Invariant Generation Techniques in Cryptographic Protocol Analysis

Catherine Meadows
Code 5543
Naval Research Laboratory
Washington, DC 20375
meadows@itd.nrl.navy.mil

Abstract

The growing interest in the application of formal methods of cryptographic protocol analysis has led to the development of a number of different techniques for generating and describing invariants that are defined in terms of what messages an intruder can and cannot learn. These invariants, which can be used to prove authentication as well as secrecy results, appear to be central to many different tools and techniques. However, since they are usually developed independently for different systems, it is often not easy to see what they have in common with each other, or to tell whether or not they can be used in systems other than the ones for which they were developed. In this paper we attempt to remedy this situation by giving an overview of several of these techniques, discussing their relationships to each other, and developing a simple taxonomy. We also discuss some of the implications for future research.

1 Introduction

Recently, a considerable body of work has grown up around the problem of applying formal methods to cryptographic protocols. Much of this work has concentrated on the problem of developing inductive techniques for reasoning about the unbounded sets of messages that can arise when dealing with an environment in which an arbitrary number of protocol executions can take place, and an intruder can apply an arbitrary number of operations to data. Normally, this is done by constructing an invariant such that actions by protocol participants leave the invariant unchanged. Examples include the languages used with the NRL Protocol Analyzer [10], the rank functions used by Schneider in his CSP based analysis [15], the ideals associated with strand spaces [18], Paulson's **analyz** and **synth** functions[14], and Cohen's secrecy invariant [4].

Not surprisingly, all of these techniques appear to have much in common. The feature that seems most widely shared is that many are characterized in terms of sets of messages that have properties that are left unchanged by operations by the intruder or by legitimate participants in the protocols.

These properties fall into two broad classes. One set has to do with characterizing terms that either can not be learned by the intruder, or can only be learned under special conditions. These include strand space ideals [18], the NRL Protocol Analyzer languages [10], and Schneider's terms of rank zero or less in [15].

The other has to do with the characterization of terms that the intruder can learn. These include the terms of rank one as constructed by Heather and Schneider in their Rank-analyzer model [7], the terms satisfying the unary predicate ok in Cohen's TAPS program, and to some extent the Paulson's analyz and synth functions. We have found that the first type falls naturally into an algebraic structure similar to that of the ideal used in ring theory (indeed, the term "ideal" used in the strand space model is inspired by the similarity to ring-theoretic ideals) while the second can often be described in terms of a subalgebra of the original crypto-algebra.

In this paper we will examine the types of invariants used by seven systems: the NRL Protocol Analyzer, the strand space model, an early and a late version of Schneider's rank model, Paulson's inductive model, a combination of Paulson's inductive model and the strand space model used by Millen and Ruess to prove a general theorem about secrecy, and Cohen's TAPS. We show how the invariants fit into the different classes, what important features they have in common, and in what important features they differ.

The remainder of the paper is organized as follows. In Section 2 we give a general discussion of models and techniques used in the application of formal methods to cryptographic protocol analysis. In Section 3 we discuss invariants defined in terms of messages not learnable by the intruder. In Section 4 we give a survey of invariants defined in terms of messages that can be learned by the intruder. In Section 5 we conclude the paper and discuss some implications of our findings and directions for future research.

2 Models and Techniques: A General Discussion

2.1 Protocol Models

From the point of view of our analysis, all the systems we are examining use essentially the same model. A protocol consists of a number of *honest* principals who attempt to send and receive messages to and from each other. Each protocol defines a finite set of *roles*, each of which consists of finite set of sequential rules governing the sending of messages and the processing of received message for the principal adopting that role. However, any message may be intercepted by the *intruder* and possibly redirected or altered. The intruder keeps a *knowledge set* of all messages that it has learned. Any message that is sent by an honest principal is added to the intruder's knowledge set. Likewise, the intruder can add to its knowledge set by performing operations on messages that are already in its knowledge set. These include at least concatenation, deconcatenation, decryption, and encryption (both public and shared key). This list may be extended, but this basic set is constant to all models. Operations may also be subject to typing restrictions; for example if a term X is encrypted with a term Y, we may be required to assume that Y is of type key.

There is one basic difference between some of the models that has an effect on the way invariants are defined. This has to do with the way decryption is represented. Some, such as the strand space model, represent the cryptoalgebra in terms of a free algebra. The effect of decryption is described by protocol rules stating that a principal who receives an encrypted message and has the corresponding decryption key can produce the decrypted message. Other models, for example, that used by the NRL Protocol Analyzer, represent encryption in terms of algebraic identities, that is, encryption and decryption are represented as separate operations, e.g. e_K and d_K, and the fact that decryption cancels out encryption is represented as an algebraic identity, i. e. $d_K(e_K(X)) = X$.

Specification in terms of an algebra with identities is more expressive than specifications in terms of free algebras, since it allows us to specify the application of the decryption operator to a message that was never encrypted. But this situation does not arise for most protocols (unless we look at them from a lower level of abstraction than that usually used by this form of analysis), so that the free algebra representation is often preferred as being simpler. As we will see, the choice of representation will have some effect on the way invariants are defined, but will not have a serious effect on their general structure.

