Modeling Adversaries in a Logic for Security Protocol Analysis

Joseph Y. Halpern and Riccardo Pucella

Department of Computer Science
Cornell University
Ithaca, NY 14853
{halpern,riccardo}@cs.cornell.edu

Abstract. Logics for security protocol analysis require the formalization of an adversary model that specifies the capabilities of adversaries. A common model is the Dolev-Yao model, which considers only adversaries that can compose and replay messages, and decipher them with known keys. The Dolev-Yao model is a useful abstraction, but it suffers from some drawbacks: it cannot handle the adversary knowing protocol-specific information, and it cannot handle probabilistic notions, such as the adversary attempting to guess the keys. We show how we can analyze security protocols under different adversary models by using a logic with a notion of algorithmic knowledge. Roughly speaking, adversaries are assumed to use algorithms to compute their knowledge; adversary capabilities are captured by suitable restrictions on the algorithms used. We show how we can model the standard Dolev-Yao adversary in this setting, and how we can capture more general capabilities including protocol-specific knowledge and guesses.

1 Introduction

Many formal methods for the analysis of security protocols rely on specialized logics to rigorously prove properties of the protocols they study.[1] Those logics provide constructs for expressing the basic notions involved in security protocols, such as secrecy, recency, and message composition, as well as providing means (either implicitly or explicitly) for describing the evolution of the knowledge or belief of the principals as the protocol progresses. Every such logic aims at proving security in the presence of hostile adversaries. To analyze the effect of adversaries, a security logic specifies (again, either implicitly or explicitly) an *adversary model*, that is, a description of the capabilities of adversaries. Almost all existing logics are based on a Dolev-Yao adversary model [9]. Succinctly, a Dolev-Yao adversary can compose messages, replay them, or decipher them if she knows the right keys, but cannot otherwise "crack" encrypted messages.

The Dolev-Yao adversary is a useful abstraction, in that it allows reasoning about protocols without worrying about the actual cryptosystem being used. It also has the advantage of being restricted enough that interesting theorems can be proved with

[1] Here, we take a very general view of logic, to encompass formal methods where the specification language is implicit, or where the properties to be checked are fixed, such as Casper [22], Cryptyc [16], or the NRL Protocol Analyzer [25].

respect to security. However, in many ways, the Dolev-Yao model is too restrictive. For example, it does not consider the information an adversary may infer from properties of messages and knowledge about the protocol that is being used. To give an extreme example, consider what we'll call the *Duck-Duck-Goose* protocol: an agent has an n-bit key and, according to her protocol, sends the bits that make up its key one by one. Of course, after intercepting these messages, an adversary will know the key. However, there is no way for security logics based on a Dolev-Yao adversary to argue that, at this point, the adversary knows the key. Another limitation of the Dolev-Yao adversary is that it does not easily capture probabilistic arguments. After all, the adversary can always be lucky and just *guess* the appropriate key to use, irrespective of the strength of the cryptosystem.. There is another important problem with the Dolev-Yao adversary. Because the Dolev-Yao model uses an abstract cryptosystem, it cannot capture subtle interactions between the cryptographic protocol and the cryptosystem. It is known that various protocols that appear secure under abstract cryptography can be problematic when implemented using specific cryptosystems [29]. Two examples of this are the phenomenon of encryption cycles [2], and the use of exclusive-or in some protocol implementations [33]. A more refined logic for reasoning about security protocols will have to be able to handle adversaries more general than the Dolev-Yao adversary.

The importance of being able to reason about different adversaries was made clear by Lowe [21] when he exhibited a man-in-the-middle attack in the well-known authentication protocol due to Needham and Schroeder [31]. Up until that time, the Needham-Schroeder protocol was analyzed under the assumption that the adversary had complete control of the network, and could inject intercept and inject arbitrary messages (up to the Dolev-Yao capabilities) into the protocol runs. However, the adversary was always assumed to be an outsider, not being able to directly interact with the protocol principals as himself. Lowe showed that if the adversary is allowed to be an *insider* of the system, that is, appear to the other principals as a bona fide protocol participant, then the Needham-Schroeder protocol does not guarantee the authentication properties it is meant to guarantee.

Because they effectively build in the adversary model, existing formal methods for analyzing protocols are not able to reason directly about the effect of running a protocol against adversaries with properties other than those built in. The problem is even worse when it is not clear exactly what assumptions are implicitly being made about the adversary.

In this paper, we introduce a logic for reasoning about security protocols that allows us to model adversaries explicitly. The idea is to model the adversary in terms of what the adversary knows. This approach has some significant advantages. Logics of knowledge [14] have been shown to provide powerful methods for reasoning about trace-based executions of protocols. They can be given semantics that is tied directly to protocol execution, thus avoiding problems of having to analyze an idealized form of the protocol, as is required, for example, in BAN logic [7]. A straightforward application of logics of knowledge allows us to conclude that in the Duck-Duck-Goose protocol, the adversary knows the key. Logics of knowledge can also be extended with probabilities [13, 18] so as to be able to deal with probabilistic phenomena. Unfortunately, traditional logics of knowledge suffer from a well-known problem known as the *logical omni-*

science problem: an agent knows all tautologies and all the logical consequences of her knowledge. The reasoning that allows an agent to infer properties of the protocol also allows an attacker to deduce properties that cannot be computed by realistic attackers in any reasonable amount of time.

To avoid the logical omniscience problem, we use the notion of *algorithmic knowledge* [14, Chapter 10 and 11]. Roughly speaking, we assume that agents (including adversaries) have "knowledge algorithms" that they use to compute what they know. The capabilities of the adversary are captured by its algorithm. Hence, Dolev-Yao capabilities can be provided by using a knowledge algorithm that can only compose messages or attempt to decipher using known keys. By changing the algorithm, we can extend the capabilities of the adversary so that it can attempt to crack the cryptosystem by factoring (in the case of RSA), using differential cryptanalysis (in the case of DES), or just by guessing keys, along the lines of a recent model due to Lowe [23]. Moreover, our framework can also handle the case of a principal sending the bits of its key, by providing the adversary's algorithm with a way to check whether this is indeed what is happening. By explicitly using algorithms, we can therefore analyze the effect of bounding the resources of the adversary, and thus make progress toward bridging the gap between the analysis of cryptographic protocols and more computational accounts of cryptography. (See [2] and the references therein for a discussion on work bridging this gap.) Note that we need both traditional knowledge and algorithmic knowledge in our analysis. Traditional knowledge is used to model a principal's beliefs about what can happen in the protocol; algorithmic knowledge is used to model the adversary's computational limitations (for example, the fact that it cannot factor).

The rest of the paper is organized as follows. In Section 2, we define our model for protocol analysis and our logic for reasoning about implicit and explicit knowledge, based on the well-understood multiagent system framework. In Section 3, we show how to model different adversaries from the literature. We discuss related work in Section 4.

2 A logic for security protocol analysis

In this section, we review the multi-agent system framework and the logic of algorithmic knowledge. We then show how these can be tailored to define a logic for reasoning about security protocols.

Multiagent systems The multiagent systems framework [14, Chapters 4 and 5] provides a model for knowledge that has the advantage of also providing a discipline for modeling executions of protocols. A multiagent system consists of n agents, each of which is in some local state at a given point in time. We assume that an agent's local state encapsulates all the information to which the agent has access. In the security setting, the local state of an agent might include some initial information regarding keys, the messages she has sent and received, and perhaps the reading of a clock. In a poker game, a player's local state might consist of the cards he currently holds, the bets made by other players, any other cards he has seen, and any information he may have about the strategies of the other players (for example, Bob may know that Alice likes to bluff, while Charlie tends to bet conservatively). The basic framework makes no assumptions about the precise nature of the local state.

