Modelling Cryptographic Protocols in a Theory of Action

James P. Delgrande and **Torsten Grote** and **Aaron Hunter**
School of Computing Science,
Simon Fraser University,
Burnaby, B.C.,
Canada V5A 1S6.
{jim,tga14,hunter}@cs.sfu.ca

Abstract

This paper proposes a framework for analysing cryptographic protocols by expressing message passing and possible attacks as a situation calculus theory. While cryptographic protocols are usually quite short, they are nonetheless notoriously difficult to analyse, and are subject to subtle and nonintuitive attacks. Our thesis is that in previous approaches for expressing protocols, underlying domain assumptions and capabilities of agents are left implicit. We propose a declarative specification of such assumptions and capabilities in the situation calculus. A protocol is then compiled into a sequence of actions to be executed by the principals. A successful attack is an executable plan by an intruder that compromises the stated goal of the plan. We argue that not only is a full declarative specification necessary, it is also much more flexible than previous approaches, permitting among other things interleaved runs of different protocols and participants with varying abilities.

Introduction

A cryptographic protocol is a formalised sequence of messages between agents, where parts of a message are protected using cryptographic functions such as encryption. These protocols are used for many purposes, including the secure exchange of information, carrying out a transaction, authenticating an agent, etc. Protocols are typically specified in the following format:

The Challenge-Response Protocol

1. $A \rightarrow B : \{N_A\}_{K_{AB}}$
2. $B \rightarrow A : N_A$

In this protocol, the goal is for agent A to determine whether B is alive on the network. The first step is for A to send B the message N_A encrypted with a shared key K_{AB}. N_A is a *nonce*, a random number assumed to be new to the network. The second step is for B to send A the message N_A unencrypted. Since only A and B have K_{AB}, and K_{AB} is assumed to be secure, it would seem that N_A could only have been decrypted by B, and so B must be alive. However, the protocol is flawed; here is an attack:

An Attack on the Challenge-Response Protocol

1. $A \rightarrow I_B : \{N_A\}_{K_{AB}}$
1.1 $I_B \rightarrow A : \{N_A\}_{K_{AB}}$
1.2 $A \rightarrow I_B : N_A$
2. $I_B \rightarrow A : N_A$

An intruder I intercepts the message in line 1 and, masquerading as B, initiates a round of the protocol with A, thereby obtaining the decrypted nonce.

While this example is simplistic, it illustrates the type of problems that arise in protocol verification. Even though protocols are usually short, they are notoriously difficult to prove correct. As a result, many different formal approaches have been developed for protocol verification. In these approaches, a protocol is generally specified as above, and then one tries to develop an attack on the protocol. However, often these approaches are difficult to apply by anyone other than the original developers (Brackin, Meadows, & Millen 1999). Part of the problem is that there is no clear agreement on exactly what an attack really is (Aiello & Massacci 2001), which leaves considerable ambiguity about the status of a protocol when no attack is found. Moreover, as we later discuss, the language for specifying a protocol is highly ambiguous, and much information is left implicit. Thus it is no surprise that protocols are hard to convincingly prove secure.

Our thesis is that all aspects of a protocol need to be explicitly specified, and moreover that protocol verification may profitably be viewed as a problem in commonsense reasoning and agent communication. The main contribution of this paper is the introduction of a declarative, commonsense theory of message passing between agents, suitable for proving results about protocols, expressed as a situation calculus theory. The framework makes explicit background assumptions, protocol goals, agent's capabilities, and the message passing environment. A protocol is *translated* into a set sequence of actions for agents to execute. These actions may be interleaved with others, and the framework allows simultaneous runnings of multiple protocols. The aim of an intruder is to construct a *plan* such that the goal of the protocol, in a precise sense, is thwarted. A protocol is secure when no such plan is possible. A valid protocol then is one which is secure, which may complete, and in which at completion the goal is provably established. The approach is flexible, and significantly more general than previous approaches since we can tailor the agents and the environment to specific applications. For example, we can model intruders with different capabilities and we can model several different protocols running at the same time. This work is intended as the first step towards a new automated verification system for protocols based on a language such as ConGolog.

