A Machine Checkable Logic of Knowledge for Specifying Security Properties of Electronic Commerce Protocols *

Edmund Clarke Somesh Jha Will Marrero

Abstract

A number of researchers have proposed various tools for checking security protocols. Most of these tools work by comparing the set of possible traces (as expressed by some model of computation) to the set of correct traces (often expressed as a set of relationships between events in a trace). In this paper we propose a new logic of knowledge in which one can express relationships between events, variables, and knowledge. This logic has a precise semantics with respect to a well defined model of computation and can be checked automatically. In addition, the fact that we can express properties about knowledge allows us to express security properties specific to electronic commerce, including anonymity.

1 Introduction

Computer security has become a very popular topic recently due to the rapid growth of such entities as "the Internet" and "the World Wide Web." As more and more people gain access to these resources, and as more services are offered, the importance of being able to provide security guarantees becomes paramount. Typically, these guarantees are provided by means of security protocols that make use of encryption. A number of researchers have proposed techniques to analyze these protocols in an attempt to find errors or to prove them correct.

One of the first attempts at formalizing the notion of a correct protocol was the Logic of Authentication, more commonly known as the BAN logic [3]. This logic proved useful in analyzing security protocols. Kindred and Wing helped to automate the use of this logic by developing a theory generator for it [5]. However, one of the drawbacks to the logic is the lack of a formal model with which to define the semantics of the logic.

*This research is sponsored by the National Science Foundation (NSF) under Grant No. CCR-9505472 and the Defense Advanced Research Projects Agency (DARPA) under Contract No. DABT63-96-C-0071. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of NSF, DARPA, or the United States Government.

There has been much work recently on formal models for security protocols. A number of researchers have used general purpose model checkers to verify authentication protocols [6, 7, 8]. In all cases, the authors must specify the "bad traces" and check to see if any of them are valid traces of the model. In [4], we provide a special purpose model checking tool for verifying authentication protocols which has a built in adversary which can construct new messages when trying to subvert a protocol.

Bella and Paulson have used theorem proving to verify authentication protocols [1]. Their method requires that one express the set of all possible traces by providing a set of rules that describe how to extend a valid trace. One then describes the relationships between events that must hold true of correct traces using the same syntax, and Isabelle tries to prove that all valid traces are also correct traces.

What seems to be missing from all of these tools, is a logic that can be used to specify the required properties and which also has a precise semantics. In this paper, we propose such a logic. This logic can be used to express relationships between events and between the variable bindings belonging to the different agents as before. In addition, however, we can now express properties involving **knowledge**. Because the manipulation of knowledge (messages) is built into our tool, this kind of checking is straightforward. We believe that similar extensions could be made to the work of Bella and Paulson, for similar reasons. However, it might prove difficult to do this for tools in which the evolution of an adversary's knowledge must be explicitly encoded in the model description.

The rest of this paper is organized as follows. In section 2 we review the most common way in which messages are modelled when verifying security protocols.. Sections 3 and 4 describe the computation model which we use to provide the semantics for the logic. This model is closely based on our actual tool. A more detailed description of our tool appears in [4]. The heart of our current work can be found in sections 5 and 6 in which we present the syntax and the semantics of our specification logic. Section 7 provides an example of how specifications could be written in this logic using the 1KP electronic commerce protocol [2] as a reference point. Section 8 concludes with some final observations and directions for future work.

2 Messages

Typically, the messages exchanged during the run of a protocol are constructed from smaller sub-messages using pairing and encryption. The smallest such sub-messages (i.e. they contain no sub-messages themselves) are called *atomic messages*. There are four kinds of *atomic messages*.

- *Keys* are used to encrypt messages. Keys have the property that every key k has an inverse k^{-1} such that for all messages m, $\{\{m\}_k\}_{k^{-1}} = m$. (Note that for symmetric cryptography the decryption key is the same as the encryption key, so $k = k^{-1}$.)

- *Principal names* are used to refer to the participants in a protocol.
- *Nonces* can be thought of as randomly generated numbers. The intuition is that no one can predict the value of a nonce; therefore, any message containing a nonce can be assumed to have been generated after the nonce was generated. (It is not an "old" message.)
- *Data* which plays no role in how the protocol works but which is intended to be communicated between principals.