We define some terms we will be using below.

Definition 1 *A* protocol execution *describes a sequence of sending and receiving messages by principals, at least some of which should be honest.*

We will say that a protocol execution is backward complete *if every message received by a principal corresponds to a message sent by a principal. We will say that a protocol execution is* forward complete *if every message sent by an honest principal corresponds to a message received by a principal.*

A correctness specification *describes the set of acceptable backward complete protocol executions. A protocol is* flawed *if there exists a backward complete protocol execution not satisfying the correctness specification.*

We will also need the notion of a state and a state transition. This is not defined for all models (the strand space model has no clear notion of state), but it is widespread enough so that we will find it necessary to use.

Definition 2 *A* state *consists of the value of the intruder's knowledge set plus the values of the local state variables of honest principals at a given point in time. A* state transition *describes the move from one state to another by the sending or receipt of a message or a change in the values of local state variables.*

The depth *of a protocol execution is the number of state transitions involved in that execution.*

2.2 Analysis Techniques

There are basically three approaches that have been used to apply formal methods to cryptographic protocol analysis. The first is the use of high-level epistemic logics developed specifically for protocol analysis, such as BAN logic [1]. This does not usually involve the generation of invariants, and so we do not consider it further here. A second is the use of theorem-proving techniques, which generally involve proving that invariants are preserved under state transition. A third is to use state exploration techniques, usually using a model checker. It is also possible to combine theorem proving and state exploration. For example, the NRL Protocol Analyzer automatically generates and proves invariants to limit the search space, and then performs an exhaustive search of the remaining finite search space. Likewise, the Athena model checker [16], which is based on the strand space model, makes use of hand-proved strand space theorems to limit the size of its search space.

Both state exploration and invariant generation can make use of forward or backward search. In the case of invariant generation and verification, one can attempt to prove that, whenever the invariant holds in a state, then it holds after any state transition (as do Paulson and Cohen). Or one can show that, whenever an invariant does not hold in a state, then it most have not have held in any state immediately preceding it (as does the NRL Protocol Analyzer). In either case, one uses the result together with a proof that the invariant always holds initially to prove that the invariant holds for any state.

State exploration techniques make use of forward search by producing a set of backwards complete executions as follows. The initial state is trivially backward complete. One attempts to construct forward complete executions by taking all sent messages that are not currently matched with received messages and matching them with rules including received messages. (Note that, if a rule contains more than one received message, all received messages must be matched in order to maintain backward completeness). Once the match is made, any sent message included in the rule is added to the execution, in the appropriate temporal order. This process continues until the execution is forward complete, or until some other stopping criterion is reached.

Backwards search is done in the following manner. Here the final state is usually assumed to be an insecure one; that is, it describes results that could only occur as the result of the protocol execution that failed to satisfy the correctness specification (indeed, the specification of the insecure final states can serve as a definition of the correctness specification in terms of its negation). The specification of the insecure state can contain information about such things as secrets learned by the intruder, inconsistent information (e.g two parties holding different values for what is intended to be a shared cryptographic key), or even specific illegal sets of events. Once the final state is specified, the analyst searches for rules whose output matches some part of the state. If these rules contain unmatched received messages, the analyst then searches for other rules containing sent messages matching the received messages. Any received messages contained by these rules are added to the execution in the appropriate temporal order. This process continues until either a backward complete execution is produced, in which case an attack has been found, or it is impossible to extend the execution further, in which case it has been shown that no attack is possible.

When backwards search is used, we will often produce desired received terms that cannot be produced by any backward complete execution, that is, terms that can never be learned by the intruder. We need some way of characterizing these terms. Often, this can be done by producing a set of terms such that the property of their not being sent remains invariant under protocol execution.

In the case of forward search, we may find ourselves in a different situation; that is, we may find ourselves generating terms that are learnable by the intruder, but are irrelevant, e.g. the result of an arbitrarily long sequence of encryptions or concatenations that will never be accepted as a legitimate message by an honest principal. In this case, we want to characterize the set of terms that *can* be learned by the intruder in a useful way without wasting time with these useless terms. We will explore these issues in more detail in the next sections.

3 Ideals and Sets of Terms Not Known by the Intruder

3.1 Defining Invariants

3.1.1 Strand Space Ideals

The strand space model uses a free crypto-algebra. This means that each term has a unique representation, which makes it straightforward to define invariants in a simple natural way as follows.

When we are attempting to determine that a set of terms $\mathbf{T}$ is not known by the intruder, we can use it to define an invariant in the following way. Clearly, if (X,T) or (T,X) is known by the intruder, where $T \in \mathbf{T}$, then the intruder can learn T. Likewise, if the intruder knows a key K and also knows $e_K(X)$, then the intruder can learn X. Thus, if T is to remain unknown, so must the result of concatenating T with X, and the result of encrypting T with any key known by intruder. This suggests that we define a set $I_{\mathbf{K}}(\mathbf{T})$ as the smallest set containing $\mathbf{T}$ closed under encryption by keys from the set $\mathbf{K}$ and under concatenation with arbitrary terms, and attempt to show that the property of the intersection of $I_{\mathbf{K}}(\mathbf{T})$ and the intruder's knowledge set being empty is invariant under protocol execution.