The next section briefly introduces work in cryptographic protocol verification. The third section motivates our approach to the problem, while the following section presents an axiomatisation of an instance of the approach in the situation calculus. The last section sketches contributions and future work.

Related Work

The standard intruder model is of a very powerful adversary, the so-called *Dolev-Yao intruder* (Dolev & Yao 1983). Informally, the intruder can read, block, intercept, or forward any message sent by an honest agent. Hence, a message recipient is never aware of the identity of the sender, except possibly via encrypted messages. The first logic-based approach to protocol verification was the *BAN logic* of (Burrows, Abadi, & Needham 1990). The logic is rather ad hoc, as it consists of a set of rules of inference with no formal semantics. However, it has been highly influential because it illustrates the importance of *knowledge* in protocol verification and also because it illustrates how protocol verification can be reduced to reasoning in a formal logical system.

One standard formal tool for reasoning about the knowledge of several agents is the multi-agent systems framework of (Fagin *et al.* 1995). In protocol verification, the *strand space* formalism provides a similar model of message passing between several agents (Thayer, Herzog, & Guttman 1999). A strand space is a formal representation of all possible traces corresponding to runs of a specified protocol; it enables a protocol analyzer to show that an intruder cannot compromise a secure protocol. It has been proven that strand spaces are actually less expressive than multi-agent systems (Halpern & Pucella 2003). One notable weakness is that the framework does not provide a suitable model of knowledge.

Formal tools developed for knowledge representation and reasoning have also been used for protocol verification. One such tool is logic programming under the stable model semantics (Gelfond & Lifschitz 1991). Cryptographic protocols have been encoded as logic programs where the stable models correspond to attacks that an intruder can perform (Aiello & Massacci 2001). There are at least two issues with the encodings. First, the logic program must be handcrafted for each protocol to be analyzed. Second, the attack must be specified in advance; new attacks are not detected automatically. (Wang & Zhang 2008) proposes a very similar approach. Neither approach is elaboration tolerant, and neither intensional.

In related work, protocols have been represented in a multi-set rewriting formalism, and then translated into the same logic programming paradigm used in the previous two approaches (Armando, Compagna, & Lierler 2004). Instead of a model checker, this translation is solved with an answer set solver, acting as an alternative back-end to the protocol verification tool AVISPA[1]. To date, this translation approach has not proven to be practical.

Hernández and Pinto propose an approach that is similar to ours; in particular they also use the situation calculus (Hernández-Orallo & Pinto 1997). However, they focus on producing proofs of correctness based on the actions of honest agents. In contrast, we explicitly model the actions of an intruder, and we view protocol verification as communicating while guarding against attack. Our treatment of the communication channel is also different: while Hernández and Pinto define an unreliable broadcast channel, we define a direct channel that allows the intruder the first opportunity to receive a message. As such, our approach is best understood as addressing a somewhat different problem than the Hernández-Pinto approach.

There has of course been extensive work in reasoning about action. Due to space limitations, we assume a familiarity with the situation calculus (Levesque, Pirri, & Reiter 1998), and we assume Reiter's solution to the frame problemn without further comment. We note that other action formalisms would have worked equally well in formalising the approach.

Motivation

Consider the Challenge-Response protocol and the attack described previously. Several things may be noted about the protocol specification. First, while the intent of the protocol and the attack are intuitively clear, the meaning of the exchanges in the protocol are ambiguous. Consider the first line of the protocol: it cannot mean that A sends a message to B, since this may not be the case, as the attack illustrates. Nor can it mean that A *intends* to send a message to B, because in the attack it certainly isn't A's intention to send the message to the intruder! Moreover, there is more than one action taking place in the first line, since A sends a message and B is involved in the (potential) receipt of a message. Hence, the specification language is inexpressive; notions of agent communication should be made explicit.

As well, the specification leaves important aspects of the problem unstated. For instance, it is not stated that the goal of the protocol is to convince A that B is alive. Nor is it stated how this goal is to be accomplished, in this case indirectly via the encryption and sending of messages. Metalevel reasoning is required to determine if a protocol is secure, or if an attack on the protocol is possible. So notions of protocol goal and attack should also be made explicit.

