Challenges for decidable epistemic logics from security protocols

R Ramanujam[1], S P Suresh[2]

[1]The Institute of Mathematical Sciences
Chennai, India
`jam@imsc.res.in`

[2]Chennai Mathematical Institute
Chennai, India
`spsuresh@cmi.ac.in`

ABSTRACT. The notion of knowledge is central to reasoning about security protocols. Formalizing the notion in epistemic logics offers several challenges: the semantics of knowledge is subtle when cryptographic primitives are employed in communications, and the unboundedness inherent in the semantics typically leads to undecidability. We propose a simple epistemic logic which is expressive enough to describe interesting security properties, and for which the protocol verification problem is decidable when there is an upper bound on the number of nonces that may be used in any run.

1 Summary

1.1 Knowledge and communication

A central question in knowledge theory relates to how knowers update their knowledge on receipt of a communication. This is important, since the very purpose of communications is (typically) to create such an update of knowledge in the recipient. However, there is often a lack of concordance between the intended update and that which occurs, leading to interesting situations and much work for knowledge theorists.

Communication protocols studied by computer scientists offer a restricted (but yet interesting) domain for the study of knowledge change. Protocol descriptions define (but also limit) how communications are to be interpreted, and in this sense potential knowledge change can be 'calculated' and messages designed accordingly, as long as participants in the protocol game can be trusted to play by the rules.

The last caveat above is significant, and a good bit of the theory of distributed systems relates to what happens when some of the players do *not* strictly adhere to the protocol diktat. If a sender cannot be trusted, a recipient is uncertain how to interpret a received message. Computer scientists solve this problem in two ways:

- The **global** way: honest participants proceed as if all the world is honest, and protocol rules ensure that desired global coordination is achieved, as long as only a fraction (say, at most one third) are untrustworthy. This is the realm of *fault tolerant* distributed algorithms [13].
- The **local** way: honest participants run a pre-protocol using special facilities (and codes) to create a trusted subnetwork and decide on a protocol to be followed within it. Once this step is completed, they use the protocol agreed on. This is the approach used by *cryptographic protocols*.

One of the earliest applications of knowledge theory in distributed computing was in fault tolerance [11, 8]. BAN logics [3] initiated the latter study, that of epistemic logics for security analysis. This domain poses challenging questions for epistemic logics, because we are called upon to determine knowledge update in the presence of a great deal of uncertainty and distrust. Since this makes the update weak, any attempt to transfer knowledge must take such distrust into account as well. As remarked above, these are pre-protocols, so communications do not have much informational content; but their *form* is quite critical.

1.2 Cryptographic protocols

Security protocols are specifications of communication patterns which are intended to let agents share secrets over a public network. They are required to perform correctly even in the presence of **malicious intruders** who listen to the message exchanges that happen over the network and also manipulate the system (by blocking or forging messages, for instance). Obvious correctness requirements include **secrecy**: an intruder cannot read the contents of a message intended for others; **authenticity**: if B receives a message that appears to be from agent A and intended for B, then A indeed sent the same message intended for B in the recent past.

Mechanisms for ensuring security typically use **encrypted communication**. However, even the use of the most perfect cryptographic tools does not always ensure the desired security goals. (See [1] for an illuminating account.) This situation arises primarily because of **logical flaws** in the design of protocols. It is widely acknowledged that security protocols are hard to analyze, bugs difficult to detect, and hence that it is desirable to look for automatic means by which **attacks** on protocols can be discovered. This means that formal models for reasoning about security protocols should be developed.

Here is a typical message exchange in a security protocol.

1. $A \rightarrow B : \{n\}_{public(B)}$
2. $B \rightarrow A : \{n\}_{public(A)}$

This notation represents the following intention on the part of the designer. A and B have been active in the network in the recent past. A wishes to talk to B, and ensure that she is talking to B. Both have strong faith in *public key encryption* and know each other's public keys. A generates a fresh nonce (a random, previously unused, unguessable number) n and sends it to B encrypted in his public key. When B receives the message, he can indeed decrypt and learn n. He returns it to A now encrypted with her public key.

Given perfect encryption, when A sends the message she *knows* that only somebody who *knows* the private key of B can *know* n. So, when she later receives n encrypted in her own public key, she *knows* that someone who knows the private key of B has accessed n (and it has to be recently, since n was previously unused).

Note the repeated use of the word *know* in the above story. It is clear that one of the main goals of security protocols is the selective transfer of some kind of knowledge to certain select members of a possibly hostile, public network. The network is hostile in that some members might be saboteurs who *actively* try to manipulate the actions of the other members to try to learn new secrets. As we can see, cryptographic methods are used to aid honest participants.

1.3 Difficulties

Consider the implications for knowledge theory from this little story. We list below some immediate questions that arise, most of which can be easily answered, individually. The difficulty is in answering them all, in *one uniform* framework.

- What details of the encryption algorithm should be known to A and B for them to employ the reasoning above?
- If the algorithm employs random elements, should they be able to ascertain that they are indeed chosen randomly (in order for their understanding to be certified as knowledge)?
- What knowledge is needed on A's part to ensure that n is previously unused? Given that n comes from an unbounded set, does the representation used by A matter?
- Once A encrypts n with B's public key, she only has a bit string from which she cannot get any new information. (If she could, so could others.) Indeed, if she does this twice, she would get a different string each time. How can she be said to know that this is the term $\{n\}_B$?
- What B receives is a bit string. How does he know it is encrypted at all, let alone with his key? In effect, this means that he knows all possible messages encrypted with his key.
- Note that B has no way of telling when the n received was generated and by whom. What precisely does B know regarding the communication, that causes him to act?

A simple answer to most of these questions is the Dolev-Yao model [6], used extensively in formal studies of security protocols, which we describe in the next section. In this view, a receiver may expect to receive a term $\{t\}_k$ according to the protocol, but unless she also has k^{-1}, she cannot get t (using only this encrypted term). This is a form of database knowledge: an agent A has only that information explicitly stored in the agent's database. However, ascribing knowledge to agents involves inference as well, and it is this more general form of knowledge that is of interest in epistemic logics.

