Towards a Completeness Result for Model Checking of Security Protocols

Gavin Lowe*

June 9, 1999

Abstract

Model checking approaches to the analysis of security protocols have proved remarkably successful. The basic approach is to produce a model of a small system running the protocol, together with a model of the most general intruder who can interact with the protocol, and then to use a state exploration tool to search for attacks. This has led to a number of new attacks upon protocols being discovered.

However, if no attack is found, this only tells us that there is no attack upon the small system we modelled; there may be an attack upon some larger system. This is the question we consider in this paper: we prove that under certain conditions on the protocol and the environment in which it operates, if there is no attack upon a particular small system (with one honest agent for each role of the protocol) leading to a breach of secrecy, then there is no attack on any larger system leading to a breach of secrecy.

*Department of Mathematics and Computer Science, University of Leicester, University Road, Leicester, LE2 7RH, UK. e-mail gavin.lowe@mcs.le.ac.uk.

Contents

1 Introduction

Many security protocols have appeared in the academic literature, with various goals, such as establishing a cryptographic key, or achieving authentication (where an agent becomes sure of the identity of the other agent taking part in the protocol). These protocols are supposed to succeed even in the presence of a malicious agent, called an *intruder*; this intruder is assumed to have complete control over the communications network, and so can intercept messages, and introduce new messages into the system using information from messages he has previously seen. Unfortunately, a large proportion of the protocols that have been suggested do not succeed in their stated goals!

Model checking has proved to be a very successful approach to analysing security protocols. The basic approach is to produce a model of a small system running the protocol (for example, with one initiator and one responder), together with a model of the most general intruder who can interact with the protocol, and to use a state exploration tool to discover if the system can enter an insecure state, that is, whether there is an attack upon the protocol. A number of different authors have used this approach, either with general purpose [10, 13, 17, 21, 4, 11] or special purpose model checkers [20, 10, 18]; some tools (for example, the NRL Protocol Analyzer [10, 19]) combine aspects of theorem proving and model checking to try to show that it is enough to consider only a finite set of states, and then to search those states. These approaches have discovered a number of attacks upon security protocols.

However, if a model checker fails to find an attack, this only means that there is no attack on the particular small system analysed: there may be an attack on some larger system running the same protocol. This is the question we consider in this paper. We define a protocol to be *small system secure* if there is no breach of secrecy on a particular small system: essentially, this is a system with one honest agent in each role, each of whom can run the protocol precisely once with the other honest agents. We will present sufficient conditions on protocols and their environment such that if the protocol is small system secure, then the protocol is secure in the sense that there is no attack on *any* system. In other words, for such protocols, model checking is complete.

Our main conditions on protocols are essentially the following two principles, taken from [1]:

Principle 3 If the identity of a principal is essential to the meaning of a message, it is prudent to mention the principal's name explicitly in the message.

Thus an agent receiving an encrypted component can tell whether it was intended for him; further, if the component originated from an honest agent then the recipient can tell who the originator was, and who the other agents in the run were.

Principle 10 If an encoding is used to present the meaning of a message, then it should be possible to tell which encoding is being used. In the common case where the encoding is protocol dependent, it should be possible to deduce ... its number in the protocol.

This principle prevents an intruder replaying one message of a protocol and having it interpreted as an instance of another message; the condition essentially implies that the different encrypted components in the protocol are textually distinct—i.e. that they have different forms.

We refer to these two conditions as the *explicitness principles*. It is well known that adherence to these principles helps to prevent many attacks. What we show in this paper is that adherence to these principles, and the other assumptions we make on protocols, prevents all attacks other than the (normally quite trivial) ones that exist on small systems.

The results of this paper are mainly geared towards the use of model checkers. However, we suggest that other approaches to verifying protocols might be made easier if one only has to consider small systems.

In the next section we present the model we will use for reasoning about protocols, and so define the conditions and assumptions on protocols and their environments that we will be making; we then formally state the main result of the paper. In Section 3 we present some examples, showing the use of this result in practice. In Section 4 we prove a number of lemmas, and then in Section 5 we prove the main result. We sum up in Section 6.

A few proofs of lemmas are omitted because of lack of space; the interested reader is referred to [16].

2 The model

In this section we formalize the model we will be using to reason about protocols, present the assumptions we are making about protocols, and so state our main result. Every mathematical model makes assumptions (either explicitly or implicitly) about the system it is modelling; it is important to ensure that these assumptions are understood to avoid mis-applying the results; it is also important for implementors to understand the obligations that these assumptions place upon them.

Most of the assumptions about the protocol's environment are fairly standard in the protocol analysis community; they identify the level of abstraction at which we are working; for example, we are not interested in attacks caused by cracking keys, so we assume perfect encryption. However, some of the conditions on protocols are slightly restrictive; they are either artefacts of the model, or are included to simplify the proof. Protocols can take many varied forms, and so being able to reason about all protocols requires a rather detailed model. We believe that most of the simplifying assumptions are not strictly necessary, and we hope to remove them in the future; in most cases, doing so will involve refining the model so as to include extra protocol features, and will add extra "noise" to the proof.

We will illustrate several aspects of our model using the following (slightly adapted) version of the Yahalom Protocol [3]:

$$\begin{array}{llcl}
\text{Message 1.} & \mathsf{init} \to \mathsf{resp} & : & \mathsf{init}, na \\
\text{Message 2.} & \mathsf{resp} \to \mathsf{serv} & : & \{\mathsf{init}, na, nb\}_{shared(\mathsf{resp},\mathsf{serv})} \\
\text{Message 3.} & \mathsf{serv} \to \mathsf{init} & : & \{\mathsf{resp}, kab, na, nb\}_{shared(\mathsf{init},\mathsf{serv})} \\
\text{Message 4.} & \mathsf{serv} \to \mathsf{resp} & : & \{\mathsf{init}, kab\}_{shared(\mathsf{resp},\mathsf{serv})} \\
\text{Message 5.} & \mathsf{init} \to \mathsf{resp} & : & \{\mathsf{init}, \mathsf{resp}, \mathsf{serv}, nb\}_{kab}.
\end{array}$$

Here init, resp and serv represent the initiator, responder and server who are taking part in the protocol, respectively; the function *shared* returns a key that is shared between its arguments.

In the original version of this protocol, the encrypted component of message 4 was sent to the initiator, who forwarded it to the responder; we have adapted the protocol so that this component is sent direct, in order for the protocol to meet some of the assumptions we will introduce below.

One can think of a protocol description in "standard notation", like the one above, as being a template, parameterized by the free variables appearing in it (here init, resp, serv, na, nb, kab): these variables are instantiated with values of the correct type in each run, possibly different values in different runs. In any single agent's view of a protocol run, the variables are instantiated consistently, in a *substitutive* manner (i.e. the agent's view of the run is obtained from the template by consistently substituting values for the free variables).

In Section 2.1, we define the space of atomic values. In Section 2.2 we formally define message templates, along with some relevant notation, and in Section 2.3 we use these to define protocol templates.

In Section 2.4 we explain how message templates are instantiated so as to produce actual messages; we then formalize the two explicitness assumptions. In Section 2.5 we extend this to explain how protocol templates are instantiated so as to produce one agent's view of a run. In Section 2.6 we formalize the abilities of the intruder, and then in Section 2.7 we define the set of all valid traces (i.e. executions) of a system running the protocol.

In Section 2.8 we formalize the goals of the protocol; in Section 2.9 we define what we mean by a small system; and finally in Section 2.10 we state the main result of the paper.

An index of notation is included in Appendix A.

2.1 Atomic values

We assume the existence of a set $Value$ of atomic values, partitioned into a number of *types* (e.g. $Agent$, $Nonce$, $PublicKey$). We assume that all atoms are uniquely typed, the so called *strong typing assumption*, so, for example, an agent's identity cannot be mistaken for a nonce: that is, the types are disjoint, and cover $Value$. If V is a value, we write $\mathsf{type}(V)$ for its type.

Some values—at least those used to encrypt messages—will be designated as keys; we write Key for the set of all keys. We assume that if one member of a type is a key, then so are all other members of the same type. All keys will have inverses defined; messages encrypted with such keys can be decrypted with the corresponding inverse. For $K \in Key$, we write $\overline{K}$ for the inverse of K; if K is a symmetric key then $\overline{K} = K$; we assume $\overline{\overline{K}} = K$ in all cases.

The set $Agent$ will represent the set of agents' identities. We partition $Agent$ into sets $Honest$, representing honest agents, and $Dishonest$, representing dishonest agents:

$$\begin{aligned} Agent &= Honest \cup Dishonest, \\ Honest \cap Dishonest &= \{\}. \end{aligned}$$

There is no difference between the capabilities of a single dishonest agent, using several identities (and related keys), and several dishonest agents working together[1].

In many protocols, some of the data items depend directly on the identities of the agents; for example, an agent's public or secret key. We call such data items *long-term*, because the same values will be used in several runs of the protocol. We represent these dependencies by functions from tuples of agents' identities to appropriate values. We therefore assume the existence of a set $Func$ of functions, ranged over by f. We write $\mathsf{arity}(f)$ for the number of arguments that f takes. Each function will return a result of a fixed type, which we write as $\mathsf{ranType}(f)$. We assume that all such functions $f \in Func$ are *injective*; this means, for example, that two agents cannot have the same public key; similarly, if a function $shared$, say, returns the key shared between its two arguments, then $shared(A, B) \neq shared(B, A)$, so different keys should be used depending upon the roles adopted by the two agents. We also assume that different functions return distinct values. We formalize our assumptions about functions as follows:

Assumption 1

1. $\forall f \in Func \bullet f \in Agent^{\mathsf{arity}(f)} \rightarrow \mathsf{ranType}(f)$, where $Agent^n$ is the set of all n-tuples of agents' identities;
2. Every $f \in Func$ is injective;
3. $\forall f, g \in Func \bullet f \neq g \Rightarrow \operatorname{ran} f \cap \operatorname{ran} g = \{\}$.

The results of some function applications will be the inverses of other function applications. For example, if PK and SK return the public and secret keys of their arguments, respectively, then $PK(A)$ and $SK(A)$ will be inverses, for all agents A. We extend the "$\overline{\ \ }$" notation, and write $\overline{f}$ for the function that returns the inverses of the values returned by f, so[2]

$$\forall Args \in Agent^{\mathsf{arity}(f)} \bullet \overline{f}(Args) = \overline{f(Args)}.$$

This implies that for all such functions, $\overline{\overline{f}} = f$.

[1] In the informal commentary that accompanies our formalisms, we will tend to refer to a single intruder, but the reader should realize that there may, in fact, be several intruders working together.

[2] We adopt the functional programming convention of using variable names ending with "s" to denote tuples, sequences and sets.

2.2 Component templates

Messages in our model will be a sequence of *components*, where a component is either an atomic value, or an encryption. In this section we formalize what we mean by a component template.

We will use a set $FreeVar$ of free variables. These free variables will represent data items such as agents' identities, nonces, session keys, and user-supplied data. Throughout this paper, we will use small letters to range over free variables, and capital letters to range over the corresponding values.

Each variable x will have a type, written as $\mathsf{type}(x)$. We assume that the types returned by functions are disjoint from the types of free variables:

$$\forall x \in FreeVar \,;\, f \in Func \bullet \mathsf{type}(x) \neq \mathsf{ranType}(f).$$

Note that we are partitioning the data items into variables and the values returned by functions. This means that we do not allow one agent to treat a data item as a variable, and another to treat it as the value returned by a function. This forbids, for example, the seven message version of the Needham Schroeder Public Key Protocol, where the initiator obtains the responder's public key from a key server; the server treats this key as the result of a function applied to the responder's identity, whereas the initiator treats it as a variable, being willing to receive any value.

Some variables will be designated as *fresh*: if an honest agent introduces a value for such a variable into a run, then the value will be new in the sense that it hasn't appeared earlier; examples include nonces and session keys. Note that timestamps do not meet this definition of freshness, because two agents could independently use the same value for a timestamp. We define $FreshVar$ to be the set of such fresh variables. We assume that freshness is a property of a type, so that if one variable is fresh, then so are all others of the same type:

$$x \in FreshVar \wedge \mathsf{type}(x) = \mathsf{type}(y) \;\Rightarrow\; y \in FreshVar.$$

This slightly restricts the class of protocol we are dealing with, because it means that it is not possible, for example, to have two keys of the same type, one of which is a long-term key, and the other of which is a fresh session key. We write $Fresh$ for the set of values of fresh types:

$$Fresh \mathrel{\widehat{=}} \bigcup\{\mathsf{type}(x) \mid x \in FreshVar\}.$$

We will formalize the condition that new values are introduced for fresh variables in Section 2.7.

Some of the free variables in the protocol description represent *roles* (e.g. initiator, responder). We define $Role$ to be the set of such variables, where

$$\forall r \in Role \bullet \mathsf{type}(r) = Agent.$$

We do not assume that the intruder is able to legitimately adopt all the roles; for example, the intruder might be able to legitimately adopt the roles of initiator and responder, but not the role of server (more precisely, we do not assume that the honest agents are willing to run the protocol with the intruder when the intruder takes the role of server). We write $IntRole$ for the set of roles that the intruder may (legitimately) adopt, where $IntRole \subseteq Role$.

It would seem natural to similarly allow the honest agents to only adopt some subset of the roles; in fact, we have no need to make such an assumption. It turns out that the explicitness assumptions effectively insulate an agent's different roles from one another, preventing messages from being replayed between the roles. Hence allowing honest agents to adopt multiple roles does not affect the security of the protocol. If a protocol is secure when the honest agents are allowed to adopt arbitrary roles, then it is certainly secure when they can only adopt restricted roles.

Some variables, namely those whose type is a key type, will have inverses defined. Variables that are inverses of one another will be instantiated to values that are inverses of one another, so messages encrypted with the instantiation of a variable can be decrypted with the instantiation of the inverse. We write $\overline{k}$ for the inverse of k, and assume $\overline{\overline{k}} = k$.

When defining a protocol, we will represent a data item formed by the application of the function f to an argument $args$ by the pair $(f, args)$; $args$ will be a tuple of free variables taken from $Role$, showing the form of the argument. For example, if the variables init and resp represent the initiator and responder, then $(f, (\mathsf{init}, \mathsf{resp}))$ represents a value formed by applying the function f to the identities of the initiator and responder. We define the set of all such function application templates by:

$$FnAppTemp \mathrel{\widehat{=}} \{(f, args) \mid f \in Func, args \in Role^{\mathsf{arity}(f)}\}.$$

We avoid the notation $f(arg)$ because we want to stress the difference between applying functions to actual values, and function application templates, such as this, where the function is only applied after the variables have been instantiated with values.

We restrict ourselves to protocols whose messages are built up from atomic values by sequencing and "standard" encryption (either symmetric or asymmetric): we do not consider other cryptographic primitives such as hash functions or bit-wise exclusive-or encryption; extending our results to deal with these operators will be an important next step. We model the space of component templates by an inductively defined datatype:[3]

$$\begin{aligned} CptTemp ::= {} & \textsc{FreeVar}\, FreeVar \mid \\ & \textsc{FnAppTemp}\, FnAppTemp \mid \\ & \textsc{Encrypt}\, CptTemp^{+}\ AtomTemp, \end{aligned}$$

where

$$\begin{aligned} AtomTemp \mathrel{\widehat{=}} {} & \{\textsc{FreeVar}\, x \mid x \in FreeVar\} \\ & \cup \{\textsc{FnAppTemp}(f, as) \mid (f, as) \in FnAppTemp\}. \end{aligned}$$

Here and henceforth we use small capitals for datatype constructors; we will omit these datatype constructors where they are obvious from the context. We will use the syntactic sugar $f(\!| args |\!)$ as shorthand for $\textsc{FnAppTemp}(f, args)$; we omit the parentheses in function application templates such as $f(\!|(a, b)|\!)$, and simply write $f(\!|a, b|\!)$. We will use the syntactic sugar $\{m_1, \ldots, m_n\}_k$ for $\textsc{Encrypt}\langle m_1, \ldots, m_n\rangle\, k$[4].

[3] X^{+} represents the set of all non-empty sequences with members taken from X.

[4] We write sequences between angle brackets.

For example, the single component in message 4 of the example Yahalom Protocol, above, is

$$\textsc{Encrypt}\,\langle \textsc{FreeVar}\,\mathsf{init}, \textsc{FreeVar}\,kab\rangle\ (\textsc{FnAppTemp}(shared, (\mathsf{resp}, \mathsf{serv})))$$

or using the syntactic sugar, and omitting the FREEVAR constructor:

$$\{\mathsf{init}, kab\}_{shared(\!|\mathsf{resp},\mathsf{serv}|\!)}.$$

Note that we do not have a separate datatype constructor for sequencing, but instead combine sequencing with encryption. This will slightly simplify the proof, because encrypted components are a natural unit to reason about. The disadvantage of this approach is that a complete message template will consist of a sequence of component templates.

We use m, m', etc. to range over component templates; later, when we instantiate these templates with actual values, we will write M, M', etc for these instantiations.

Implicit in the above datatype definition is that two component templates are equivalent only if they have identical fields, so, for example, encryption is injective:

$$\{ms\}_k = \{ms'\}_{k'} \Leftrightarrow ms = ms' \wedge k = k'.$$

This is often called the *perfect encryption assumption* [18]. This also means that we are assuming that encryption satisfies no interesting algebraic properties, which rules out some common encryption algorithms.