Let $\mathcal{A}$ denote the space of *atomic messages*. The set of all messages $\mathcal{M}$ over some set of atomic messages $\mathcal{A}$ is defined inductively as follows:

- If $a \in \mathcal{A}$ then $a \in \mathcal{M}$. (Any *atomic message* is a message.)
- If $m_1 \in \mathcal{M}$ and $m_2 \in \mathcal{M}$ then $m_1 \cdot m_2 \in \mathcal{M}$. (Two messages can be paired together to form a new message.)
- If $m \in \mathcal{M}$ and key $k \in \mathcal{A}$ then $\{m\}_k \in \mathcal{M}$. (A message M can be encrypted with key k to form a new message.)

We would also like to generalize the notion of messages to *message templates*. A message template can be thought of as a message containing one or more message variables. To extend messages to message templates we add the following to the inductive definition of messages:

- If v is a message variable, then $v \in \mathcal{M}$.

Because keys have inverses, we take this space modulo $\{\{m\}_k\}_{k^{-1}} = m$. It is also important to note that we make the following perfect encryption assumption. The only way to generate $\{m\}_k$ is from m and k. In other words, for all messages m, m_1, and m_2 and keys k, $\{m\}_k \neq m_1 \cdot m_2$, and $\{m\}_k = \{m'\}_{k'} \Rightarrow m = m' \wedge k = k'$.

We also need to consider how new messages can be created from already known messages by encryption, decryption, pairing (concatenation), and projection. The following rules capture this relationship by defining how a message can be derived from some initial set of information I.

1. If $m \in I$ then $I \vdash m$.
2. If $I \vdash m_1$ and $I \vdash m_2$ then $I \vdash m_1 \cdot m_2$. (pairing)
3. If $I \vdash m_1 \cdot m_2$ then $I \vdash m_1$ and $I \vdash m_2$. (projection)
4. If $I \vdash m$ and $I \vdash k$ for key k, then $I \vdash \{m\}_k$. (encryption)
5. If $I \vdash \{m\}_k$ and $I \vdash k^{-1}$ then $I \vdash m$. (decryption)

This defines the most common derivability relation used to model the capabilities of the adversary in the literature. Given some base set of messages I, we can define all the messages that can be derived from I as $\overline{I}$, the closure of I under the rules above. For example, if I_0 is some finite set of messages overheard by the adversary, then $\overline{I_0}$ represents the set of all messages known to the adversary.

In general, $\overline{I}$ is infinite, but researchers have taken advantage of the fact that one need not actually compute $\overline{I}$. It suffices to check $m \in \overline{I}$ for some finite number of messages m. However, checking if $m \in \overline{I}$ must still be decidable. For a detailed discussion of this question, see [4].

3 The Model

We model a protocol by the asynchronous composition of a set of named communicating processes which model the honest agents and the adversary. We would like to model an insecure and lossy communication medium, in which a principal has no guarantees about the origin of a message, and where the adversary is free to eavesdrop on all communications. Therefore, in the model, we insist that all communications go through the adversary. In other words, all sent messages are intercepted by the adversary and all messages received by honest agents were actually sent by the adversary. In addition, the adversary is allowed to create new messages from the information it gains by eavesdropping, in an attempt to subvert the protocol.

In order to make the model finite, we must place a bound on the number of sessions that a principal may attempt. A session will be modelled as an *incarnation* of a principal's role in the protocol. Each incarnation is a separate copy or instantiation of a principal and consists of a single execution of the sequence of actions that make up that agent's role in the protocol, along with all the variable bindings and knowledge acquired during the execution. An agent can have multiple incarnations, but each incarnation is executed once. When we combine these with a single incarnation of the adversary, we get the entire model for the protocol.

Each incarnation of an honest principal is modelled as a 5-tuple $\langle N, S, I, B, P\rangle$ where:

- $N \in$ *names* is the name of the principal.
- S is a unique *session ID* for this incarnation.
- B: $vars(N) \rightarrow \mathcal{M}$ is a set of bindings for $vars(N)$, the set of variables appearing in principal N, which are bound for a particular session as it receives messages.
- $I \subseteq \mathcal{M}$ is the set of messages known to the principal to the session S.
- P is a process description (similar in style to CSP) given as a sequence of actions to be performed. This actions include the pre-defined actions

send and receive. as well as user defined internal actions such as commit and debit.

The model of the adversary, Ω has some similarities; however, the adversary is not bound to follow the protocol and so it doesn't make sense to include either a sequence of actions P_Ω or a set of bindings B_Ω for the adversary. Instead, at any time, the adversary can receive any message or it can send any message it can generate from its set of known messages I_Ω.

