe [26,21]. Our derivation is based on the specification of authentication from Section 4, but, for simplicity reasons, restricted to singleton sets of honest components, viz., $A = \{a\}$ and $B = \{b\}$.

The protocol can use encryption using public and private keys. The key $k.x$ denotes the public key of component x, and the key $\overline{k.x}$ denotes the private key of component x. Each component has access to its own private key, and to the public key of each component. The validity of this scheme demands that the private keys of the honest components remain secret, in particular to the intruders, which requires an invariant

$$(\forall x ::\ D.\overline{k.x}\ \ \Rightarrow\ \ x \neq a\ \wedge\ x \neq b)$$

This invariant might be easier to interpret in contrapositive form, but we prefer this shape for homogeneity. Thus we obtain the following specification:

Pre: $\neg D_\omega.\overline{k.a} \ \wedge\ \neg D_\omega.\overline{k.b}$

$\ldots$	$\ldots$
$\{?\ m_a = b \ \Rightarrow\ m_b = a\}$	$\{?\ m_b = a \ \Rightarrow\ m_a = b\}$

Inv: $(\forall x :: \ D.\overline{k.x} \ \ \Rightarrow\ \ x \neq a \ \wedge\ x \neq b)$

As we are heading for an asymmetric protocol, we start with only one of the two post-assertions. Regarding this choice, the usual role names, viz., initiator for component a and responder for component b, are not helpful. That is, they refer to the direction of the *first* communication, while our construction starts at the end of the protocol. However, since the specification is symmetric, the decision does not really matter, and we just choose to start with the post-assertion in component b.

It turns out that all current and future assertions in this derivation only refer to constants and local variables, and hence we do not need to consider their global correctness.

5.1 Last message communication

The queried assertion $m_b = a \ \Rightarrow\ m_a = b$ in component b refers to constants from both components, viz., m_a and m_b. The way to establish such an assertion is to import it via the receipt of a message, say a constant x, using an invariant

$$D.x \quad \Rightarrow \quad (m_b = a \ \Rightarrow\ m_a = b)$$

To be able to initialise this invariant, we must ensure that message x cannot be derived initially by the intruders. So, in particular, message x should not just be some predictable fixed value.

An obvious proposal would be to authenticate the message using the private key from the sending component. However, this requires initially that the intruders cannot derive any such encrypted messages, which is usually not considered to be reasonable. For example, when the protocol has been run before, there is the danger of a so-called replay attack.

Another proposal is to introduce in component b a fresh constant n_b, which is usually called a nonce, and which is only accessible by component b. It is fresh in the sense that it has not been used before, i.e., initially $\neg D_\omega.n_b$, and it is different from the nonces of the other components (to be introduced later on).

To ensure that the nonces can be used for future communications after successful execution of the authentication protocol, and to avoid so-called known plaintext attacks [4], it sometimes turns out to be desired that the intruders cannot derive the nonce in case $m_a = b \ \wedge\ m_b = a$. Unfortunately, this could not be specified in the original specification, since it does not refer to nonces at all. Moreover, it is not a general requirement on nonces, e.g., in some protocols [8] the nonces are transmitted in plaintext, but we need this requirement to be able to motivate some message details in the authentication protocol of Needham, Schroeder and Lowe. To formally specify this property, we require the following additional invariant

$$D.n_b \quad \Rightarrow \quad m_a \neq b \ \vee \ m_b \neq a$$

Again, this invariant might be easier to interpret in contrapositive form, but we prefer this shape for homogeneity.

Thus we would replace the message to be received by nonce n_b, and require for the assertion in component b the invariant

$$D.n_b \quad \Rightarrow \quad (m_b = a \ \Rightarrow \ m_a = b)$$

This invariant can be initialised using the freshness of nonce n_b, and it is trivially maintained under composition. However, the latter requirement on nonce n_b excludes the possibility of sending a message n_b in case $m_a = b \ \wedge \ m_b = a$, which would be required for progress. Therefore we propose to encrypt the nonce in the message using the public key of the intended receiver, which is a way to preserve confidentiality. After replacing the message by $[k.b : \ n_b]$ instead, the required invariant is

$$D.[k.b : \ n_b] \quad \Rightarrow \quad (m_b = a \ \Rightarrow \ m_a = b)$$

Thus we obtain the following intermediate program:

Pre: $\neg D_\omega.\overline{k.a} \ \wedge \ \neg D_\omega.\overline{k.b} \ \wedge \ \neg D_\omega.n_b$

$\cdots$	$\cdots$
	rcv $[k.b : \ n_b]$
$\{\mathbf{?} \ m_a = b \ \Rightarrow \ m_b = a\}$	$\{m_b = a \ \Rightarrow \ m_a = b\}$

Inv: $(\forall x :: \ D.\overline{k.x} \quad \Rightarrow \quad x \neq a \ \wedge \ x \neq b)$
Inv: $\mathbf{?} \ D.n_b \quad \Rightarrow \quad m_a \neq b \ \vee \ m_b \neq a$
Inv: $\mathbf{?} \ D.[k.b : \ n_b] \quad \Rightarrow \quad (m_b = a \ \Rightarrow \ m_a = b)$