2.2.1 Notation for component templates

In this section we define some notation for component templates; an index of notation is included in Appendix A.

We define the free variables of a component template, $\mathsf{FV}(m) \subseteq FreeVar$, by:

$$\begin{aligned}
\mathsf{FV} &: CptTemp \to \mathbb{P}(FreeVar),\\
\mathsf{FV}(\textsc{FreeVar}\,x) &\mathrel{\widehat{=}} \begin{cases}\{x, \overline{x}\}, & \text{if } \overline{x} \text{ is defined}\\ \{x\}, & \text{otherwise}\end{cases}\\
\mathsf{FV}(f(\!|a_1, \ldots, a_n|\!)) &\mathrel{\widehat{=}} \{a_1, \ldots, a_n\},\\
\mathsf{FV}(\{m_1, \ldots, m_n\}_k) &\mathrel{\widehat{=}} \mathsf{FV}(m_1) \cup \ldots \cup \mathsf{FV}(m_n) \cup \mathsf{FV}(k).
\end{aligned}$$

Note that: (1) if a message is encrypted with a variable, then that variable is one of the message's free variables; (2) if a free variable of a message has an inverse, then that inverse is also a free variable of the message; (3) the arguments of a function are free variables of the message.

The definition of the atoms of a message template is similar, except we consider function applications to be atomic, and we do not include decrypting keys:

$$\begin{aligned}
\mathsf{atoms} &: CptTemp \to \mathbb{P}(AtomTemp),\\
\mathsf{atoms}(\textsc{FreeVar}\,x) &\mathrel{\widehat{=}} \{\textsc{FreeVar}\,x\},\\
\mathsf{atoms}(f(\!|as|\!)) &\mathrel{\widehat{=}} \{f(\!|as|\!)\},\\
\mathsf{atoms}(\{m_1, \ldots, m_n\}_k) &\mathrel{\widehat{=}} \mathsf{atoms}(m_1) \cup \ldots \cup \mathsf{atoms}(m_n) \cup \{k\}.
\end{aligned}$$

Later we will show that the intruder can build up a message from its atoms.

We define $\mathsf{subcpts}(m)$ to be the sub-components of m:[5]

$$
\begin{array}{l}
\mathsf{subcpts} : CptTemp \to \mathbb{P}(CptTemp), \\
\mathsf{subcpts}(\text{FREEVAR}\, x) \mathrel{\widehat{=}} \{\text{FREEVAR}\, x\} \\
\mathsf{subcpts}(f\,(\!|as|\!)) \mathrel{\widehat{=}} \{f\,(\!|as|\!)\} \\
\mathsf{subcpts}(\text{ENCRYPT}\langle m_1, \ldots, m_n\rangle\, k) \mathrel{\widehat{=}} \\
\qquad \{\text{ENCRYPT}\langle m_1, \ldots, m_n\rangle\, k\} \cup \mathsf{subcpts}(m_1) \cup \ldots \cup \mathsf{subcpts}(m_n).
\end{array}
$$

And we extend the $\mathsf{subcpts}$ notation to sequences of component templates:

$$\mathsf{subcpts}(\langle m_1, \ldots, m_n\rangle) \;\widehat{=}\; \mathsf{subcpts}(m_1) \cup \ldots \cup \mathsf{subcpts}(m_n).$$

For convenience, we define a predicate over message templates which is true when the argument is an encryption[6]:

$$
\begin{array}{rcl}
\mathsf{encrypted} & : & CptTemp \to \mathbb{B}, \\
\mathsf{encrypted}(\text{FREEVAR}\, x) & \widehat{=} & false, \\
\mathsf{encrypted}(f\,(\!|as|\!)) & \widehat{=} & false, \\
\mathsf{encrypted}(\{ms\}_k) & \widehat{=} & true.
\end{array}
$$

2.3 Protocols

We define a *communication template* to be a 4-tuple of the form

$$(n, a, b, \langle m_1, \ldots, m_l\rangle),$$

where n is a positive integer, a and b are free variables representing distinct roles, and m_1, …, m_l are component templates; this represents a communication of a message number n, from a to b, with contents $m_1, \ldots, m_l$; in more standard notation:

$$\text{Message } n.\quad a \to b \;:\; m_1, \ldots, m_l\ .$$

For example, message 4 of the example Yahalom Protocol is represented by:

$$(4, \mathsf{serv}, \mathsf{resp}, \langle\{\mathsf{init}, kab\}_{shared(\!|\mathsf{resp},\mathsf{serv}|\!)}\rangle).$$

Formally, we define the set of communication templates by:

$$
\begin{array}{l}
CommTemp \;\widehat{=}\; \{(n, a, b, ms) \mid n \in \mathbb{Z}^+ \wedge a, b \in Role \wedge \\
\qquad\qquad\qquad\qquad\qquad a \neq b \wedge ms \in CptTemp^+\}.
\end{array}
$$

We now define a *protocol template* to be a sequence of communication templates, where the message numbers (in order) form an initial prefix of the positive integers[7].

$$
\begin{array}{l}
ProtTemp \;\widehat{=}\; \{pt \mid pt \in CommTemp^+ \wedge \\
\qquad\qquad\qquad \exists\, N \in \mathbb{Z}^+ \bullet \langle n \mid (n,\_,\_,\_) \leftarrow pt\rangle = \langle 1 \mathinner{..} N\rangle\},
\end{array}
$$

[5] We use the ENCRYPT constructor here, rather than the syntactic sugared form, to avoid confusion between set brackets and encryption brackets.

[6] We write $\mathbb{B}$ for the set of boolean values.

[7] Our notation for sequence comprehensions follows the style of functional programming languages such as Haskell; the notation $x \leftarrow xs$ represents that x takes each of the values from xs in order.

Here and henceforth we follow standard function programming notation by using the wildcard "_" to represent a variable whose value we are not interested in; this allows the reader to concentrate on the important variables.

For example, the example Yahalom Protocol is represented by:

$$\begin{array}{l}\langle (1, \mathsf{init}, \mathsf{resp}, \langle \mathsf{init}, na \rangle), \\ \;(2, \mathsf{resp}, \mathsf{serv}, \langle \{\mathsf{init}, na, nb\}_{shared(\!|\mathsf{resp},\mathsf{serv}|\!)} \rangle), \\ \;(3, \mathsf{serv}, \mathsf{init}, \langle \{\mathsf{resp}, kab, na, nb\}_{shared(\!|\mathsf{init},\mathsf{serv}|\!)} \rangle), \\ \;(4, \mathsf{serv}, \mathsf{resp}, \langle \{\mathsf{init}, kab\}_{shared(\!|\mathsf{resp},\mathsf{serv}|\!)} \rangle), \\ \;(5, \mathsf{init}, \mathsf{resp}, \langle \{\mathsf{init}, \mathsf{resp}, \mathsf{serv}, nb\}_{kab} \rangle) \,\rangle.\end{array}$$

The above definition allows protocols that appear infeasible; for example, it allows protocols where there doesn't seem to be a causal link between successive messages, where data items seem to be transmitted "magically". Later we will add a condition about fresh variables being created by a unique agent; however, it turns out that we will not have to assume anything more about feasibility of protocols.

Some free variables and their associated values will be designated as *secret*: it will be the intention of the protocol that only those agents legitimately taking part in the protocol run can learn these secrets. Thus if an honest agent introduces a new value for a secret variable, then the intruder should never learn that value, unless the honest agent was deliberately running the protocol with the intruder; also, if an honest agent accepts a value for a secret variable, apparently in a run with other honest agents, then the intruder should not know and should never learn that value. We call these *short-term secrets*, and define $STSecrets$ to be the set of all such short-term-secret variables. In the running example, kab is intended to remain secret; in addition, nb remains secret, and it turns out (because of Assumption 6 below) that we need to specify it as such; so we take $STSecrets = \{kab, nb\}$. We assume that all short-term secrets are fresh.

Assumption 2 (All short-term secrets are fresh) All short-term-secret variables are fresh variables:

$$STSecrets \subseteq FreshVar.$$

Similarly, we classify functions as either public or secret, depending upon whether the values they return are supposed to be public information (e.g. public keys) or secret (e.g. secret keys, or long-term shared keys). We define $LTSecrets$ (long-term secrets) to be the set of such secret functions. In the running example, the shared keys returned by function $shared$ are supposed to remain secret, so $LTSecrets = \{shared\}$.

For convenience, we define $Secrets$ to be the union of short- and long-term-secret templates:

$$Secrets \mathrel{\widehat{=}} STSecrets \cup \{f(\!|args|\!) \mid f \in LTSecrets, args \in Role^{\mathsf{arity}(f)}\}.$$

Note that instantiations of templates of the form $f(\!|args|\!)$ are supposed to remain secrets from the intruder only when the arguments are instantiated to identities of honest agents; and instantiations of members of $STSecrets$ are supposed to remain secret from the intruder only when they are used in runs between honest agents.

In the rest of this paper we assume the protocol is given, so the rest of the definitions should be interpreted with being with respect to a certain fixed protocol description $pt \in ProtTemp$, along with associated fixed values for $FreeVar$, $Func$, $STSecrets$, etc.

For ease of reference, we define N to be the number of messages in the protocol, and sender_i, $\mathsf{receiver}_i$ and ms_i to be the sender, receiver and template for message i; so:

$$pt \ = \ \langle (i, \mathsf{sender}_i, \mathsf{receiver}_i, \mathsf{ms}_i) \mid i \leftarrow \langle 1 \mathinner{..} N \rangle \rangle.$$

We assume that no message within the protocol contains values returned by secret functions (although such values may be used to encrypt messages):

Assumption 3 (No long-term secrets within messages) No component template m in pt has a long-term function application amongst its subcomponents (recall that $\mathsf{subcpts}(m)$ excludes encrypting keys)[8]:

$$\forall (\_, \_, \_, ms) \ \mathsf{in} \ pt \bullet \forall m \ \mathsf{in} \ ms \bullet \forall f \in LTSecrets \bullet \forall args \in Role^{\mathsf{arity}(f)} \bullet \\ f(\!| args |\!) \notin \mathsf{subcpts}(m).$$

The above assumption prevents us from applying our results to protocols such as SET [25], where a "card secret"—a long-term secret—is included within a message.

Note that for each free variable x, there will be an earliest protocol message containing that variable; we call the number of this message $\mathsf{first}(x)$:

$$\mathsf{first}(x) \ \widehat{=} \ \min\{i \mid (i, \_, \_, \mathsf{ms}_i) \ \mathsf{in} \ pt \wedge x \in \mathsf{FV}(\mathsf{ms}_i)\}.$$

Similarly, for each free variable x and for each role r, there will be an earliest protocol message that r sees containing that variable; we call the number of this message $\mathsf{first}(x, r)$:

$$\mathsf{first}(x, r) \ \widehat{=} \ \min\{i \mid (i, \mathsf{sender}_i, \mathsf{receiver}_i, \mathsf{ms}_i) \ \mathsf{in} \ pt \wedge x \in \mathsf{FV}(\mathsf{ms}_i) \\ \wedge (r = \mathsf{sender}_i \vee r = \mathsf{receiver}_i)\}.$$

$\mathsf{first}(x, r)$ is undefined if no message that r sees includes x.

2.4 Message components

In the previous sections we considered message templates, containing free variables. We now consider actual messages, in which the free variables are instantiated with appropriate values.

We model the set of all message components by an inductively defined datatype Cpt, built up from the atomic values using encryption:

$$Cpt \ ::= \ \textsc{Atom}\, Value \mid \textsc{Encrypt}\, Cpt^{+} Value$$

As with templates, we will use the syntactic sugar $\{M_1, \ldots, M_n\}_K$ for $\textsc{Encrypt}\langle M_1, \ldots, M_n \rangle K$, where appropriate.

[8] The notation x in xs represents that x is an element of the *sequence* xs.

2.4.1 Instantiations

We define a valuation function to be a function from variables to values, giving the values assigned to variables by an honest agent in his view of a particular run of the protocol. Formally, a valuation is a function:
$val : FreeVar \to Value$
such that:

1. val respects the types of the variables:

   $\forall x \in FreeVar \bullet val(x) \in \mathsf{type}(x),$

2. val respects inverses:

   $\overline{x}$ is defined if and only if $\overline{val(x)}$ is defined, and in this case $val(\overline{x}) = \overline{val(x)}$,

3. val allows dishonest agents to adopt only roles in $IntRole$:

   $\forall r \in Role \bullet val(r) \in Dishonest \Rightarrow r \in IntRole.$

We write Val for the set of all valid valuation functions for the given protocol.

We write $\mathsf{inst}_{val}(m)$ for the instantiation of m under val[9]:

$$\begin{aligned} \mathsf{inst}_{val} &: CptTemp \to Cpt, \\ \mathsf{inst}_{val}(\text{FREEVAR}\,x) &\mathrel{\widehat{=}} \text{ATOM}(val(x)), \\ \mathsf{inst}_{val}(f(\!|a_1, \ldots, a_n|\!)) &\mathrel{\widehat{=}} \text{ATOM}(f(val(a_1), \ldots, val(a_n))), \\ \mathsf{inst}_{val}(\{m_1, \ldots, m_n\}_k) &\mathrel{\widehat{=}} \{\mathsf{inst}_{val}(m_1), \ldots, \mathsf{inst}_{val}(m_n)\}_{\mathsf{inst}_{val}(k)}. \end{aligned}$$

We extend the inst notation to deal with sequences of messages:
$\mathsf{inst}_{val}\langle m_1, \ldots, m_n\rangle \mathrel{\widehat{=}} \langle \mathsf{inst}_{val}(m_1), \ldots, \mathsf{inst}_{val}(m_n)\rangle.$

The fact that we only consider instantiation functions that respect the types of variables (part 1, above) means that we assume that agents can tell the types of values they receive, and only accept values of the expected type, thus preventing type flaw attacks, caused by a confusion of types. This assumption is fairly common amongst protocol analysis techniques, but is quite hard to achieve in implementations; we believe that discovering ways to prevent type flaw attacks is an important area of future research.

2.4.2 Notation for message components

We extend the atoms, $\mathsf{subcpts}$ and $\mathsf{encrypted}$ functions to message components:

$$\begin{aligned} &\mathsf{atoms} : Cpt \to \mathbb{P}(Value), \\ &\mathsf{atoms}(\text{ATOM}\,A) \mathrel{\widehat{=}} \{A\}, \\ &\mathsf{atoms}(\text{ENCRYPT}\langle M_1, \ldots, M_n\rangle K) \mathrel{\widehat{=}} \mathsf{atoms}(M_1) \cup \ldots \cup \mathsf{atoms}(M_n) \cup \{K\}, \\ &\mathsf{subcpts} : Cpt \to \mathbb{P}(Cpt), \\ &\mathsf{subcpts}(\text{ATOM}\,A) \mathrel{\widehat{=}} \{\text{ATOM}\,A\}, \\ &\mathsf{subcpts}(\text{ENCRYPT}\langle M_1, \ldots, M_n\rangle K) \mathrel{\widehat{=}} \\ &\quad \{\text{ENCRYPT}\langle M_1, \ldots, M_n\rangle K\} \cup \mathsf{subcpts}(M_1) \cup \ldots \cup \mathsf{subcpts}(M_n), \\ &\mathsf{subcpts}(\langle M_1, \ldots, M_n\rangle) \mathrel{\widehat{=}} \mathsf{subcpts}(M_1) \cup \ldots \cup \mathsf{subcpts}(M_n), \\ &\mathsf{encrypted}(\text{ATOM}\,A) \mathrel{\widehat{=}} false, \\ &\mathsf{encrypted}(\{Ms\}_K) \mathrel{\widehat{=}} true. \end{aligned}$$

[9] inst_{val} can be thought of as the homomorphic extension of val to all of $CptTemp$

The following lemma relates these functions to the functions of the same names over component templates:

Lemma 1 Let m be a component template, and val a valuation; then:

$$\begin{aligned} \mathsf{atoms}(\mathsf{inst}_{val}(m)) &= \{\mathsf{inst}_{val}(a) \mid a \in \mathsf{atoms}(m)\}, \\ \mathsf{subcpts}(\mathsf{inst}_{val}(m)) &= \{\mathsf{inst}_{val}(m') \mid m' \in \mathsf{subcpts}(m)\}, \\ \mathsf{encrypted}(\mathsf{inst}_{val}(m)) &\Leftrightarrow \mathsf{encrypted}(m). \end{aligned}$$

Let $ProtMsgs$ be the set of all instantiations of maximal components in the protocol description, and let $ProtCpts$ be the set of all the sub-messages of such instantiations:

$$\begin{aligned} ProtMsgs &\mathrel{\widehat{=}} \{\mathsf{inst}_{val}(m) \mid (i, a, b, ms) \text{ in } pt \wedge m \text{ in } ms \wedge val \in Val\}, \\ ProtCpts &\mathrel{\widehat{=}} \bigcup\{\mathsf{subcpts}(M) \mid M \in ProtMsgs\}. \end{aligned}$$

Note that elements of $ProtMsgs$ are maximal in the sense that they do not appear inside any larger component, whereas elements of $ProtCpts$ may be proper submessages, and may appear inside encryptions. From Lemma 1, we have the following equality:

$$ProtCpts = \{\mathsf{inst}_{val}(m) \mid (i, a, b, ms) \text{ in } pt \wedge val \in Val \wedge m \in \mathsf{subcpts}(ms)\}.$$

2.4.3 Formalizing explicitness

We now formalize the two explicitness assumptions mentioned in the introduction. Principle 3 of [1] stated that the identity of each agent taking part in the protocol run should be mentioned explicitly; we weaken this slightly to allow such identities to be inferred, for example from encryptions with public or secret keys of those agents. Principle 10 stated that it should be possible to tell a message's number from its encoding; we take this to mean that the different messages should be textually distinct. In both cases, we take the term "message" to refer to each encrypted component of the protocol.