The global model is then the asynchronous composition of the models for each session, including the adversary. Each possible execution of the model corresponds to a *trace*. a finite, alternating sequence of global states and actions $\pi = \sigma_0\alpha_1\sigma_1\alpha 1 \cdots \alpha_n\sigma_n$ for some $n \in \mathbb{N}$, such that $\sigma_{i-1} \xrightarrow{\alpha_i} \sigma_i$ for $0 < i \leq n$ and for the transition relation $\rightarrow$ defined in section 4.

4 Actions

The actions allowed during the execution of a protocol include the two predefined actions SEND and RECEIVE as well as possibly some user defined actions. The model transitions between global states as a result of actions taken by the incarnations. More formally, we define a transition relation $\rightarrow \subseteq \Sigma \times S \times A \times \mathcal{M} \times \Sigma$ where Σ is the set of global states, S again is the set of session IDs, A is the set of action names (which includes **Send** and **Rec**), and $\mathcal{M}$ is the set of all possible messages. As at the end of section 3, we will use the notation $\sigma \xrightarrow{s \cdot a \cdot m} \sigma'$ in place of $(\sigma, s, a, m, \sigma') \in \rightarrow$ when it is more convenient. In the definitions below, we will denote the adversary as $\Omega = \langle N_\Omega, S_\Omega, \phi, I_\Omega, \phi \rangle$ and the incarnations as $\Psi_i = \langle N_i, S_i, B_i, I_i, P_i \rangle$. We will use $\sigma = \langle \Omega, \Psi_1, \ldots, \Psi_n \rangle$ to denote the global state before the transition and $\sigma' = \langle \Omega', \Psi'_1, \ldots, \Psi'_n \rangle$ to denote the global state after the transition. In addition, we will use the notation $\hat{B}$ to denote the obvious extension of a set of bindings B from the domain of variables to the domain of message templates. In other words, $\hat{B}(m)$ is result of substituting $B(v)$ for v in the message template m for all the variables v appearing in m.

- $\sigma \xrightarrow{s \cdot \mathbf{Send} \cdot m} \sigma'$

  An incarnation with session ID s can send message m in global state σ and the new global state is σ' if and only if

  1. $I_{\Omega'} = I_\Omega \cup m$. (The adversary adds m to the set of messages it knows.)
  2. There is an incarnation $\Psi_i = \langle N_i, s, B_i, I_i, \text{SEND}(s\text{-}msg).P'_i \rangle$ in σ such that in σ'. $\Psi'_i = \langle N_i, s. B_i, I_i, P'_i \rangle$ and $m = \hat{B}_i(s\text{-}msg)$. (There is an incarnation that is ready to send message m.)
  3. $\Psi_j = \Psi'_j$ for all $j \neq i$. (All other incarnations remain unchanged.)

- $\sigma \xrightarrow{s \cdot \mathbf{Rec} \cdot m} \sigma'$

An incarnation with session ID s can receive message m in global state σ and the new global state is σ' if and only if

1. $m \in \overline{I_\Omega}$. (The adversary can generate the message m.)
2. There is an incarnation $\Psi_i = \langle N_i, s, B_i, I_i, \text{RECEIVE}(r\text{-}msg).P_i' \rangle$ in σ such that in σ'. $\Psi_i' = \langle N_i, s, B_i', I_i', P_i' \rangle$, $I_i' = I_i \cup m$, and B_i' is the smallest extension of B_i such that $\hat{B'}_i(r\text{-}msg) = m$. (There is an incarnation ready to receive a message of the form of m and the its bindings are updated correctly in the next state.)
3. $\Psi_j = \Psi_j'$ for all $j \neq i$. (All other incarnations remain unchanged.)

- $\sigma \xrightarrow{s \cdot A \cdot m} \sigma'$

  A incarnation with session ID s can perform some user defined internal action A with argument m in global state σ and the new global state is σ' if and only if

  1. There is an incarnation $\Psi_i = \langle N_i, s, B_i, I_i, A(msg).P_i' \rangle$ in σ such that in σ', $\Psi_i' = \langle N_i, s, B_i, I_i, P_i' \rangle$ and $m = \hat{B}_i(msg)$. (There is an incarnation s that is ready to perform action A with argument m.)
  2. $\Psi_j = \Psi_j'$ for all $j \neq i$. (All other incarnations remain unchanged).

5 Syntax

We will use a first order logic where quantifiers range over the finite set of incarnations. The atomic propositions are used to characterize states, actions, and knowledge in the model. The arguments to the atomic propositions are terms expressing incarnations or messages. We begin by a formal description of terms.