Initial correctness of the queried invariants follows from the freshness of nonce n_b, i.e., $\neg D_\omega.n_b$. For the maintenance under composition, we only need to consider the invariant on $D.[k.b : \ n_b]$. Since the public key $k.b$ may be derivable, we must focus on the contents n_b, and require an invariant

$$D.n_b \quad \Rightarrow \quad (m_b = a \ \Rightarrow \ m_a = b)$$

Also this invariant can be initialised using the freshness of nonce n_b, and it is trivially maintained under composition. Although it can be simplified using the other invariant on $D.n_b$, we prefer to maintain the structure and combine it with the original invariant on $D.[k.b : \ n_b]$ using disjunction in the antecedent of the implication.

For progress in case $m_a = b \;\wedge\; m_b = a$, we must ensure that the message to be received $[k.b :\; n_b]$ can actually be sent. Since the components cannot refer to each other's constants, we introduce in component a a variable p of type nonce and a send statement for a message $[k.m_a : p]$.

What remains is to guarantee maintenance of the invariants under this send statement, including decomposition. We obtain the following series of proof obligations, one for each relevant invariant:

- $D.\overline{k.m_a} \;\wedge\; p = n_b \;\;\Rightarrow\;\; m_a \neq b \;\vee\; m_b \neq a$
- $k.m_a = k.b \;\wedge\; p = n_b \;\;\Rightarrow\;\; (m_b = a \;\Rightarrow\; m_a = b)$
- $D.\overline{k.m_a} \;\wedge\; p = n_b \;\;\Rightarrow\;\; (m_b = a \;\Rightarrow\; m_a = b)$

The first proof obligation follows from the existing invariant $D.\overline{k.m_a} \;\Rightarrow\; m_a \neq b$. We strengthen the last two proof obligations and require them as a combined pre-assertion $p = n_b \;\;\Rightarrow\;\; (m_b = a \;\Rightarrow\; m_a = b)$ of the send statement. In addition we copy the original post-assertion of component a as a pre-assertion of the send statement, since it contains no occurrences of D.

Pre: $\neg D_\omega.\overline{k.a} \;\wedge\; \neg D_\omega.\overline{k.b} \;\wedge\; \neg D_\omega.n_b$

$\cdots$	$\cdots$
$\{?\; m_a = b \;\Rightarrow\; m_b = a\}$	
$\{?\; p = n_b \;\Rightarrow\; (m_b = a \;\Rightarrow\; m_a = b)\}$	
snd $[k.m_a :\; p]$	**rcv** $[k.b :\; n_b]$
$\{m_a = b \;\Rightarrow\; m_b = a\}$	$\{m_b = a \;\Rightarrow\; m_a = b\}$

Inv: $(\forall x :: \; D.\overline{k.x} \;\;\Rightarrow\;\; x \neq a \;\wedge\; x \neq b)$

Inv: $D.n_b \;\;\Rightarrow\;\; m_a \neq b \;\vee\; m_b \neq a$

Inv: $D.[k.b :\; n_b] \;\vee\; D.n_b \;\;\Rightarrow\;\; (m_b = a \;\Rightarrow\; m_a = b)$

Notice that termination is guaranteed if initially $m_a = b \wedge p = n_b$, provided that the intruders behave like a proper communication channel.

5.2 Middle message communication

For queried assertion $m_a = b \;\Rightarrow\; m_b = a$ in component a we could repeat the same idea. That is, introducing a fresh nonce n_a (in particular, different from n_b) with an invariant

$$D.n_a \;\;\Rightarrow\;\; m_a \neq b \;\vee\; m_b \neq a$$

and introducing the receipt of a message $[k.a :\; n_a]$ in component a, using invariant

$$D.[k.a :\; n_a] \;\vee\; D.n_a \;\;\Rightarrow\;\; (m_a = b \;\Rightarrow\; m_b = a)$$

By construction these two invariants are maintained under composition, and they can be initialised using the freshness of nonce n_a. Furthermore, in component b we would introduce a variable q of type nonce, and a send statement for a message $[k.m_b : \ q]$ with the required pre-assertion $q = n_a \ \Rightarrow \ (m_a = b \ \Rightarrow \ m_b = a)$.

Before studying the effect of the previous statements on these new invariants, and vice versa, we first focus on the other queried assertion in component a, viz., the assertion $p = n_b \ \Rightarrow \ (m_b = a \ \Rightarrow \ m_a = b)$, which also refers to variables and constants from both components. The most direct way to establishing it would be to use the proposed receive statement using an invariant

$$D.[k.a : \ n_a] \quad \Rightarrow \quad (p = n_b \ \Rightarrow \ (m_b = a \ \Rightarrow \ m_a = b))$$

This invariant can be initialised using the freshness of nonce n_a, but regarding composition the shape of the existing invariants is not very helpful. The last invariant on $D.n_b$ might become useful, and although nonce n_b is not accessible in component a, we can try to exploit the term $p = n_b$.