Note that we are using a variety of forms of knowledge, each of which has been extensively studied by knowledge theorists. We have *propositional knowledge*: for instance, that $\{\{x\}_k\}_{k^{-1}} = x$; *process knowledge*: that of efficacy of encryption; *algorithmic knowledge*: that of how to extract messages from codes; *implicit knowledge*: that of how B responds to a message; *explicit knowledge*: that of n in A's database; and so on. It is worth noting that the difference between *propositional knowledge* and *sentential knowledge* emphasized by Parikh [15] is also crucially relevant when we talk of knowing encoded text.

1.4 Decidability issues

A central aim of formal methods in security theory is to find algorithmic solutions to the verification problem: do all runs of a given security protocol Pr satisfy a given security property ϕ ? When the answer is no, the counterexample is termed an *attack* on the protocol, and the need for automatic methods arises from the fact that finding attacks can be quite complicated, whereas the cost of any potential attack is very high.

Since the space of all runs of a security protocol typically constitute an infinite state system, even simple reachability properties are undecidable, and hence the project of automatic verification of security protocols is doomed to failure, unless some restrictions are imposed. Typically this is in the form of bounded verification, whereby we assume that the number of concurrent multisessions possible at any time is bounded, and syntactic conditions ensure that the term space of communications is also bounded.

When the security property ϕ involves epistemic modalities, there is a further complication. Suppose that we place external bounds so that all runs use only a fixed finite set of terms T. Then the semantics defines a finite state system, and the verification problem is decidable. However, with Hintikka-style semantics of knowledge, this also implies that T is common knowledge in the system, which goes against the very basics of security theory. In the example discussed above, we crucially used the fact that a nonce n was freshly generated and hence not known to others.

An alternative is that we check ϕ only over runs that use T, but let the knowledge modality range over all runs, modelling the fact that agents do not know T. However, this is easily seen to lead to undecidability as well.

An interesting complication arises due to the fact that we are talking of decidability of the verification problem and not that of the *satisfiability* problem for the logic. When the logic is sufficiently simple, deciding the satisfiability of knowledge properties of security protocols is not very different from that of other "interpreted" systems. However, when we are considering the runs of a given protocol, the situation is different: asserting the existence of an equivalent run requires a witness that is admissible according to the given protocol. In knowledge theory, this is typically achieved by forming a product with the given finite state system being checked. In the case of a security protocol, the system is infinite state, and hence a simple product does not suffice. As we will see below, we will let the knowledge formula guide us to an abstraction of the infinite state system to one that is finite state and then proceed to verify it in the latter.

1.5 This paper

In this paper, we look for a minimal logic of knowledge that addresses some of the semantic issues discussed above, and for which the verification problem is decidable. The main idea is to limit expressiveness so that knowledge of data and encryption is described by an underlying inference system that operates at the level of terms used in messages, and propositional epistemic connectives are defined on top of this system.

The logic we study is a standard Hintikka style propositional logic of knowledge, where the atomic propositions have specific structure related to security protocols.

1.6 The proposal

The syntax of the logic is as given below:

$$\mathscr{L} ::= A\ \mathit{has}\ x \mid \mathit{sent}(A,B,x) \mid \mathit{received}(A,B,x) \mid \neg\alpha \mid \alpha \vee \beta \mid \mathbf{G}\alpha \mid \mathbf{H}\alpha \mid \mathbf{K}_A\alpha$$

$\mathbf{G}\alpha$ asserts that α holds *always* in the future. Its dual $\mathbf{F}\alpha$ says that α holds *sometime* in the future. Similarly $\mathbf{H}\alpha$ and $\mathbf{P}\alpha$ refer to all the time points and some time point, respectively, in the past. $\mathbf{L}_A\alpha$ is the dual of $\mathbf{K}_A\alpha$, as usual.

The basic propositions require some explanation: the x that figures in the syntax stands for **secret nonces or keys** exchanged in an execution of a protocol. *A has x* asserts that A has the secret x in her database. $\mathit{sent}(A,B,x)$ specifies that a communication event happened with A sending a term containing x intended for B. $\mathit{received}(A,B,x)$ makes a similar assertion about A receiving a term purportedly from B.

The logic is admittedly rather weak. Note the absence of next-time or previous-time modalities, so we are not describing system transitions here. Moreover information accumulation is monotone, as we will see below. Critically, there are no encrypted terms in the syntax of formulas, and hence we cannot describe cryptographic protocols. All this is in the spirit of an abstract specification logic in which we only wish to specify security requirements, not describe mechanisms for implementing them.

One important reason for studying such a minimal logic is to address the semantic difficulties discussed earlier. The main idea is to limit the expressiveness of the logic as follows:

- The model uses an explicit primitive for talking about what is and what is not decipherable by agents, but this is not referred to in the logic.
- An indistinguishability relation is defined, which in effect makes pieces of messages that can only have originated with the intruder/adversary indistinguishable from those generated by honest principals. Knowledge is defined in terms of this relation.
- Thus knowledge quantifies over intruder capabilities, and describes properties invariant on an equivalence class of messages (compatible with a given protocol).
- Formulas, in themselves, describe only what principals know or do not know about who has access to which secret. Protocols use cryptographic mechanisms to achieve this, and the verification problem determines whether, under the perfect encryption assumption, the specifications are met.

Even in this limited specification language, we can easily specify many desirable properties of protocols. For instance, here is a simple version of secrecy, which says that in any run where A sends B (distinct from I) a secret m, I cannot get hold of m.

$$\mathbf{G}[\mathit{sent}(A, B, m) \supset \neg(I\ \mathit{has}\ m)]$$

An even stronger version states that A in fact knows this (and can therefore base her further actions on this knowledge, if she chooses to).

$$\mathbf{G}[\mathit{sent}(A, B, m) \supset \mathbf{K}_A \neg(I\ \mathit{has}\ m)]$$

Authentication is simply stated as:

$$\mathbf{G}[\mathit{received}(A, B, m) \supset \mathbf{P}\mathit{sent}(B, A, m)]$$

The main theorem of the paper asserts that the protocol verification problem of the logic is elementarily decidable, as long as we consider protocol runs with a fixed upper bound on the number of concurrent multi-sessions. Despite this bound, the inexact nature of agents' knowledge forces us to consider an infinite set of runs as possible, and hence the decision question is nontrivial.