We begin by enumerating the encrypted component templates of the protocol. We define a function $encs$ that returns the encrypted subcomponent templates of a message template in some standard order[10]:

$$\begin{aligned} encs &: CptTemp \rightarrow CptTemp^* \\ encs(\textsc{FreeVar}\, x) &\mathrel{\widehat{=}} \langle\rangle, \\ encs(f\,(\!|as|\!)) &\mathrel{\widehat{=}} \langle\rangle, \\ encs(\{m_1, \ldots, m_n\}_k) &\mathrel{\widehat{=}} \langle\{m_1, \ldots, m_n\}_k\rangle ^\frown encs(m_1) ^\frown \ldots ^\frown encs(m_n). \end{aligned}$$

We now enumerate the encrypted subcomponent templates of the protocol in question, in some standard order:

$$EncCpts \mathrel{\widehat{=}} \mathsf{concat}\langle encs(m) \mid (i, A, B, ms) \leftarrow pt, m \leftarrow ms\rangle,$$

(where concat is list concatenation). This allows us to formalize that the identities of all the agents taking part in this protocol run can be inferred from any encrypted component from the run.

[10] The precise choice of order is not important; we just need to enumerate the encrypted components in *some* order. The notation X^* represents the set of all finite sequences with members taken from X; "$^\frown$" is sequence concatenation.

Assumption 4 (Identities inferable) All encrypted subcomponents of protocol message templates include all the roles amongst their free variables:

$\forall m \text{ in } EncCpts \bullet Role \subseteq \mathsf{FV}(m)$.

The above condition may be met by including the identities of the agents explicitly within each encrypted component. Recall, though, that when a component is encrypted with the result of a function application, the arguments of the function are considered as free variables of the component, so the above condition may also be met by the agents' identities "appearing" in the encrypting key; for example, in message 4 of the running example, the message template is $\{\mathsf{init}, kab\}_{shared(\!|\mathsf{resp},\mathsf{serv}|\!)}$, and so resp and serv are in the free variables of the template (as, more obviously, is init), and so this template satisfies the above assumption. In implementations, the agents who deal with this message (resp and serv) would be able to deduce these identities from the keys, because of the assumed injectivity of the functions producing these keys. Similarly, if a message is encrypted with a secret key, a recipient can deduce the identity of the sender from this key; this, of course, assumes that each agent knows the public key function, which effectively removes the need for key certificates.

We also formalize our assumption that different encrypted components within the protocol messages have textually distinct forms.

Assumption 5 (Encrypted components are textually distinct) No two different elements of $EncCpts$ can be instantiated to the same value: if two encrypted component templates of protocol messages (say from positions i and j of $EncCpts$) can be instantiated (possibly under different instantiation functions) to the same message, then they are in fact the same template:

$\mathsf{inst}_{val}(EncCpts(i)) = \mathsf{inst}_{val'}(EncCpts(j)) \;\Rightarrow\; i = j$.

This means that an agent receiving an encrypted component will know which message it belongs to, and which component from that message. The above condition can be easily met, for example by including a component number within each encrypted component.

2.4.4 The @ notation

Assumption 5 means that for any encrypted component of a protocol message M, there is a unique message template m such that M is an instantiation of m. If M is an encrypted component which is an instantiation of encrypted component template m via instantiating function val:

$M \;=\; \mathsf{inst}_{val}(m)$,

and x is a free variable of m, then we write $M@x$ for the value with which x is instantiated in M, i.e. $val(x)$ (if x is not a free variable of m then $M@x$ is undefined). This is well defined by the above argument and the following lemma:

Lemma 2 If two instantiations of the same template are equal, then the instantiating functions must agree on all variables that appear in the message template:

$\mathsf{inst}_{val}(m) = \mathsf{inst}_{val'}(m) \;\Rightarrow\; \forall x \in \mathsf{FV}(m) \bullet val(x) = val'(x)$.

The proof of this lemma relies upon the injectivity of the functions (part 3 of Assumption 1).

Recall that if x is a free variable of m and $\overline{x}$ is defined, then $\overline{x}$ is also considered to be a free variable of m, and so in this case $M@\overline{x}$ is defined and equal to $\overline{M@x}$.

Assumption 4 tells us that for any encrypted component $M \in ProtCpts$, the corresponding template has each role $r \in Role$ amongst its free variables, and so $M@r$ is defined.

Let Ms be an instantiation of ms_i. We define $Ms@_i x$ to be V if Ms, when interpreted as a message i, has value V in the place of variable x. Formally:

$$Ms@_i x = V \Leftrightarrow x \in \mathsf{FV}(\mathsf{ms}_i) \wedge \exists\, val \in Val \bullet Ms = \mathsf{inst}_{val}(\mathsf{ms}_i) \wedge val(x) = V.$$

If x is not a free variable of ms_i, or Ms is not an instance of ms_i, then $Ms@_i x$ is undefined.

The following example should make it clear why we need to refer to the message number in the above definition: let $\mathsf{ms}_i = \langle \mathsf{init}, \mathsf{resp} \rangle$, $\mathsf{ms}_j = \langle \mathsf{resp}, \mathsf{init} \rangle$, and $Ms = \langle A, B \rangle$; then $Ms@_i\mathsf{init} = A$ but $Ms@_j\mathsf{init} = B$.

2.5 Strands

We now consider a single honest agent's view of a particular run of the protocol. We term this view a *strand*[11]. We assume the existence of a set $StrandId$ of *strand identifiers*, ranged over by α, β, etc, and write $\alpha.i$ for message i of strand α. We will ensure that each strand identifier is used in a unique strand.

Each strand will consist of a sequence of send and receive events, where events can take two forms:

send event: an event of the form $\mathsf{send}(\alpha.i, A, B, Ms)$ will represent honest agent A sending a message number i in strand α, intended for B, with contents Ms;

receive event: an event of the form $\mathsf{receive}(\alpha.i, A, B, Ms)$ will represent honest agent B receiving a message number i in strand α, apparently from A, with contents Ms.

Note that only a *single* honest agent will be involved in each such event.

We assume that each agent will only carry out runs where the variables are consistently instantiated: an agent will accept a new message that it receives only if the values assigned to free variables are consistent with what has gone before in this run.

One effect is that we restrict our attention to protocols where an agent is able to get at all parts of every message it receives. This means we do not consider protocols where an agent does not decrypt an encrypted component but instead simply forwards it to a third party. In the original Yahalom protocol, the message we have called message 4 was sent from the server to the initiator before being forwarded to the responder; we have redirected the protocol so as to fit into our model. We discuss the implications of this in the next section.

[11] The terminology is taken from [27]

Let r be a role, α a strand identifier, and val a valuation; we write $\mathsf{inst}_{r,\alpha,val}(i)$ for message $\alpha.i$ instantiated under val, as seen by the honest agent taking role r (i.e. agent $val(r)$); recalling that the message i in the protocol description pt is of the form $(i, \mathsf{sender}_i, \mathsf{receiver}_i, \mathsf{ms}_i)$, this is defined by:

$$\begin{array}{l} \mathsf{inst}_{r,\alpha,val}(i) \mathrel{\widehat{=}} \\ \qquad \begin{array}{ll} \mathsf{send}(\alpha.i, val(\mathsf{sender}_i), val(\mathsf{receiver}_i), \mathsf{inst}_{val}(\mathsf{ms}_i)), & \text{if } \mathsf{sender}_i = r \\ \mathsf{receive}(\alpha.i, val(\mathsf{sender}_i), val(\mathsf{receiver}_i), \mathsf{inst}_{val}(\mathsf{ms}_i)), & \text{if } \mathsf{receiver}_i = r \end{array} \end{array}$$

$\mathsf{inst}_{r,\alpha,val}(i)$ is undefined if role r does not take part in message i, i.e. if $\mathsf{sender}_i \neq r \wedge \mathsf{receiver}_i \neq r$.

A (partial) strand is a sequence of such events, corresponding to one agent's view of a (partial) run; this means:

- all the communications have the same strand identifier;
- the agent in question is the sender or receiver of each message;
- the message numbers (in order) form some prefix of the message numbers of the messages that this agent takes part in (i.e. sends or receives);
- the sender, receiver and message components are consistently instantiated, that is there exists some valuation val such that the communication corresponding to message i is of the form $\mathsf{inst}_{r,\alpha,val}(i)$.

Thus each strand may be parameterized by the role of the agent in question, the strand identifier, the valuation function, and the largest message number in the strand; this prompts the following definition:

$$\begin{array}{l} \mathsf{strandOf}(r, \alpha, val, n) \mathrel{\widehat{=}} \\ \qquad \langle \mathsf{inst}_{r,\alpha,val}(i) \mid i \leftarrow \langle 1 \mathinner{..} n \rangle, \mathsf{sender}_i = r \vee \mathsf{receiver}_i = r \rangle. \end{array}$$

The set $Strands$, then, consists of all such strands relating to honest agents:

$$\begin{array}{rl} Strands \mathrel{\widehat{=}} & \{\mathsf{strandOf}(r, \alpha, val, n) \mid r \in Role \wedge \alpha \in StrandId \wedge n \leq N \\ & \qquad \wedge\, val \in Val \wedge val(r) \in Honest\}. \end{array}$$

We make the following definitions:

$$\begin{array}{rcl} \mathsf{principal}(\mathsf{strandOf}(r, \alpha, val, n)) & \mathrel{\widehat{=}} & val(r), \\ \mathsf{partners}(\mathsf{strandOf}(r, \alpha, val, n)) & \mathrel{\widehat{=}} & \{val(r') \mid r' \in Role \wedge r' \neq r\}, \\ \mathsf{roleOf}(\mathsf{strandOf}(r, \alpha, val, n)) & \mathrel{\widehat{=}} & r. \end{array}$$

A few moments thought should persuade the reader that for strand s, $\mathsf{principal}(s)$ and $\mathsf{roleOf}(s)$ are well defined provided $s \neq \langle\rangle$, and $\mathsf{partners}(s)$ is well defined provided s contains at least one encrypted component.

Note that nothing in the above formalization prevents a single honest agent from taking several roles within a strand (i.e., where the valuation function maps several roles to the same identity); in this case, if this agent really is taking several roles (i.e. the intruder isn't faking the other roles) then there will be several strands for that agent, one for each role.

2.6 The intruder

We now consider the capabilities of the intruder. We assume that the intruder is able to produce an encrypted message only if he knows the encrypting key, and that he is able to decrypt messages only if he knows the decrypting key. We define a relation formalizing how the intruder can deduce new messages from a set of components he has previously seen. We write $Bs \vdash M$ if the intruder can deduce component M from the set of components Bs. The relation $\vdash$ is defined by the following three rules:

member $M \in Bs \Rightarrow Bs \vdash M$,

encrypt $Bs \vdash M_1 \wedge \ldots \wedge Bs \vdash M_n \wedge Bs \vdash K \Rightarrow Bs \vdash \{M_1, \ldots, M_n\}_K$,

decrypt $Bs \vdash \{M_1, \ldots, M_n\}_K \wedge Bs \vdash \overline{K} \Rightarrow Bs \vdash M_i$, for $i = 1, \ldots, n$.

The rule **member** says that if the intruder has already seen M, then he can certainly produce it. The rule **encrypt** says that the intruder can encrypt messages he can produce with a key he can produce. The rule **decrypt** says that if the intruder can produce an encrypted message, and he can produce the corresponding decrypting key, then he can perform the decryption.

In Section 2.7, we will specify that an intruder can send a message with contents Ms only if he can produce all the parts of Ms, (i.e. every M in Ms) from the messages that have gone before and his initial knowledge. This means that we are assuming that the intruder does not learn anything from outside the protocol, for example by guessing keys, applying cryptographic techniques, replaying messages from this protocol into runs of another protocol to learn information, or "rubber-hose cryptography" [24].

We now state a few properties of the deduction relation $\vdash$.

Lemma 3

1. If the intruder can deduce M from Bs, then he can deduce M from any larger set:

   $$Bs \vdash M \wedge Bs \subseteq Bs' \Rightarrow Bs' \vdash M.$$

2. The intruder can use messages he can deduce in further deductions: if Bs' is finite and $\forall M' \in Bs' \bullet Bs \vdash M'$ and $Bs \cup Bs' \vdash M$ then $Bs \vdash M$.

3. Messages that the intruder can deduce contain only atoms that appear somewhere in the deducing set; this is true whether we include or exclude the encrypting keys:

   $$\begin{aligned} Bs \vdash M \wedge V \in \mathsf{atoms}(M) &\Rightarrow \exists B \in Bs \bullet V \in \mathsf{atoms}(B), \\ Bs \vdash M \wedge V \in Value \wedge V \in \mathsf{subcpts}(M) &\Rightarrow \exists B \in Bs \bullet V \in \mathsf{subcpts}(B). \end{aligned}$$

4. Messages can by built up from their atoms: $\mathsf{atoms}(M) \vdash M$.

2.6.1 The intruder's initial knowledge

We assume that the intruder has some initial knowledge, which we define as follows:

Definition 2.1 (Intruder's initial knowledge) The intruder's initial knowledge is a set IIK of messages such that:

1. The intruder initially knows only atoms[12]:

$$IIK \subseteq \{\textsc{Atom}\, A \mid A \in Value\}.$$

2. The intruder does not initially know the values returned by long-term-secret functions when applied to tuples of honest agents' identities:

$$\forall f \in LTSecrets\,;\, As \in Honest^{\mathsf{arity}(f)} \bullet f(As) \notin IIK.$$

Part (2) coupled with the fact that no protocol messages contain long-term secrets will be enough to prove that the intruder never learns any such secrets (see Section 4.1).

2.7 Traces

We define a trace to be a sequence tr of events that represents a valid history of a system running the protocol. Thus each strand within tr should represent a valid (partial) execution of the protocol. Also, every message received by an honest agent should either:

- have been previously sent by another honest agent, or
- have been created and sent by the intruder, in which case it must be deducible from what has gone before (under the $\vdash$ relation);

note that the first of these alternatives is included in the second (essentially because of the **member** clause of the definition of $\vdash$). Further, values of fresh datatypes really should be fresh.

Recall that the set $Strands$ of strands represents the set of single agent's views of a run; to specify that each agent follows the protocol, we just need to specify that the restriction of the trace to each strand identifier is a strand:

$$\forall \alpha \in StrandId \bullet tr \upharpoonright \alpha \in Strands, \tag{1}$$

where $tr \upharpoonright \alpha$ is the restriction of trace tr to strand α:

$$tr \upharpoonright \alpha \mathrel{\widehat=} \langle Ev \mid Ev \leftarrow tr, \mathsf{strandId}(Ev) = \alpha \rangle,$$

where $\mathsf{strandId}(Ev)$ is the strand identifier of event Ev:

$$\mathsf{strandId}(\mathsf{send}(\alpha.\_, \_, \_, \_)) = \mathsf{strandId}(\mathsf{receive}(\alpha.\_, \_, \_, \_)) = \alpha.$$

We now add the condition that if an honest agent receives a message, then this is a message that the intruder could create from what went earlier.

[12] From now on, we will tend to drop the "$\textsc{Atom}$" when referring to atomic members of IIK.

We define $\mathsf{data}(tr)$ to be the set of message components sent by honest agents during tr:

$$\mathsf{data}(tr) \mathrel{\widehat{=}} \{M \mid \mathsf{send}(\_,\_,\_,Ms) \text{ in } tr \wedge M \text{ in } Ms\}.$$

And define $\mathsf{knowledge\text{-}after}(tr, IIK)$ to be the set of components that the intruder can produce from having seen those messages, starting with initial knowledge IIK; that is, the set of components deducible (under $\vdash$) from $\mathsf{data}(tr) \cup IIK$:

$$\mathsf{knowledge\text{-}after}(tr, IIK) \mathrel{\widehat{=}} \{M \mid \mathsf{data}(tr) \cup IIK \vdash M\}.$$

Note that $\mathsf{knowledge\text{-}after}(tr, IIK)$ is a set of *components*, i.e. atoms and encrypted components. We note in passing that

$$\mathsf{knowledge\text{-}after}(tr \frown \langle \mathsf{receive}(\_,\_,\_,Ms)\rangle, IIK) = \mathsf{knowledge\text{-}after}(tr, IIK),$$

because the intruder does not learn anything new from messages he himself sends.

We then specify that all received messages are indeed producible from what has gone before:

$$\begin{array}{l} tr' \frown \langle \mathsf{receive}(\_,\_,\_,Ms)\rangle \leq tr \Rightarrow \\ \qquad \forall M \text{ in } Ms \bullet M \in \mathsf{knowledge\text{-}after}(tr', IIK). \end{array} \tag{2}$$

Note that this allows the intruder to simply forward messages that have appeared earlier, for if $(\_,\_,\_,Ms)$ in tr' then certainly $\forall M$ in $Ms \bullet M \in \mathsf{knowledge\text{-}after}(tr', IIK)$. However, it also allows the intruder to alter messages, and to introduce completely new messages, as allowed by the $\vdash$ relation.