We can then view the whole system as being in some global state, which is a tuple consisting of each agent' local state, together with the state of the environment, where the environment consists of everything that is relevant to the system that is not contained in the state of the agents. Thus, a global state has the form $(s_e, s_1, \ldots, s_n)$, where s_e is the state of the environment and s_i is agent i's state, for $i = 1, \ldots, n$. The actual form of the agents' local states and the environment's state depends on the application.

A system is not a static entity. To capture its dynamic aspects, we define a run to be a function from time to global states. Intuitively, a run is a complete description of what happens over time in one possible execution of the system. A *point* is a pair (r, m) consisting of a run r and a time m. For simplicity, we take time to range over the natural numbers in the remainder of this discussion. At a point (r, m), the system is in some global state $r(m)$. If $r(m) = (s_e, s_1, \ldots, s_n)$, then we take $r_i(m)$ to be s_i, agent i's local state at the point (r, m). We formally define a system $\mathcal{R}$ to consist of a set of runs (or executions). It is relatively straightforward to model security protocols as systems. Note that the adversary in a security protocol can be modeled as just another agent. The adversary's information at a point in a run can be modeled by its local state.

The basic logic The aim is to be able to reason about properties of such systems, including properties involving the knowledge of agents in the system. To formalize this type of reasoning, we first need a language. The logic of algorithmic knowledge [14, Chapters 10 and 11] provides such a framework. It extends the classical logic of knowledge by adding algorithmic knowledge operators.

The syntax of the logic $\mathcal{L}_n^{\mathrm{KX}}$ for algorithmic knowledge is straightforward. Starting with a set Φ_0 of primitive propositions, which we can think of as describing basic facts about the system, such as "the key is k" or "agent A sent the message m to B", complicated formulas of $\mathcal{L}_n^{\mathrm{KX}}(\Phi_0)$ are formed by closing off under negation, conjunction, and the modal operators $K_1, \ldots, K_n$ and $X_1, \ldots, X_n$.

Syntax of $\mathcal{L}_n^{\mathrm{KX}}(\Phi_0)$:

$p, q \in \Phi_0$	Primitive propositions	
$\varphi, \psi \in \Phi ::=$	Formulas	
p		Primitive proposition
$\neg\varphi$		Negation
$\varphi \wedge \psi$		Conjunction
$K_i\varphi$		Implicit knowledge of φ ($i \in 1..n$)
$X_i\varphi$		Explicit knowledge of φ ($i \in 1..n$)

The formula $K_i\varphi$ is read as "agent i (implicitly) knows the fact φ", while $X_i\varphi$ is read as "agent i explicitly knows fact φ". In fact, we will read $X_i\varphi$ as "agent i can compute fact φ". This reading will be made precise when we discuss the semantics of the logic. As usual, we take $\varphi \vee \psi$ to be an abbreviation for $\neg(\neg\varphi \wedge \neg\psi)$ and $\varphi \Rightarrow \psi$ to be an abbreviation for $\neg\varphi \vee \psi$.

The standard models for this logic are based on the idea of possible worlds and Kripke structures [19]. Formally, a Kripke structure M is a tuple $(S, \pi, \mathcal{K}_1, \ldots, \mathcal{K}_n)$, where S is a set of states or possible worlds, π is an interpretation which associates with each state in S a truth assignment to the primitive propositions (i.e., $\pi(s)(p) \in \{\mathbf{true}, \mathbf{false}\}$ for each state $s \in S$ and each primitive proposition p), and $\mathcal{K}_i$ is an equivalence relation on S (recall that an equivalence relation is a binary relation which is reflexive, symmetric, and transitive). $\mathcal{K}_i$ is agent i's possibility relation. Intuitively, $(s, t) \in \mathcal{K}_i$ if agent i cannot distinguish state s from state t (so that if s is the actual state of the world, agent i would consider t a possible state of the world).

A system can be viewed as a Kripke structure, once we add a function π telling us how to assign truth values to the primitive propositions. An *interpreted system* $\mathcal{I}$ consists of a pair $(\mathcal{R}, \pi)$, where $\mathcal{R}$ is a system and π is an interpretation for the propositions in Φ that assigns truth values to the primitive propositions at the global states. Thus, for every $p \in \Phi$ and global state s that arises in $\mathcal{R}$, we have $\pi(s)(p) \in \{\mathbf{true}, \mathbf{false}\}$. Of course, π also induces an interpretation over the points of $\mathcal{R}$; simply take $\pi(r, m)$ to be $\pi(r(m))$. We refer to the points of the system $\mathcal{R}$ as points of the interpreted system $\mathcal{I}$.

The interpreted system $\mathcal{I} = (\mathcal{R}, \pi)$ can be made into a Kripke structure by taking the possible worlds to be the points of $\mathcal{R}$, and by defining $\mathcal{K}_i$ so that $((r, m), (r', m')) \in \mathcal{K}_i$ if $r_i(m) = r'_i(m')$. Clearly $\mathcal{K}_i$ is an equivalence relation on points. Intuitively, agent i considers a point (r', m') possible at a point (r, m) if i has the same local state at both points. Thus, the agents' knowledge is completely determined by their local states.

To account for X_i, we provide each agent with a knowledge algorithm that he uses to compute his knowledge. We will refer to $X_i\varphi$ as *algorithmic knowledge*. An *interpreted algorithmic system* is of the form $(\mathcal{R}, \pi, \mathtt{A}_1, \ldots, \mathtt{A}_n)$, where $(\mathcal{R}, \pi)$ is an interpreted system, and $\mathtt{A}_i$ is the knowledge algorithm of agent i. In local state ℓ, the agent computes whether he knows φ by applying the knowledge algorithm $\mathtt{A}$ to input (φ, ℓ). The output is either "Yes", in which case the agent knows φ to be true, "No", in which case the agent does *not* know φ to be true, or "?", which intuitively says that the algorithm has insufficient resources to compute the answer. It is the last clause that allows us to deal with resource-bounded reasoners.

We define what it means for a formula φ to be true (or satisfied) at a point (r, m) in an interpreted system $\mathcal{I}$, written $(\mathcal{I}, r, m) \models \varphi$, inductively as follows.

Satisfiability relation: $(\mathcal{I}, r, m) \models \varphi$

$(\mathcal{I}, r, m) \models p$	if $\pi(r, m)(p) = \mathbf{true}$
$(\mathcal{I}, r, m) \models \neg\varphi$	if $(\mathcal{I}, r, m) \not\models \varphi$
$(\mathcal{I}, r, m) \models \varphi \wedge \psi$	if $(\mathcal{I}, r, m) \models \varphi$ and $(\mathcal{I}, r, m) \models \psi$
$(\mathcal{I}, r, m) \models K_i\varphi$	if $(\mathcal{I}, r, m) \models \varphi$ for all (r', m') such that $r_i(m) = r'_i(m')$
$(\mathcal{I}, r, m) \models X_i\varphi$	if $\mathtt{A}_i(\varphi, r_i(m)) =$ "Yes"

The first clause shows how we use the π to define the semantics of the primitive propositions. The next two clauses, which define the semantics of $\neg$ and $\wedge$, are the standard clauses from propositional logic. The fourth clause is designed to capture the intuition that agent i knows φ exactly if φ is true in all the worlds that i thinks are possible. The last clause captures the fact that explicit knowledge is determined using the knowledge algorithm of the agent.