1.7 BAN logic

It is important to highlight the similarities and differences of our approach with that of BAN logic [3]. BAN logic is a highly abstract logic that is intended to be used for assertional reasoning about protocols, much in the style of Hoare logics. It is a propositional modal logic with formulas like *A believes* α, *A sees* α, *A sent* α, *fresh* n, *A controls* α etc. More critically, the logic comes with a deduction system with plausible rules like the following:

$$\frac{A \textit{ sees } \alpha \qquad A \textit{ believes } (\textit{fresh } \alpha)}{A \textit{ believes } \alpha}$$

$$\frac{A \textit{ believes } (C \textit{ controls } \alpha) \qquad A \textit{ believes } (C \textit{ sent } \alpha)}{A \textit{ believes } \alpha}$$

There are many rules but the above two are crucial, as they pertain to *information transfer*. The key to reasoning in this logic is the process of *idealisation*, where message exchanges are themselves *propositionalized*. For example, a communication $A \to B : k$ may be rendered as

$$B \textit{ sees } (A \textit{ sent } (k \textit{ is a good key for } A \textit{ and } B))$$

Based on all this, non-trivial reasoning about the behaviour of protocols can be carried out assertionally. Many protocols have been proved correct using BAN logic, and flaws in many protocols have been detected using it as well. But this approach – the idealisation process, in particular – has met with a lot of criticism over the years ([14], for example), as there are many examples of protocols where there is a mismatch between the logic and the semantics of the protocol. The point is that the intruder behaviour is too rich to be captured by a simple set of rules.

Recent work by Cohen and Dam [4, 5] has made significant progress by providing a Kripke semantics for BAN logic that addresses the logical omniscience problem, and by exploring various completeness and expressibility issues.

An important feature of the logic discussed here is that the idealisation step is limited to assertions about which (atomic) secrets an agent has access to. This has implications for reasoning, as well as decidable verification, as we will see below.

2 Security protocol modelling

We briefly present our model for protocols in this section. A more detailed presentation can be found in [16]. Most of the elements of the model are standard in the literature on modelling security protocols. In particular, we use the Dolev-Yao adversary model [6].

Terms and actions

We start with a (potentially infinite) set of **agents** *Ag*, which includes the **intruder** *I*, and the others, who are called **honest agents**. Fix a countable set of

fresh secrets $\mathscr{N}$. (This includes *random, nonguessable nonces* as well as *temporary session keys.*) $\mathscr{K}$, the set of **potential keys,** is given by $\mathscr{N} \cup \{public(A), private(A) \mid A \in Ag\} \cup \{shared(A,B) \mid A, B \in Ag\}$. Here $public(A)$, $private(A)$, and $shared(A,B)$ denote the public key of A, private key of A, and (long-term) shared key between A and B. We assume an inverse $\overline{k}$ for each $k \in \mathscr{K}$ such that $\overline{\overline{k}} = k$.

$\mathscr{T}_0 \overset{\text{def}}{=} \mathscr{K} \cup Ag$ is the set of **basic terms.**

The set of **information terms** is defined to be

$$\mathscr{T} \ ::= \ m \mid (t_1, t_2) \mid \{t_1\}_k$$

where m ranges over $\mathscr{T}_0$, t_1 and t_2 range over $\mathscr{T}$, and k ranges over $\mathscr{K}$. Here (t_1, t_2) denotes the pair consisting of t_1 and t_2, and $\{t_1\}_k$ denotes the term t_1 encrypted using k. These are the terms used in the message exchanges (which will be presently introduced). We use $st(t)$ to denote the set of subterms of t.

We model communication between agents by *actions.* An action is either a *send action* of the form $A!B{:}\,t$ or a *receive action* of the form $A?B{:}\,t$. Here A and B are distinct agents, A is honest, and t is a term. For an action a of the form $A!B{:}\,t$ or $A?B{:}\,t$, we define $term(a)$ to be t. The agent B is (merely) the **intended receiver** in $A!B{:}\,t$ and the **purported sender** in $A?B{:}\,t$. Since the intruder is assumed to have access to the entire communication network at all times, every send action can be seen as an instantaneous receive by the intruder, and similarly, every receive action is an instantaneous send by the intruder.

Protocol specifications

A protocol is given by the roles it contains, and a **role** is a finite sequence of actions. A **parametrized role** $\eta[m_1, \cdots, m_k]$ is a role in which the *basic terms* $m_1, \ldots, m_k$ are singled out as parameters. The idea is that an agent participating in the protocol can execute many *sessions* of a role in the course of a single run, by instantiating the parameters in many different ways. All the basic terms occurring in a parametrized role that are not counted among the parameters are the **constants** of the role. They do not change their meaning over different sessions of the role.

Suppose $\eta = a_1 \cdots a_k$ is a parametrized role. We say that a nonce n **originates** at $i (\leq k)$ in η if:

- n is a parameter of η,
- a_i is a send action, and
- $n \in st(term(a_i))$ and for all $j < i$, $n \notin st(term(a_j))$.

If a nonce n originates at i in a role it means that the agent sending the message a_i uses n for the first time in the role. This usually means that, in any session of that role, the agent playing the role has to generate a fresh, nonguessable random number and send it as a challenge. Subsequent receipt of the same number in the same session plays a part in convincing the agent that the original message reached the intended recipient.

A **protocol** is a finite set of parametrized roles $\{\eta_1, \ldots, \eta_n\}$. The set of constants of Pr, denoted $\mathbf{C}(Pr)$, consists of all constants of all roles of Pr. The semantics of a protocol is given by the set of all its runs. A run is got by instantiating each role of the protocol in an appropriate manner, and forming admissible interleavings of such instantiations. We present the relevant definitions below.

Substitutions and events

A **substitution** σ is a map from $\mathscr{T}_0$ to $\mathscr{T}_0$ such that $\sigma(Ag) \subseteq Ag$, $\sigma(\mathscr{N}) \subseteq \mathscr{N}$, and $\sigma(I) = I$. For any $T \subseteq \mathscr{T}_0$, σ is said to be a T-substitution iff for all $x \in \mathscr{T}_0$, $\sigma(x) \in T$. A substitution σ is **suitable for a parametrized role** η if $\sigma(m) = m$ for all constants m of η. We say that σ is **suitable for a protocol** Pr if $\sigma(m) = m$ for all constants m of Pr.