We now specify that values of fresh datatypes really are fresh: two honest agents will not introduce the same value into protocol runs; and fresh data items are unpredictable so the intruder will not know (or be able to guess) any fresh values before they are introduced (although he may be able to deduce a fresh value from messages containing it).

Definition 2.2 (Freshness) We say that a trace satisfies freshness if:

1. Whenever messages $\alpha.i$ and $\beta.j$ are both sent by honest agents, V is a fresh value which appears in these messages in the place of variables x and y, and these are the first messages in these strands containing x and y; then in fact these are the same message and the same variable:

$$\begin{pmatrix} \mathsf{send}(\alpha.i, A, \_, Ms), \mathsf{send}(\beta.j, B, \_, Ms') \text{ in } tr \\ \wedge\, V \in Fresh \wedge Ms@_i x = V \wedge Ms@_j y = V \\ \wedge\, \mathsf{first}(x) = i \wedge \mathsf{first}(y) = j \end{pmatrix} \Rightarrow \alpha.i = \beta.j \wedge x = y.$$

2. Whenever message $\alpha.i$ is sent by an honest agent, V is a fresh value which appears in this messages in the place of variable x, and this is the first message in this strand containing x; then the intruder does not initially know V:

$$\begin{pmatrix} \mathsf{send}(\alpha.i, A, \_, Ms) \text{ in } tr \\ \wedge\, V \in Fresh \wedge Ms@_i x = V \wedge \mathsf{first}(x) = i \end{pmatrix} \Rightarrow V \notin IIK.$$

Definition 2.3 (Validity) We say that a trace is *valid* if it satisfies Equations (1) and (2), and the definition of freshness.

2.7.1 Notation

If α is a strand identifier appearing in trace tr, then we extend the $\mathsf{principal}$, $\mathsf{partners}$ and roleOf functions in the obvious way:

$$\begin{aligned}\mathsf{principal}(\alpha)(tr) &\mathrel{\widehat{=}} \mathsf{principal}(tr \upharpoonright \alpha),\\ \mathsf{partners}(\alpha)(tr) &\mathrel{\widehat{=}} \mathsf{partners}(tr \upharpoonright \alpha),\\ \mathsf{roleOf}(\alpha)(tr) &\mathrel{\widehat{=}} \mathsf{roleOf}(tr \upharpoonright \alpha),\end{aligned}$$

where the functions appearing on the right hand side of the definitions were those defined in Section 2.5.

We will write $(\alpha @ x)(tr)$ for the value with which variable x is instantiated throughout strand α; this is well defined providing x appears in at least one of the messages of α.

In each of the above definitions, we will drop the tr where the trace in question is obvious from the context.

2.8 Defining secrecy

In this section we formalize the goals of the protocol, and thus define what we mean for a protocol to be secure.

We say that S is a (short-term) secret of strand α in trace tr if S instantiates a secret variable s, and the partner in strand α is an honest agent.

$$\begin{aligned}& S \ \mathsf{secret\text{-}of}\ \alpha(tr) \Leftrightarrow \\ & \quad \exists\, s \in FreeVar \bullet s \in STSecrets \wedge \alpha @ s(tr) = S \\ & \qquad\qquad\qquad\qquad \wedge\ \mathsf{partners}(\alpha)(tr) \subseteq Honest.\end{aligned}$$

In other words, the principal of α believes that S is a secret shared only with the (honest) partners of α. In this case the intruder should not learn the value of S.

We say there is a *breach of strong secrecy* if the intruder learns a value S that an honest agent thought was a secret:

$$\begin{aligned}& \exists\, S \in Value\,;\, \alpha \in StrandId \bullet \\ & \quad S\ \mathsf{secret\text{-}of}\ \alpha(tr) \wedge S \in \mathsf{knowledge\text{-}after}(tr, IIK).\end{aligned}$$

Note that the above definition allows the run α to be incomplete, so the agent in question does not need to have completed the run. This is quite a strong definition of secrecy: some definitions of secrecy only refer to completed runs, on the grounds that if the agent in question does not complete the protocol run, then he is not going to do anything with that secret, and so it doesn't matter if the intruder learns that value. We discuss this further in the conclusions.

Note that we are assuming that if the intruder legitimately takes *any* part in the protocol run then he is allowed to learn the short-term secrets: our result does not deal with protocols whose intention is to establish a shared secret between a proper subset of these agents; in practice this doesn't appear to significantly restrict the class of protocols we can consider.

Consider a trace tr including a breach of strong secrecy concerning value S, variable s; let tr' be the shortest prefix with such a breach, and consider the last communication of tr' (tr' must be non-empty); there are two possibilities:

- the last communication is a send event: then the intruder learns S from this communication, either directly, or indirectly;
- the last communication in tr' is a receive event: then the intruder gets an honest agent to accept as a secret (with some third party) a value that the intruder already knows.

We say that a protocol is *secure* if there is no valid trace that has a breach of strong secrecy.

2.9 Small systems

We now formally define what we mean by a small system. We begin with the intruder's initial knowledge; we take this initial knowledge to contain at least:

- one value of each type that contains short-term secrets, and the corresponding decrypting key where this value is a key;
- all the results of non-secret functions.

Formally:

Definition 2.4 (Intruder's initial knowledge in small systems) An intruder's initial knowledge $SSIIK$ is an intruder's initial knowledge for a small system if:

1. For every type T containing short-term secrets (those types T such that there is some variable $s \in STSecrets$ such that $\mathsf{type}(s) = T$):
$$\exists V \in T \cdot V \in SSIIK \wedge (V \in Key \Rightarrow \overline{V} \in SSIIK).$$
2. $\forall f \in Func \setminus LTSecrets\,;\, As \in Agent^{\mathsf{arity}(f)} \cdot f(As) \in SSIIK.$

This definition defines the minimal knowledge we must give the intruder when model checking small systems, if we are to apply the main result of this paper.

2.9.1 No temporary secrets

We now make a rather unusual assumption about the protocol, namely that protocol runs do not make use of any temporary secrets: if a particular data item is not one that is supposed to be a secret, then an intruder watching the communications is immediately able to learn the value from any execution in which it appears (or the value may be public information, so the intruder might already know it).

Assumption 6 (No temporary secrets) Let $SSIIK$ be an intruder's initial knowledge set for a small system, and let tr be a trace. If a message Ms is sent as a message i, which has a non-secret variable x instantiated with value V, then the intruder will know the value V:

$$\mathsf{send}(\_.i, \_, \_, Ms) \text{ in } tr \wedge x \in \mathsf{FV}(\mathsf{ms}_i) \wedge Ms@_i x = V \wedge x \notin STSecrets \Rightarrow V \in \mathsf{knowledge\text{-}after}(tr, SSIIK).$$

Recall that the inverses of free variables of messages (where defined) are also free variables, so one consequence of the above is that if Ms contains a key K instantiating a variable k whose inverse is non-secret, then the intruder will know the value of $\overline{K}$:

$$\begin{aligned}&\mathsf{send}(\_.i,\_,\_,Ms) \text{ in } tr \wedge k \in \mathsf{FV}(\mathsf{ms}_i) \wedge Ms@_i k = K \wedge \overline{k} \notin STSecrets \Rightarrow \\ &\quad \overline{K} \in \mathsf{knowledge\text{-}after}(tr, SSIIK).\end{aligned}$$

This assumption means that we cannot deal, for example, with a protocol where an encrypted nonce is sent in one message, and the nonce is then returned as plaintext in the following message; such a nonce would be a temporary secret. We know of no such protocol in the literature, but the following—made up—protocol illustrates this:

$$\begin{array}{lll}\text{Message 1.} & \mathsf{init} \rightarrow \mathsf{resp} & : \{\mathsf{init}, k, n\}_{PK(\!|\mathsf{resp}|\!)}, \{\{\mathsf{init}, n\}_{PK(\!|\mathsf{resp}|\!)}\}_{SK(\!|\mathsf{init}|\!)} \\ \text{Message 2.} & \mathsf{resp} \rightarrow \mathsf{init} & : n\ .\end{array}$$

The above protocol also demonstrates why we need Assumption 6. The protocol is supposed to establish a secret session key k. It is easily verified, using model checking, that this goal is met in a small system with a single initiator and a single responder, each of whom can run the protocol once. However, when we consider a larger system, where the responder B can run the protocol twice, the following attack is possible:

$$\begin{array}{lll}\text{Message 1.} & A \rightarrow B & : \{A, K, N\}_{PK(B)}, \{\{A, N\}_{PK(B)}\}_{SK(A)} \\ \text{Message 2.} & B \rightarrow A & : N \\ \text{Message 1}'. & I_A \rightarrow B & : \{A, K', N\}_{PK(B)}, \{\{A, N\}_{PK(B)}\}_{SK(A)} \\ \text{Message 2}'. & B \rightarrow I_A & : N\ .\end{array}$$

The notation I_A represents the intruder I imitating A to either fake or intercept a message. The intruder watches a run, α, between A and B so as to learn a component $\{\{A, N\}_{PK(B)}\}_{SK(A)}$ together with the value of N; it then fakes a message 1 to B using this component, but with a new first component, using the same nonce N but a different key K'; B will then incorrectly accept this new key as being a secret shared with A.

This is a rather unusual assumption; however, it is satisfied by most protocols in the academic literature. It means that we cannot deal with bit commitment protocols, where the bits are temporary secrets. It also means that we cannot consider the effects of key compromise (our model of the intruder is another reason why we cannot consider key compromise).

2.9.2 Small system traces

As explained in the introduction, we define a small system to be a collection of agents running the protocol, where there is a single honest agent taking each role, each running the protocol precisely once, with one another. The intruder's initial knowledge will be as in Definition 2.4. We need to place a restriction upon the values that honest agents introduce into runs; we will then define the traces of the small system.

Let $Role = \{r_1, \ldots, r_p\}$. If an honest agent introduces a value for variable x, the value will be $honval(x)$, where the valuation $honval$, is as follows:

Definition 2.5 (Valuations of honest agents in small systems) A valuation for the honest agents in a small system (with intruder's initial knowledge $SSIIK$) is a valuation *honval* such that:

- There are distinct honest agents $A_1, \ldots, A_p$, such that

$$\forall\, i \in 1\,..\,p \bullet honval(r_i) = A_i.$$

- *honval* maps different fresh variables to different values; and the intruder does not initially know the values assigned to any fresh variables under *honval*:

$$\begin{array}{l} \forall\, x, y \in FreshVar \bullet x \neq y \Rightarrow honval(x) \neq honval(y) \\ \wedge\, \forall\, x \in FreshVar \bullet honval(x) \notin SSIIK. \end{array} \tag{3}$$

The following definition describes the traces of this system:

Definition 2.6 (Small system traces) Let $SSIIK$ be as in Definition 2.4, and let *honval* and $A_1, \ldots, A_p$ be as in Definition 2.5. We define the corresponding *small system traces* to be all traces tr such that:

1. There are strand identifiers $\alpha_1, \ldots, \alpha_p$, such that each strand α_i is a strand of A_i taking role r_i, and each other role r_j is taken by A_j, and these strands account for all the events[13]:

$$\begin{array}{l} \forall\, i \in 1\,..\,p \bullet tr \upharpoonright \alpha_i \neq \langle\rangle \Rightarrow \mathsf{roleOf}(\alpha_i)(tr) = r_i \wedge \mathsf{principal}(\alpha_i)(tr) = A_i \\ \qquad\qquad\qquad \wedge\, \forall\, j \in 1\,..\,p \bullet \alpha_i @ r_j \text{ defined} \Rightarrow \alpha_i @ r_j = A_j \\ \wedge\, \forall\, ev \text{ in } tr \bullet \mathsf{strandId}(ev) \in \{\alpha_1, \ldots, \alpha_p\}. \end{array}$$

2. When an honest agent introduces a value for a variable x, the value is $honval(x)$:

$$\begin{array}{l} \forall\, x \in FreeVar\,;\, i \in 1\,..\,p \bullet \\ \qquad \mathsf{send}(\alpha_i.n, A_i, \_, Ms) \text{ in } tr \wedge n = \mathsf{first}(x, r_i) \Rightarrow \alpha_i @ x = honval(x). \end{array}$$

3. tr is a valid trace with respect to $SSIIK$.

Note, in particular, that the honest agents are not able to run the protocol with the intruder in a small system. Note also that one consequence of part 2, above, is that if a variable x is not uniquely originating, i.e. it is initially shared by two roles, then all the agents introduce the same value $honval(x)$ for that variable.

Finally, note that we assume that the honest agents will accept any values for free variables—other than agent identities—that they do not themselves introduce. This means that we do not allow the agents to perform any tests on such values, for example testing for replays, or checking the timeliness of timestamps.

[13] The "$tr \upharpoonright \alpha_i \neq \langle\rangle$" is to ensure that $\mathsf{roleOf}(\alpha_i)(tr)$ and $\mathsf{principal}(\alpha_i)(tr)$ are well defined.

2.10 Statement of main result

We say that a protocol is *small system secure* if there is some choice of *honval* and $SSIIK$ such that there is no corresponding small system trace that contains a breach of strong secrecy. Note that for a particular choice of *honval* and $SSIIK$, this can be tested by model checking the particular system described above.

Our main result is that subject to Assumptions 1–6, above, if a protocol is small system secure, then it is secure.

Note that the result tells us that we only have to analyse a single small system (i.e. a single choice of *honval* and $SSIIK$); if we discover an attack on the small system, then the protocol is clearly insecure; if we do not discover an attack, then the result tells us that there is no attack on any system (and in particular, there is no attack on any other small system).

3 Examples

In this section we consider two examples, and show how they relate to our assumptions. In some cases we have to adapt the protocols slightly so as to meet the explicitness conditions described in the previous section—in each case, these adaptations prevent attacks that would otherwise be possible! We consider having to adapt the protocols slightly to be a price well worth paying: the extra information that has to be included to ensure explicitness adds very little to the complexity of the protocol.

We also use these examples to illustrate some of the considerations one has to make in order to apply the results. The examples were analysed using the CSP [7, 23] model checker FDR [6]; see [13, 17] for a description of this technique. The CSP descriptions of the system were produced using the CSP security protocol compiler Casper [15]; the Casper scripts are available on the World Wide Web[14].

3.1 The Yahalom Protocol

Consider, again, the following version of the Yahalom Protocol:

$$\begin{array}{lrcl}
\text{Message 1.} & \mathsf{init} \to \mathsf{resp} & : & \mathsf{init}, na \\
\text{Message 2.} & \mathsf{resp} \to \mathsf{serv} & : & \{\mathsf{init}, na, nb\}_{shared(\mathsf{resp},\mathsf{serv})} \\
\text{Message 3.} & \mathsf{serv} \to \mathsf{init} & : & \{\mathsf{resp}, kab, na, nb\}_{shared(\mathsf{init},\mathsf{serv})} \\
\text{Message 4.} & \mathsf{serv} \to \mathsf{resp} & : & \{\mathsf{init}, kab\}_{shared(\mathsf{resp},\mathsf{serv})} \\
\text{Message 5.} & \mathsf{init} \to \mathsf{resp} & : & \{\mathsf{init}, \mathsf{resp}, \mathsf{serv}, nb\}_{kab}.
\end{array}$$

This protocol clearly satisfies the two explicitness conditions. The protocol establishes a secret shared key kab; in addition, the nonce nb remains secret; we therefore take the set $STSecrets$ of short-term secrets to be $\{kab, nb\}$. Note that na is immediately available to the intruder from message 1, as required by Assumption 6 (no temporary secrets). Assumption 6 further requires that

[14] `http://www.mcs.le.ac.uk/~glowe/Security/Casper/index.html`

the values of the role variables init, resp and serv are available to the intruder immediately they appear; the easiest way to ensure this is to include all the agents' identities in the intruder's initial knowledge.

We can use a model checker to verify that there are no attacks upon a small system with a single initiator, responder and server, each of whom runs the protocol once with one another; the initial knowledge for the intruder must contain the honest agents' identities, as explained above, along with a single session key and a single nonce, to meet the requirements of Definition 2.4. We may then use the result of this paper to deduce that there is no attack upon any larger system.

We have had to adapt this protocol slightly to fit the conditions of this paper:

- In the original protocol, the component of message 4 was sent to the initiator before being forwarded to the responder; our model does not allow such forwarding. It seems intuitively clear to me that transforming a protocol to redirect such components to their final recipient does not alter the security of the protocol: if there is an attack on the original, then there is a similar attack on the transformed protocol where the relevant components get sent direct, and vice versa. In [8] we introduced the idea of a *safe simplifying transformation*: a transformation such that if the original protocol is insecure, then so is the transformed version. We believe that redirecting transformations are safe in this sense, although the results of [8] aren't adequate to show this; this is the subject of current research.
- In the original protocol, the fields init, resp and serv were not included in the final component; we have added them so as to satisfy Assumption 4 (identities inferable). This means that our results don't immediately tell us anything about the original protocol. However, we would argue that in many cases omitting identities allows attacks, and so it is better to include such identities: such adaptations will often strengthen the protocol.

3.2 The Davis Swick Protocol

Consider the following protocol from [5]:

Message 1. $\mathsf{init} \rightarrow \mathsf{serv}\ :\ \{\mathsf{resp}, text\}_{shared(\mathsf{init},\mathsf{serv})}$

Message 2. $\mathsf{serv} \rightarrow \mathsf{resp}\ :\ \{text, \mathsf{init}\}_{shared(\mathsf{resp},\mathsf{serv})}$.