The protocol specification also does not state the fact that N_A is a freshly generated nonce, nor the fact that the key K_{AB} is only known to A and B. Moreover, the capabilities of agents are not specified. For example, the intruder is assumed to be able to intercept and redirect messages; however it can decrypt a message only if has the appropriate key.

Last, there is no recognition that a protocol execution will take place in a broader context that includes other agent actions and other protocol executions. Nor does it take into account the interleaving of actions with the execution of a given instance of a protocol. For example, it is quite possible that a protocol could fail via what might be called a "stupidity attack". Consider the following exchange:

Another Attack on the Challenge-Response Protocol

1. $A \rightarrow I_B : \{N_A\}_{K_{AB}}$
1.1 $A \rightarrow I_B : N_A$
2. $I_B \rightarrow A : N_A$

[1] http://avispa-project.org/

In this case A sends the unencrypted nonce to the intruder. This of course is outlandish, but it nonetheless represents a logically possible compromise of the protocol (and in fact any other "secure" protocol). The point is that, much like the qualification problem in planning, there is an assumption that "nothing untoward happens" in a protocol execution. However, it may well be that there are "untoward happenings" much more subtle than the stupidity attack; consequently, it is desirable to have a framework for specifying protocols that is general enough to take such possibilities into account.

We argue that in order to provide a robust demonstration of the security and correctness of a protocol, all of the above points need to be addressed. We suggest that an explicit, logical formalisation in the situation calculus provides a suitable framework. Broadly speaking, our primary aim is to clearly formalize exactly what is going on in a cryptographic protocol in a declarative action formalism; such a formalization will provide a more flexible model of agent communication.

Approach

We present an outline of a formalization for cryptographic protocols, using the Challenge-Response protocol as an example. While we don't completely cover all points raised in the previous section, given space constraints, it should be clear that any omissions are easily addressable.

Vocabulary

We formalize message passing systems in the situation calculus. For our purposes, there are four main sorts of objects (beyond actions and situations): *agents, keys, messages* and *nonces*. In this section, we briefly describe each sort.

Agents: The term *agent* refers to both honest agents and to the malicious intruder. We reserve the term *principal* to refer to an honest agent. Variables a, a_1, ... range over agents. The constant $intr$ denotes the intruder. Unary predicates $Agent$ and $Intruder$ have their obvious meanings.

Fluent $Alive(a, s)$ indicates that a is alive in situation s. It is a precondition for executing any action; for brevity however we omit it in action preconditions. $Has(a, x, s)$ means that a has access to x in situation s, where the variable x ranges over messages, keys and nonces. This can be seen as a kind of knowledge, but we use the epistemically neutral term Has and interpret the meaning in terms of "access" to information. We use $Bel(a, f, s)$ to indicate that a believes that the fluent f is true in situation s. The semantics of Bel can be defined using the treatment of belief in (Scherl & Levesque 2003) (where they use $Knows$ for Bel).

Messages: Communication in our framework involves the exchange of messages. Variables $m, m_1, \ldots$ range over messages. Unary predicate Msg is true of messages. Messages are considered to be atemporal, and so are not indexed by a situation. Messages are composed of a finite sequence of parts, which may be nonces, agent names, or keys; each part may be encrypted. We assume an appropriate situation calculus axiomatization of lists, including the constructor $list(p_1, \ldots, p_n)$ and selectors $first(m)$, $second(m)$, etc. A useful state constraint[2] is that if an agent Has a message, then it has the message parts, for example:

$$Has(a, m, s) \wedge Msg(m) \supset Has(a, first(m), s).$$

Keys: Variables k, k_1, ... range over keys. Predicate $Key(k)$ indicates that k is a key, while $SymKey(k)$ and $AsymKey(k_1, k_2)$ have their expected meaning for symmetric and asymmetric keys respectively. $ShKey(a_1, a_2, k)$ indicates that k is a shared (symmetric) key for agents a_1, a_2. $PubKey(a, k)$ and $PrivKey(a, k)$ give public and private keys, respectively, of an agent.