A run of a protocol is any sequence of actions that can possibly be performed by the various agents taking part in the protocol. We model each run as a sequence of *event occurrences*, which are actions with some extra information about causality. (From now on, we will gloss over the difference between events and event occurrences.) An event of a protocol Pr is a triple (η, σ, lp) such that η is a role of Pr, σ is a substitution, and $1 \leq lp \leq |\eta|$. A T-event is one which involves a T-substitution. For an event $e = (\eta, \sigma, lp)$ with $\eta = a_1 \cdots a_\ell$, $act(e) \overset{\text{def}}{=} \sigma(a_{lp})$. If $e = (\eta, \sigma, lp)$ and $lp < |\eta|$ and $e' = (\eta, \sigma, lp + 1)$, then we say that e *locally precedes* e' and denote it by $e <_\ell e'$. We say that a nonce n is **uniquely originating** in a set of events E of a protocol Pr if there is at most one event (η, σ, lp) of E and at most one nonce m such that m originates at lp in η and $\sigma(m) = n$. (Note that the fact m originates in η implies that m is a parameter of η.)

Message generation rules

We intend a run of a protocol to be an admissible sequence of events. A very important ingredient of the admissibility criterion is the enabling of events given a particular information state. To treat this formally, we need to define how the agents (particularly the intruder) can build new messages from old. This is formalised by the notion of derivations.

A **sequent** is of the form $T \vdash t$ where $T \subseteq \mathscr{T}$ and $t \in \mathscr{T}$. A **derivation** or a **proof** π of $T \vdash t$ is a tree whose nodes are labelled by sequents and connected by one of the *analz*-rules or *synth*-rules in Figure 1; whose root is labelled $T \vdash t$; and whose leaves are labelled by instances of the Ax rule. We will use the notation $T \vdash t$ to denote both the sequent, and the fact that it is derivable. For a set of terms T, $\overline{T} \overset{\text{def}}{=} \{t \mid T \vdash t\}$ is the *closure* of T. Note that $\overline{T}$ is in general infinite

$$\frac{}{T \cup \{t\} \vdash t}\,Ax \qquad \frac{T \vdash t_1 \quad T \vdash t_2}{T \vdash (t_1, t_2)}\,pair$$

$$\frac{T \vdash (t_1, t_2)}{T \vdash t_i}\,split_i (i = 1, 2) \qquad \frac{T \vdash t \quad T \vdash k}{T \vdash \{t_1\}_k}\,encrypt$$

$$\frac{T \vdash \{t\}_k \quad T \vdash \overline{k}}{T \vdash t}\,decrypt$$

analz-rules *synth*-rules

Figure 1: Message generation rules.

even when T is finite, and hence the following proposition is useful. It is quite well known in the literature. A proof can be found in [18], for instance.

Proposition 1. *Suppose that T is a finite set of terms and t is a term. Checking whether $T \vdash t$ is decidable.*

Information states, updates, and runs

An **information state** (or just *state*) is a tuple $(s_A)_{A \in Ag}$, where $s_A \subseteq \mathscr{T}$ for each $A \in Ag$. The **initial state** of Pr, denoted by $init(Pr)$ is the tuple $(s_A)_{A \in Ag}$ such that for all $A \in Ag$,

$$s_A = \mathbf{C}(Pr) \cup Ag \cup \{private(A)\} \cup \{public(B), shared(A, B) \mid B \in Ag\}.$$

The notion of information state as simply a set of terms is rudimentary, but suffices for our purposes; it can be considered as a database of explicit but restricted knowledge. The notions of an action **enabled** at a state, and $update(s, a)$, the **update** of a state s on an action a, are defined as follows:

- A send action a is enabled at s iff $term(a) \in \overline{s_A}$.
- A receive action a is enabled at s iff $term(a) \in \overline{s_I}$.
- $update(s, A!B{:}\, t) \overset{\text{def}}{=} s'$ where $s'_I = s_I \cup \{t\}$, and for all agents C other than I, $s'_C = s_C$.
- $update(s, A?B{:}\, t) \overset{\text{def}}{=} s'$ where $s'_B = s_B \cup \{t\}$ and for all agents C other than B, $s'_C = s_C$.

$update(s, \eta)$ for a state s and a sequence of actions η is defined in the obvious manner. Thus if s_A is a form of explicit knowledge, then $\overline{s_A}$ is a form of implicit knowledge, but one that is algorithmically constructible by the agent.

Given a protocol Pr and a sequence $\xi = e_1 \cdots e_k$ of events of Pr, $infstate(\xi)$ is defined to be $update(init(Pr), act(e_1) \cdots act(e_k))$. Given a protocol Pr, a sequence $e_1 \cdots e_k$ of events of Pr is said to be a **run** of Pr iff the following conditions hold:

- for all $i, j \leq k$ such that $i \neq j$, $e_i \neq e_j$,

- for all $i \leq k$ and for all e such that $e \overset{+}{\to}_\ell e_i$, there exists $j < i$ such that $e_j = e$,
- for all $i \leq k$, $act(e_i)$ is enabled at $infstate(e_1 \cdots e_{i-1})$, and
- every nonce that is not a constant of *Pr* is uniquely originating in $\{e_1, \ldots, e_k\}$.

We say that ξ is a T-run of *Pr*, for any given $T \subseteq \mathscr{T}_0$, if for all $i \leq k$, $st(e_i) \cap \mathscr{T}_0 \subseteq T$. We say that ξ is a b-run of *Pr*, for any given $b \in \mathbb{N}$, if there are at most b nonces uniquely originating in ξ. We let $\mathscr{R}(Pr)$, $\mathscr{R}_T(Pr)$, and $\mathscr{R}_b(Pr)$ denote respectively the set of all runs, all T-runs, and all b-runs of *Pr*.