For the maintenance under composition, we propose to add to the receive statement an assignment to variable p, which will be needed anyhow for progress. The received message becomes $[k.a : n_a, p]$, and we require, after exploiting the antecedent $p = n_b$, the invariant

$$D.[k.a : \ n_a, n_b] \quad \Rightarrow \quad (m_b = a \ \Rightarrow \ m_a = b)$$

This invariant can be initialised using the freshness of nonce n_b (or nonce n_a), and it is maintained under composition thanks to nonce n_b.

For progress reasons, the proposed send statement in component b must be extended with a value n_b into $[k.m_b : \ q, n_b]$, and moreover, for the maintenance of the invariant, with a pre-assertion $q = n_a \ \Rightarrow \ (m_b = a \ \Rightarrow \ m_a = b)$. This is almost the original post-assertion in component b again, and it is the place where the original Needham-Schroeder protocol [26] turns out to be vulnerable [21] to a so-called man-in-the-middle attack in case $m_b = a \wedge m_a \neq b$ holds and $q = n_a$ is established. Although this attack can easily be reconstructed from these formulae, we will not provide it as it is not relevant for the derivation.

To eliminate this pre-assertion, we aim to establish the consequent $m_a = b$. We propose to add the constant m_a to the receive statement, yielding the receipt of a message $[k.a : n_a, p, m_a]$. Thus we require the invariant

$$D.[k.a : \ n_a, n_b, m_a] \quad \Rightarrow \quad (m_b = a \ \Rightarrow \ m_a = b)$$

This invariant can still be initialised, and it is still maintained under composition, both using nonce n_b. For progress reasons, the corresponding send statement in component b must be extended with the value b into $[k.m_b : \ q, n_b, b]$, and hence it does not require a pre-assertion for this invariant.

Notice that both extensions of the message are necessary, as only considering the last one would again give problems with composition. Also notice that this strategy for the term $m_a = b$ would not help directly for establishing the post-assertion of component b, because the resulting assertion in component

a is also related to an invariant on $D.n_b$.

Given this expanded message, the earlier mentioned invariants resulting from assertion $m_a = b \;\Rightarrow\; m_b = a$ in component a become

- $D.n_a \;\Rightarrow\; m_a \neq b \;\vee\; m_b \neq a$
- $(\exists x ::\; D.[k.a:\; n_a, x, m_a]) \;\vee\; D.n_a \;\Rightarrow\; (m_a = b \;\Rightarrow\; m_b = a)$

These invariants can still be initialised using the freshness of nonce n_a, and they are maintained under composition. The proposed required pre-assertion $q = n_a \;\Rightarrow\; (m_a = b \;\Rightarrow\; m_b = a)$ for the new send statement $[k.m_b:\; q, n_b, b]$ remains unchanged, using the freshness assumption $n_a \neq n_b$.

After separately studying the two message communications, i.e., the one from this section and the one from Section 5.1, we must consider their combination. For the new invariants, (recursive) decomposition of the previous message $[k.m_a:\; p]$ leads to the following two proof obligations:

- $D.\overline{k.m_a} \;\wedge\; p = n_a \;\Rightarrow\; m_a \neq b \;\vee\; m_b \neq a$
- $D.\overline{k.m_a} \;\wedge\; p = n_a \;\Rightarrow\; (m_a = b \;\Rightarrow\; m_b = a)$

Notice that the amount of proof obligations is quite small, because the shapes of the two messages are different. Both proof obligations follow from invariant $D.\overline{k.m_a} \;\Rightarrow\; m_a \neq b$.

Maintenance of the previous invariants under the (recursive) decomposition of the new message $[k.m_b:\; q, n_b, b]$ results in the following two proof obligations:

- $D.\overline{k.m_b} \;\Rightarrow\; m_a \neq b \;\vee\; m_b \neq a$
- $D.\overline{k.m_b} \;\Rightarrow\; (m_b = a \;\Rightarrow\; m_a = b)$

Both proof obligations follow from invariant $D.\overline{k.m_b} \;\Rightarrow\; m_b \neq a$. Thus we obtain the following program:

Pre: $\neg D_\omega.\overline{k.a} \;\wedge\; \neg D_\omega.\overline{k.b} \;\wedge\; \neg D_\omega.n_a \;\wedge\; \neg D_\omega.n_b \;\wedge\; n_a \neq n_b$

$\ldots$	$\ldots$
rcv $p \leftarrow [k.a:\; n_a, p, m_a]$	$\{?\; q = n_a \;\Rightarrow\; (m_a = b \;\Rightarrow\; m_b = a)\}$
; $\{m_a = b \;\Rightarrow\; m_b = a\}$	**snd** $[k.m_b:\; q, n_b, b]$
$\{p = n_b \;\Rightarrow\; (m_b = a \;\Rightarrow\; m_a = b)\}$	;
snd $[k.m_a:\; p]$	**rcv** $[k.b:\; n_b]$
$\{m_a = b \;\Rightarrow\; m_b = a\}$	$\{m_b = a \;\Rightarrow\; m_a = b\}$