Three functions are associated with keys: The value of $encKey(x)$ is the key which has been used to encrypt x. The value of $enc(x, k)$ is the result of encrypting x with k; and $dec(x, k)$ returns the corresponding decrypted message. The following state constraint relates enc and $encKey$; others (omitted here) relate public and private keys, etc.:

$$m_1 = enc(m_2, k) \ \supset \ k = encKey(m_1).$$

Nonces: Variables $n, n_1, \ldots$ range over nonces. The most important feature of nonces is that they must be *freshly generated* during the current protocol run. The fluent $IsFresh(n, s)$ is intended to be true if and only if the nonce n has been generated "recently" with respect to the situation s. To this end, the functional fluent $fresh(s)$ is used to model the generation of new nonces during a protocol run using the axiom $fresh(s) = fresh(s') \supset s = s'$.

Actions: There are two classes of action terms. The class of *basic actions* is comprised of actions for encryption and decryption, sending and receiving messages, and composing messages. These actions are described next. *Protocol-specific actions* are described later, in the section on representing a protocol in an action theory.

To ease readability we omit sort predicates. The variable conventions given above implicitly specify the sort of each variable. As usual, free variables are implicitly universally quantified.

1. $encrypt(a, x, k)$ – Agent a encrypts nonce or message x using key k.

   Precondition:

   $Poss(encrypt(a, x, k), s) \ \equiv \ (Has(a, x, s) \wedge$
   $(Has(a, k, s) \vee \exists a' PublicKey(a', k)))$

   Effect:

   $Has(a, enc(x, k), do(encrypt(a, x, k), s))$

2. $decrypt(a, x, k)$ – Agent a decrypts x using key k.

   Precondition:

   $Poss(decrypt(a, x, k), s) \ \equiv \ (Has(a, x, s) \wedge$
   $Has(a, k, s) \ \wedge \ [(SymKey(k) \ \wedge \ k = encKey(x)) \vee$
   $(AsymKey(k, k') \wedge k' = encKey(x))])$

[2] State constraints can be problematic, and are not part of a *basic action theory* (Reiter 2001). Nonetheless they are useful in a representational context, in initially specifying a theory.

Effect:

$Has(a, dec(x, k), do(decrypt(a, x, k), s))$

3. $send(a_1, a_2, m)$ – Agent a_1 sends m intended for a_2. The intruder can masquerade as the sender. Fluent $Sent$ indicates that a message is in some fashion "posted", that is can be received by an agent.

   Precondition:

   $Poss(send(a_1, a_2, m), s) \equiv$
   $((Has(a_1, m, s) \wedge a_1 \neq a_2) \vee Has(intr, m, s))$

   Effect:

   $Sent(a_1, a_2, m, do(send(a_1, a_2, m), s))$

4. $receive(a_1, a_2, m)$ – a_1 receives message m from a_2. The intruder can intercept messages. $\neg Sent$ indicates that the message is no longer available to be received.

   Precondition:

   $Poss(receive(a_1, a_2, m), s) \equiv (Sent(a_2, a_1, m, s) \vee$
   $(a_1 = intr \wedge \exists a' \, Sent(a_2, a', m, s)))$

   Effect:

   $Has(a_1, m, do(receive(a_1, a_2, m), s)) \wedge$
   $\neg Sent(a_2, a_1, m, do(receive(a_1, a_2, m), s)) \wedge$
   $Recd(a_1, a_2, m, do(receive(a_1, a_2, m), s))$

5. $compose(a, m, x)$ – Agent a composes message m having body x.

   Precondition:

   $Poss(compose(a, m, list(x_1, \ldots x_n)), s) \equiv$
   $(Has(a, x_1, s) \wedge \ldots Has(a, x_n, s))$

   Effect:

   $Has(a, m, do(compose(a, m, list(x_1, \ldots x_n)), s)) \wedge$
   $Msg(m) \wedge first(m) = x_1 \wedge second(m) = x_2 \wedge \ldots$

   Since messages in a protocol always have fixed length, an alternative is to have *compose* take message parts as arguments. Thus there would be a set of *compose* actions, one for each possible message length.