As we pointed out, we think of K_i as representing *implicit knowledge*, facts that the agent implicitly knows, given its information, while X_i represents *explicit knowledge*, facts whose truth the agent can compute explicitly. As defined, there is no necessary connection between $X_i\varphi$ and $K_i\varphi$. An algorithm could very well claim that agent i knows φ (i.e., output "Yes") whenever it chooses to, including at points where $K_i\varphi$ does not hold. Although algorithms that make mistakes are common, we are often interested in knowledge algorithms that are correct. A knowledge algorithm is *sound* for agent i in the system $\mathcal{I}$ if for all points (r, m) of $\mathcal{I}$ and formulas φ, $\mathtt{A}(\varphi, r_i(m)) =$ "Yes" implies $(\mathcal{I}, r, m) \models K_i\varphi$, and $\mathtt{A}(\varphi, r_i(m)) =$ "No" implies $(\mathcal{I}, r, m) \models \neg K_i\varphi$. Thus, a knowledge algorithm is sound if its answers are always correct.

Specializing to security The systems and logic we describe earlier in this section are fairly general. We have a particular application in mind, namely reasoning about security protocols, especially authentication protocols. We now specialize the framework above by describing the local states of the systems under considerations, as well as pinning down the primitive propositions in our logic, and the interpretation of those propositions.

For the purposes of analyzing security protocols, we consider *security systems* to be those where the local state of a principal consists of the principal's initial information followed by the sequence of events that the principal has been involved in. An event is either the reception $\mathsf{recv}(\mathsf{m})$ of a message m, or the sending $\mathsf{send}(i, \mathsf{m})$ of a message m to another agent i. To model the fact that adversaries can intercept all the messages exchanged by the principals, we assume that the adversary's local state includes the set of messages exchanged between all the principals.

Since typically security protocols involve agents exchanging encrypted messages, we need to discuss the cryptographic concepts we need. We assume a set $\mathcal{P}$ of plaintexts, as well as a set $\mathcal{K}$ of keys. We define a cryptosystem $\mathcal{C}$ over $\mathcal{P}$ and $\mathcal{K}$ to be the closure $\mathcal{M}$ of $\mathcal{P}$ and $\mathcal{K}$ under a concatenation operation $conc : \mathcal{M} \times \mathcal{M} \to \mathcal{M}$ and an encryption operation $encr : \mathcal{M} \times \mathcal{K} \to \mathcal{M}$. We often write $\mathsf{m}_1 \cdot \mathsf{m}_2$ for $conc(\mathsf{m}_1, \mathsf{m}_2)$ and $\{\!|\mathsf{m}|\!\}_{\mathsf{k}}$ for $encr(\mathsf{m}, \mathsf{k})$. There is no difficulty in adding more operations to the cryptosystems, for instance, to model hashes, signatures, or the ability to take the exclusive-or of two terms. The messages exchanged by security systems will be taken from this set $\mathcal{M}$ of messages.

We assume that the set $\mathcal{K}$ of keys is closed under inverses; for a given key $\mathsf{k} \in \mathcal{K}$, we assume an inverse key $\mathsf{k}^{-1} \in \mathcal{K}$ such that encrypting with respect to the key k^{-1} is equivalent to decrypting a message, that is, $\{\!|\{\!|\mathsf{m}|\!\}_{\mathsf{k}}|\!\}_{\mathsf{k}^{-1}} = \mathsf{m}$. If the cryptosystem uses symmetric keys, then $\mathsf{k}^{-1} = \mathsf{k}$; for public key cryptosystems, k and k^{-1} are different. Formally, we define $\mathcal{M}$ to be the smallest set containing both $\mathcal{P}$ and $\mathcal{K}$, such that if $\mathsf{m}_1, \mathsf{m}_2 \in \mathcal{M}$ and $\mathsf{k} \in \mathcal{K}$, then $\{\!|\mathsf{m}|\!\}_{\mathsf{k}} \in \mathcal{M}$ and $\mathsf{m}_1 \cdot \mathsf{m}_2 \in \mathcal{M}$. We make no assumption in the general case as to the properties of encryption.

Define a relation $\sqsubseteq$ on $\mathcal{M}$ as the smallest relation satisfying the following constraints:

1. $\mathsf{m} \sqsubseteq \mathsf{m}$,
2. if $\mathsf{m} \sqsubseteq \mathsf{m}_1$, then $\mathsf{m} \sqsubseteq \mathsf{m}_1 \cdot \mathsf{m}_2$,
3. if $\mathsf{m} \sqsubseteq \mathsf{m}_2$, then $\mathsf{m} \sqsubseteq \mathsf{m}_1 \cdot \mathsf{m}_2$,

4. if $\mathsf{m} \sqsubseteq \mathsf{m}_1$, then $\mathsf{m} \sqsubseteq \{\!|\mathsf{m}_1|\!\}_{\mathsf{k}}$.

Intuitively, $\mathsf{m}_1 \sqsubseteq \mathsf{m}_2$ if m_1 *could* be used in the construction of m_2. For example, if $\mathsf{m} = \{\!|\mathsf{m}_1|\!\}_{\mathsf{k}} = \{\!|\mathsf{m}_2|\!\}_{\mathsf{k}}$, then both $\mathsf{m}_1 \sqsubseteq \mathsf{m}$ and $\mathsf{m}_2 \sqsubseteq \mathsf{m}$. Therefore, if we want to establish that $\mathsf{m}_1 \sqsubseteq \mathsf{m}_2$ for a given m_1 and m_2, then we have to look at all the possible ways in which m_2 can be taken apart, either by concatenation or encryption, to finally decide if m_1 can be derived from m_2.

To reason about security protocols, we consider a specific set of primitive propositions Φ_0^{S}.

Primitive propositions for security:

$p, q \in \Phi_0^{\mathrm{S}} ::=$

$\mathsf{send}_i(\mathsf{m})$	Agent i sent message m
$\mathsf{recv}_i(\mathsf{m})$	Agent i received message m
$\mathsf{has}_i(\mathsf{m})$	Agent i has message m

Intuitively, $\mathsf{send}_i(\mathsf{m})$ is true when agent i has sent message m at some point, and $\mathsf{recv}_i(\mathsf{m})$ is true when agent i has received message m at some point. Agent i *has* a submessage m_1 at a point (r, m), written $\mathsf{has}_i(\mathsf{m}_1)$, if there exists a message $\mathsf{m}_2 \in \mathcal{M}$ such that $\mathsf{recv}(\mathsf{m}_2)$ is in $r_i(m)$, the local state of agent i, and $\mathsf{m}_1 \sqsubseteq \mathsf{m}_2$. Note that the has_i predicate is not restricted by issues of encryption. If $\mathsf{has}_i(\{\!|\mathsf{m}|\!\}_{\mathsf{k}})$ holds, then so does $\mathsf{has}_i(\mathsf{m})$, whether or not agent i knows the key k^{-1}. Intuitively, the has_i predicate characterizes the messages that agent i has implicitly in his possession, given the messages that have been exchanged between the principals.

An *interpreted security system* is simply an interpreted system $\mathcal{I} = (\mathcal{R}, \pi_{\mathcal{R}}^{\mathrm{S}})$ where $\mathcal{R}$ is a system for security protocols, and $\pi_{\mathcal{R}}^{\mathrm{S}}$ is the following canonical interpretation of the primitive propositions in $\mathcal{R}$.