Inv: $(\forall x ::\; D.\overline{k.x} \;\Rightarrow\; x \neq a \;\wedge\; x \neq b)$

Inv: $D.n_a \;\vee\; D.n_b \;\Rightarrow\; m_a \neq b \;\vee\; m_b \neq a$

Inv: $(\exists x :: D.[k.a:\; n_a, x, m_a]) \;\vee\; D.n_a \;\Rightarrow\; (m_a = b \;\Rightarrow\; m_b = a)$

Inv: $D.[k.a:\; n_a, n_b, m_a] \;\vee\; D.[k.b:\; n_b] \;\vee\; D.n_b \;\Rightarrow\; (m_b = a \;\Rightarrow\; m_a = b)$

Notice that termination is guaranteed if initially $m_a = b \;\wedge\; m_b = a \;\wedge\; q = n_a$, provided that the intruders behave like a proper communication channel.

5.3 First message communication

The assertion $q = n_a \;\Rightarrow\; (m_a = b \;\Rightarrow\; m_b = a)$ in component b can be treated similarly to the assertion $p = n_b \;\Rightarrow\; (m_b = a \;\Rightarrow\; m_a = b)$ in component a. That is, receiving a nonce into variable q by receiving a message $[k.b :\; q, m_b]$ in component b, using a required invariant

$$D.[k.b :\; n_a, m_b] \quad\Rightarrow\quad (m_a = b \;\Rightarrow\; m_b = a)$$

This invariant can be initialised, and it is maintained under composition, both using nonce n_a. For progress we introduce a send statement for a message $[k.m_a :\; n_a, a]$ in component a, which does not require a pre-assertion for this invariant.

This invariant is also trivially maintained by the other send statements, and the new send statement also maintains the earlier invariants using the freshness assumption $n_a \neq n_b$. Thus we obtain the following program:

Pre: $\neg D_\omega.\overline{k.a} \;\wedge\; \neg D_\omega.\overline{k.b} \;\wedge\; \neg D_\omega.n_a \;\wedge\; \neg D_\omega.n_b \;\wedge\; n_a \neq n_b$

snd $[k.m_a :\; n_a, a]$	**rcv** $q \leftarrow [k.b :\; q, m_b]$
; **rcv** $p \leftarrow [k.a :\; n_a, p, m_a]$	; $\{q = n_a \;\Rightarrow\; (m_a = b \;\Rightarrow\; m_b = a)\}$
; $\{m_a = b \;\Rightarrow\; m_b = a\}$	**snd** $[k.m_b :\; q, n_b, b]$
$\{p = n_b \;\Rightarrow\; (m_b = a \;\Rightarrow\; m_a = b)\}$	;
snd $[k.m_a :\; p]$	**rcv** $[k.b :\; n_b]$
$\{m_a = b \;\Rightarrow\; m_b = a\}$	$\{m_b = a \;\Rightarrow\; m_a = b\}$

Inv: $(\forall x :: D.\overline{k.x} \quad\Rightarrow\quad x \neq a \;\wedge\; x \neq b)$
Inv: $D.n_a \;\vee\; D.n_b \quad\Rightarrow\quad m_a \neq b \;\vee\; m_b \neq a$
Inv: $(\exists x :: D.[k.a :\; n_a, x, m_a]) \;\vee\; D.[k.b :\; n_a, m_b] \;\vee\; D.n_a \quad\Rightarrow\quad (m_a = b \;\Rightarrow\; m_b = a)$
Inv: $D.[k.a :\; n_a, n_b, m_a] \;\vee\; D.[k.b :\; n_b] \;\vee\; D.n_b \quad\Rightarrow\quad (m_b = a \;\Rightarrow\; m_a = b)$

Notice that termination is guaranteed in case $m_a = b \;\wedge\; m_b = a$, provided that the intruders behave like a proper communication channel. Since there are no more queried assertions or invariants, this is the end of our derivation.

5.4 Agreement on the data items

It may also be desired [22] that the components agree on the specific data items that are used in the protocol, viz., the nonces. This program dependent part of the specification can be modelled using a post-assertion

$$m_a = b \;\wedge\; m_b = a \quad\Rightarrow\quad p = n_b \;\wedge\; q = n_a$$

in both honest components. We discuss this property after the main derivation, since it only introduces some more annotation, and in particular it does not drive the derivation.

This post-assertion in component b can be established by the last receive statement using an invariant

$$D.[k.b:\ n_b] \ \Rightarrow\ (m_a = b\ \wedge\ m_b = a\ \ \Rightarrow\ \ p = n_b\ \wedge\ q = n_a)$$

In turn, this invariant can be initialised using the freshness of nonce n_b, and it is maintained under composition using the first invariant on $D.n_b$. The required pre-assertion for the corresponding send statement in component a follows from a copy of the corresponding required post-assertion in component a.