State Constraints

Some state constraints have already been mentioned. For proving properties about protocols, some *epistemic* constraints are useful, for example, an agent knows what actions it carried out. In the Challenge-Response protocol we use the following:

$$Sent(a_1, a_2, m, s) \supset Bel(a_1, Sent(a_1, a_2, m), s)$$
$$Recd(a_1, a_2, m, s) \supset Bel(a_1, Recd(a_1, a_2, m), s)$$

We can then state that if an agent a_1 sends a fresh nonce encrypted in the key it shares with a_2, and gets the unencrypted nonce back, then a_1 believes that a_2 is alive:

$$(Bel(a_1, Sent(a_1, a_2, en), s) \wedge en = enc(n, k) \wedge Fresh(n) \wedge ShKey(a_1, a_2, k) \wedge Bel(a_1, Recd(a_1, x, n), s)) \supset Bel(a_1, Alive(a_2), s)$$

Initial Situation

The initial situation contains information about the number of agents, their keys, etc. Since the details are straightforward, we just outline what is required. For example, using the $Agent$ predicate, a finite set of principals is specified, along with the intruder, $intr$. For each agent, we specify a combination of private, public, and shared keys. Typically, this is all we need to specify in the initial situation.

Adding Control Constraints

Parallelism is simulated by allowing concurrent interleaving of actions. We model a Dolev-Yao intruder through the following scheme, which allows the intruder to perform an arbitrary number of actions before an honest agent can act:

```
loop {
  Intruder executes some actions;
  A principal executes one action
}
```

This can be implemented in our action theory as follows; assume that fluent OkP isn't used in the theory. Informally OkP states that it is ok for a principal to execute an action. Basic actions are modified as follows:

- For a principal: Each precondition $Poss(a, s) \equiv \phi$ is modified to $Poss(a, s) \equiv (\phi \wedge OkP(s))$. Each effect axiom $\psi(do(a, s))$ is replaced by $\psi(do(a, s)) \wedge \neg OkP(do(a, s))$.
- Only the intruder can make $OkP(s)$ true. A new action $onOkP$ is introduced with precondition $Poss(onOkP(a), s) \equiv (a = intr)$ and effect $OkP(do(onOkP, s))$.

An advantages of this framework is that other models of concurrency can be easily expressed. For example, it is straightforward to specify that the intruder may carry out one action, followed by some agent carrying out an action. In this case, the intruder is limited in that it may not be able to compromise all protocol runs. On the other hand, there are some principal actions that an intruder cannot compromise, such as encryptions and decryptions. So from an efficiency standpoint it would make sense to allow an agent to execute a full sequence of such "uncompromisable" actions. To this end, a full implementation could make use of higher-level imperative constructs, such as a sequence of actions as given in Golog's *Do* (Levesque *et al.* 1997).

Representing a Protocol in an Action Theory

The goal of the preceding framework is to completely and explicitly specify a theory of agent communication involving encryption, freshly generated nonces, and a hostile intruder. In this setting, a protocol is regarded as a high-level description of prescribed agent actions, designed to achieve some goal in a dynamic, unpredictable, hostile environment. Hence there are two things that remain to be specified:

1. how the protocol corresponds to sets of agent actions, and
2. the goal of the protocol.

Compiling a Protocol into an Action Theory Our goal is to express a protocol such as the Challenge-Response protocol in terms of our action theory. Our ultimate goal is to *automate* this process, so that any protocol can be translated and integrated with our situation calculus theory. Hence the ultimate goal is to provide a *compiler* for protocols into action theories. At present we hand code a translation, giving the Challenge-Response protocol as an example below. We suggest via this example that a specification of a translator presents no great technical difficulty.

There are two general methodologies for translating a protocol specification into our action theory, corresponding to two levels of granularity:

1. Compile lines of a protocol into new, protocol-specific actions.
2. Compile each line of a protocol into two sequences of previously-defined, basic actions. The first sequence captures the implicit composition and sending of a message; while the second captures the implicit receipt and decrypting of a message.