- If S is a session ID, then S is an incarnation term.
- If s is an incarnation variable, then s is an incarnation term.
- If M is a message, then M is a message term.
- If m is a message variable, then m is a message term.
- If s is an incarnation term, then $pr(s)$ is a message representing the principal that is executing incarnation s.
- If s is an incarnation term and m is a message variable then $s.m$ is a message term representing the binding of m in the incarnation s.
- If m_1 and m_2 are message terms, then $m_1 \cdot m_1$ is a message term.
- If m_1 and m_2 are message terms, then $\{m_1\}_{m_2}$ is a message term.

As in standard first order logic, atomic propositions are constructed from terms using relation symbols. The predefined relation symbols are "=", "**Knows**", "**Send**", and "**Rec**". Just as **Send** and **Rec** correspond to the send and receive actions in the model, the user can define other relation symbols which would correspond to user defined actions in the model.

The syntax for atomic propositions is as follows. (All relation symbols are used infix.)

- If m_1 and m_2 are message terms, then $m_1 = m_2$ is an atomic proposition.
- If s is an incarnation term and m is a message term, then s **Knows** m is an atomic proposition which intuitively means that incarnation s knows the message m.
- If s is an incarnation term, m is a message term, and **Act** is an action relation symbol, then s **Act** m is an atomic proposition which intuitively means that incarnation s performed action **Act** with message m as an argument.

Finally, well-formed formulas (wffs) are built up from atomic propositions with the usual connectives from first-order logic.

- if f is an atomic proposition, then f is a wff.
- if f is a wff, then $\neg f$ is a wff.
- if f_1 and f_2 are wffs, then $f_1 \wedge f_2$ is a wff.
- if f is a wff and s is an incarnation variable, then $\forall s.f$ is a wff.

We also use the following common shorthand:

- $f_1 \vee f_2 \equiv \neg(\neg f_1 \wedge \neg f_2)$
- $f_1 \rightarrow f_2 \equiv \neg f_1 \vee f_2$
- $f_1 \leftrightarrow f_2 \equiv f_1 \rightarrow f_2 \wedge f_2 \rightarrow f_1$
- $\exists s.f \equiv \neg\forall s.\neg f$

6 Semantics

Again, we begin with the terms of the logic.

- A session ID S refers to the incarnation in the model with that session ID.
- A incarnation variable s ranges over all the session IDs in the model.
- An atomic message M is an atomic message in the model.

- A message variable v varies over messages in the model and can be defined as a binding variable in a particular principal.
- The function pr maps session IDs to principal names. If s is a session ID, then $pr(s)$ is the principal executing the incarnation with session ID s.
- We use "." as a scoping operator. If s an incarnation term and v is a message variable, then $s.v$ refers to the variable v bound in incarnation s. The interpretation $\sigma(s.v)$ of $s.v$ in a particular state σ is $B_s(v)$, the value bound to the variable v in incarnation s in state σ.
- Message terms can be concatenated using "·" just as messages are concatenated.
- Similarly a message term m_1 can be encrypted with another message term m_2 just as messages are encrypted in the model.

The wffs of the logic will be interpreted over the traces of a particular model. Recall that a trace consists of a finite, alternating sequence of states and actions $\pi = \sigma_0\alpha_1\sigma_1 \ldots s_n$. We give the semantics of wffs in our model via a recursive definition of the satisfaction relation $\models$. We will write $\langle \pi, i \rangle \models f$ to mean that the ith state in π, satisfies the formula f. We begin with atomic propositions.

- $\langle \pi, i \rangle \models m_1 = m_2$ iff $\sigma_i(m_1) = \sigma_i(m_2)$ (i.e. the interpretation of m_1 and the interpretation of m_2 in the state σ_i are indeed equal).
- $\langle \pi, i \rangle \models s$ **Knows** m iff $\sigma_i(m) \in \overline{I_j}$ for some incarnation Ψ_j in σ_i such that $S_j = s$ (i.e. the incarnation with session ID s can derive the message m from its known set of messages in the state σ_i).
- $\langle \pi, i \rangle \models s\ A\ m$ for some action A (including the pre-defined actions **Send** and **Rec**) iff $\alpha_i = s \cdot A \cdot m$ (i.e. the transition into state σ_i was taken because the incarnation with session ID s took action A with argument m).

The extension of the satisfaction relation to the logical connectives is the same as for standard first order logic. We use the notation $[f/s \mapsto s_0]$ to denote the result of substituting every free occurrence of the incarnation variable s in f with the session ID s_0.