3 The semantics of the logic

We now formally present the syntax and semantics of our logic. From the model for security protocols presented earlier, it is clear that protocol descriptions mention abstract names for agents, nonces, and keys, but the runs use different instantiations for these abstract names. It is these concrete systems determined by protocols that we wish to specify properties of and verify. To be abstract, the specification logic should also mention only the abstract names, but the semantics must translate them into concrete names used in runs. A difficulty is that the denotation of an abstract name differs at different points in runs, so we have to find a way of resolving the different meanings. This is akin to the problem of **rigid designators** in first order modal logic.

We use the simple device of using logical variables. Their meaning is given by assignments, just as in first-order logic. But we do not allow quantification over variables in the logic itself. We let $\mathscr{X}$ denote the infinite set of logical variables. These variables are supposed to stand for nonces. We also need to use concrete agent names in the logic for specifications based on agent-based knowledge modalities to make sense. It bears emphasizing that variables do not have the same status as the abstract names in the protocol specification. These indeed refer to concrete nonces that occur in concrete executions of the protocol, but quite often we do not want to bother about which particular concrete nonce is being talked about.

Recall the syntax of the logic:

$$\mathscr{L} \ ::= \ A\ has\ x \mid sent(A, B, x) \mid received(A, B, x) \mid \neg\alpha \mid \alpha \vee \beta \mid \mathbf{G}\alpha \mid \mathbf{H}\alpha \mid \mathbf{K}_A \alpha$$

The semantics of the logic crucially hinges on an equivalence relation on runs, which is defined as follows. Intuitively, an agent cannot distinguish a term $\{t\}_k$ from any other bitstring in a state where she has no information about k.

We define the set $\mathscr{P}$ of patterns as follows (where $\square$ denotes an unknown pattern):

$$P, Q \in \mathscr{P} ::= m \in \mathscr{T}_0 \mid (P, Q) \mid \{P\}_k \mid \square$$

An *action pattern* is an action that uses a pattern instead of a term, and an *event pattern* is an event that uses an action pattern instead of an action.

We can now define the patterns derivable by an agent on seeing a term t in the context of a set of terms S. In a sense, this is the only *certain* knowledge that the agent can rely on at that state. A similar notion has been defined in [20] in the context of providing semantics for a BAN-like logic.

$$\begin{aligned}
\mathit{pattern}(m,S) &= \begin{cases} m & \text{if } m \in \mathscr{T}_0 \cap S \\ \square & \text{if } m \in \mathscr{T}_0 \setminus S \end{cases} \\
\mathit{pattern}((t_1,t_2),S) &= (\mathit{pattern}(t_1,S), \mathit{pattern}(t_2,S)) \\
\mathit{pattern}(\{t\}_k,S) &= \begin{cases} \{\mathit{pattern}(t,S)\}_k & \text{if } \overline{k} \in \overline{S} \\ \square & \text{otherwise} \end{cases}
\end{aligned}$$

We extend the definition to $\mathit{pattern}(a,S)$ and $\mathit{pattern}(e,S)$ for an action a, event e and a set of terms S in the obvious manner. Note that $\mathit{pattern}(a,S)$ is an action pattern and $\mathit{pattern}(e,S)$ is an event pattern. For $\xi = e_1 \cdots e_n$, we define $\mathit{pattern}(\xi,S)$ to be the sequence $\mathit{pattern}(e_1,S) \cdots \mathit{pattern}(e_n,S)$.

Definition 2. *An agent A's* view *of a run ξ, denoted $\xi{\upharpoonright}A$, is defined as $\mathit{pattern}(\xi',S)$, where ξ' is the subsequence of all A-events of ξ, and $S = \mathit{infstate}(\xi)$. For two runs ξ and ξ' of Pr and an agent A, we define ξ and ξ' to be A-equivalent (in symbols $\xi \sim_A \xi'$) iff $\xi{\upharpoonright}A = \xi'{\upharpoonright}A$. For $\xi = e_1 \cdots e_n$, $\xi' = e'_1 \cdots e'_{n'}$, $i \le n$, and $i' \le n'$, we say that $(\xi, i) \sim_A (\xi', i')$ when $e_1 \cdots e_i \sim_A e'_1 \cdots e'_{i'}$.*

The next issue in giving the semantics of $\mathscr{L}$ is how to handle the logical variables. This is standard. Along with the protocol we have an assignment $\mathbf{a} : \mathscr{X} \to \mathscr{N}$. A T-assignment (for $T \subseteq \mathscr{N}$) is one which maps logical variables only to nonces in T.

The semantics of formulas in such logics of knowledge is typically given at *points*: $(\xi, i) \vDash_{\mathbf{a}} \alpha$ [10]. However, to emphasize the fact that knowledge semantics crucially depends on the set of runs being considered, we present it as $\mathscr{R}, (\xi, i) \vDash_{\mathbf{a}} \alpha$ below. As we vary the set $\mathscr{R}$, for instance to consider a subset, what is common knowledge to agents changes, and hence the semantics as well.

Fix a protocol Pr and an assignment $\mathbf{a}$. For any subset $\mathscr{R}$ of $\mathscr{R}(Pr)$, we define the satisfaction relation $\mathscr{R}, (\xi, i) \vDash_{\mathbf{a}} \alpha$ as follows, where $\xi \in \mathscr{R}$, $i \le |\xi|$, and α is a formula:

- $\mathscr{R}, (\xi, i) \vDash_{\mathbf{a}} A\ \mathit{has}\ x$ iff $\mathbf{a}(x) \in \overline{s_A}$ for $s = \mathit{infstate}((\xi, i))$.
- $\mathscr{R}, (\xi, i) \vDash_{\mathbf{a}} \mathit{sent}(A, B, x)$ iff $\mathit{act}(\xi(i)) = A!B{:}t$ for some t such that $\mathbf{a}(x)$ occurs as a non-key subterm of t.
- $\mathscr{R}, (\xi, i) \vDash_{\mathbf{a}} \mathit{received}(A, B, x)$ iff $\mathit{act}(\xi(i)) = A?B{:}t$ for some t such that $\mathbf{a}(x)$ occurs as a non-key subterm of t.
- $\mathscr{R}, (\xi, i) \vDash_{\mathbf{a}} \mathbf{G}\alpha$ iff for all i such that $i \le i' \le |\xi|$, it is the case that $\mathscr{R}, (\xi, i') \vDash_{\mathbf{a}} \alpha$.