This post-assertion in component a can be established by the preceding receive statement using an invariant

$$(\forall x ::\ D.[k.a:\ n_a, x, m_a]\ \Rightarrow\ (m_a = b\ \wedge\ m_b = a\ \Rightarrow\ x = n_b\ \wedge q = n_a))$$

In turn, this invariant can be initialised using the freshness of nonce n_a, and it is maintained under composition using the first invariant on $D.n_a$. The corresponding send statement in component b does not require any pre-assertion.

The occurrences of variables p and q in the invariants and in the assertions from the other component demand for a global correctness proof, viz., regarding the first two receive statements. Since there is only one assignment to them, the global correctness can be achieved by assuming as a pre-assertion of the receive statement (and precondition of the program) that $p \neq n_b$ and $q \neq n_a$ holds respectively, which ensures that the conditions $p = n_b$ and $q = n_a$ cannot be violated. Since the variables p and q are assigned a value before they are inspected, this precondition is only a trick for the annotation, and it can be omitted in the final program without influencing its behaviour.

5.5 Afterthought

The hardest part of the construction was in fact the introduction of the nonces, in the beginning of the derivation. The problem is that our specification does not mention the possibility that the protocol might be run more than once by a single component. Apart from running the protocol several times in succession, this could also include running it several times in parallel. To make this explicit in the specification of authentication, it must be possible to not only distinguish the different components, but also to distinguish the different runs of the components. We expect that such an identification mechanism would guide the derivation more easily towards the notion of a nonce.

One could argue that we have quickly adopted a few important design decisions, like the particular encryption scheme and the nonce concept. These are certainly not the only valid choices [8], but even given these two concepts, it is not trivial to design a correct authentication protocol. Also notice that the component identifiers in the messages (in particular in the first communication) are *not* intended to be able to send a reply to the right component. Our broadcast model clearly points out that there are more important reasons, viz., to avoid a dual of the man-in-the-middle attack from [21] in case $m_a = b \ \wedge \ m_b \neq a$.

It is useful [2] to investigate which properties we have used about the nonces and keys. We have assumed that the nonces are fresh, which means that initially they are not known by the intruders, i.e., $(\forall c :: \ \neg D_\omega.n_c)$, and that they are different from all other nonces, i.e., $(\forall c, d :: \ n_c = n_d \ \Rightarrow \ c = d)$. Notice that this gives no problem [1] in expressing that "the intruders can invent nonces, but they are not lucky enough to guess the random nonces on which the protocol depends". Regarding the keys, we have left the exact knowledge of the intruders unspecified; the only requirement is that they do not know the private keys of the honest components. In particular we have not assumed that the number of intruders is limited, nor that the private keys (nor the public keys) of the honest components are different. Hence, the encryption scheme is just used as a smoke screen against the intruders.

Observing the many symmetries in the protocol and its annotation, it may be tempting to strengthen the implication in the post-assertion of component b into an equivalence. However, this would not be valid for this protocol, and problems would manifest itself in maintenance under decomposition for the required composition invariant $D.n_b \ \Rightarrow \ (m_a = b \ \Rightarrow \ m_b = a)$, viz., in case $m_a = b \ \wedge \ m_b \neq a$. A similar argument applies to the post-assertion of component a.

It may seem to be more realistic to model the constants m_a, m_b, n_a and n_b as variables that are explicitly assigned a value by the components. A disadvantage of such a more-dynamic model is the additional amount of proof obligations. In particular, the correctness proof would demand some additional assertions to address the cases in which the variables have not yet been initialised; see also the discussion about variables p and q at the end of the previous section. In our opinion this would primarily increase the amount of work and formulae, while adding nothing to the understanding of the essence of this protocol.

6 Public-key distribution protocol

In contrast to the previous section where we have been very elaborate, our derivation of a simple public-key distribution protocol in this section will be more compact. Such a protocol is used in case the public keys are not yet accessible by all (honest) components. As the public keys are not required to be secret, the main security aspect is guaranteeing that the received key has not been modified by the intruders.

To be more precise, such a protocol enables any component a to obtain the public key $k.m_a$, assuming that component a has access to the public key $k.s$ from any honest (key server) component s that has access to $k.m_a$. For this protocol we assume that each private key is different from each public key, and that the public keys and the private keys are each other's inverses, i.e., for any key z, $\overline{\overline{z}} = z$.

6.1 Specification

Given an encryption scheme with public and private keys, a protocol between the honest components a and s that copies $k.m_a$ into variable i can be specified as follows.

Pre: $\neg D_\omega.\overline{k.a} \;\wedge\; \neg D_\omega.\overline{k.s}$

$\cdots$ $\{?\ i = k.m_a\}$	$\cdots$

Inv: $(\forall x ::\ D.\overline{k.x} \;\Rightarrow\; x \neq a \;\wedge\; x \neq s)$

Termination of the protocol only needs to be guaranteed if the intruders behave like a proper communication channel between the components a and s.