We are currently implementing the first approach. Each line of a protocol is implicitly made up of two parts, the first involving the composition and sending of a message, and the second involving the receiving and decrypting of the message. Thus in the first line of the Challenge-Response protocol, the intent is that A compose a message and send it, followed by B receiving it and decrypting it. However, note that for every pair of successive lines in a protocol, the implicit $\mathit{receive}$ of one line can be combined with the send of the next. Thus in the Challenge-Response protocol, B's receiving of a message from A can be combined with a sending of an unencrypted nonce back to A. Hence a n-line protocol can compile into $n+1$ protocol-specific actions – one for the first line of the protocol, one for the last line, and one for each of the $n-1$ successive pair of lines. Thus the Challenge-Response protocol would compile into three new protocol-specific actions:

$CR.1.\mathit{send}$**:** Agent a_1 composes a message with a fresh nonce, encrypted in the key shared with a_2, and sends it to a_2.

$CR.1.\mathit{rec}.2.\mathit{send}$**:**[3] a_2 receives the message, decrypts it, and sends a message with the nonce to a_1.

$CR.2.\mathit{rec}$**:** a_1 receives the unencrypted nonce from a_2.

We introduce the following constants and fluents: $\langle \mathit{pid}\rangle$ is an identifier inserted by the compiler giving the protocol type and instance of the run. (We also use pid without angle brackets as a variable.) Predicate Type extracts the protocol type from its argument; here $\mathit{Type}(\mathit{pid}) = $ "CR". Fluent Expect expresses control knowledge, that after initiating a run of the protocol, a_1 expects at some point to receive a message from a_2 comprising the second step in this instance of the protocol. In this way, multiple instances of multiple protocols may be concurrently executed. Fluent $\mathit{Completed}$ indicates that the protocol has completed successfully.

We have the following action preconditions and effects:

$CR.1.\mathit{send}$:

Precondition:

$$\begin{aligned}&\mathit{Poss}(CR.1.\mathit{send}(a_1,a_2,m,k,n),s) \equiv\\ &\quad \mathit{ShKey}(a_1,a_2,k) \wedge n = \mathit{fresh}(s) \wedge\\ &\quad m = \mathit{list}(\langle \mathit{pid}\rangle, \mathit{enc}(n,k))\end{aligned}$$

Effect: Let $s' = do(CR.1.\mathit{send}(a_1,a_2,m,k,n),s)$.

$$\begin{aligned}&\mathit{Sent}(a_1,a_2,m,s') \wedge \mathit{Has}(a_1,m,s') \wedge \mathit{Has}(a_1,n,s') \wedge\\ &\quad \mathit{Has}(a_1,\mathit{enc}(n,k),s') \wedge \mathit{Expect}(a_1,a_2,\langle \mathit{pid}\rangle,2,s')\end{aligned}$$

$CR.1.\mathit{rec}.2.\mathit{send}$:

Precondition:

$$\begin{aligned}&\mathit{Poss}(CR.1.\mathit{rec}.2.\mathit{send}(a_2,a_1,m,m'),s) \equiv\\ &\quad \mathit{Sent}(a_1,a_2,m,s) \wedge \mathit{Type}(\mathit{first}(m)) = \text{"}CR\text{"} \wedge\\ &\quad \mathit{Has}(a_2,\mathit{encKey}(m),s) \wedge\\ &\quad m' = \mathit{list}(\mathit{first}(m),\mathit{dec}(\mathit{second}(m),\mathit{encKey}(m)))\end{aligned}$$

The precondition is cumbersome, reflecting the fact that several actions (including a receive and send) are combined into one protocol-specific action.

Effect: Let $s' = do(CR.1.\mathit{rec}.2.\mathit{send}(a_2,a_1,m,m'),s)$.

$$\begin{aligned}&\mathit{Recd}(a_2,a_1,m,s') \wedge \mathit{Has}(a_2,m,s') \wedge\\ &\quad \mathit{Has}(a_2,\mathit{first}(m),s') \wedge \mathit{Has}(a_2,\mathit{second}(m),s') \wedge\\ &\quad \mathit{Has}(a_2,\mathit{dec}(\mathit{second}(m),\mathit{encKey}(m)),s') \wedge\\ &\quad \neg\mathit{Sent}(a_1,a_2,m,s') \wedge \mathit{Sent}(a_2,a_1,m',s')\end{aligned}$$

The effect is likewise cumbersome: a_2 has the message and all its parts; the original message is marked as unavailable; and a new message is sent to a_1.