Canonical interpretation:

$\pi_{\mathcal{R}}^{\mathrm{S}}(r, m)(\mathsf{send}_i(\mathsf{m})) = \textbf{true}$ iff $\exists j$ such that $\mathsf{send}(j, \mathsf{m}) \in r_i(m)$
$\pi_{\mathcal{R}}^{\mathrm{S}}(r, m)(\mathsf{recv}_i(\mathsf{m})) = \textbf{true}$ iff $\mathsf{recv}(\mathsf{m}) \in r_i(m)$
$\pi_{\mathcal{R}}^{\mathrm{S}}(r, m)(\mathsf{has}_i(\mathsf{m})) = \textbf{true}$ iff $\exists \mathsf{m}'$ such that $\mathsf{m} \sqsubseteq \mathsf{m}'$ and $\mathsf{recv}(\mathsf{m}') \in r_i(m)$

An *interpreted algorithmic security system* is similarly defined as an interpreted algorithmic system $(\mathcal{R}, \pi_{\mathcal{R}}^{\mathrm{S}}, \mathtt{A}_1, \ldots, \mathtt{A}_n)$, where $\mathcal{R}$ is a security system, and $\pi_{\mathcal{R}}^{\mathrm{S}}$ is the canonical interpretation in $\mathcal{R}$.

3 Modeling adversaries

As we outlined in the last section, interpreted algorithmic security systems provide a foundation for representing security protocols, and support a logic for writing properties based on knowledge, both traditional and algorithmic. For the purposes of analyzing security protocols, we use traditional knowledge to model a principal's beliefs about what can happen in the protocol, while we use algorithmic knowledge to model the adversary's capabilities, possibly resource-bounded. To interpret algorithmic knowledge, we rely on a knowledge algorithm for each agent in the system. We haven't said anything yet as to what kind of algorithm we might consider, short of the fact that we typically care about sound knowledge algorithms. For the purpose of security, the knowledge algorithms we give to the adversary are the most important, as they capture the facts that the adversary can compute given what he has seen. In this section, we show how we can capture different capabilities for the adversary rather naturally in this framework. We first show how to capture the standard model of adversary due to Dolev and Yao. We then show how to account adversaries in the Duck-Duck-Goose protocol, and the adversary due to Lowe that can perform self-validating guesses.

The Dolev-Yao adversary As a first example of this, consider the standard Dolev-Yao adversary [9]. This model is a combination of assumptions on the cryptosystem used and the capabilities of the adversaries. Specifically, the cryptosystem is seen as the free algebra generated by $\mathcal{P}$ and $\mathcal{K}$ over abstract operations $\cdot$ and $\{\!|\,|\!\}$. Perhaps the easiest way to formalize this is to view the set $\mathcal{M}$ as the set of abstract expressions generated by the grammar

$$\mathsf{m} ::= \mathsf{p} \mid \mathsf{k} \mid \{\!|\mathsf{m}|\!\}_{\mathsf{k}} \mid \mathsf{m} \cdot \mathsf{m}$$

(with $\mathsf{p} \in \mathcal{P}$ and $\mathsf{k} \in \mathcal{K}$). We then identify elements of $\mathcal{M}$ under the equivalence $\{\!|\{\!|\mathsf{m}|\!\}_{\mathsf{k}}|\!\}_{\mathsf{k}^{-1}} = \mathsf{m}$. Notice that this model of cryptosystem implicitly assumes that there are no collisions; messages always have a unique decomposition. The only way that $\{\!|\mathsf{m}|\!\}_{\mathsf{k}} = \{\!|\mathsf{m}'|\!\}_{\mathsf{k}'}$ is if $\mathsf{m} = \mathsf{m}'$ and $\mathsf{k} = \mathsf{k}'$. We make the standard assumption that concatenation and encryption have enough redundancy to recognize that a term is in fact a concatenation $\mathsf{m}_1 \cdot \mathsf{m}_2$ or an encryption $\{\!|\mathsf{m}|\!\}_{\mathsf{k}}$.

The Dolev-Yao model can be formalized by a relation $H \vdash_{DY} \mathsf{m}$ between a set H of messages and a message m. (Our formalization is equivalent to many other formalizations of Dolev-Yao in the literature, and is similar in spirit to that of Paulson [32].) Intuitively, $H \vdash_{DY} \mathsf{m}$ means that an adversary can "extract" message m from a set of received messages and keys H, using the allowable operations. The derivation is defined using the following inference rules:

Dolev-Yao derivation: $H \vdash_{DY} \mathsf{m}$

$$\frac{\mathsf{m} \in H}{H \vdash_{DY} \mathsf{m}} \qquad \frac{H \vdash_{DY} \{\!|\mathsf{m}|\!\}_{\mathsf{k}} \quad H \vdash_{DY} \mathsf{k}^{-1}}{H \vdash_{DY} \mathsf{m}} \qquad \frac{H \vdash_{DY} \mathsf{m}_1 \cdot \mathsf{m}_2}{H \vdash_{DY} \mathsf{m}_1} \qquad \frac{H \vdash_{DY} \mathsf{m}_1 \cdot \mathsf{m}_2}{H \vdash_{DY} \mathsf{m}_2}$$

In our framework, to capture the capabilities of a Dolev-Yao adversary, we specify how the adversary can tell if she in fact *has* a message, by defining a knowledge algorithm $\mathtt{A}_i^{\mathrm{DY}}$ for adversary i. Recall that a knowledge algorithm for agent i takes as input a formula and agent i's local state (which we are assuming contains the messages received by i). The most interesting case in the definition of $\mathtt{A}_i^{\mathrm{DY}}$ is when the formula is $\mathsf{has}_i(\mathsf{m})$. To compute $\mathtt{A}_i^{\mathrm{DY}}(\mathsf{has}_i(\mathsf{m}), \ell)$, the algorithm simply checks, for every message m' received by the adversary, whether m is a submessage of m', according to the keys that are known to the adversary. We assume that the adversary's initial state consists of the set of keys initially known by the adversary. This will typically contain, in a public-key cryptography setting, the public keys of all the agents. We use $initkeys(\ell)$ to denote the set of initial keys known by agent i in local state ℓ. (Recall that a local state for agent i is the sequence of events pertaining to agent i, including any initial information in the run, in this case, the keys initially known.) Checking whether m is a submessage of m' is performed by a function $submsg$, which can take apart messages created by concatenation, or decrypt messages as long as the adversary knows the decryption key.

Dolev-Yao knowledge algorithm (extract):

$\mathtt{A}_i^{\mathrm{DY}}(\mathsf{has}_i(\mathsf{m}), \ell) \triangleq K = keysof(\ell)$
    for each $\mathsf{recv}(\mathsf{m}')$ in ℓ do
        if $submsg(\mathsf{m}, \mathsf{m}', K)$ then
            return "Yes"
    return "No"

The full algorithm can be found in the appendix.

According to the Dolev-Yao model, the adversary cannot explicitly compute anything interesting about what other messages agents have. Hence, for other primitives, including $\mathsf{has}_j(\mathsf{m})$ for $j \neq i$, $\mathtt{A}_i^{\mathrm{DY}}$ returns "?". For formulas of the form $K_j\varphi$ and $X_j\varphi$, $\mathtt{A}_i^{\mathrm{DY}}$ also returns "?". For Boolean combinations of formulas, $\mathtt{A}_i^{\mathrm{DY}}$ returns the corresponding Boolean combination (where the negation of "?" is "?", the conjunction of "No" and "?" is "No", and the conjunction of "Yes" and "?" is "?") of the answer for each $\mathsf{has}_i(\mathsf{m})$ query.

The following result shows that an adversary using $\mathtt{A}_i^{\mathrm{DY}}$ recognizes (i.e., returns "Yes" to) $\mathsf{has}_i(\mathsf{m})$ in state ℓ iff m exactly the messages determined to be in the set of messages that can be derived (according to $\vdash_{DY}$) from the messages received in that state together with the keys initially known, Moreover, if a $\mathsf{has}_i(\mathsf{m})$ formula is derived at the point (r, m), the $\mathsf{has}_i(\mathsf{m})$ is actually true at (r, m) (so that $\mathtt{A}_i^{\mathrm{DY}}$ is sound).