6.2 Derivation

Given that the key $k.m_a$ is accessible by component s, the queried assertion can be imported using a receive statement that assigns a value to variable i. The most obvious candidate message is just a key, leading to the following invariant

$$(\forall x ::\ D.x \;\Rightarrow\; x = k.m_a)$$

However, this cannot be initialised whenever the intruder can derive any key other than $k.m_a$. A second candidate message would therefore authenticate the message by encrypting the key using the private key of the key server, leading to the following invariant

$$(\forall x ::\ D.[\overline{k.s} :\ x] \;\Rightarrow\; x = k.m_a)$$

Unfortunately, this cannot be initialised whenever any other key has already been transmitted in this way. Therefore, in addition to the last message, we introduce a field denoting the value m_a, yielding the following intermediate program:

Pre: $\neg D_\omega.\overline{k.a} \;\wedge\; \neg D_\omega.\overline{k.s} \;\wedge\; (\forall x ::\; D_\omega.[\overline{k.s}:\; x, m_a] \;\;\Rightarrow\;\; x = k.m_a)$

$\ldots$	$\ldots$
rcv $i \leftarrow [\overline{k.s}:\; i, m_a]$	
$\{i = k.m_a\}$	

Inv: $(\forall x ::\; D.\overline{k.x} \;\;\Rightarrow\;\; x \neq a \;\wedge\; x \neq s)$

Inv: **?** $(\forall x ::\; D.[\overline{k.s}:\; x, m_a] \;\;\Rightarrow\;\; x = k.m_a)$

The remaining queried invariant can usually be initialised, and it is maintained under composition using the first invariant on $D.\overline{k.s}$. For progress reasons, we must introduce in component s a send statement for a message $[\overline{k.s}:\;\; k.v, v]$, for any variable v.

As the public key $k.s$ may be derivable, the transmitted key $k.v$ may be revealed by the intruders via decomposition. Regarding this new key, the invariants are maintained, using the assumption that the private and public keys are different. Regarding the earlier transmitted messages, we strengthen the invariants using our generalisation and the singleton key set $\{k.v\}$. This turns out to maintain their correctness, thus yielding the following intermediate program:

Pre: $\neg D_\omega.\overline{k.a} \;\wedge\; \neg D_\omega.\overline{k.s} \;\wedge\; (\forall x ::\; D_\omega.[\overline{k.s}:\; x, m_a] \;\;\Rightarrow\;\; x = k.m_a)$

$\ldots$	$\ldots$
rcv $i \leftarrow [\overline{k.s}:\; i, m_a]$	
$\{i = k.m_a\}$	**snd** $[\overline{k.s}:\;\; k.v, v]$

Inv: $(\forall x ::\; D_{\{k.v\}}.\overline{k.x} \;\;\Rightarrow\;\; x \neq a \;\wedge\; x \neq s)$

Inv: $(\forall x ::\; D_{\{k.v\}}.[\overline{k.s}:\; x, m_a] \;\;\Rightarrow\;\; x = k.m_a)$

There are no more queried assertions or invariants, but for progress reasons we must still ensure that variable v in component s can be set to m_a. To this end we can safely introduce a message communication, without any invariants or security concerns. By assigning any value to v, the invariants must be strengthened again, but maintenance remains guaranteed. We use $(c ::\;\; k.c)$ to denote the set of all public keys, and obtain the following final program:

Pre: $\neg D_\omega.\overline{k.a} \;\wedge\; \neg D_\omega.\overline{k.s} \;\wedge\; (\forall x ::\; D_\omega.[\overline{k.s}:\; x, m_a] \;\;\Rightarrow\;\; x = k.m_a)$

snd m_a	**rcv** $v \leftarrow v$
; **rcv** $i \leftarrow [\overline{k.s}:\; i, m_a]$	;
$\{i = k.m_a\}$	**snd** $[\overline{k.s}:\;\; k.v, v]$

Inv: $(\forall x ::\; D_{(c::\; k.c)}.\overline{k.x} \;\;\Rightarrow\;\; x \neq a \;\wedge\; x \neq s)$

Inv: $(\forall x ::\; D_{(c::\; k.c)}.[\overline{k.s}:\; x, m_a] \;\;\Rightarrow\;\; x = k.m_a)$

Notice that termination is guaranteed if the intruders behave like a proper communication channel. Since there are no more queried assertions, this is the end of our derivation.

Notice that the first (plaintext) message does not contain the value a, which might be useful for sending the reply to the right component. However, as we only consider broadcasts, this is not needed in our model. Moreover, we can easily extend this example to distribute keys to a series of components, for which a repetition (or a parallel composition) is likely to be introduced in the server component.

7 Composition

In this section we use the results from Section 5 and 6 to reason about composed security protocols.

7.1 Protocol integration

For reasoning about the concatenation of the public-key distribution protocol with the authentication protocol, the occurrences of D_K in the invariants must first be made homogeneous. In this case we strengthen the invariants for the authentication protocol by replacing each occurrence of D by $D_{(c::\ k.c)}$. Assuming that the public and private keys are different, none of the transmitted messages in the authentication protocol uses encryption with any key from $(c ::\ \overline{k.c})$, and hence these stronger invariants are also maintained. The required precondition might become stronger, but in this case it already depends on the even stronger notion D_ω.