- $\mathscr{R}, (\xi, i) \models_{\mathbf{a}} \mathbf{H}\alpha$ iff for all i' such that $1 \le i' \le i$, it is the case that $\mathscr{R}, (\xi, i') \models_{\mathbf{a}} \alpha$.
- $\mathscr{R}, (\xi, i) \models_{\mathbf{a}} \mathbf{K}_A\alpha$ iff for all $\xi' \in \mathscr{R}$ and $i' \le |\xi'|$ such that $(\xi, i) \sim_A (\xi', i')$, it is the case that $\mathscr{R}, (\xi', i') \models_{\mathbf{a}} \alpha$.

For a protocol *Pr* and a formula α, we say that α is **valid** over *Pr* iff for all assignments **a**, and all runs (ξ, i) of *Pr*, $\mathscr{R}(Pr), (\xi, i) \models_{\mathbf{a}} \alpha$.

For a protocol *Pr*, formula α, and a fixed $T \subseteq \mathscr{T}_0$, we say that α is T-**valid** over *Pr* iff for all T-assignments **a**, and all T-runs (ξ, i) of *Pr*, $\mathscr{R}_T(Pr), (\xi, i) \models_{\mathbf{a}} \alpha$.

For a protocol *Pr*, formula α, and $b \ge |T|$, we say that α is b-**valid** over *Pr* iff for all assignments **a**, and all b-runs (ξ, i) of *Pr*, $\mathscr{R}_b(Pr), (\xi, i) \models_{\mathbf{a}} \alpha$.

Note that we have defined the semantics of formulas not over arbitrary sequences, but relative to the runs of a protocol. This is because we are not studying abstract notions of consistency in the logic but the more concrete questions of attacks on security protocols. If a formula is satisfiable as a sequence of information states which cannot be obtained as an admissible run of a protocol, such a property is deemed to be uninteresting. Since the logic itself has no access to encrypted terms and hence cannot describe protocols, it cannot constrain satisfiability to range over such admissible runs either.

4 Decidability

The technical problem we now consider is the *verification problem* for our logic. This asks for a given protocol *Pr* and a given formula α whether $Pr \models \alpha$. The first thing to note is that this problem is undecidable in general.

Theorem 3. *The verification problem for $\mathscr{L}$ is undecidable.*

Undecidability in the context of unboundedly long messages (but boundedly many nonces) was shown by [9] and for unboundedly many nonces (but bounded message length) by [7], by different techniques. Chapter 3 of [19] presents a uniform framework in which such undecidablity results can be seen.

We therefore look at restrictions of the problem and try to obtain decidability. A natural restriction is to confine our interest only to T-runs, for a fixed finite set $T \subseteq \mathscr{T}_0$. The following assertion shows that this is of no help.

Theorem 4. *For a given Pr and α, and for a fixed finite set $T \subseteq \mathscr{T}_0$, checking whether all T-runs of Pr satisfy α is undecidable.*

The proof is based on the same kind of Turing machine codings used in the proof of Theorem 3. The point is that even though we evaluate α only on T-runs of *Pr*, the knowledge modalities range over *all runs* of *Pr*, not just over all T-runs. A "simple" formula like $\mathbf{L}_A(I\ has\ m)$ forces one to consider runs (including those in which A does not play a crucial role) in which I tries to obtain m through complicated means, and which might involve the honest agents generating lots of new nonces. The trouble is that the "simple" atomic formula *I has m* packs a lot of expressive power, and allows a trivial coding of secrecy.

In this context, it is reasonable to try to obtain decidability by restricting the semantics of protocols (since even very weak logics are undecidable). One approach would be to ask if α is T-valid over *Pr*, given *Pr* and α. But that would amount to common knowledge of T, and goes against the grain of security theory: after all, agents are assumed to have the ability to generate random nonces about which others know nothing!

We follow an approach that is standard in security analysis. We place an upper bound on the number of nonces that can be used in any run. This also implies a bound on the number of concurrent multi-sessions an agent can participate in. More formally, given $b > 0$, we ask if it is the case that α is b-valid over *Pr*.

Before we proceed with our specific decidability result, we make two preliminary observations.

PROPOSITION 5. *Given a protocol Pr and a formula α, there exists a* ***finite*** *set of assignments $\mathscr{A}$ such that $Pr \vDash \alpha$ iff for all runs (ξ, i) of Pr and all assignments* ***a*** *from $\mathscr{A}$, $(\xi, i) \vDash_{\boldsymbol{a}} \alpha$.*

PROOF: Fix an enumeration $x_0, x_1, \ldots$ of $\mathscr{X}$ and an enumeration $n_0, n_1, \ldots$ of $\mathscr{N}$. Let (without loss of generality) $T_0 = \{n_0, \ldots, n_{K-1}\}$ be the *constants* of *Pr* and α (the constants of α are just the nonces occurring in α). Let $x_K, \ldots, x_{L-1}$ be the variables occurring in α.

For every $\mathbf{a} : \mathscr{X} \to \mathscr{N}$, consider $\mathbf{a}^{-1} : \mathscr{N} \to \mathscr{N} \cup \{x_K, \ldots, x_{L-1}\}$ defined as follows:

$$\mathbf{a}^{-1}(n) = \begin{cases} n & \text{if } n \in T_0 \text{ or } n \notin \mathbf{a}(\{x_K, \ldots, x_{L-1}\}) \\ x_i & \text{if } i \text{ is the least such that } a(x_i) = n \end{cases}$$

Let $\widehat{\mathbf{a}} : \{x_K, \ldots, x_{L-1}\} \to \{n_K, \ldots, n_{L-1}\}$ be given by:

$$\widehat{\mathbf{a}}(x_i) = \begin{cases} n & \text{if } \mathbf{a}(x_i) = n \in T_0 \\ n_i & \text{if } \mathbf{a}(x_i) \notin T_0 \end{cases}$$

For every assignment $\mathbf{a}$, let $\widetilde{\mathbf{a}} : \mathscr{N} \to \mathscr{N}$ be given by:

$$\widetilde{\mathbf{a}}(n) = \begin{cases} \mathbf{a}^{-1}(n) & \text{if } \mathbf{a}^{-1}(n) \in \mathscr{N} \\ \widehat{\mathbf{a}}(\mathbf{a}^{-1}(n)) & \text{otherwise} \end{cases}$$

It is now straightforward to prove that for all runs (ξ, i) of *Pr*, all assignments $\mathbf{a}$, and all subformulas β of α:

$$(\xi, i) \vDash_{\mathbf{a}} \beta \text{ iff } (\widetilde{\mathbf{a}}(\xi), i) \vDash_{\widehat{\mathbf{a}}} \beta$$

and the proposition immediately follows. ⊣

On the strength of the above theorem, in what follows we only consider the problem of checking whether a particular set of runs of a protocol satisfy a formula under a fixed assignment (which we shall not mention explicitly).