Proposition 3.1. *In the interpreted algorithmic security system* $\mathcal{I} = (\mathcal{R}, \pi^{\mathrm{S}}_{\mathcal{R}}, \mathtt{A}_1, \ldots, \mathtt{A}_n)$, *where* $\mathtt{A}_i = \mathtt{A}_i^{\mathrm{DY}}$, *we have that* $(\mathcal{I}, r, m) \models X_i(\mathsf{has}_i(\mathsf{m}))$ *iff* $\{\mathsf{m} : \mathsf{recv}(\mathsf{m}) \in r_i(m)\} \cup initkeys(\ell) \vdash_{DY} \mathsf{m}$. *Moreover, if* $(\mathcal{I}, r, m) \models X_i(\mathsf{has}_i(\mathsf{m}))$ *then* $(\mathcal{I}, r, m) \models \mathsf{has}_i(\mathsf{m})$.

The Duck-Duck-Goose adversary The key advantage of our framework is that we can easily change the capabilities of the adversary beyond those prescribed by the Dolev-Yao model. For example, we can capture the fact that if the adversary knows the protocol, she can derive more information than she could otherwise. For instance, in the Duck-Duck-Goose example, assume that the adversary maintains in her local state a list of all the bits received corresponding to the key of the principal. We can easily write the algorithm so that if the adversary's local state contains all the bits of the key of the principal, then the adversary can decode messages that have been encrypted with that key. Specifically, assume that key k is being sent in the Duck-Duck-Goose example. Then for an adversary i, $\mathsf{has}_i(\mathsf{k})$ will be false until all the bits of the key have been received. This translates immediately into the following algorithm $\mathtt{A}_i^{\mathrm{DDG}}$:

Duck-Duck-Goose knowledge algorithm:

$\mathtt{A}_i^{\mathrm{DDG}}(\mathsf{has}_i(\mathsf{k}), \ell) \triangleq$ if all the bits recorded in ℓ form k then
    return "Yes" else return "No"

$\mathtt{A}_i^{\mathrm{DDG}}$ handles other formulas in the same way as $\mathtt{A}_i^{\mathrm{DY}}$.

Of course, nothing keeps us from combining algorithms, so that we can imagine an adversary intercepting both messages and key bits, and using an algorithm $\mathtt{A}_i$ which is a combination of the Dolev-Yao algorithm and the Duck-Duck-Goose algorithm, such as:

$$\mathtt{A}_i(\varphi, \ell) = \text{if } \mathtt{A}_i^{\mathrm{DY}}(\varphi, \ell) = \text{“Yes” then} \\ \text{return “Yes”} \\ \text{else return } \mathtt{A}_i^{\mathrm{DDG}}(\varphi, \ell)$$

This assumes that the adversary knows the protocol, and hence knows when the key bits are being sent. The algorithm above captures this protocol-specific knowledge.

The Lowe adversary For a more realistic example of an adversary model that goes beyond Dolev-Yao, consider the following adversary model introduced by Lowe [23] to analyze protocols subject to guessing attacks. The intuition is that some protocols provide for a way to "validate" the guesses of an adversary. For a simple example of this, here is a simple challenge-based authentication protocol:

$$\begin{array}{l} A \rightarrow S : A \\ S \rightarrow A : n_s \\ A \rightarrow S : \{\!|n_s|\!\}_{p_a} \end{array}$$

Intuitively, A tells the server S that she wants to authenticate herself. S replies with a challenge n_s. A sends back to S the challenge encrypted with her password p_a. Presumably, S knows the password, and can verify that she gets $\{\!|n_s|\!\}_{p_a}$. Unfortunately, an adversary can overhear both n_s and $\{\!|n_s|\!\}_{p_a}$, and can "guess" a value g for p_a and verify his guess by checking if $\{\!|n_s|\!\}_g = \{\!|n_s|\!\}_{p_a}$. The key feature of this kind of attack is that the guessing (and the validation) can be performed offline, based only on the intercepted messages.

To account for this capability of adversaries is actually fairly complicated. We present a slight variation of Lowe's description, mostly to make it notationally consistent with the rest of the section; we refer the reader to Lowe [23] for a discussion of the design choices.

Lowe's model relies on a basic one-step reduction function, $S \rhd_l \mathsf{m}$, saying that the messages in S can be used to derive the message m. This is essentially the same as $\vdash_{DY}$, except that it represents a single step of derivation. Moreover, the relation is "tagged" by the kind of derivation performed (l):

One-step reduction: $S \rhd_l \mathsf{m}$

$\{\mathsf{m}, \mathsf{k}\} \rhd_{\mathsf{enc}} \{\!|\mathsf{m}|\!\}_{\mathsf{k}}$
$\{\{\!|\mathsf{m}|\!\}_{\mathsf{k}}, \mathsf{k}^{-1}\} \rhd_{\mathsf{dec}} \mathsf{m}$
$\{\mathsf{m}_1 \cdot \mathsf{m}_2\} \rhd_{\mathsf{fst}} \mathsf{m}_1$
$\{\mathsf{m}_1 \cdot \mathsf{m}_2\} \rhd_{\mathsf{snd}} \mathsf{m}_2$

Lowe also includes a reduction to derive $\mathsf{m}_1 \cdot \mathsf{m}_2$ from m_1 and m_2. We do not add this reduction to simplify the presentation. It is straightforward to extend the work in this section to account for this augmented derivation.

Given a set H of message, and a sequence t of one-step reductions, we define inductively the set $[H]_t$ of messages obtained from the one-step reductions given in t.

Messages obtained from one-step reductions: $[H]_t$

$[H]_{\langle\rangle} \triangleq H$

$$[H]_{\langle S \rhd_l \mathsf{m}\rangle \cdot t} \triangleq \begin{cases} [H \cup \{\mathsf{m}\}]_t & \text{if } S \subseteq H \\ \mathit{undefined} & \text{otherwise} \end{cases}$$

Here, $\langle\rangle$ denotes the empty trace, and $t_1 \cdot t_2$ denotes trace concatenation. A trace t is said to be *monotone* if, intuitively, it does not perform any one-step reduction that "undoes" a previous one-step reduction. For example, the reduction $\{\mathsf{m}, \mathsf{k}\} \rhd \{\!|\mathsf{m}|\!\}_\mathsf{k}$ undoes the reduction $\{\{\!|\mathsf{m}|\!\}_\mathsf{k}, \mathsf{k}^{-1}\} \rhd \mathsf{m}$. (See Lowe [23] for more details on undoing reductions.)

We say that a set H of messages *validates* a guess m if, intuitively, H contains enough information to verify that m is indeed a good guess. Intuitively, this happens if a value v (called a validator) can be derived from the messages in $H \cup \{\mathsf{m}\}$ in a way that uses the guess m, and either that (a) validator v can be derived in a different way from $H \cup \{\mathsf{m}\}$, (b) the validator v is already in $H \cup \{\mathsf{m}\}$, or (c) the validator v is a key whose inverse is derivable from $H \cup \{\mathsf{m}\}$. For example, in the protocol exchange at the beginning of this section, the adversary sees the messages $H = \{n_s, \{\!|n_s|\!\}_{p_a}\}$, and we can check that H validates the guess $\mathsf{m} = p_a$: clearly, $\{n_s, \mathsf{m}\} \rhd_{\mathsf{enc}} \{\!|n_s|\!\}_{p_a}$, and $\{\!|n_s|\!\}_{p_a} \in H \cup \{\mathsf{m}\}$. In this case, the validator $\{\!|n_s|\!\}_{p_a}$ is already present in $H \cup \{\mathsf{m}\}$. For other examples of validation, we again refer to Lowe [23].