After this strengthening of the invariants, the annotated programs can easily be concatenated and all invariants turn out to be maintained under the additional statements. This example looks promising with respect to the important field of protocol composition; see also for example [11].

7.2 Proof reduction

The protocol so far deals with an instance of the specification from Section 4, viz., mutual authentication between one pair of components. Given a common assumption on the keys and nonces, we present a reduction technique that shows that the same protocol can deal with the general specification from Section 4, viz., mutual authentication between any number of disjoint pairs of components. Although this more general case can also be addressed directly using our method, the reduction simplifies matters as it does not refer to the details of the annotation.

The specification of authentication from Section 4 consists of a conjunction of safety requirements. Namely, establish for each pair (a, b) from $A \times B$ the post-assertion $m_a = b \ \Rightarrow\ m_b = a$ in component a, and the post-assertion

$m_b = a \Rightarrow m_a = b$ in component b. Also the requirement that the private keys should remain secret can be specified as a conjunction of invariants $\neg D.\overline{k.a}$ and $\neg D.\overline{k.b}$ for each pair of components (a, b). Such conjunctions can be exploited as, for any system, the required assertions and invariants can be proved separately.

Given our earlier focus on one pair of components, we propose to divide the requirements according to such pairs. By symmetry we only need to focus on the requirements for one pair of components, but this time in a system with an arbitrary number of honest components. To reuse our earlier correctness results in the more general setting, we must somehow deal with the additional honest components. In what follows we investigate whether these can be modelled together with the intruders, as the possible behaviour of the general intruder model clearly contains the behaviour of each honest component. So what remains to consider is the precondition, which requires that initially the private key and the nonce of each component may not occur in the other components, and for any key x and component y, the message $[\overline{k.s} : x, y]$ can only occur if $x = k.y$. In particular this means that the nonces and private keys of the honest components must be different. These assumptions are common, e.g., in [21,28,30], but we do not suggest that the requirement on the private keys is strictly necessary for this protocol.

8 Conclusions and further work

At the end of this exploration of the derivation of security protocols, we must assess what we have achieved. The general style of reasoning from the programming method of Feijen and van Gasteren also turns out to be effective for this application area. In particular it avoids the well-known attack on the original authentication protocol of Needham and Schroeder in a natural way. After introducing the notion of a nonce, this protocol practically develops itself. The annotation developed along the derivation can even be used for reasoning about the composition of protocols.

To achieve these results, we have developed simple specifications of secrecy and authentication, in a familiar style of pre- and post-conditions and invariants. For dealing with the interference caused by the intruders, we have slightly modified an existing approach to deal with traditional message communication. The resulting method benefits from such a well-understood basis, and it is flexible enough to address patterns for both public-key confidentiality and private-key authentication.

The emphasis has been on keeping the style of reasoning manageable, such that it can also be applied manually. The resulting annotated programs are created in a demand driven way, and their correctness can easily be checked. Their annotation records the design decisions, and it provides a safe basis for reasoning about the messages. To reduce the size of the annotation, some of the invariants have been combined. Although this makes these invariants less appealing, they remain highly structured. A continued attention to notational concerns is an important piece of further work for effectively deriving and reasoning about protocols.

In contrast to automated approaches, we have already noticed that afterwards experimenting with minor variations of the protocols is a bit tricky, in which case the annotation should be checked carefully. In this sense it might be a useful piece of further work to develop an easy way to automatically verify the developed annotation using theorem provers, e.g., along the lines of [25,24]. Nevertheless we want to stress the importance of keeping the derivation techniques manageable, in order to avoid the introduction of errors, and to emphasise the design decisions.

For further work we want to develop some heuristics to better predict the effects of the design decisions, and hence simplify the derivations. It is also further work to specify other security properties in this style. Our specification of authentication should be extended to cover other notions of authentication [22], components that act both as an initiator and a responder, multi-party synchronisation [10], and components that perform the protocol several times.

It is also further work to study different intruder and encryption models, such that the derivations can highlight where the specific assumptions are used. We are also interested in studying the effect of this approach on other security protocols, and to create a taxonomy of protocols that are presented in the same way. Apart from studying some of the protocols for two or three components from [8], we want to consider various protocols with multiple components and sessions.

Acknowledgement

The author thanks the anonymous reviewers for their insightful comments.

References

[1] Abadi, M. and A. D. Gordon, *A calculus for cryptographic protocols: The spi calculus*, Information and Computation **148** (1999), pp. 1–70.

[2] Abadi, M. and R. Needham, *Prudent engineering practice for cryptographic protocols*, IEEE Transactions on Software Engineering **22** (1996), pp. 6–15.

[3] Alves-Foss, J. and T. Soule, *A weakest precondition calculus for analysis of cryptographic protocols*, in: *Proceedings of the DIMACS Workshop on Design and Formal Verification of*

Security Protocols, 1997.