For a protocol *Pr* and a set of agents Ag', call ξ an Ag'-run of *Pr* if only agents from Ag' occur in ξ. The following proposition is easy to prove.

Proposition 6. *Given a protocol Pr and a formula α, there is a finite set of agents Ag' such that $Pr \models \alpha$ iff $(\xi, 0) \models \alpha$ for all Ag'-runs ξ of Pr.*

Theorem 7. *Fix a bound b. The problem of checking for a given protocol Pr and a formula α of $\mathscr{L}$ whether α is b-valid over Pr is decidable in time $2^{(n \cdot b \cdot d)^{O(1)}}$, where n is the size of the protocol specification, and d is the modal depth of α.*

Fix a bound b for the rest of the section. Also fix a protocol *Pr* and a formula α. We want to check whether α is b-valid over *Pr*.

We let T be the set consisting of the constants of *Pr*, and the nonces occurring in α. For ease of notation, we refer to $\mathscr{R}_b(Pr)$ by $\mathscr{R}$ for the rest of the section. Assume that the modal depth of the formula is d. This means that it is enough to consider "chains" of runs of length at most d to determine the truth of the given formula on any run. Thus if we find an *finite* set of runs $\mathscr{R}'$ that is d-bisimilar to $\mathscr{R}$, it will turn out to be enough to check the truth of α over $\mathscr{R}'$, in order to verify α over $\mathscr{R}$.

Towards this, we define a set of *nonce patterns* $\{\Box_{i,j} \mid i \leq d, j \leq b\}$, and define for each $i \leq d$ the set $\mathscr{P}_i = \{\Box_{i',j} \mid i' \leq i, j \leq b\}$. We denote by $\mathscr{R}_i (i \leq d)$ the set of runs which only use nonces from T and patterns from $\mathscr{P}_i$ (in place of nonces). A run belonging to $\mathscr{R}_i$ is also referred to as an i-run.

For $i \leq d$, a *zap function of rank i* is a *one-to-one, partial map* $\mu : \mathscr{N} \to T \cup \mathscr{P}_i$ such that for all $n \in T$, $\mu(n) = n$. We say that a zap function μ is suitable for a b-run ξ of the protocol if it is defined for all nonces occurring in ξ. For two b-runs ξ and ξ' of *Pr*, we say that $\xi \approx_i \xi'$ iff there exist zap functions μ, μ' of rank i suitable for ξ and ξ', respectively, such that $\mu(\xi) = \mu'(\xi')$. It is clear that $\approx_i$ is an equivalence relation for all $i \leq d$.

The heart of the proof is the following lemma:

Lemma 8. *For all $i < d$, b-runs ξ_1 and ξ_2 such that $\xi_1 \approx_i \xi_2$, the following holds:*

> *for all agents A, all b-runs ξ_1', and all positions $\ell \leq |\xi_1|$, $\ell' \leq |\xi_1'|$ such that $(\xi_1, \ell) \sim_A (\xi_1', \ell')$, there exists a b-run ξ_2' such that $\xi_1' \approx_{i+1} \xi_2'$ and $(\xi_2, \ell) \sim_A (\xi_2', \ell')$.*

Proof: Let μ_1, μ_2 be appropriate zap functions of rank i such that $\mu_1(\xi_1) = \mu_2(\xi_2)$. Let $\mathscr{N}'$ be the nonces occurring in $\xi_1 \upharpoonright A$. Define ξ_2' to be $\tau(\xi_1')$ where τ is defined as follows (assuming a set of "fresh nonces" $n_1, \ldots, n_b$):

$$\tau(n) = \begin{cases} \text{undefined} & \text{if } n \text{ does not occur in } \xi_1' \\ \mu_2^{-1}(\mu_1(n)) & \text{if } n \in \mathscr{N}' \\ n_i & \text{if } n \text{ is the } i^{\text{th}} \text{ nonce occurring in } \xi_1' \text{ and not in } \mathscr{N}' \end{cases}$$

It is clear that we can find zap functions μ_1' and μ_2' of rank $i+1$ such that $\mu_1'(\xi_1') = \mu_2'(\xi_2')$. (They mimic μ_1 respectively on $\mathscr{N}'$ and map the other nonces (in order of occurrence) to $\square_{i+1,0}, \square_{i+1,1}, \ldots, \square_{i+1,b}$).

Now we show that $(\xi_2, \ell) \sim_A (\xi_2', \ell')$. Note that $\xi_2 = \mu_2^{-1}(\mu_1(\xi_1))$ and $\xi_2' = (\mu_2')^{-1}(\mu_1'(\xi_1'))$. Also $\xi_1 \upharpoonright A = \xi_1' \upharpoonright A$. But notice that on $\mathscr{N}'$, μ_2' and μ_1' agree with μ_1, and therefore $\xi_2' \upharpoonright A = \mu_1^{-1}(\mu_1(\xi_1' \upharpoonright A)) = \xi_1' \upharpoonright A$. $\dashv$

From this, it is a standard argument to prove the following lemma.