We can now define the relation $H \vdash_L \mathsf{m}$ that says that m can be derived from H by a Lowe adversary. Intuitively, $H \vdash_L \mathsf{m}$ if m can be derived by Dolev-Yao reductions, or m can be guessed and validated by the adversary, and hence susceptible to an attack.

Lowe derivation: $H \vdash_L m$

$H \vdash_L \mathsf{m}$ iff $H \vdash_{DY} \mathsf{m}$ or there exists a monotone trace t, a set S, and a "validator" v such that:

(1) $[H \cup \{\mathsf{m}\}]_t$ is defined,
(2) $S \rhd_l v$ is in t,
(3) there is no trace t' such that $S \subseteq [H]_{t'}$, and
(4) either:
 (a) there exists $(S', l') \neq (S, l)$ with $S' \rhd_{l'} v$ in t,
 (b) $v \in H \cup \{\mathsf{m}\}$, or
 (c) $v \in \mathcal{K}$ and $v^{-1} \in [H \cup \{\mathsf{m}\}]_t$.

One can verify that the above formalization captures the intuition about validation given earlier. Specifically, condition (1) says that the trace t is well-formed, condition (2) says that the validator v is derived from $H \cup \{\mathsf{m}\}$, condition (3) says that deriving the validator v depends on the guess m, and condition (4) specifies when a validator v validates a guess m, as given earlier.

We would now like to define a knowledge algorithm $\mathtt{A}_i^{\mathrm{L}}$ to capture the capabilities of the Lowe adversary. Again, the only case of real interest is what $\mathtt{A}_i^{\mathrm{L}}$ does on input $\mathsf{has}_i(\mathsf{m})$.

Lowe knowledge algorithm (extract):

$\mathtt{A}_i^{\mathrm{L}}(\mathsf{has}_i(\mathsf{m}), \ell) \triangleq$ if $\mathtt{A}_i^{\mathrm{DY}}(\mathsf{has}_i(\mathsf{m}), \ell) =$ "Yes" then
    return "Yes"
  if $guess(\mathsf{m}, \ell)$ then
    return "Yes"
  return "No"

The full algorithm can be found in the appendix. (We have not concerned ourselves with matters of efficiency in the description of $\mathtt{A}_i^{\mathrm{L}}$; again, see Lowe [23] for a discussion of implementation issues.)

As before, we can check the correctness and soundness of the algorithm:

Proposition 3.2. *In the interpreted algorithmic security system* $\mathcal{I} = (\mathcal{R}, \pi_{\mathcal{R}}^{\mathrm{S}}, \mathtt{A}_1, \ldots, \mathtt{A}_n)$, *where* $\mathtt{A}_i = \mathtt{A}_i^{\mathrm{L}}$, *we have that* $(\mathcal{I}, r, m) \models X_i(\mathsf{has}_i(\mathsf{m}))$ *iff* $\{\mathsf{m} : \mathsf{recv}(\mathsf{m}) \in r_i(m)\} \cup initkeys(\ell) \vdash_L \mathsf{m}$. *Moreover, if* $(\mathcal{I}, r, m) \models X_i(\mathsf{has}_i(\mathsf{m}))$ *then* $(\mathcal{I}, r, m) \models \mathsf{has}_i(\mathsf{m})$.

4 Related work

The issues we raise in this paper are certainly not new, and have been addressed, up to a point, in the literature. In this section, we review this literature, and discuss where we stand with respect to other approaches that have attempted to tackle some of the same problems.

As we mentioned in the introduction, the Dolev-Yao adversary is the most widespread adversary in the literature. Part of its attraction is its tractability, making it possible to develop formal systems to automatically check for safety with respect to such adversaries [27, 28, 32, 22, 25]. The idea of moving beyond the Dolev-Yao adversary is not new. As we pointed out in Section 3, Lowe [23] developed an adversary that can encode some amount of off-line guessing; we showed in Section 3 that we could capture such an adversary in our framework. Other approaches have the possibility of extending the adversary model. For instance, the framework of Paulson [32], Clarke, Jha and Morrero [8], and Lowe [22] describe the adversary via a set of derivation rules, which could be modified by adding new derivation rules. We could certainly capture these adversaries by appropriately modifying our $\mathtt{A}_i^{\mathrm{DY}}$ knowledge algorithm. However, it does not seem that these other approaches have the flexibility of our approach in terms of capturing adversaries. Not all adversaries can be conveniently described in terms of derivation rules.

There are other approaches that weaken the Dolev-Yao adversary assumptions by either taking concrete cryptosystems into account, or at least adding new algebraic identities to the algebra of messages. Bieber [6] does not assume that the cryptosystem is a free algebra, following an idea due to Merritt and Wolper [26]. Even *et al.* [11] analyze ping-pong protocols under RSA, taking the actual cryptosystem into account. The applied π-calculus of Abadi and Fournet [1] permits the definition of an equational theory over the messages exchanged between processes, weakening some of the abstract cryptosystem assumptions when the applied π-calculus is used to analyze security protocols. Since the cryptosystem used in our framework is a simple parameter to the logic, there is no difficulty in modifying our logic to reason about a particular cryptosystem, and hence we can capture these approaches in our framework. But again, it is not clear the extent to which these other approaches have the same flexibility as ours.

On a related note, the work of Abadi and Rogaway [2], building on previous work by Bellare and Rogaway [5], compare the results obtained by a Dolev-Yao adversary with those obtained by a more computational view of cryptography, and show that under various conditions, the former is sound with respect to the latter, that is, terms that are indistinguishable under abstract encryption remain indistinguishable under a concrete encryption scheme. It would be interesting to recast their analysis in our setting, which, as we argued, can capture both the Dolev-Yao adversary and more concrete adversaries.

The use of a logic based on knowledge/belief is also not new. A number of formal logics for analysis of security protocols that involve knowledge and belief have been introduced, going back to BAN logic [7], such as [3, 6, 15, 34, 35, 4]. The main problem with some of those approaches is that semantics of the logic (to the extent that one is provided) is typically not tied to protocol executions or attacks. As a result, protocols are analyzed in an idealized form, and this idealization is itself error-prone and difficult to formalize [24].[2] While some of these approaches have a well-defined semantics and do not rely on idealization (e.g., [6, 4]), they are still restricted to (a version of) the Dolev-Yao adversary. In contrast, our framework goes beyond Dolev-Yao, as we have seen, and our semantics is directly tied to protocol execution.

The problem of logical omniscience in logics of knowledge is well known, and the literature describes numerous approaches to try to circumvent it. (See [14, Chapter 10 and 11] for an overview.) In the context of security, this takes the form of using different semantics for knowledge, either by introducing *hiding* operators that hide part of the local state for the purpose of indistinguishability (as done, for example, in [3]), or by using notions such as *awareness* [12] to capture an intruder's inability to decrypt [4].[3] Roughly speaking, the semantics for awareness can specify for every point a set of formulas of which an agent is aware. For instance, an agent may be aware of a formula without being aware of its subformulas. A general problem with awareness is determining the set of formulas of which an agent is aware at any point.

[2] While more recent logical approaches (e.g., [10]) do not suffer from an idealization phase and are more tied to protocol execution, but they also do not attempt to capture knowledge and belief in any general way.

[3] A notion of algorithmic knowledge was defined by Moses [30], and used by Halpern, Moses and Tuttle [17] to analyze zero-knowledge protocols. Although related to algorithmic knowledge defined here, Moses' approach does not use an explicit algorithm. Rather, it checks whether these exists an algorithm of a certain class (for example, a polynomial-time algorithm) that could compute such knowledge.

One interpretation of algorithmic knowledge is that it prescribes algorithmically what formulas an agent is aware of: those for which the algorithm says "Yes". In that sense, we subsume approaches based on awareness by providing them with an intuition. (We should note that the use of awareness by Accorsi *et al.* [4] is not motivated by the desire to model more general adversaries, but by the desire to restrict the number of states one needs to consider in models. Hence, the thrust of their work is different from ours.)