- $\langle \pi, i \rangle \models \neg f$ iff $\langle \pi, i \rangle \not\models f$.
- $\langle \pi, i \rangle \models f_1 \wedge f_2$ iff $\langle \pi, i \rangle \models f_1$ and $\langle \pi, i \rangle \models f_2$
- $\langle \pi, i \rangle \models \forall s.f$ iff $\langle \pi, i \rangle \models [f/s \mapsto s_0]$ for all sessions s_0 in the model.

7 Example

We now demonstrate how this system might be used on an example. We will look at the 1KP Protocol from [2]. In the protocol description below, "C" refers to the customer, "M" to the merchant, and "A" to the credit card authority.

- **Basic Fields**

  DESC is the description of the goods being purchased.

  PRICE is the previously agreed upon price.

  CC# is the customer's credit card number.

  $\mathcal{E}_A$ is the authority's public key.

  $\mathcal{H}$ is a hash function.

  R_C is a nonce generated by the customer.

  ID_M is the merchant's ID.

  TID_M is a transaction ID.

  SALT_C is a random number generated by the customer.

- **Starting Information:**

  Customer: DESC, PRICE, CC#, $\mathcal{E}_A$

  Merchant: DESC, PRICE, $\mathcal{E}_A$

- **Composite Fields:**

  $\mathrm{CID} = \mathcal{H}(R_C, \mathrm{CC\#})$

  $\mathrm{Common} = \mathrm{PRICE}, \mathrm{ID}_M, \mathrm{TID}_M, \mathrm{DATE}, \mathrm{NONCE}_M, \mathrm{CID}, \mathcal{H}(\mathrm{DESC}, \mathrm{SALT}_C)$

  $\mathrm{SLIP} = \mathrm{PRICE}, \mathcal{H}(\mathrm{Common}), \mathrm{CC\#}, R_C$

  $\mathrm{Clear} = \mathrm{ID}_M, \mathrm{TID}_M, \mathrm{DATE}, \mathrm{NONCE}_M, \mathcal{H}(\mathrm{Common})$

- **Protocol Flows:**

  1. Initiate:
     $C \rightarrow M$: $\mathrm{SALT}_C, \mathrm{CID}$
  2. Invoice:
     $M \rightarrow C$: $\mathrm{ID}_M, \mathrm{TID}_M, \mathrm{DATE}, \mathrm{NONCE}_M, \mathcal{H}(\mathrm{Common})$
  3. Payment:
     $C \rightarrow M$: $\mathcal{E}_A(\mathrm{SLIP})$
  4. Auth-Request:
     $M \rightarrow A$: $\mathrm{Clear}, \mathcal{H}(\mathrm{DESC}, \mathrm{SALT}_C), \mathcal{E}_A(\mathrm{SLIP})$
  5. Auth-Response:
     $A \rightarrow M$: $\mathrm{Y/N}, \mathcal{S}_A(\mathrm{Y/N}, \mathcal{H}(\mathrm{Common}))$

6. Confirm:
   $M \rightarrow C$: Y/N, $\mathcal{S}_A(\text{Y/N}.\mathcal{H}(\text{Common}))$

We first construct a model of each agent in the protocol using the incarnation model we discussed in section 3. Each can be characterized as a sequence of messages it sends and messages it expects to receive. Again, [4] contains a detailed example of how to write a model from a protocol description. In addition we might include extra actions for things such as money transfers and commit points. For example, the credit card authority may take a DEBIT action immediately before replying with the Auth-Response in message 5. The customer may take a COMMIT action immediately before sending the payment information in message 3.

Next, we specify the properties we expect the protocol to satisfy in the logic we have described. We now consider some of the properties the designers of the 1KP protocol discuss in their paper.

- *Proof of Transaction Authorization by Customer.* When the credit card authority debits a certain credit card account by a certain amount, the authority must be in possession of an unforgeable proof that the owner of the credit card has authorized the payment. We will assume that the signed slip $\mathcal{E}_A(\text{SLIP})$ provides this proof. This is somewhat indirect; we are not proving that $\mathcal{E}_A(\text{SLIP})$ is actually such a proof. We can express this property as follows:

$$\forall A_0 \,.\, pr(A_0) = A \wedge A_0 \text{ DEBIT } (A_0.CC\# \cdot A_0.PRICE) \rightarrow$$
$$A_0 \textbf{ Knows } \mathcal{E}_A(\text{SLIP})$$

  This formula states that for all incarnations A_0, if A_0 is an incarnation of the authority, and A_0 debits the credit card account $CC\#$ by $PRICE$, then A_0 can produce (knows) the appropriate purchase slip.