(We have again adapted the protocol to send an encrypted component directly.) The intention of the protocol is to transfer some secret text $text$ from initiator to responder, using the server as a key translator; we therefore take $STSecrets = \{text\}$. This protocol satisfies the explicitness conditions (it wouldn't if the order of the fields in either encrypted component were reversed). We can use a model checker to verify that there are no attacks upon a small system with a single initiator, responder and server, each of whom runs the protocol once with one another, and where the intruder's initial knowledge contains a single text (to satisfy Definition 2.4) and all the agents' identities (to satisfy Assumption 6). We may then use the result of this paper to deduce that there is no attack upon any larger system.

4 Some lemmas

In this section we prove a number of lemmas that we will use in the proof of the main result. In Section 4.1 we prove that the intruder can never learn the values of any long-term secrets, and in Section 4.2 we prove that if a protocol is small system secure, then all secrets in the protocol are encrypted with the inverses of secrets. In Section 4.3 we prove two uniformity results, namely that if the intruder never learns a particular value, then that value appears instantiating at most one variable, and all messages containing that value have the same agents taking each of the roles. In Section 4.4 we prove a technical results concerning the deduction relation. Then in Section 4.5 we consider a certain property of protocol message templates, and prove that if there is a certain form of attack upon the protocol, then the template of the last message in the attack satisfies this property, and that this is enough for the intruder to be able to produce other instantiations of this template, which will be enough to produce small system attacks.

4.1 On long-term secrets

We now prove that the intruder can never learn the value of a long-term secret.

Lemma 4 (Intruder learns no long-term secrets) If tr is a valid trace of a system running the protocol, then

$$\forall f \in LTSecrets\,; Args \in Honest^{\mathsf{arity}(f)} \bullet \\ f(Args) \notin \mathsf{knowledge\text{-}after}(tr, IIK).$$

Proof: We use part 3 of Lemma 3 to prove a slightly stronger result, that the intruder cannot learn any message containing any value that is formed by applying a long-term-secret function to a tuple of honest agents' identities:

$$\forall f \in LTSecrets\,; Args \in Honest^{\mathsf{arity}(f)}\,; M \in \mathsf{knowledge\text{-}after}(tr, IIK) \bullet \\ f(Args) \notin \mathsf{subcpts}(M).$$

From part 2 of Definition 2.1 the intruder did not initially know any such value:

$$\forall f \in LTSecrets\,; Args \in Honest^{\mathsf{arity}(f)}\,; M \in IIK \bullet \\ f(Args) \notin \mathsf{subcpts}(M).$$

From Assumption 3, no such value appears in any protocol message:

$$\forall f \in LTSecrets\,; Args \in Honest^{\mathsf{arity}(f)}\,; M \in \mathsf{data}(tr) \bullet \\ f(Args) \notin \mathsf{subcpts}(M).$$

Hence we may deduce the result from part 3 of Lemma 3. □

4.2 On encrypting keys and small-system secure protocols

In this section we show that if a protocol is small system secure than all secrets are encrypted with inverses of secrets; this means that an intruder cannot learn such secrets by simply observing a run of the protocol, and decrypting using non-secret information.

We begin by defining a relation that identifies which keys are used to encrypt a particular variable in a component template.

Definition 4.1 (Encrypting keys) We write

$$ks \ \mathsf{encrypts}\ x \ \mathsf{in}\ m$$

if an occurrence of variable x in component template m is encrypted by the keys in ks, and no others:

$$ks \ \mathsf{encrypts}\ x \ \mathsf{in}\ \textsc{FreeVar}\, y \Leftrightarrow x = y \wedge ks = \{\},$$

$$ks \ \mathsf{encrypts}\ x \ \mathsf{in}\ f(\!| as |\!) \Leftrightarrow false,$$

$$ks \ \mathsf{encrypts}\ x \ \mathsf{in}\ \{m_1, \ldots, m_n\}_k \Leftrightarrow$$
$$k \in ks \wedge (\exists i \in 1 \mathinner{..} n \bullet ks \ \mathsf{encrypts}\ x \ \mathsf{in}\ m_i \vee ks \setminus \{k\} \ \mathsf{encrypts}\ x \ \mathsf{in}\ m_i).$$

We extend the notation to deal with message templates:

$$ks \ \mathsf{encrypts}\ x \ \mathsf{in}\ ms \ \Leftrightarrow \ \exists\, m \ \mathsf{in}\ ms \bullet ks \ \mathsf{encrypts}\ x \ \mathsf{in}\ m.$$

For example, $\{k\}$ $\mathsf{encrypts}$ x in $\{x, y\}_k$ because $\{k\} \setminus \{k\} = \{\}$ $\mathsf{encrypts}$ x in x; and $\{k\}$ $\mathsf{encrypts}$ x in $\{\{x\}_k\}_k$ because $\{k\}$ $\mathsf{encrypts}$ x in $\{x\}_k$.

The following lemma is an immediate consequence of the above definition:

Lemma 5 If x is encrypted by ks in m:

$$ks \ \mathsf{encrypts}\ x \ \mathsf{in}\ m$$

then an instantiation of x can be deduced from the corresponding instantiations of m and the inverses of the members of ks:

$$\forall\, val \in Val \bullet \{\mathsf{inst}_{val}(m)\} \cup \{\mathsf{inst}_{val}(\overline{k}) \mid k \in ks\} \vdash \mathsf{inst}_{val}(x).$$

Lemma 6 (Secrets are encrypted with inverses of secrets in small-system secure protocols) Suppose a protocol is small-system secure; then all secrets are encrypted with the inverse of a secret: more formally, if s is a short-term-secret variable, and ks is a set of keys encrypting s in the template of message n:

$$ks \ \mathsf{encrypts}\ s \ \mathsf{in}\ \mathsf{ms}_n$$

then there is some key k in ks such that k is the inverse of a (short- or long-term) secret:

$$\exists\, k \in ks \bullet \overline{k} \in Secrets.$$

Proof: We prove the contra-positive. Suppose not all secrets are encrypted with inverses of secrets. Then there is a secret variable s, which is encrypted by a set of keys ks in message ms_n, say,

$$ks \ \mathsf{encrypts}\ s \in \mathsf{ms}_n$$

and none of the inverses of the keys in ks are secrets.

Then the intruder can learn the value that honest agents assign to s by simply watching a normal run. More precisely, we produce an attack on a small system as follows.

Let $Role = \{r_1, \ldots, r_p\}$. Following Definition 2.6, let $honval$ be a valuation respecting freshness (equation (3)), and such that

$$\forall\, i \in 1 \mathinner{..} p \bullet honval(r_i) = A_i,$$

with the A_i honest; and let $SSIIK$ be a corresponding intruder's initial knowledge for a small system. Let $\alpha_1, \ldots, \alpha_p$ be distinct strand identifiers. Consider the trace tr formed by interleaving the strands:

$$\mathsf{strandOf}(r_i, \alpha_i, honval, n) \qquad \text{for } i = 1, \ldots, p,$$

where the intruder simply forwards messages from one strand to another, so that if message l is a send event in α_i and a receive event in α_j (i.e. $\mathsf{sender}_l = r_i$ and $\mathsf{receiver}_l = r_j$) then $\alpha_i.l$ precedes $\alpha_j.l$.

Clearly this trace is valid because every receive event has the same contents as the previous send event. It satisfies the other conditions for being a small system trace, by construction.

We claim that the intruder learns the value $honval(s)$ that all the agents believe is secret. Note that in both instances of message n, $honval(s)$ is encrypted with

$$\{\mathsf{inst}_{honval}(k) \mid k \in ks\}.$$

We show that the intruder will know the inverses of each of these keys; let $k \in ks$, and perform a case analysis; either k is a free variable, or it is a function application:

- If $k \in FreeVar$, we have $\overline{k} \notin STSecrets$ by assumption, and so by Assumption 6 (no temporary secrets), the intruder will know $honval(\overline{k}) = \overline{\mathsf{inst}_{honval}(k)}$.
- If k is of the form $f(\!|a_1, \ldots, a_n|\!)$, we have $\overline{f} \notin LTSecrets$ by assumption, and so by part 2 of Definition 2.4:

$$\begin{aligned} \overline{f}(honval(a_1), \ldots, honval(a_n)) &= \overline{f(honval(a_1), \ldots, honval(a_n))} \\ &\in SSIIK. \end{aligned}$$

Hence the intruder will know the inverses of all the keys encrypting $honval(s)$, and so can decrypt these encryptions so as to learn the value of $honval(s)$. $\square$

4.3 Two uniformity results

In this section we prove two uniformity results, showing that, under certain circumstances, values instantiate variables in a uniform manner. We show that if the intruder never learns a particular fresh value V, then:

1. there is a variable x such that all occurrences of V instantiate x; that is, V cannot instantiate different variables in different places of a valid trace;
2. all messages containing V have the same agents taking each of the roles.

4.3.1 Uniformity of variable

We begin by showing that if all message components the intruder has seen contain a particular value V only in the place of variable x (if anywhere), and the intruder cannot deduce V itself, then all messages that the intruder can produce contain V only in the place of x (if anywhere).

Lemma 7 Let $Bs \subseteq ProtCpts$ and suppose all occurrences of the value V in Bs are in positions given by variable x:

$$\forall M' \in Bs\,;\, y \in FreeVar \bullet \mathsf{encrypted}(M') \wedge M'@y = V \Rightarrow y = x, \tag{4}$$

and V itself cannot be deduced from Bs:

$$Bs \nvdash V,$$

then all encrypted components that can be deduced from Bs contain V only in positions given by variable x (if anywhere):

$$\begin{array}{l} \forall M \in ProtCpts\,;\, y \in FreeVar \bullet \\ \quad \mathsf{encrypted}(M) \wedge Bs \vdash M \wedge M@y = V \Rightarrow y = x. \end{array}$$

As a corollary, we deduce that if the intruder does not know a particular value V, and all previous occurrences of V were in positions given by variable x, then any message sent by the intruder has V in positions given by x, if anywhere:

Corollary 8 If tr is a valid trace during which the intruder sends message Ms:

$$tr' \frown \langle \mathsf{receive}(\_, \_, \_, Ms) \rangle \leq tr,$$

and the intruder cannot deduce V itself from tr'

$$V \notin \mathsf{knowledge\text{-}after}(tr', IIK) \tag{5}$$

(which implies that all previous occurrences of V were within encrypted components), and all previous occurrences of value V were in positions given by variable x:

$$\forall M' \in \mathsf{data}(tr') \bullet \forall y \in FreeVar \bullet M'@y = V \Rightarrow y = x, \tag{6}$$

then all occurrences of value V in Ms are in positions given by variable x:

$$\forall M \text{ in } Ms \bullet \forall y \in FreeVar \bullet \mathsf{encrypted}(M) \wedge M@y = V \Rightarrow y = x.$$

Proof: Suppose the conditions of the corollary. By the validity of tr:

$$\forall M \text{ in } Ms \bullet IIK \cup \mathsf{data}(tr') \vdash M.$$

From (6),

$$\forall M' \in \mathsf{data}(tr') \bullet \forall y \in FreeVar \bullet \mathsf{encrypted}(M') \wedge M'@y = V \Rightarrow y = x,$$

and trivially (because IIK contains only atoms):

$$\forall M' \in IIK \bullet \forall y \in FreeVar \bullet \mathsf{encrypted}(M') \wedge M'@y = V \Rightarrow y = x.$$

Further, from (5):

$$IIK \cup \mathsf{data}(tr') \nvdash V.$$

Hence $IIK \cup \mathsf{data}(tr')$ satisfies the conditions of Lemma 7 (for Bs), so we may deduce:

$$\forall y \in FreeVar \bullet \mathsf{encrypted}(M) \wedge M@y = V \Rightarrow y = x.$$

□

We now prove a related result for the honest agents. First we show that if an agent introduces a value for a fresh variable, then that value must never have appeared before:

Lemma 9 If tr is a valid trace containing a message in which a fresh value V is introduced by an honest agent for variable x:

$$tr' \frown \langle \mathsf{send}(\_.i, \_, \_, Ms) \rangle \leq tr \wedge Ms@_i x = V \wedge \mathsf{first}(x) = i \wedge V \in Fresh$$

then this value has not appeared previously:

$$\forall M' \in \mathsf{data}(tr') \cdot V \notin \mathsf{atoms}(M').$$

Proof: Assume the conditions of the lemma, and suppose for a contradiction that V did occur earlier. Consider the first message containing V; this is either a send event or a receive event:

Send event Then there is some message $\mathsf{send}(\_.j, \_, \_, Ms')$ in tr' where V is freshly introduced, say for variable y: $Ms'@_j y = V \wedge \mathsf{first}(y) = j$. But this contradicts part 1 of Definition 2.2 freshness).

Receive event So there is some message Ms' with $V \in \mathsf{atoms}(Ms')$, and some trace tr'', such that

$$tr'' \frown \langle \mathsf{receive}(\_, \_, \_, Ms') \rangle \leq tr'.$$

Then, by the validity of the trace,

$$\forall M' \text{ in } Ms' \cdot IIK \cup \mathsf{data}(tr'') \vdash M'.$$

Hence by part 3 of Lemma 3, there is some $M'' \in IIK \cup \mathsf{data}(tr'')$ such that $V \in \mathsf{atoms}(M'')$. If $M'' \in IIK$, then we have a contradiction of part 2 of Definition 2.2 (freshness). If $M'' \in \mathsf{data}(tr'')$ then this contradicts the assumption that message Ms' contained the first occurrence of V.

□

We use the previous lemma to show that if fresh value V has appeared previously, but only in positions given by variable x, then in any new messages sent by honest agents, V appears only in positions given by x, if anywhere:

Lemma 10 If tr is a valid trace during which an honest agent sends message Ms:

$$tr' \frown \langle \mathsf{send}(\alpha.i, \_, \_, Ms) \rangle \leq tr,$$

and V is a fresh value that has occurred previously

$$V \in Fresh \wedge \exists M' \in \mathsf{data}(tr') \cdot V \in \mathsf{atoms}(M'),$$

but only in the position given by fresh variable x,

$$\begin{array}{l} \forall \mathsf{send}(\_.j, \_, \_, Ms') \text{ in } tr' \cdot \forall y \in FreeVar \cdot Ms'@_j y = V \Rightarrow y = x \\ \wedge \\ \forall \mathsf{receive}(\_.j, \_, \_, Ms') \text{ in } tr' \cdot \forall y \in FreeVar \cdot Ms'@_j y = V \Rightarrow y = x, \end{array}$$

then V occurs in Ms only in positions given by x, if anywhere:

$$\forall y \in FreeVar \cdot Ms@_i y = V \Rightarrow y = x.$$

Proof: Assume the conditions of the lemma, and suppose $Ms@_i y = V$. If this is the first occurrence of y in this strand, i.e. $i = \mathsf{first}(y)$, then we have a contradiction of Lemma 9. Otherwise ($i > \mathsf{first}(y)$) there must have been a previous message in strand α containing V in the position given by y; hence $y = x$ by the conditions of the lemma. □

We now put together the previous results to deduce that if the intruder never learns fresh value V then V occurs instantiating at most one variable:

Corollary 11 If tr is a valid trace, during which the intruder never learns a particular fresh value V:

$V \in Fresh \wedge V \notin \mathsf{knowledge\text{-}after}(tr, IIK)$,

then V occurs in the trace instantiating at most one variable:

$\exists\, x \in FreeVar \bullet \forall\, M \in \mathsf{data}(tr) \bullet \forall\, y \in FreeVar \bullet M@y = V \Rightarrow y = x.$

Proof: By induction on the length of tr.

- The result is vacuously true for the empty trace.
- If the trace is of the form $tr' \frown \langle \mathsf{send}(\_, \_, \_, Ms) \rangle$, then by the inductive hypothesis the result is true for tr'; there are two subcases to consider:
  - If V appears in tr' instantiating some variable x, then the result follows from Lemma 10;
  - If V does not appear in tr', then if it appears in Ms, it can do some instantiating only one variable, from the definition of freshness.
- If the trace is of the form $tr' \frown \langle \mathsf{receive}(\_, \_, \_, Ms) \rangle$, then by the inductive hypothesis the result is true for tr'; we again consider two subcases:
  - If V appears in tr' instantiating some variable x, then the result follows from Corollary 8;
  - If V does not appear in tr', then $\forall\, M' \in IIK \cup \mathsf{data}(tr') \bullet V \notin \mathsf{atoms}(M')$; hence V cannot appear in M, from part 3 of Lemma 3.

□

4.3.2 Uniformity of agents

Throughout this section, we take $Role = \{r_1, \ldots, r_p\}$. We prove that if the intruder never learns a fresh value V, then in all messages containing V, the roles are consistently instantiated: there are agents $A_1, \ldots, A_p$ such that all such messages have role r_i taken by A_i, for each $i \in 1 \mathrel{..} p$. We omit the proofs, because they are very similar to proofs in the previous section.