5 Conclusion

We have presented a framework for security analysis using algorithmic knowledge. The knowledge algorithm can be tailored to account for both the capabilities of the adversary and the specifics of the protocol under consideration. Of course, it is always possible to take a security logic and extend it in an ad hoc way to reason about adversary with different capabilities. Our approach has many advantages over ad hoc approaches: it is a general framework (we simply need to change the algorithm used by the adversary to change its capabilities, or add adversaries with different capabilities), and it permits reasoning about protocol-specific issues (we can capture the cases such as the agent sending the bits of its key).

Another advantage of our approach is that it naturally extends to the probabilistic setting. For instance, we can easily handle probabilistic protocols, by moving to multi-agent systems with an associated probability distribution on the runs (see Halpern and Tuttle [18]). In a slightly less trivial setting, we can also deal with knowledge algorithms that are probabilistic. This leads to some difficulty, since the semantics for X_i only really make sense if the knowledge algorithm is deterministic. We have extended the theory to handle such cases, and hope to report on a study of the obtained framework in future work. This would allow us to capture probabilistic adversaries of the kind studied by Lincoln *et al.* [20].

The goal of this paper was to introduce a general framework for handling different adversary models in a natural way, not specifically to devise new attacks or adversary capabilities. With this framework, it should be possible to put on a formal foundation new attacks that are introduced by the community. We gave a concrete example of this with the "guess-and-confirm" attacks of Lowe [23]. We are in the process of incorporating the ideas of this paper into a logic for reasoning about security protocols.

It is fair to ask at this point what we can gain by using this framework. For one thing, we believe that the ability of the framework to describe the capabilities of the adversary will make it possible to specify the properties of security protocols more precisely. Of course, it may be the case that to prove correctness of a security protocol with respect to certain types of adversaries (for example, polynomial-time bounded adversaries), we will not be able to do much within the logic—we will need to appeal to techniques developed in the cryptography community. However, we believe that it may well be possible to extend current model-checking techniques to handle more restricted adversaries (for example, Dolev-Yao extended with random guessing). This is a topic that deserves further investigation. In any case, having a logic where we can specify the abilities of adversaries is a necessary prerequisite to using model-checking techniques.

Acknowledgments This research was inspired by discussions between the first author, Pat Lincoln, and John Mitchell, on a wonderful hike in the Dolomites. We also think Sabina Petride for useful comments. Authors supported in part by NSF under grant CTC-0208535, by ONR under grants N00014-00-1-03-41 and N00014-01-10-511, by the DoD Multidisciplinary University Research Initiative (MURI) program administered by the ONR under grant N00014-01-1-0795, and by AFOSR under grant F49620-02-1-0101.

A Algorithms

Dolev-Yao knowledge algorithm:

$\mathtt{A}_i^{\mathrm{DY}}(\mathsf{has}_i(\mathsf{m}), \ell) \triangleq K = \mathit{keysof}(\ell)$
  for each $\mathsf{recv}(\mathsf{m}')$ in ℓ do
    if $\mathit{submsg}(\mathsf{m}, \mathsf{m}', K)$ then
      return "Yes"
  return "No"

$\mathit{submsg}(\mathsf{m}, \mathsf{m}', K) \triangleq$ if $\mathsf{m} = \mathsf{m}'$ then
    return true
  if m' is $\{\!|\mathsf{m}_1|\!\}_{\mathsf{k}}$ and $\mathsf{k}^{-1} \in K$ then
    return $\mathit{submsg}(\mathsf{m}, \mathsf{m}_1, K)$
  if m' is $\mathsf{m}_1 \cdot \mathsf{m}_2$ then
    return $\mathit{submsg}(\mathsf{m}, \mathsf{m}_1, K) \vee \mathit{submsg}(\mathsf{m}, \mathsf{m}_2, K)$
  return false

$\mathit{getkeys}(\mathsf{m}, K) \triangleq$ if $\mathsf{m} \in \mathcal{K}$ then
    return $\{\mathsf{m}\}$
  if m' is $\{\!|\mathsf{m}_1|\!\}_{\mathsf{k}}$ and $\mathsf{k}^{-1} \in K$ then
    return $\mathit{getkeys}(\mathsf{m}_1, K)$
  if m' is $\mathsf{m}_1 \cdot \mathsf{m}_2$ then
    return $\mathit{getkeys}(\mathsf{m}_1, K) \cup \mathit{getkeys}(\mathsf{m}_2, K)$
  return $\{\}$

$\mathit{keysof}(\ell) \triangleq K \leftarrow \mathit{initkeys}(\ell)$
  loop until no change in K
    $K \leftarrow \bigcup_{\mathsf{recv}(\mathsf{m}) \in \ell} \mathit{getkeys}(\mathsf{m}, K)$
  return K

Lowe knowledge algorithm:

$\mathtt{A}_i^{\mathtt{L}}(\mathsf{has}_i(\mathsf{m}), \ell) \triangleq$ if $\mathtt{A}_i^{\mathtt{DY}}(\mathsf{has}_i(\mathsf{m}), \ell) =$ "Yes" then
    return "Yes"
  if $\mathit{guess}(\mathsf{m}, \ell)$ then
    return "Yes"
  return "No"

$\mathit{guess}(\mathsf{m}, \ell) \triangleq H \leftarrow \mathit{reduce}(\{\mathsf{m} \ : \ \mathsf{recv}(\mathsf{m}) \text{ in } \ell\} \cup \mathit{initkeys}(\ell)) \cup \{\mathsf{m}\}$
  $\mathit{reds} \leftarrow \{\}$
  loop until $\mathit{reductions}(H) - \mathit{reds}$ is empty
    $(S, l, v) \leftarrow$ pick an element of $\mathit{reductions}(H) - \mathit{reds}$
    if $\exists (S', l', v) \in \mathit{reds}$ s.t. $S' \neq S$ and $l' \neq l$ then
      return "Yes"
    if $v \in H$ then
      return "Yes"
    if $v \in \mathcal{K}$ and $v^{-1} \in H$ then
      return "Yes"
    $\mathit{reds} \leftarrow \mathit{reds} \cup \{(S, l, v)\}$
    $H \leftarrow H \cup \{v\}$
  return "No"

$\mathit{reduce}(H) \triangleq$ loop until no change in H
    $r \leftarrow \mathit{reductions}(H)$
    for each (S, l, v) in r
      $H \leftarrow H \cup \{v\}$
  return H

$\mathit{reductions}(H) \triangleq \mathit{reds} \leftarrow \{\}$
  for each $\mathsf{m}_1 \cdot \mathsf{m}_2$ in H
    $\mathit{reds} \leftarrow \{(\{\mathsf{m}\}, \mathsf{fst}, \mathsf{m}_1), (\{\mathsf{m}\}, \mathsf{snd}, \mathsf{m}_2)\}$
  for each $\mathsf{m}_1, \mathsf{m}_2$ in H
    if $\mathsf{m}_2 \in \mathcal{K}$ and $\mathit{sub}(\{\!|\mathsf{m}_1|\!\}_{\mathsf{m}_2}, H)$ then
      $\mathit{reds} \leftarrow \{(\{\mathsf{m}_1, \mathsf{m}_2\}, \mathsf{enc}, \{\!|\mathsf{m}_1|\!\}_{\mathsf{m}_2})\}$
    if m_1 is $\{\!|\mathsf{m}'|\!\}_{\mathsf{k}}$ and m_2 is k^{-1} then
      $\mathit{reds} \leftarrow \{(\{\mathsf{m}_1, \mathsf{m}_2\}, \mathsf{dec}, \mathsf{m}')\}$
  return reds

$$
\begin{aligned}
sub(\mathsf{m}, H) \triangleq\ & \text{if } H = \{\mathsf{m}\} \text{ then} \\
& \quad \text{return } true \\
& \text{if } H = \{\mathsf{m}_1 \cdot \mathsf{m}_2\} \text{ then} \\
& \qquad \text{return } sub(\mathsf{m}, \{\mathsf{m}_1\}) \vee sub(\mathsf{m}, \{\mathsf{m}_2\}) \\
& \text{if } H = \{\{\!|\mathsf{m}'|\!\}_{\mathsf{k}}\} \text{ then} \\
& \qquad \text{return } sub(\mathsf{m}, \{\mathsf{m}'\}) \\
& \text{if } |H| > 1 \text{ and } H = \{\mathsf{m}'\} \cup H' \text{ then} \\
& \qquad \text{return } sub(\mathsf{m}, \{\mathsf{m}'\}) \vee sub(\mathsf{m}, H') \\
& \text{return } false
\end{aligned}
$$