LEMMA 9. *For any $i \leq d$, any formula β of modal depth $d - i$, and any two runs ξ and ξ' such that $\xi \approx_i \xi'$, and all $\ell \leq |\xi|$, $(\xi, \ell) \vDash \beta$ iff $(\xi, \ell) \vDash \beta$.*

Theorem 7 now follows from the fact that the verification problem reduces to verifying the truth of α over a finite set $\mathscr{R}'$ of runs of Pr (got from $\mathscr{R}_d$ by using a "fresh nonce" $n_{x,y}$ in place of each $\square_{i,j} \in \mathscr{P}_d$). A naive decision procedure is to generate the set of all such runs and check whether the given formula is true. Proposition 1 is used crucially both to check the truth of atomic formulas of the form *A has x*, and to check admissibility conditions on actions sequences. We get the bound on $\mathscr{R}'$ as follows: each event of a run in $\mathscr{R}'$ is specified by an action a in the protocol specification, and nonces of the form $n_{i,j}$ that instantiate the ones in a. Thus the set of events we need to consider is of size $B = n \cdot (b \cdot d)^{O(1)}$. Now a run in $\mathscr{R}'$ is determined by a subset of the events and an ordering on them. Thus the number of runs is $(B+1)^B$. Thus we get the bound specified in the theorem.

5 Discussion

The central idea of the decision procedure above is to *lift* the decidability of the question $T \vdash t$ to that of the logic. This suggests that we can extend the result to protocols whose message terms over richer cryptographic primitives, as long as the question $T \vdash t$ remains decidable. We could include algebraic properties of the encryption operators, or extend the term algebra with primitives such as *blind signatures*, and yet achieve such decidability. In [2], we show this in the case of blind pairing.

The use of the inference system on terms suggests that knowledge of agents reduces to *provability* in the weaker system. However, there are many cryptographic contexts such as *zero knowledge proofs* where agents can verify that a term has a particular structure, without being able to construct it. Such a consideration takes us beyond Dolev-Yao models.

Halpern and Pucella [12] make an attempt to go beyond the Dolev Yao model, by also considering probabilistic notions and the intruder attempting to guess nonces and keys. They employ the idea of modelling adversary capabilities by restrictions on the algorithms used by adversaries; this is represented in our set-up by indexing the message derivation system: $T \vdash_A t$ describes the algorithm employed by A. This approach is explored in [17]. But our emphasis has been on decidability, and it will be interesting to explore decision questions in rich logics like the one in [12].

Another consideration, largely ignored in this paper, relates to how security protocols may be implemented in a manner consistent with knowledge specifications [21]. This line of work is important, but what we have attempted to argue here is that such considerations lead us not only to interesting security theory, but also new theories of knowledge.

References

[1] Ross Anderson and Roger M. Needham. Programming Satan's computer. In *Computer Science Today*, volume 1000 of *Lecture Notes in Computer Science*, pages 426–441, 1995.

[2] A. Baskar, R. Ramanujam, and S.P. Suresh. Knowledge-based modelling of voting protocols. In Dov Samet, editor, *Proceedings of the 11th Conference on Theoretical Aspects of Rationality and Knowledge*, pages 62–71, 2007.

[3] Michael Burrows, Martin Abadi, and Roger M. Needham. A logic of authentication. *ACM Transactions on Computer Systems*, 8(1):18–36, Feb 1990.

[4] Mika Cohen and Mats Dam. A completeness result for BAN logic. In *2005 International Workshop on Methods for Modalities (M4M-05)*, pages 202–219, 2005.

[5] Mika Cohen and Mats Dam. A complete axiomatization of knowledge and cryptography. In *22nd IEEE Symposium on Logic in Computer Science (LICS 2007)*, pages 77–88. IEEE Computer Society, 2007.

[6] Danny Dolev and Andrew Yao. On the Security of public-key protocols. *IEEE Transactions on Information Theory*, 29:198–208, 1983.

[7] Nancy A. Durgin, Patrick D. Lincoln, John C. Mitchell, and Andre Scedrov. The undecidability of bounded security protocols. In *Proceedings of the Workshop on Formal Methods and Security Protocols (FMSP'99)*, 1999.

[8] Cynthia Dwork and Yoram Moses. Knowledge and Common Knowledge in a Byzantine Environment: Crash Failures. *Information and Computation*, 88(2):156–186, 1990.

[9] Shimon Even and Oded Goldreich. On the security of multi-party ping-pong protocols. Technical Report 285, Technion - Israel Institute of Technology, 1983.

[10] Ronald Fagin, Joseph Y. Halpern, Yoram Moses, and Moshe Y. Vardi. *Reasoning about Knowledge*. M.I.T. Press, 1995.

[11] Joseph Y. Halpern and Yoram Moses. Knowledge and common knowledge in a distributed environment. *Journal of the ACM*, 3(3):549–587, 1990.

[12] Joseph Y. Halpern and Riccardo Pucella. Modeling adversaries in a logic for security protocol analysis. In *Formal Aspects of Security, First International Conference, FASec 2002*, volume 2629 of *Lecture Notes in Computer Science*, pages 115–132, 2003.

[13] Nancy Lynch. *Distributed Algorithms*. Morgan Kaufmann Publishers, 1996.

[14] D. M. Nessett. A critique of the Burrows, Abadi and Needham logic. *ACM Operating systems review*, 24(2):35–38, 1990.

[15] Rohit Parikh. Logical omniscience and common knowledge: WHAT do we know and what do WE know? In *Proceedings of TARK 2005*, pages 62–77, 2005.

[16] R. Ramanujam and S. P. Suresh. Decidability of context-explicit security protocols. *Journal of Computer Security*, 13(1):135–165, 2005.

[17] R. Ramanujam and S. P. Suresh. A (restricted) quantifier elimination for security protocols. *Theoretical Computer Science*, 367:228–256, 2006.

[18] Michaël Rusinowitch and Mathieu Turuani. Protocol Insecurity with Finite Number of Sessions and Composed Keys is NP-complete. *Theoretical Computer Science*, 299:451–475, 2003.

[19] S.P. Suresh. *Foundations of Security Protocol Analysis*. PhD thesis, The Institute of Mathematical Sciences, Chennai, India, November 2003. Madras University. Available at `http://www.cmi.ac.in/~spsuresh`.

[20] Paul F. Syverson and P.C. van Oorschot. On unifying some cryptographic protocol logics. In *Proceedings of the 13th IEEE Symposium on security and privacy*, pages 14–28. IEEE Press, 1994.

[21] Ron van der Meyden and Thomas Wilke. Preservation of Epistemic Properties in Security Protocol Implementations. In Dov Samet, editor, *Proceedings of TARK '07*, pages 212–221, 2007.