- *Unauthorized Payment is Impossible.* We will interpret this to mean that whenever the customer's account is debited, the customer must be in a state consistent with it having made a purchase whose price corresponds to the debited amount.

$$\forall A_0 \,.\, pr(A_0) = A \wedge A_0 \text{ DEBIT } (A_0.CC\# \cdot A_0.PRICE) \rightarrow$$
$$\exists C_0 \,.\, pr(C_0) = C \wedge C_0.PRICE = PRICE$$

  This formula states that for all incarnations A_0 if A_0 is an incarnation of the authority A, and A_0 debits the credit card account $CC\#$ by $PRICE$, then there exists an incarnation C_0 of the customer C and the customer's price $C_0.PRICE$ is equal to the debited price $PRICE$.

- *Privacy.* The order information and credit card numbers should not be revealed. In other words, only the appropriate principals should know the order information and the credit card number.

  $$\forall S\,.\forall C_0\,.\,(pr(C_0) = C) \wedge (S\ \mathbf{Knows}\ C_0.DESC) \rightarrow [pr(S) = C \vee pr(S) = M]$$

  $$\forall S\,.\forall C_0\,.\,(pr(C_0) = C) \wedge (S\ \mathbf{Knows}\ C_0.CC\#) \rightarrow [pr(S) = C \vee pr(S) = A]$$

  This first formula states that for all incarnations S, if S is knows the customer's description of the transaction, then S is an incarnation of either the customer or the merchant.

  The second states that for all incarnations S, if S knows the customer's credit card number, then S is an incarnation of either the customer or the authority.

- *Anonymity.* We may also want to insure that the merchant does not gain knowledge of the identity of the customer. Assuming the merchant doesn't know the name of the customer before execution of the protocol, then the following specification would guarantee anonymity.

  $$\forall S\ .\ (S\ \mathbf{Knows}\ C) \rightarrow [pr(S) = C \vee pr(S) = A]$$

  This formula states that for all incarnations S, if S knows the customer's name C, then S must be an incarnation of the custormer C or the authority A.

8 Conclusion

In this paper we have proposed a logic in which we can express relationships between messages, variables, actions, and knowledge. This logic has a precise semantics with respect to a well defined model of computation. We are currently implementing this extension to our already existing model checker. As before, because we consider only finite models, it is seems clear that the decision procedure we suggest terminates.

The 1KP example discussed in the paper suggests that reasoning about knowledge is useful, especially for electronic commerce protocols. To our knowledge, no one else has suggested a definition of anonymity which can be checked automatically. In addition, being able to specify requirements in a language that is different from the one in which the protocol is modelled suggests an added measure of confidence for the verification.

The most obvious area of future work is to finish implementing the ideas presented in this paper. Currently, it is not clear how much the complexity of the model checker will increase with the introduction of this general logic. It should also be interesting to test how this kind of extension might work with a theorem proving system like the one built by Bella and Paulson.

References

[1] G. Bella and L. C. Paulson. Using isabelle to prove properties of the kerberos authentication system. In *DIMACS Workshop on Design and Formal Verification of Security Protocols*, 1997.

[2] M. Bellare, J. Garay, R. Hauser, A. Herberg, H. Krawczyk, M. Steiner, G. Tsudik, and M. Waidner. *i*KP - a family of secure electronic payment protocols. In *Proceedings of the 1st USENIX Workshop on Electronic Commerce*, July 1995.

[3] M. Burrows, M. Abadi, and R. Needham. A logic of authentication. Technical Report 39, DEC Systems Research Center, February 1989.

[4] 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. To appear.

[5] D. Kindred and J. M. Wing. Closing the idealization gap with theory generation. In *DIMACS Workshop on Design and Formal Verification of Security Protocols*, 1997.

[6] G. Lowe. Casper: A compiler for the analysis of security protocols. In *Proceedings of the 1997 IEEE Computer Society Symposium on Research in Security and Privacy*, pages 18–30, 1997.

[7] J. C. Mitchell, M. Mitchell, and U. Stern. Automated analysis of cryptographic protocols using murϕ. In *Proceedings of the 1997 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, 1997.

[8] A. W. Roscoe. Intensional specifications of security protocols. In *9th Computer Security Foundations Workshop*, 1996.