References

1. M. Abadi and C. Fournet. Mobile values, new names, and secure communication. In *Proceedings of the 28th ACM Symposium on Principles of Programming Languages (POPL'01)*, pages 104–115, 2001.
2. M. Abadi and P. Rogaway. Reconciling two views of cryptography (the computational soundness of formal encryption). In *Proceedings of the IFIP International Conference on Theoretical Computer Science (IFIP TCS2000)*, volume 1872 of *Lecture Notes in Computer Science*, pages 3–22. Springer-Verlag, 2000.
3. M. Abadi and M. R. Tuttle. A semantics for a logic of authentication. In *Proc. 10th ACM Symp. on Principles of Distributed Computing*, pages 201–216, 1991.
4. R. Accorsi, D. Basin, and L. Viganò. Towards an awareness-based semantics for security protocol analysis. In Jean Goubault-Larrecq, editor, *Electronic Notes in Theoretical Computer Science*, volume 55. Elsevier Science Publishers, 2001.
5. M. Bellare and P. Rogaway. Entity authentication and key distribution. In *Proceedings of the 13th Annual International Cryptology Conference (CRYPTO '93)*, volume 773 of *Lecture Notes in Computer Science*, pages 232–249, 1993.
6. P. Bieber. A logic of communication in hostile environment. In *Proceedings of the Computer Security Foundations Workshop*, pages 14–22. IEEE Computer Society Press, 1990.
7. M. Burrows, M. Abadi, and R. Needham. A logic of authentication. *ACM Transactions on Computer Systems*, 8(1):18–36, 1990.
8. E.M. Clarke, S. Jha, and W. Marrero. Using state space exploration and a natural deduction style message derivation engine to verify security protocols. In *Proceedings of the IFIP Working Conference on Programming Concepts and Methods (PROCOMET)*, 1998.
9. D. Dolev and A. C. Yao. On the security of public key protocols. *IEEE Transactions on Information Theory*, 29(2):198–208, 1983.
10. N. Durgin, J. Mitchell, and D. Pavlovic. A compositional logic for protocol correctness. In *Proceedings of the Computer Security Foundations Workshop*, pages 241–255. IEEE Computer Society Press, 2001.
11. S. Even, O. Goldreich, and A. Shamir. On the security of ping-pong protocols when implemented using the RSA. In *Proceedings of Crypto'85*, volume 218 of *Lecture Notes in Computer Science*, pages 58–72. Springer-Verlag, 1985.
12. R. Fagin and J. Y. Halpern. Belief, awareness, and limited reasoning. *Artificial Intelligence*, 34:39–76, 1988.
13. R. Fagin and J. Y. Halpern. Reasoning about knowledge and probability. *Journal of the ACM*, 41(2):340–367, 1994.
14. R. Fagin, J. Y. Halpern, Y. Moses, and M. Y. Vardi. *Reasoning about Knowledge*. The MIT Press, 1995.

15. L. Gong, R. Needham, and R. Yahalom. Reasoning about belief in cryptographic protocols. In *Proc. IEEE Symposium on Security and Privacy*, pages 234–248, May 1990.
16. A. D. Gordon and A. Jeffrey. Authenticity by typing for security protocols. In *Proceedings of the 14th IEEE Computer Security Foundations Workshop (CSFW 2001)*, pages 145–159. IEEE Computer Society Press, 2001.
17. J. Y. Halpern, Y. Moses, and M. R. Tuttle. A knowledge-based analysis of zero knowledge. In *Proc. 20th ACM Symp. on Theory of Computing*, pages 132–147, 1988.
18. J. Y. Halpern and M. R. Tuttle. Knowledge, probability, and adversaries. *Journal of the ACM*, 40(4):917–962, 1993.
19. S. Kripke. A semantical analysis of modal logic I: normal modal propositional calculi. *Zeitschrift für Mathematische Logik und Grundlagen der Mathematik*, 9:67–96, 1963. Announced in *Journal of Symbolic Logic 24*, 1959, p. 323.
20. P. Lincoln, J. C. Mitchell, M. Mitchell, and A. Scedrov. A probabilistic poly-time framework for protocol analysis. In *Proceedings of the ACM Conference on Computer and Communications Security*, pages 112–121, 1998.
21. G. Lowe. An attack on the Needham-Schroeder public-key authentication protocol. *Information Processing Letters*, 56:131–133, 1995.
22. G. Lowe. Some new attacks upon security protocols. In *Proc. 9th IEEE Computer Security Foundations Workshop*, pages 162–169, 1996.
23. G. Lowe. Analysing protocols subject to guessing attacks. In *Proceedings of the Workshop on Issues in the Theory of Security (WITS'02)*, 2002.
24. W. Mao. An augmentation of BAN-like logics. In *Proceedings of the 8th IEEE Computer Security Foundations Workshop*, pages 44–56. IEEE Computer Society Press, 1995.
25. C. Meadows. The NRL protocol analyzer: An overview. *Journal of Logic Programming*, 26(2):113–131, 1996.
26. M. Merritt and P. Wolper. States of knowledge in cryptographic protocols. Unpublished manuscript, 1985.
27. J. K. Millen, S. C. Clark, and S. B. Freedman. The Interrogator: Protocol security analysis. *IEEE Transactions on Software Engineering*, 13(2):274–288, 1987.
28. J. Mitchell, M. Mitchell, and U. Stern. Automated analysis of cryptographic protocols using murφ. In *Proceedings of the IEEE Symposium on Research in Security and Privacy*, pages 141–151. IEEE Computer Society Press, 1997.
29. J. H. Moore. Protocol failures in cryptosystems. *Proceedings of the IEEE*, 76(5):594–602, 1988.
30. Y. Moses. Resource-bounded knowledge. In *Proc. Second Conference on Theoretical Aspects of Reasoning about Knowledge*, pages 261–276. 1988.
31. R. M. Needham and M. D. Schroeder. Using encryption for authentication in large networks of computers. *Communications of the ACM*, 21(12):993–999, 1978.
32. L. C. Paulson. The inductive approach to verifying cryptographic protocols. *Journal of Computer Security*, 6(1/2):85–128, 1998.
33. P. Y. A. Ryan and S. A. Schneider. An attack on a recursive authentication protocol: A cautionary tale. *Information Processing Letters*, 65(1):7–10, 1998.
34. S. Stubblebine and R. Wright. An authentication logic supporting synchronization, revocation, and recency. In *Proc. Third ACM Conference on Computer and Communications Security*, pages 95–105, 1996.
35. P. Syverson. A logic for the analysis of cryptographic protocols. NRL Report 9305, Naval Research Laboratory, 1990.