$CR.2.\mathit{rec}$:

Precondition:

$$\begin{aligned}&\mathit{Poss}(CR.2.\mathit{rec}(a_1,a_2,m),s) \equiv\\ &\quad \mathit{Sent}(a_2,a_1,m,s) \wedge \mathit{Type}(\mathit{first}(m)) = \text{"}CR\text{"} \wedge\\ &\quad \mathit{Expect}(a_1,a_2,\mathit{first}(m),2,s)\end{aligned}$$

Effect: Let $s' = do(CR.2.\mathit{rec}(a_1,a_2,m),s)$.

$$\begin{aligned}&\mathit{Recd}(a_1,a_2,m,s') \wedge \mathit{Has}(a_1,m,s') \wedge\\ &\quad \mathit{Has}(a_1,\mathit{first}(m),s') \wedge \mathit{Has}(a_1,\mathit{second}(m),s') \wedge\\ &\neg\mathit{Sent}(a_2,a_1,m,s') \wedge \mathit{Completed}(a_1,a_2,\mathit{first}(m),s')\end{aligned}$$

Expressing the Goal of a Protocol The goal of a protocol will often have epistemic components. For the Challenge-Response protocol, the overall goal is that if a protocol run successfully completes, then the initiating agent will believe that the responding agent is alive; and moreover, it is not possible that the initiating agent believe that the second agent is alive when in fact it is not. (That is, the initiating agent's belief is indeed knowledge.)

$$\begin{aligned}&(\mathit{Completed}(a_1,a_2,x,s) \wedge \mathit{Type}(x) = \text{"}CR\text{"}) \supset\\ &\quad (\mathit{Bel}(a_1,\mathit{Alive}(a_2),s) \equiv \mathit{Alive}(a_2,s))\end{aligned}$$

This assumes that principals are alive or dead on the network, independent of the situation. A more nuanced representation would take into account the possibility that an agent may become not Alive.

There are other parts to a successful protocol specification that need to be specified. First, it must be possible for there to be a successful run:

$$\exists s.\, \mathit{Completed}(a_1,a_2,x,s) \wedge \mathit{Type}(x) = \text{"}CR\text{"}$$

That is, a protocol that can never complete will vacuously

[3] The naming here is awkward, but is intended to be mnemonic for the protocol name (CR), along with the receive part of one line ($1.\mathit{rec}$) and the send part of the next ($2.\mathit{send}$).

never be compromised, but is of no use. Second, it would be desirable to prove that if the intruder carries out no actions, then the protocol is guaranteed to succeed.

The Attack on the CR Protocol

We now illustrate the approach by describing the attack on the Challenge-Response protocol:[4]

1. Agent a_1 initiates a round of the protocol with action:
   $CR.1.send(a_1, a_2, (\text{"}CR\text{"}, enc(n, k)), k, n)$
   One effect is $Sent(a_1, a_2, (\text{"}CR\text{"}, enc(n, k)))$
2. The intruder intercepts the sent message:
   $receive(intr, a_2, (\text{"}CR\text{"}, enc(n, k)))$
3. The intruder sends a message to a_1, masquerading as a_2:
   $send(a_2, a_1, (\text{"}CR\text{"}, enc(n, k)))$
4. The message is received by a_1 who understands it as an initiation of a new round of the CR protocol by a_2, and so responds with:
   $CR.1.rec.2.send(a_1, a_2, (\text{"}CR\text{"}, enc(n, k)), (\text{"}CR\text{"}, n))$
   This action has an effect $Sent(a_1, a_2, (\text{"}CR\text{"}, n))$.
5. The intruder intercepts this message:
   $receive(intr, a_1, (\text{"}CR\text{"}, n))$.
   This has effects $Has(intr, (\text{"}CR\text{"}, n))$, $Has(intr, n)$.
6. The intruder sends the nonce to a_1, masquerading as a_2:
   $send(a_2, a_1, (\text{"}CR\text{"}, n))$
7. The message is received by a_1:
   $CR.2.rec(a_1, a_2, (\text{"}CR\text{"}, n))$
   a_1 understands it as the completion of the original protocol; thus a_1 believes a_2 alive in the resulting situation.