Lemma 12 Let $Bs \subseteq ProtCpts$, $A_1, \ldots, A_p \in Agent$, and suppose all occurrences of the value V in Bs occur in components with each role r_i taken by the corresponding agent A_i:

$\forall\, M \in Bs \bullet \mathsf{encrypted}(M) \wedge V \in \mathsf{atoms}(M) \Rightarrow \forall\, i \in 1 \mathrel{..} p \bullet M@r_i = A_i,$

and V itself cannot be deduced from Bs:

$Bs \nvdash V,$

then all encrypted components that can be deduced from Bs and that contain V have the same agents in each role:

$$\begin{array}{l} \forall M \in ProtCpts \bullet \\ \quad \mathsf{encrypted}(M) \wedge Bs \vdash M \wedge V \in \mathsf{atoms}(M) \Rightarrow \forall i \in 1 \mathinner{..} p \bullet M@r_i = A_i. \end{array}$$

Corollary 13 If tr is a valid trace during which the intruder sends message Ms:

$$tr' \frown \langle \mathsf{receive}(\_, \_, \_, Ms) \rangle \leq tr,$$

and the intruder cannot deduce V itself from tr'

$$V \notin \mathsf{knowledge\text{-}after}(tr', IIK),$$

(which implies that all previous occurrences of V were within encrypted components), and all previous occurrences of value V were in messages where each role r_i is taken by agent A_i:

$$\forall M' \in \mathsf{data}(tr') \bullet V \in \mathsf{atoms}(M') \Rightarrow \forall i \in 1 \mathinner{..} p \bullet M'@r_i = A_i,$$

then all occurrences of value V in Ms are in messages where each role r_i is taken by agent A_i:

$$\forall M \text{ in } Ms \bullet V \in \mathsf{atoms}(M) \Rightarrow \forall i \in 1 \mathinner{..} p \bullet M@r_i = A_i.$$

Lemma 14 If tr is a valid trace during which an honest agent sends message Ms:

$$tr' \frown \langle \mathsf{send}(\alpha.k, \_, \_, Ms) \rangle \leq tr,$$

and V is a fresh value that has occurred previously

$$V \in Fresh \wedge \exists M' \in \mathsf{data}(tr') \bullet V \in \mathsf{atoms}(M'),$$

but only in messages where each role r_i is taken by agent A_i,

$$\begin{array}{l} \forall \mathsf{send}(\_.j, \_, \_, Ms') \text{ in } tr' \bullet \\ \quad V \in \mathsf{atoms}(Ms') \Rightarrow \forall i \in 1 \mathinner{..} p \bullet Ms'@_j r_i = A_i \\ \wedge \\ \forall \mathsf{receive}(\_.j, \_, \_, Ms') \text{ in } tr' \bullet \\ \quad V \in \mathsf{atoms}(Ms') \Rightarrow \forall i \in 1 \mathinner{..} p \bullet Ms'@_j r_i = A_i, \end{array}$$

then V occurs in Ms only in messages where each role r_i is taken by agent A_i:

$$V \in \mathsf{atoms}(Ms) \Rightarrow \forall i \in 1 \mathinner{..} p \bullet Ms@_k r_i = A_i.$$

Corollary 15 (Uniformity of agents) If tr is a valid trace during which the intruder never learns a particular fresh value V:

$$V \in Fresh \wedge V \notin \mathsf{knowledge\text{-}after}(tr, IIK),$$

then V occurs in messages with the roles uniquely instantiated:

$$\begin{array}{l} \exists A_1, \ldots, A_p \in Agent \bullet \forall M \in \mathsf{data}(tr) \bullet \\ \quad V \in \mathsf{atoms}(M) \Rightarrow \forall i \in 1 \mathinner{..} p \bullet M@r_i = A_i. \end{array}$$

4.4 Reachable subcomponents

We define a function $\mathsf{reachable}$ such that $\mathsf{reachable}(Bs)$ is the set of messages that can be obtained from Bs by performing decryptions with deducible keys; formally, we define $\mathsf{reachable}(Bs)$ to be the smallest set satisfying the following two equations:

$$M \in Bs \;\Rightarrow\; M \in \mathsf{reachable}(Bs)$$

$$\{Ms\}_K \in \mathsf{reachable}(Bs) \wedge Bs \vdash \overline{K} \;\Rightarrow\; \forall\, M \text{ in } Ms \bullet M \in \mathsf{reachable}(Bs).$$

$\mathsf{reachable}(Bs)$ defines the components that the intruder is able to replay in different contexts. Note that if $M \in \mathsf{reachable}(Bs)$ then $Bs \vdash M$.

If the intruder is unable to produce a message without replaying any of its encrypted components, then the intruder must know all the atoms.

Lemma 16 If $Bs \vdash M$ and

$\forall\, M' \in \mathsf{subcpts}(M) \bullet \mathsf{encrypted}(M') \Rightarrow M' \notin \mathsf{reachable}(Bs),$

then

$\forall\, A \in \mathsf{atoms}(M) \bullet Bs \vdash A.$

4.5 A property of message templates

In this section we study a particular property of component templates, which we denote $\mathsf{producible}$. In Section 4.5.1 we show that, under certain circumstances (essentially that the intruder cannot learn any secrets from earlier messages), if an intruder can produce an instantiation of a component template relating to honest agents, then that template satisfies $\mathsf{producible}$. Then in Section 4.5.2 we show that if a template satisfies $\mathsf{producible}$, then the intruder can produce instantiations of that message from fairly limited knowledge (essentially the knowledge the intruder would have from having observed a single run of the protocol up to that point).

In Section 5 we will use these results to prove that if there is an attack on the protocol where the intruder gets an honest agent to accept a secret in a message that the intruder sends, where the intruder knows the secret but the honest agent believes that the secret is shared with other honest agents, then the components of that message satisfy $\mathsf{producible}$; we will then show that the intruder could launch a similar attack in a small system.

Let ss be a set of short-term-secret variables:

$ss \subseteq STSecret.$

Later, ss will represent the set of short-term variables whose instantiations (in some message) are known to the intruder. We define the predicate $\mathsf{producible}_{ss}(m)$ as follows:

$\mathsf{producible}_{ss}(\textsc{FreeVar}\, a) \mathrel{\widehat{=}} a \in STSecrets \Rightarrow a \in ss,$

$\mathsf{producible}_{ss}(f(\!|as|\!)) \mathrel{\widehat{=}} f \notin LTSecrets,$

$\mathsf{producible}_{ss}(\{ms\}_k) \mathrel{\widehat{=}}$
$\quad \forall\, a \in \mathsf{FV}(\{ms\}_k) \bullet a \notin ss$
$\quad \vee\ \mathsf{producible}_{ss}(k) \wedge \overline{k} \notin Secrets \wedge \forall\, m \text{ in } ms \bullet \mathsf{producible}_{ss}(m)$
$\quad \vee\ \forall\, a \in \mathsf{atoms}(\{ms\}_k) \bullet \mathsf{producible}_{ss}(a).$

Essentially, $\mathsf{producible}_{ss}(m)$ is true if the intruder can produce the instantiation of m under a valuation val from the values given to the variables in ss by val, all non-secret information, and the instantiation of m under val', where val and val' agree on all variables other than those in ss. The two atomic cases basically say that the message is either non-secret or is in ss. The first disjunct of the encryption case says that m contains no variables from ss, so the instantiation under val is the same as under val'. The second disjunct says that the intruder can undo the encryption, produce the instantiations of the subcomponents, and then re-encrypt. The third disjunct says that all the atoms can be produced, so the instantiation can be built up from its atoms.

The truth of $\mathsf{producible}_{ss}(m)$ is independent of members of ss that do not appear in m:

Lemma 17 If ss and ss' are two sets of variables that contain the same free variables of m:

$$ss \cap \mathsf{FV}(m) \;=\; ss' \cap \mathsf{FV}(m),$$

then m satisfies $\mathsf{producible}$ for ss if and only if it does so for ss':

$$\mathsf{producible}_{ss}(m) \;\Leftrightarrow\; \mathsf{producible}_{ss'}(m).$$

4.5.1 Producible messages satisfy $\mathsf{producible}$

We now prove a result which says that, under certain circumstances, if an intruder can produce an instantiation of a component template, then that template satisfies $\mathsf{producible}$.

Lemma 18 Let

$$Bs \subseteq ProtMsgs \cup Value$$

be a set of maximal protocol components (i.e. components that appear in messages without enclosing encryptions) and atomic values. Suppose that for all (necessarily encrypted) members of Bs relating to honest agents, the values of any secrets cannot be deduced from Bs:

$$\begin{aligned}&\forall M \in Bs\,;\, s \in STSecrets\,;\, S \in Value \bullet\\ &\qquad (\forall r \in Role \bullet M@r \in Honest) \wedge M@s = S \Rightarrow Bs \nvdash S.\end{aligned} \tag{7}$$

(Recall that the @ notation is defined for the inverses of keys appearing in the component, so the above implies that no secret decrypting keys can be deduced.) Further, suppose that no long-term secrets of honest agents can be deduced:

$$\forall f \in LTSecrets\,;\, Args \in Honest^{\mathsf{arity}(f)} \bullet Bs \nvdash f(Args). \tag{8}$$

Let val be a valuation such that:

$$\forall r \in Role \bullet val(r) \in Honest. \tag{9}$$

Let m be a template component, and let ss be the set of short-term-secret variables that are atoms of m, and whose values, under val, can be deduced from Bs:

$$ss \;\widehat{=}\; \{s \in STSecrets \mid s \in \mathsf{FV}(m) \wedge Bs \vdash val(s)\}.$$

Then if the instantiation of m can be deduced from Bs, then m satisfies the property $\mathsf{producible}_{ss}$:

$$Bs \vdash \mathsf{inst}_{val}(m) \;\Rightarrow\; \mathsf{producible}_{ss}(m).$$

The idea is that Bs will represent the knowledge of the intruder in a state where he has not yet deduced the values assigned to secrets in any messages between honest agents (equation 7); the lemma then says that if the intruder can fake a message from one honest agent to another, then the corresponding message template must satisfy $\mathsf{producible}_{ss}$, where ss is the set of short-term secrets in the message whose values the intruder knows.

Proof: Assume the conditions of the lemma, and suppose $Bs \vdash \mathsf{inst}_{val}(m)$.

Firstly, if m is of the form $\textsc{FreeVar}\,a$, then from the definition of ss, if $a \in STSecret$ then $a \in ss$; hence $\mathsf{producible}_{ss}(m)$.

Next, if m is a function application of the form $f(\!|as|\!)$, then $f \notin LTSecrets$, because otherwise we would have a contradiction of (8), using the fact that all the arguments would be honest agents; hence $\mathsf{producible}_{ss}(m)$.

For non-atomic cases, let $m = \{ms\}_k$, and proceed by induction over the derivation of $Bs \vdash \mathsf{inst}_{val}(m)$:

member Case $\mathsf{inst}_{val}(m) \in Bs$; then from (9),

$$\forall r \in Role \bullet \mathsf{inst}_{val}(m)@r = val(r) \in Honest,$$

so from (7):

$$\forall s \in STSecrets \bullet s \in \mathsf{FV}(m) \Rightarrow Bs \nvdash val(s).$$

Hence $ss = \{\}$. Thus the first disjunct of the definition of $\mathsf{producible}_{ss}(m)$ holds.

encrypt Case $Bs \vdash \mathsf{inst}_{val}(k)$ and $\forall m'$ in $ms \bullet Bs \vdash \mathsf{inst}_{val}(m')$. Then by the inductive hypothesis, and making use of Lemma 17, $\mathsf{producible}_{ss}(k)$ and $\forall m'$ in $ms \bullet \mathsf{producible}_{ss}(m')$. If $\overline{k} \notin Secrets$ then we have the second disjunct in the definition of $\mathsf{producible}_{ss}$. So suppose $\overline{k} \in Secrets$; we show that no instantiation of an encrypted subcomponent of ms is reachable:

$$\forall m'' \in \mathsf{subcpts}(ms) \bullet \mathsf{encrypted}(m'') \Rightarrow \mathsf{inst}_{val}(m'') \notin \mathsf{reachable}(Bs). \quad (10)$$

Suppose, for a contradiction, that for some such m'', $\mathsf{inst}_{val}(m'')$ is in $\mathsf{reachable}(Bs)$; then from the definition of $\mathsf{reachable}$, it must appear as a subcomponent of some message of Bs, say M, encrypted only with keys that the intruder can deduce:

$$M \in Bs \wedge \mathsf{inst}_{val}(m'') \in \mathsf{subcpts}(M).$$

(There may be several members of Bs containing $\mathsf{inst}_{val}(m'')$, but let M be one from which $\mathsf{inst}_{val}(m'')$ can be obtained, as in the definition of $\mathsf{reachable}$ in Section 4.4.) From Assumption 5 (encrypted components are textually distinct), the instantiation of m'' must appear within an instantiation of $\{ms\}_k$, say under instantiation function val':

$$\mathsf{inst}_{val}(m'') \in \mathsf{subcpts}(\mathsf{inst}_{val'}(\{ms\}_k)) \wedge \mathsf{inst}_{val'}(\{ms\}_k) \in \mathsf{subcpts}(M),$$

such that (by the definition of M) the intruder can deduce $val'(\overline{k})$:

$$Bs \vdash val'(\overline{k}).$$

But $\overline{k} \in \mathit{Secrets}$, and $\forall r \in \mathit{Role} \bullet M@r = val(r) \in \mathit{Honest}$, which contradicts equation (7) in the statement of the lemma. Hence we deduce (10).

Hence, recalling that $\forall m' \text{ in } ms \bullet Bs \vdash \mathsf{inst}_{val}(m')$, we have from Lemma 16:

$$\forall m' \text{ in } ms \bullet \forall a \in \mathsf{atoms}(m') \bullet Bs \vdash \mathsf{inst}_{val}(a).$$

Thus, for all atoms a of ms, as in the atomic cases above, and using Lemma 17, we have $\mathsf{producible}_{ss}(a)$. We have already proved $\mathsf{producible}_{ss}(k)$. Thus

$$\forall a \in \mathsf{atoms}(\{ms\}_k) \bullet \mathsf{producible}_{ss}(a),$$

which is the third disjunct of $\mathsf{producible}_{ss}(\{ms\}_k)$.

decrypt Case $Bs \vdash \{Ms\}_K$, for some Ms, K with $\mathsf{inst}_{val}(m)$ in Ms and $Bs \vdash \overline{K}$. Then $\{Ms\}_K$ can be written in the form $\mathsf{inst}_{val'}(\{ms'\}_{k'})$ for some template $\{ms'\}_{k'}$ such that m in ms', and for some valuation val' such that

$$\forall a \in \mathsf{FV}(m) \bullet val(a) = val'(a).$$

Define ss' by:

$$ss' \mathrel{\widehat{=}} \{s \in \mathit{STSecrets} \mid s \in \mathsf{FV}(\{ms'\}_{k'}) \wedge Bs \vdash val'(s)\}.$$

Then by the inductive hypothesis:

$$\mathsf{producible}_{ss'}(\{ms'\}_{k'}).$$

Expanding the definition of $\mathsf{producible}_{ss'}$:

$$\begin{array}{l} \forall a \in \mathsf{FV}(\{ms'\}_{k'}) \bullet a \notin ss' \\ \vee\ \mathsf{producible}_{ss'}(k') \wedge \overline{k'} \notin \mathit{Secrets} \wedge \forall m' \text{ in } ms' \bullet \mathsf{producible}_{ss'}(m') \\ \vee\ \forall a \in \mathsf{atoms}(\{ms'\}_{k'}) \bullet \mathsf{producible}_{ss'}(a). \end{array}$$

In particular (because $m = \{ms\}_k$ in ms'):

$$\begin{array}{l} \forall a \in \mathsf{FV}(\{ms\}_k) \bullet a \notin ss' \\ \vee\ \mathsf{producible}_{ss'}(\{ms\}_k) \\ \vee\ \forall a \in \mathsf{atoms}(\{ms\}_k) \bullet \mathsf{producible}_{ss'}(a), \end{array}$$

which is the definition of $\mathsf{producible}_{ss'}(m)$. Finally, we use Lemma 17 to deduce $\mathsf{producible}_{ss}(m)$.

□

4.5.2 Producing instantiations of templates that satisfy producible

We now prove that if a template satisfies producible, then the intruder can produce instantiations of that message from fairly limited knowledge.

Lemma 19 Let

$$ss \mathrel{\widehat{=}} \{s_1, \ldots, s_l\} \subseteq \mathit{STSecrets}$$

be a set of short-term-secret variables; let $S_1, \ldots, S_l$ be values such that each s_i and S_i have the same type, and if s_i is a key, let $\overline{S_i}$ be the inverse of S_i; let val be a valuation; and let m be a message template such that

$$\mathsf{producible}_{ss}(m).$$

Define[15]

$$\begin{aligned} val' &\mathrel{\widehat{=}} val \oplus \{s_i \mapsto S_i \mid i \in 1\,..\,l\} \oplus \{\overline{s_i} \mapsto \overline{S_i} \mid i \in 1\,..\,l \wedge s_i \in Key\}, \\ LHS(m) &\mathrel{\widehat{=}} \{\mathsf{inst}_{val}(m)\} \\ &\quad \cup \{S_i \mid i \in 1\,..\,l\} \\ &\quad \cup \{\overline{S_i} \mid i \in 1\,..\,l \wedge s_i \in Key\} \\ &\quad \cup \{val(a) \mid a \in \mathsf{FV}(m) \wedge a \notin STSecrets\} \\ &\quad \cup \{f(As) \mid f \in Func \setminus LTSecrets \wedge As \in Agent^{\mathsf{arity}(f)}\}. \end{aligned}$$

Then

$$LHS(m) \vdash \mathsf{inst}_{val'}(m).$$

$LHS(m)$ represents a set of values that the intruder could know if he has seen $\mathsf{inst}_{val}(m)$. The idea is that if the intruder sees $\mathsf{inst}_{val}(m)$, he can learn

$$\{val(a) \mid a \in \mathsf{FV}(m) \wedge a \notin Secrets\}$$

(because of assumption 6, no temporary secrets), and can invent new values S_i and, where appropriate, the corresponding $\overline{S_i}$, and will know

$$\{f(As) \mid f \in Func \setminus LTSecrets \wedge As \in Agent^{\mathsf{arity}(f)})\},$$

because of part 2 of Definition 2.4; the lemma says that the intruder can then produce m instantiated under the new valuation val'.