[4] Bird, R., I. Gopal, A. Herzberg, P. Janson, S. Kutten, R. Molva and M. Yung, *Systematic design of a family of attack-resistant authentication protocols*, IEEE Journal on Selected Areas in Communications **11** (1993), pp. 679–693.

[5] Burrows, M., M. Abadi and R. Needham, *A logic of authentication*, ACM Transactions on Computer Systems **8** (1990), pp. 18–36.

[6] Butler, M. J., *On the use of data refinement in the development of secure communication systems*, Formal Aspects of Computing **14** (2002), pp. 2–34.

[7] Buttyán, L., *Formal methods in the design of cryptographic protocols (state of the art)*, Technical report SCC/1999/38, Swiss Federal Institute of Technology (EPFL) (1999).

[8] Clark, J. and J. Jacob, *A survey of authentication protocol literature*, Technical report, Department of Computer Science, University of York (1997).

[9] Clarke, E. M., S. Jha and W. R. Marrero, *Using state space exploration and a natural deduction style message derivation engine to verify security protocols*, in: D. Gries and W. P. de Roever, editors, *Proceedings of the IFIP working conference on Programming Concepts and Methods* (1998), pp. 87–106.

[10] Cremers, C. J. F. and S. Mauw, *Generalizing Needham-Schroeder-Lowe for multi-party authentication*, Computer Science Report 06-04, Technische Universiteit Eindhoven (2006).

[11] Datta, A., A. Derek, J. C. Mitchell and D. Pavlovic, *A derivation system and compositional logic for security protocols*, Journal of Computer Security **13** (2005), pp. 423–482.

[12] Dijkstra, E. W., "A Discipline of Programming," Prentice Hall, 1976.

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

[14] Dongol, B. and D. Goldson, *Extending the theory of Owicki and Gries with a logic of progress*, Logical Methods in Computer Science **2** (2006), pp. 1–25.

[15] Dongol, B. and A. J. Mooij, *Progress in deriving concurrent programs: emphasizing the role of stable guards*, in: T. Uustalu, editor, *Proceedings of the conference on Mathematics of Program Construction*, LNCS **4014** (2006), pp. 140–161.

[16] Dongol, B. and A. J. Mooij, *Streamlining progress-based derivations of concurrent programs*, Technical Report SSE-2006-06, School of Information Technology and Electrical Engineering, The University of Queensland (2006), to appear in the Formal Aspects of Computing journal.

[17] Feijen, W. H. J. and A. J. M. van Gasteren, "On a method of multiprogramming," Springer-Verlag, 1999.

[18] Feijen, W. H. J., A. J. M. van Gasteren and B. Schieder, *An elementary derivation of the alternating bit protocol*, in: J. Jeuring, editor, *Proceedings of the conference on Mathematics of Program Construction*, LNCS **1422** (1998), pp. 175–187.

[19] Fidge, C. J., *A survey of verification techniques for security protocols*, Technical Report 01-22, Software Verification Research Centre, The University of Queensland (2001).

[20] Hoogerwoord, R. R., *A formal derivation of a sliding window protocol*, Computer Science Report 06-31, Technische Universiteit Eindhoven (2006).

[21] Lowe, G., *Breaking and fixing the Needham-Schroeder public-key protocol using FDR*, in: T. Margaria and B. Steffen, editors, *Proceedings of the conference on Tools and Algorithms for the Construction and Analysis of Systems*, LNCS **1055** (1996), pp. 147–166.

[22] Lowe, G., *A hierarchy of authentication specifications*, in: *Proceedings of the Computer Security Foundations Workshop* (1997), pp. 31–44.

[23] Meadows, C., *Invariant generation techniques in cryptographic protocol analysis*, in: *Proceedings of the Computer Security Foundations Workshop* (2000), pp. 159–167.

[24] Mooij, A. J., "Constructive formal methods and protocol standardization," Ph.D. thesis, Technische Universiteit Eindhoven (2006).

[25] Mooij, A. J. and J. W. Wesselink, *Incremental verification of Owicki/Gries proof outlines using PVS*, in: K.-K. Lau and R. Banach, editors, *Proceedings of the conference on Formal Engineering Methods*, LNCS **3785** (2005), pp. 390–404.

[26] Needham, R. and M. Schroeder, *Using encryption for authentication in large networks of computers*, Communications of the ACM **21** (1978), pp. 993–999.

[27] Owicki, S. and D. Gries, *An axiomatic proof technique for parallel programs I*, Acta Informatica **6** (1976), pp. 319–340.

[28] Paulson, L. C., *The inductive approach to verifying cryptographic protocols*, Journal of Computer Security **6** (1998), pp. 85–128.

[29] Perrig, A. and D. Song, *A first step towards the automatic generation of security protocols*, in: *Proceedings of the Network and Distributed System Security symposium* (2000).

[30] Thayer Fábrega, F. J., J. C. Herzog and J. D. Guttman, *Strand spaces: proving security protocols correct*, Journal of Computer Security **7** (1999), pp. 191–230.