Discussion

Thus far our focus has been on the development of an appropriate situation calculus formalization of cryptographic protocols. Our formalism is highly elaboration tolerant, in the sense that it is easy to axiomatize agents and intruders with different capabilities. For example, if we had information about the topology of a particular network, it would be easy to restrict an intruder to only intercept messages between particular principals. In most existing logical approaches to protocol verification, it is not straightforward to modify agent capabilities for a specific application.

In many cases, proofs of protocol correctness rely on the assumption that honest agents do not perform actions that compromise secret information; however, it is not always clear which actions are safe in this sense. In our framework, we can discover these undesirable actions and we can formally specify axioms that restrict honest agents from performing them. To the best of our knowledge, this problem has not been addressed in related formalisms.

There are two natural directions for future work on our framework. First, as noted previously, we would like to be able to directly compile protocol specifications into situation calculus theories. At present, we perform this encoding by hand; it would be desirable to automate this process, thereby facilitating the analysis of a wider range of protocols. The second direction is to implement a system for automatically finding attacks based on our situation calculus formalization. Our intention is to implement the system using ConGolog.

References

Aiello, L., and Massacci, F. 2001. Verifying security protocols as planning in logic programming. *ACM Transactions on Computational Logic* 2(4):542–580.

Armando, A.; Compagna, L.; and Lierler, Y. 2004. Automatic compilation of protocol insecurity problems into logic programming. In Alferes, J., and Leite, J., eds., *JELIA'04*, volume 3239 of *LNAI*, 617–627.

Brackin, S.; Meadows, C.; and Millen, J. 1999. CAPSL interface for the NRL protocol analyzer. In *Proceedings of ASSET 99*. IEEE Computer Society Press.

Burrows, M.; Abadi, M.; and Needham, R. 1990. A logic of authentication. *ACM TOCS* 8(1):18–36.

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

Fagin, R.; Halpern, J. Y.; Moses, Y.; and Vardi, M. Y. 1995. *Reasoning about Knowledge*. Cambridge, MA: The MIT Press.

Gelfond, M., and Lifschitz, V. 1991. Classical negation in logic programs and deductive databases. *New Generation Computing* 9:365–385.

Halpern, J., and Pucella, R. 2003. On the relationship between strand spaces and multi-agent systems. *CoRR* cs.CR/0306107.

Hernández-Orallo, J., and Pinto, J. 1997. Formal modelling of cryptographic protocols in situation calculus. (Published in Spanish as: Especificación formal de protocolos criptográficos en Cálculo de Situaciones, *Novatica*, 143, pp. 57-63, 2000).

Levesque, H.; Reiter, R.; Lin, F.; and Scherl, R. 1997. Golog: A logic programming language for dynamic domains. *Journal of Logic Programming* 31.

Levesque, H.; Pirri, F.; and Reiter, R. 1998. Foundations for the situation calculus. *Linköping Electronic Articles in Computer and Information Science* 3(18).

Reiter, R. 2001. *Knowledge in Action: Logical Foundations for Specifying and Implementing Dynamical Systems*. Cambridge, MA: The MIT Press.

Scherl, R., and Levesque, H. 2003. Knowledge, action, and the frame problem. *Artificial Intelligence* 144(1-2):1–39.

Thayer, F.; Herzog, J.; and Guttman, J. 1999. Strand spaces: Proving security protocols correct. *JCS* 7(1).

Wang, S., and Zhang, Y. 2008. A logic programming based framework for security protocol verification. In *Proc. ISMIS 2008*, volume 4994 of *LNAI*, 638–643. Springer.

[4] We suppress situation arguments in fluents for readability.