Proof: We begin by showing

$$a \in \mathsf{atoms}(m) \wedge \mathsf{producible}_{ss}(a) \;\Rightarrow\; LHS(m) \vdash inst_{val'}(a), \tag{11}$$

by a case analysis:

- Case $a \in STSecrets$; then $a \in ss$ from the definition of $\mathsf{producible}_{ss}(a)$, say $a = s_i$; so $\mathsf{inst}_{val'}(a) = S_i \in LHS(m)$, and so $LHS(m) \vdash \mathsf{inst}_{val'}(a)$.
- Case $a \in FreeVar \setminus STSecrets$ and $\overline{a} \in ss$; then $\overline{a} = s_i$, say, so $\mathsf{inst}_{val'}(a) = val'(a) = \overline{S_i} \in LHS(m)$, and so $LHS(m) \vdash \mathsf{inst}_{val'}(a)$.
- Case $a \in FreeVar \setminus STSecrets$, and $\overline{a}$ undefined or $\overline{a} \notin ss$; then $\mathsf{inst}_{val'}(a) = val'(a) = val(a) \in LHS(m)$; and so $LHS(m) \vdash \mathsf{inst}_{val'}(a)$.
- Case a is a function application of the form $f(\!|as|\!)$; then $f \notin LTSecrets$, because of the definition of $\mathsf{producible}_{ss}(a)$; so $\mathsf{inst}_{val'}(a) \in LHS(m)$; and so $LHS(m) \vdash \mathsf{inst}_{val'}(a)$.

We now prove the main result by induction on the structure of m. If m is an atom, then the result follows immediately from (11). Otherwise, let $m = \{ms\}_k$, and perform a case analysis on the definition of $\mathsf{producible}_{ss}(m)$:

- Case $\forall a \in \mathsf{FV}(m) \cdot a \notin ss$; then $\mathsf{inst}_{val'}(m) = \mathsf{inst}_{val}(m)$ (since val and val' agree on all the free variables of m); and so $LHS(m) \vdash \mathsf{inst}_{val'}(m)$.
- Case $\mathsf{producible}_{ss}(k) \wedge \overline{k} \notin Secrets \wedge \forall m' \;\mathsf{in}\; ms \cdot \mathsf{producible}_{ss}(m')$. From (11):

  $$LHS(m) \vdash \mathsf{inst}_{val'}(k). \tag{12}$$

  Let m' in ms. By the inductive hypothesis:

  $$LHS(m') \vdash \mathsf{inst}_{val'}(m'). \tag{13}$$

[15] The notation $f \oplus g$ represents function f overwritten by function g.

We now show

$$\forall m'' \in LHS(m') \cdot LHS(m) \vdash m'', \tag{14}$$

by a case analysis:

- To show $LHS(m) \vdash \mathsf{inst}_{val}(m')$, note that $\overline{k} \notin Secrets$ and so $\mathsf{inst}_{val}(\overline{k}) \in LHS(m)$ (whether $\overline{k}$ is a variable or a function application); also $\mathsf{inst}_{val}(\{ms\}_k) \in LHS(m)$, by definition; so from the **decrypt** clause of the definition of $\vdash$, $LHS(m) \vdash \mathsf{inst}_{val}(m')$.
- All members of $LHS(m')$ other than $\mathsf{inst}_{val}(m')$ are also members of $LHS(m)$, so the result follows from the **member** clause of the definition of $\vdash$.

Hence applying part 2 of Lemma 3 to equations (13) and (14),

$$LHS(m) \vdash \mathsf{inst}_{val'}(m'),$$

for all m' in ms. Hence, from the **encrypt** clause of the definition of $\vdash$, using equation 12:

$$LHS(m) \vdash \mathsf{inst}_{val'}(m).$$

- Case $\forall a \in \mathsf{atoms}(m) \cdot \mathsf{producible}_{ss}(a)$. From (11), $\forall a \in \mathsf{atoms}(m) \cdot LHS(m) \vdash \mathsf{inst}_{val'}(a)$. Hence by part 4 of Lemma 3, $LHS(m) \vdash \mathsf{inst}_{val'}(m)$.

$\square$

5 Proof of the main theorem

In this section we prove the main theorem of the paper, that if a protocol is small system secure, then it secure.

We require one more piece of terminology. If $Bs \vdash M$ then we say that the length of the deduction of M from Bs is the minimum number of applications of the defining rules of $\vdash$ that are needed to demonstrate this deduction. Let Ss be the set of secrets that may be deduced (in a given situation). Then we say that $S \in Ss$ is an *initial secret* if no other secret in Ss has a shorter deduction. That is, S might be the first secret that the intruder learns. An immediate consequence is that no other secret in Ss is used in the deduction of S.

Theorem 1 Suppose the protocol in question is small system secure. Then the protocol is secure.

Proof: Suppose the protocol is small-system secure, and suppose for a contradiction that there is a breach of secrecy in trace tr, where tr is valid with respect to intruder's knowledge IIK; more specifically, let the breach of security concern strand α, where the (honest) principal of α believes that the value of secret variable $s \in STSecrets$ should be secret from the intruder:

$$\mathsf{partners}(\alpha) \subseteq Honest, \tag{15}$$

but the intruder learns the value S assigned to s:

$$\alpha@s = S \wedge S \in \mathsf{knowledge\text{-}after}(tr, IIK).$$

Further, without loss of generality suppose there is no breach of security on any proper prefix of tr, and that S is an initial secret. From the definition of breaches of security, tr must be non-empty, so must be of one of the following forms:

$$tr' \frown \langle \mathsf{send}(\beta.n, A, B, Ms) \rangle \quad \text{or} \quad tr' \frown \langle \mathsf{receive}(\beta.n, A, B, Ms) \rangle,$$

where there is no breach of security on tr', that is, the breach of security occurs in event $\beta.n$ itself. We consider the two possibilities below.

Send event Suppose the last event is a send event, so tr is of the form:

$$tr = tr' \frown \langle \mathsf{send}(\beta.n, A, B, Ms) \rangle,$$

where $A \in Honest$.

We must have that the intruder did not know the value of S before $\beta.n$. For suppose otherwise; then by the definition of secrecy, and the fact that there is no breach of security on tr', we must have that A introduces S in message $\beta.n$ as a new secret; but this contradicts the assumption that all secrets are fresh (Assumption 2). Hence the intruder learns S from $\beta.n$.

From Corollary 11, all occurrences of S in tr' are instantiating variable s; and from Lemma 10, all occurrences of S in Ms also instantiate s. Further, from Corollary 15, all instances of S in tr' are in components that agree upon the agents taking each role; and from Lemma 14, all components in Ms containing S also agree upon the agents taking each role.

In order for the intruder to learn S, it must appear in some message, say $\gamma.i$ ($\gamma.i$ may or may not be the same message as $\beta.n$):

$$(\gamma.i, \_, \_, Ms') \in tr \wedge S \in \mathsf{atoms}(Ms')$$

where the corresponding variable s is encrypted with some set of keys ks in the corresponding message template ms_i:

$$ks \ \mathsf{encrypts}\ s \ \mathsf{in}\ \mathsf{ms}_i$$

and the intruder learns all the inverses of the values with which the elements of ks are instantiated:

$$\forall k \in ks \bullet \overline{\gamma @ k} \in \mathsf{knowledge\text{-}after}(tr, IIK).$$

(where we are extending the @ notation for function applications with $\gamma @ f(\!| a_1, \ldots, a_n |\!) = f(\gamma @ a_1, \ldots, \gamma @ a_n)$).

We show that one of the partners in run γ is dishonest; for suppose, for a contradiction, that $\mathsf{partners}(\gamma) \subseteq Honest$. By Lemma 6 (secrets are encrypted with inverses of secrets in small-system secure protocols), the inverse of at least one of these encrypting keys must be secret, say the inverse of k_s:

$$\overline{k_s} \in Secrets.$$

By Lemma 4 (intruder learns no long-term secrets) we can deduce:

$$\overline{k_s} \in STSecrets.$$

Hence, the learning of $\overline{k_s}$ itself constitutes a breach of security; further, this breach precedes the learning of S, which contradicts the assumption that S is an initial secret.

We deduce that one of the partners in strand γ is dishonest:

$$\exists I \in \mathsf{partners}(\gamma) \bullet I \in \mathit{Dishonest}. \tag{16}$$

Clearly, then, from equation (15), $\alpha \neq \gamma$. Recall that the intruder does not learn the value S of s before the final message of the trace. Hence, from Corollary 15 (uniformity of agents), all (necessarily encrypted) components in strands α and γ that contain S must agree upon the values assigned to each role. But this contradicts equations (15) and (16).

Receive event Suppose, now, that the last message is a receive event, so tr is of the form:

$$tr' \frown \langle \mathsf{receive}(\beta.n, B, A, Ms) \rangle.$$

Suppose Ms instantiates ms_n via valuation val:

$$Ms = \mathsf{inst}_{val}(\mathsf{ms}_n),$$

and let A's and B's roles in β be r_A and r_B:

$$\mathsf{sender}_n = r_B \wedge \mathsf{receiver}_n = r_A \wedge val(r_A) = A \wedge val(r_B) = B.$$

For this to constitute the first breach of security, it must be the case that all agents in β are honest:

$$\forall r \in \mathit{Role} \bullet \beta@r = val(r) \in \mathit{Honest},$$

and A receives at least one secret in $\beta.n$ that the intruder previously knew. Let ss be the set of secret variables whose values in $\beta.n$ are known by the intruder:

$$ss \mathrel{\widehat{=}} \{s \in \mathit{STSecret} \mid s \in \mathsf{FV}(\mathsf{ms}_n) \wedge val(s) \in \mathsf{knowledge\text{-}after}(tr', \mathit{IIK})\}.$$

So $ss \neq \{\}$. Let ss be of the form:

$$ss = \{s_i \mid i \in 1 \,..\, l\}.$$

By the assumption that message $\beta.n$ represents the first failure of secrecy, none of the elements of ss can appear in messages of β before message n:

$$\forall s_i \in ss \bullet \mathsf{first}(s_i, r_A) = n.$$

We now show:

$$\forall m \text{ in } \mathsf{ms}_n \bullet \mathit{prop}_{ss}(m).$$

For each m in ms_n, we use Lemma 18, with $\mathit{IIK} \cup \mathsf{data}(tr')$ in the place of Bs, and with $ss' \mathrel{\widehat{=}} ss \cap \mathsf{FV}(m)$ in the place of ss, to show $\mathit{prop}_{ss'}(m)$. We check the conditions of the lemma as follows:

- $\mathit{IIK} \cup \mathsf{data}(tr') \subseteq \mathit{ProtMsgs} \cup \mathit{Value}$, by definition.
- By assumption, there have been no breaches of security on tr', and so:

$$\begin{array}{l}\forall M \in \mathit{IIK} \cup \mathsf{data}(tr') \,;\, s \in \mathit{STSecrets} \,;\, S \in \mathit{Value} \bullet \\ \quad (\forall r \in \mathit{Role} \bullet M@r \in \mathit{Honest}) \wedge M@s = S \Rightarrow \mathit{IIK} \cup \mathsf{data}(tr') \nvdash S.\end{array}$$

- From Lemma 4, the intruder does not learn any long-term secrets:

$$\forall f \in \mathit{LTSecrets} \,;\, \mathit{Args} \in \mathit{Honest}^{\mathsf{arity}(f)} \bullet \mathit{IIK} \cup \mathsf{data}(tr') \nvdash f(\mathit{Args}).$$

- $\forall r \in Role \bullet val(r) \in Honest$, as above.
- By construction:

$$ss' = \{s \in STSecrets \mid s \in \mathsf{FV}(m) \wedge IIK \cup \mathsf{data}(tr') \vdash val(s)\}.$$

- By the validity of tr, $IIK \cup \mathsf{data}(tr') \vdash \mathsf{inst}_{val}(m)$.

Hence, using Lemma 18, we deduce $prop_{ss'}(m)$. Thus, from Lemma 17 we have $prop_{ss}(m)$.

We can construct a small system attack as follows. Let $Role = \{r_1, \ldots, r_p\}$. Following Definition 2.6, let $honval$ be a valuation respecting freshness (equation (3)), and such that

$$\forall i \in 1 \mathinner{..} p \bullet honval(r_i) = A_i,$$

with the A_i honest; and let $SSIIK$ be a corresponding intruder's initial knowledge for a small system.

For each $i \in 1 \mathinner{..} l$, let S_i be the element of the intruder's initial knowledge of type $\mathsf{type}(s_i)$, and if S_i is a key let $\overline{S_i}$ be the corresponding inverse (as defined by part 1 of Definition 2.4). Define val' by:

$$val' \mathrel{\widehat{=}} honval \oplus \{s_i \mapsto S_i \mid i \in 1 \mathinner{..} l\} \oplus \{\overline{s_i} \mapsto \overline{S_i} \mid i \in 1 \mathinner{..} l \wedge s_i \in Key\}.$$

Let $\alpha_1, \ldots, \alpha_p$ be distinct strand identifiers, and without loss of generality, let A's role in the original trace (r_A) be r_p. Consider the strands:

$$\mathsf{strandOf}(r_i, \alpha_1, honval, n) \qquad \text{for } i \in 1 \mathinner{..} p-1$$

and

$$\mathsf{strandOf}(r_p, \alpha_p, val', n),$$

and consider the trace formed by interleaving these strands, with the intruder replaying messages between the strands; for example, if message k is a send event in α_i and a receive event in α_j (i.e. $\mathsf{sender}_k = r_i$ and $\mathsf{receiver}_k = r_j$), then message $\alpha_i.k$ precedes $\alpha_j.k$.

This trace satisfies most of the conditions for being a small system trace, by construction. We must show that the trace is valid. For messages not involving strand α_p, each message is a direct replay. Also, recall that $\forall s_i \in ss \bullet \mathsf{first}(s_i, r_p) = n$, and so $honval$ and val' agree on all variables in messages $1 \mathinner{..} n-1$ of strand α_p, so

$$\forall i \in 1 \mathinner{..} n-1 \bullet \mathsf{inst}_{honval}(\mathsf{ms}_i) = \mathsf{inst}_{val'}(\mathsf{ms}_i)$$

and so these messages are also direct replays. It remains to show that the intruder can produce the final message, numbered n. This must be in strand α_p, because $\mathsf{receiver}_n = r_A = r_p$, so the message is of the form:

$$\mathsf{receive}(\alpha_p.n, val'(\mathsf{sender}_n), val'(\mathsf{receiver}_n), \mathsf{inst}_{val'}(\mathsf{ms}_n)).$$

Let IK be the set of data from which the intruder must produce this message, i.e.:

$$IK \mathrel{\widehat{=}} SSIIK \cup \{\mathsf{inst}_{honval}(m) \mid m \text{ in } \mathsf{ms}_i,\ i \in 1 \mathinner{..} n\}.$$

We must show that

$$\forall m \text{ in } \mathsf{ms}_n \bullet IK \vdash \mathsf{inst}_{val'}(m).$$

Following Lemma 19, for each m in ms_n, we define:

$$\begin{aligned} LHS(m) \mathrel{\widehat{=}} & \{\mathsf{inst}_{honval}(m)\} \\ & \cup \{S_i \mid i \in 1\,..\,l\} \\ & \cup \{\overline{S_i} \mid i \in 1\,..\,l \wedge s_i \in Key\} \\ & \cup \{honval(a) \mid a \in \mathsf{FV}(m) \wedge a \notin STSecrets\} \\ & \cup \{f(As) \mid f \in Func \setminus LTSecrets \wedge As \in Agent^{\mathsf{arity}(f)}\}. \end{aligned}$$

This satisfies the conditions of Lemma 19, and so we deduce:

$$LHS(m) \vdash \mathsf{inst}_{val'}(m).$$

From part 2 of Lemma 3, in order to deduce $IK \vdash \mathsf{inst}_{val'}(m)$, it will be enough to prove

$$\forall M' \in LHS(m) \bullet IK \vdash M'.$$

We consider the different sorts of elements of $LHS(m)$ in turn:

- $\mathsf{inst}_{honval}(m)$ is in IK, so $IK \vdash \mathsf{inst}_{honval}(m)$;
- for $i \in 1\,..\,l$, S_i is in $SSIIK$ by construction, and hence in IK, and similarly for the inverse $\overline{S_i}$, where defined;
- $honval(a)$ for $a \in \mathsf{FV}(m)$ and $a \notin STSecrets$ is deducible from IK by Assumption 6 (no temporary secrets);
- $f(As)$ for $f \in Func \setminus LTSecrets$ and $As \in Agent^{\mathsf{arity}(f)}$ is in $SSIIK$ by part 2 of Definition 2.4.

Hence the intruder can produce the final message, namely message $\alpha_p.n$, and so the trace constructed above is valid.

Finally, we note that the trace is an attack because the intruder knows all the values that the agent $A_p = val'(\mathsf{receiver}_n)$ gives to the secret variables of ss in strand α_p. $\square$

6 Conclusions

In this paper we have proved that if a protocol satisfies a number of conditions, and there is no attack upon an appropriate small system, then there is no attack upon an arbitrary system. Most of the conditions on protocols are easily checked; where they do not hold, it is easy to adapt the protocol so that they do. We have also given a recipe that explains how to create the appropriate small system for model checking.

In this final section, we discuss some of the limitations of the result of this paper, and give pointers to future extensions.

6.1 Limitations and prospects

The two explicitness principles clearly limit the scope of these results. In most cases it is easy to adapt a protocol so that it satisfies these principles, by explicitly adding identities so as to satisfy Assumption 4 (identities inferable), or by rearranging fields in a component so as to satisfy Assumption 5 (encrypted components are textually distinct); in many cases, such changes produce a better protocol. However, there are cases where this technique of adding identities cannot be used; for example, the EKE protocol [2] is designed to work in an environment where the intruder is able to guess weak secrets, such as passwords; the inclusion of predictable quantities such as identities within such protocols would often allow the intruder to verify his guess.

Our result refers to a rather strong definition of a breach of secrecy: if the intruder learns a value that is intended to be a secret, it represents a breach of secrecy, regardless of whether or not the protocol run is completed. This is appropriate if the secret is some sort of secret user data. However, it is perhaps less appropriate if the secret is a fresh value such as a key that the protocol is trying to establish, because the agents involved will presumably not do anything with the key if the run is not completed. The reason that we use the stronger definition of secrecy is that it seems more appropriate for performing proofs by induction: the weaker definition does not induct easily.

Clearly, if we can show that a protocol is secure according to the definition of this paper, then it is secure according to the weaker definition that only considers complete runs. However, if a protocol is secure according to the weaker definition, but not the stronger, then we will be unable to apply these results. The only well known protocol that we are aware of with this property is (ironically) the adapted Needham Schroeder Public Key Protocol [12][16], the first message of which is:

$$\text{Message 1.}\quad \mathsf{init} \rightarrow \mathsf{resp} \ : \ \{\mathsf{init}, na\}_{PK(\mathsf{resp})} \,,$$

where $PK(\mathsf{resp})$ is the public key of the responder, and na is a nonce, which is intended to remain secret. Clearly, the initiator is not authenticated within the first message; thus there is nothing to prevent an intruder faking this first message, using a nonce he knows; this would be an attack under the strong secrecy definition; however, the attack cannot be extended to a full run, so this is not an attack under the weaker definition. This protocol is unusual in having a secret that is introduced in one message, but which is authenticated only in a later message.

We would like to remove some of the simplifying assumptions, so as to be able to deal with more protocols. Protocols can take many varied forms, and so being able to reason about all protocols requires a rather detailed model. As an analogy, the early versions of the CSP protocol compiler Casper [15] did not deal with all protocol features; the features not dealt with initially are roughly the same as those features we have not dealt with in the current paper; Casper has since been extended to deal with such features.

At the moment we are unable to deal with protocols where one agent treats a data item as the result of a function application but another treats it as a variable; or where an agent performs some test upon data it receives, refusing to accept certain values. We conjecture that our main result will still hold

[16]Another protocol with this property is the ISO bilateral key exchange with public key protocol [9] which is based upon the Needham Schroeder protocol.

in these cases; in order to prove this fact, we will need to extend our model appropriately.

We also assumed that no long-term secrets were included within protocol messages. This allowed us to easily prove that the intruder never learnt any such secrets. We conjecture that if we relax this assumption, we will be able to prove that if the intruder never learns any long-term secrets in a small system, then all long-term secrets are encrypted with the inverses of secrets (as in Section 4.2); and we will then be able to extend the main result to prove that the intruder won't learn any long-term secrets in larger systems.

We would like to extend our results to deal with more cryptographic functions, such as hash functions and bit-wise exclusive-or. I believe that hash functions will be easy to deal with, because they share many properties with standard encryption; probably the hardest challenge will be extending the $prop$ property appropriately. However, bit-wise exclusive-or might prove harder, because this operator has many important algebraic properties which we would have to model and reason about.

Our model does not deal with protocols where one agent cannot decrypt a message that it receives, but simply forwards it to another agent. As noted in Section 3, we believe that redirecting this message, so that it is sent direct, does not lose any attacks, but this remains to be proven.

6.2 Towards authentication

The result in this paper has only dealt with secrecy properties. We intend to extend these results to deal with various flavours of authentication. Note, though, that our definition of a breach of secrecy does include many authentication-type attacks. For example, consider an attack where an intruder impersonates an agent A to get another agent B to accept a key the intruder knows; this is both a failure of authentication, because B wrongly thinks he is talking to A, and a failure of secrecy, because the intruder knows the key that B thinks is a secret shared only with A.

We believe that the results of this paper should prove invaluable towards that end: in the proof we established a lot of general results (such as the uniformity results) that are useful for proving properties of protocols; further, proving secrecy is a good step towards proving authentication, because most authentication protocols rely upon the secrecy of some data item to prove authentication.

There are many different forms of authentication, and our result does not extend immediately to all of these. For example, a common authentication requirement, called *agreement* in [14], is that when two agents complete a run with one another, they should agree upon a data item, such as a key. Consider the following version of the Otway Rees protocol (adapted slightly to meet the assumptions of this paper):

$$\begin{array}{lrl}
\text{Message } 1a. & \mathsf{init} \to \mathsf{resp} & : na \\
\text{Message } 1b. & \mathsf{init} \to \mathsf{serv} & : \{\mathsf{init}, na, \mathsf{resp}\}_{shared(\mathsf{init},\mathsf{serv})} \\
\text{Message } 2. & \mathsf{resp} \to \mathsf{serv} & : na, nb, \{na, \mathsf{init}, \mathsf{resp}\}_{shared(\mathsf{resp},\mathsf{serv})} \\
\text{Message } 3. & \mathsf{serv} \to \mathsf{resp} & : na, \{\mathsf{init}, nb, kab\}_{shared(\mathsf{resp},\mathsf{serv})} \\
\text{Message } 4. & \mathsf{serv} \to \mathsf{init} & : \{na, \mathsf{resp}, kab\}_{shared(\mathsf{init},\mathsf{serv})} .
\end{array}$$

This protocol satisfies agreement when one considers only the small system from Section 2.9. However, if one considers a slightly larger system, where the server can run the protocol twice, then the following attack is possible, where two agents A and B end up with different keys (this attack is essentially the attack of Thayer et al. from [26]):

$$\begin{array}{lrl}
\text{Message } 1a. & A \to B & : Na \\
\text{Message } 1b. & A \to S & : \{A, Na, B\}_{shared(A,S)} \\
\text{Message } 2. & B \to S & : Na, Nb, \{Na, A, B\}_{shared(B,S)} \\
\text{Message } 3. & S \to B & : Na, \{A, Nb, Kab\}_{shared(B,S)} \\
\text{Message } 4. & S \to I_A & : \{Na, B, Kab\}_{shared(A,S)} \\
\text{Message } 1b'. & I_A \to S & : \{A, Na, B\}_{shared(A,S)} \\
\text{Message } 2'. & I_B \to S & : Na, Nb, \{Na, A, B\}_{shared(B,S)} \\
\text{Message } 3'. & S \to I_B & : Na, \{A, Nb, Kab'\}_{shared(B,S)} \\
\text{Message } 4'. & S \to A & : \{Na, B, Kab'\}_{shared(A,S)}
\end{array}$$

Hence the results of this paper do not extend to agreement specifications.

Similarly, our results do not extend to the so-called *injective* agreement properties of [14], where each run of one agent should correspond to a unique run of their partner. Many protocols allow attacks where an agent A can end up in a state where he believes he has completed two runs of the protocol with B, but where B believes he has completed only a single run with A. It is clear that in order to find such attacks, one has to consider a system where A can run the protocol twice.

Techniques for analysing security protocols have advanced significantly in the last four years. Before then, protocol design was something of a black art, and many incorrect protocols were suggested, and their flaws went unnoticed for years. We now understand much better how we should avoid such flaws, for example, through adherence to the explicitness principles. Further, we now have powerful techniques for analysing protocols, and for proving them correct.

I believe that we are now getting to the point where it is possible to design and verify a protocol in a matter of a few hours.

Acknowledgements

I would like to thank the anonymous referees for useful comments and suggestions for improving the presentation of this paper. This work was supported by grants from the UK Engineering and Physical Sciences Research Council, the UK Defence Evaluation and Research Agency, and the US Office of Naval Research.

References

[1] Martín Abadi and Roger Needham. Prudent engineering practice for cryptographic protocols. *IEEE Transactions on Software Engineering*, 22(1):6–15, 1996. Also in Proceedings of the 1994 IEEE Symposium on Security and Privacy, and Digital Equipment Corporation Systems Research Center, Research Report 125.

[2] S. M. Bellovin and M. Merritt. Encrypted key exchange: Password-based protocols secure against dictionary attacks. In *Proceedings of the 1992 IEEE Computer Security Conference on Research in Security and Privacy*, pages 72–84, 1992.

[3] Michael Burrows, Martín Abadi, and Roger Needham. A logic of authentication. *Proceedings of the Royal Society of London A*, 426:233–271, 1989. A preliminary version appeared as Digital Equipment Corporation Systems Research Center report No. 39, 1989.

[4] Z. Dang and R. A. Kemmerer. Using the ASTRAL model checker for cryptographic protocol analysis. In *Proceedings of the DIMACS Workshop on Design and Formal Verification of Security Protocols*, 1997. Available via URL: `http://dimacs.rutgers.edu/Workshops/Security/program2/program.html`.

[5] D. Davis and R. Swick. Network security via private-key certificates. *Operating Systems Review*, pages 64–67, 1990.

[6] Formal Systems (Europe) Ltd. *Failures-Divergence Refinement—FDR 2 User Manual*, 1997. Available via URL `http://www.formal.demon.co.uk/FDR2.html`.

[7] C. A. R. Hoare. *Communicating Sequential Processes.* Prentice Hall, 1985.

[8] Mei Lin Hui and Gavin Lowe. Safe simplifying transformations for security protocols. In *To appear in Proceedings of the 12th IEEE Computer Security Foundations Workshop*, 1999.

[9] ISO/IEC. Information technology—security techniques—entity authentication mechanisms part 3: Entity authentication using a public key algorithm, 1995.

[10] Richard Kemmerer, Catherine Meadows, and Jonathan Millen. Three systems for cryptographic protocol analysis. *Journal of Cryptology*, 7(2), 1994.

[11] T. Kozlowski and S. A. Smolka. Digital signatures with encryption: Fact and fiction (extended abstract). In *Proceedings of the DIMACS Workshop on Design and Formal Verification of Security Protocols*, 1997. Available via URL: `http://dimacs.rutgers.edu/Workshops/Security/program2/program.html`.

[12] Gavin Lowe. An attack on the Needham-Schroeder public-key authentication protocol. *Information Processing Letters*, 56:131–133, 1995.

[13] Gavin Lowe. Breaking and fixing the Needham-Schroeder public-key protocol using FDR. In *Proceedings of TACAS*, volume 1055 of *Lecture Notes in Computer Science*, pages 147–166. Springer Verlag, 1996. Also in *Software—Concepts and Tools*, 17:93–102, 1996.

[14] Gavin Lowe. A hierarchy of authentication specifications. In *Proceedings of 10th IEEE Computer Security Foundations Workshop*, 1997.

[15] Gavin Lowe. Casper: A compiler for the analysis of security protocols. *Journal of Computer Security*, 6:53–84, 1998. World Wide Web home page at URL `http://www.mcs.le.ac.uk/~glowe/Security/Casper/index.html`.

[16] Gavin Lowe. Towards a completeness result for model checking of security protocols. Technical Report 1998/6, Department of Mathematics and Computer Science, University of Leicester, 1998. Available from `http://www.mcs.le.ac.uk/~glowe/Security/Papers/completeness.ps.gz`.

[17] Gavin Lowe and Bill Roscoe. Using CSP to detect errors in the TMN protocol. *IEEE Transactions on Software Engineering*, 23(10):659–669, 1997.

[18] Will Marrero, Edmund Clarke, and Somesh Jha. A model checker for authentication protocols. In *Proceedings of the DIMACS Workshop on Design and Formal Verification of Security Protocols*, 1997. Available via URL: `http://dimacs.rutgers.edu/Workshops/Security/program2/program.html`.

[19] Catherine Meadows. The NRL Protocol Analyzer: An overview. *Journal of Logic Programming*, 26(2):113–131, 1996.

[20] J. K. Millen, S. C. Clark, and S. B. Freedman. The interrogator: Protocol security analysis. *IEEE Transactions on software Engineering*, 13(2), 1987.

[21] John C. Mitchell, Mark Mitchell, and Ulrich Stern. Automated analysis of cryptographic protocols using Murϕ. In *IEEE Symposium on Security and Privacy*, 1997.

[22] Lawrence C. Paulson. The inductive approach to verifying cryptographic protocols. *Journal of Computer Security*, 6:85–128, 1998.

[23] A. W. Roscoe. *The Theory and Practice of Concurrency*. Prentice Hall, 1997.

[24] Bruce Schneier. *Applied Cryptography*. Wiley, 1994.

[25] Secure Electronic Transaction (SET) specification. Available from `http://www.setco.org/download.html`, 1997.

[26] F. Javier Thayer Fábrega, Jonathan C. Herzog, and Joshua D. Guttman. Honest ideals on strand spaces. In *11th IEEE Computer Security Foundations Workshop*, 1998.

[27] F. Javier Thayer Fábrega, Jonathan C. Herzog, and Joshua D. Guttman. Strand spaces: Why is a security protocol correct? In *1998 IEEE Symposium on Security and Privacy*, 1998.

A Index of notation

Notation	Description	Section
$@$	Infix operator; $M@x$ is value with which x is instantiated in message M	2.4
$@_i$	Infix operator; $Ms@_ix$ is value with which x is instantiated in Ms, when interpreted as a message i	2.4
$\vdash$	Relation defining how intruder can produce new components	2.6
$\overline{K}$	Inverse of key K	2.1
$\overline{k}$	Inverse of variable k	2.2
$\overline{f}$	Function that returns values that are the inverses of values returned by f	2.1
$Agent$	Set of agents' identities	2.1
arity	Function returning number of arguments of function	2.1
ATOM	Datatype constructor for components	2.4
atoms	Atoms of component template, including encrypting keys	2.2
	Atoms of component, including encrypting keys	2.4
$AtomTemp$	Set of templates for atomic components	2.2
Cpt	Set of message components	2.4
$CptTemp$	Set of component templates	2.2
data(tr)	Set of components sent by honest agents during tr	2.7
$Dishonest$	Set of dishonest agents' identities	2.1
ENCRYPT	Datatype constructor for component templates	2.2
	Datatype constructor for components	2.4
encrypted	Predicate on component templates, returning true for encryptions	2.2
	Predicate on components, returning true for encryptions	2.4
encrypts ... in	ks encrypts x in m is true if an occurrence of variable x in template m is encrypted with set of keys ks	4.2

$\mathsf{first}(x)$	Message number of first message of protocol containing variable x	2.3
$\mathsf{first}(x, r)$	Message number of first message of protocol seen by role r, containing variable x	2.3
$FnAppTemp$	Set of applications of functions to free variables	2.2
FnAppTemp	Datatype constructor for component templates	2.2
$FreeVar$	Set of free variables	2.2
FreeVar	Datatype constructor for component templates	2.2
$Func$	Set of functions used in the protocol	2.1
FV	Free variables of component template	2.2
$Honest$	Set of honest agents' identities	2.1
IIK	Intruder's initial knowledge	2.6
$\mathsf{inst}_{val}(m)$	Instantiation of template m under valuation val	2.4
$\mathsf{inst}_{r,\alpha,val}(i)$	Message $\alpha.i$, as seen by the agent taking role r, under valuation val	2.5
$IntRole$	Set of roles that the intruder may legitimately adopt	2.2
Key	Set of all keys	2.1
knowledge-after	Function returning set of components that the intruder can produce following trace	2.7
$LTSecrets$	Functions returning long-term secrets of protocol	2.3
ms_i	Message template for message i	2.3
partners	Function returning set of partners of the principal of a strand	2.5
principal	Function returning principal of strand	2.5
$ProtCpts$	Components of protocol messages (possibly appearing within enclosing encryptions)	2.4
$ProtMsgs$	Message components appearing in protocol messages without enclosing encryptions	2.4
$ProtTemp$	Set of protocol templates	2.3
pt	Protocol template for protocol in question	2.3
ranType	Function returning range (or result) type of function	2.1
reachable	Components derivable from set of components via decryptions with deducible keys	4.4
$\mathsf{receiver}_i$	Receiver of message i	2.3
$Role$	Set of free variables representing agents' roles	2.2
roleOf	Role of principal in strand	2.5
	Role of principal in strand within trace	2.7

$Secrets$	All atomic values representing secrets	2.3
sender_i	Sender of message i	2.3
$StrandId$	Set of all strand identifiers	2.5
$\mathsf{strandId}$	Function returning strand identifier of event	2.7
$\mathsf{strandOf}$	$\mathsf{strandOf}(r, \alpha, val, n)$ is strand relating to role r, strand identifier α, under instantiation val, with messages $1, \ldots, n$	2.5
$STSecrets$	Short-term-secrets of protocol	2.3
$\mathsf{subcpts}$	Subcomponents of component template, message template	2.2
	Subcomponents of component, message	2.4
type	Type of value	2.1
	Type of variable	2.2
Val	Set of all valuation functions	2.4