The notation $I_{\mathbf{K}}(T)$ is no accident; the set we have described above is of course exactly describes the strand space ideal described in [18]. However, there are models that use invariants that behave very similarly. We will describe them in the next two sections.

3.1.2 NRL Protocol Analyzer Languages

One such is the *languages* used by the NRL Protocol Analyzer. Speaking generally, the languages used by the NRL Protocol Analyzer are simply sets of irreducible terms described using a BNF grammar. But as a matter of fact, languages are always produced by the NRL Protocol Analyzer. using an automatic procedure that tends to produce languages of a form very similar to ideals. It starts with a generator (*seedword* in the NRL Protocol Analyzer nomenclature), which is actually a set of generators, since it usually contains variable subterms. The NRL Protocol Analyzer then searches backwards from the state in which the intruder knows the seedword, producing all protocol executions of a predefined depth that could produce that seedword. (The requirement for a predefined depth is there in order to prevent executions of unbounded depth.)

For each such execution, the NRL Protocol Analyzer checks to see if it contains a state transition that requires the intruder to already know the seedword, that is, that requires a principal to receive the seedword as a message. If not, it attempts to add a language rule that would imply that at least one of the terms required as input messages by the execution belong to the language. After this is done, it takes each language rule and uses it to formulate a goal. It then searches backwards from that goal, again producing all executions of a given depth that lead up to that goal. It then checks each execution to see if each execution requires the intruder to know a word in the language. If not, it adds new language rules, and proceeds as before. This process is iterated until a fixed point is reached, that is, every execution produced requires that the intruder know a word in the language, and thus no new language rules need to be produced. (Note that it is not guaranteed that a fixed point will be reached). At the end, a set $\mathbf{L}$ has been produced such that, if the intruder learns a term in $\mathbf{L}$, then it must have already known a term in $\mathbf{L}$.

The NRL Protocol Analyzer includes state transition rules that say that, if the intruder knows (X, Y), then it can learn X and Y, and, if the intruder knows $e_K(Y)$ and K, then it can learn Y. Thus, whenever the language generator attempts to produce the set of executions that will lead to the intruder learning a term W, it will almost always produce an execution requiring the intruder to know (X, W) for some X, another requiring the intruder to know (W, X) for some X, and another requiring the intruder to know $e_K(W)$ for some key K. This leads the language generator to include rules saying that, if the T is in the language, then (T, X) and (X, T) is in the language for any X, and $e_K(T)$ is in the language for any K. It will also generate similar rules for other operations. Executions that involve the use of protocol rules describing honest principals receiving messages may also generate language rules, but these rules are often subsumed by the language rules generated as the result of intruder actions; they are usually not included in the language as it is finally defined. Thus the language generator generates invariants that are in many ways similar to ideals: they have a set of generators, and they are closed under concatenation, encryption, and any other operation that is defined by the protocol. In some ways these languages are more general than strand space ideals (more operations may be included) and in some way more restrictive. For example, in the strand space model it is possible to specify a particular set of keys that will be used for encryption. This cannot be done in the NRL Protocol Analyzer.

There are also some other differences between ideals and languages. The NRL Protocol Analyzer, unlike the strand space model, does not use a free algebra. Instead, it models operations such as encryption and decryption in terms of algebraic identities; e.g., encryption and decryption cancel each other out. This has the potential of causing problems when defining languages, e.g. should $d_K(e_K(X))$ be considered a member of the language generated by $e_K(X)$ or not? The problems are avoided by restricting the algebraic identities to those that form a Noetherian (that is, terminating), confluent set of rewrite rules. These properties are fairly easy to check for, and also have the advantage that it is easy to find a unique canonical representative of each equivalence class. The language is then taken to be the intersection of the set generated by the BNF grammar and the set of canonical representatives.

3.1.3 Schneider's Rank Functions

A rank function is an integer function defined on messages. One uses rank functions both to characterize the messages that can be sent in a protocol execution (the messages of positive rank), and the messages that can't be sent, (the messages of rank 0 or less). One then attempts to show that certain messages can never be sent or learned by the intruder because they are of rank 0 or less. This is done by showing that the property of rank 0 messages not being known by the intruder remains invariant under state transition.

Consider for example the strategy used by Schneider in his analysis of the Needham-Schroeder and Needham-Schroeder-Lowe public key protocols [15], which use encryption by public and secret keys (represented by two different operations), and concatenation. Schneider's strategy, for each of his rank functions, is to first start out by assigning rank to atomic terms. Those that may be learnable by an intruder are assigned rank one; those that should not be learnable are assigned rank zero. The result of encrypting with a public key is defined for the most part to be rank-preserving, as is the result of encrypting with a private key. The exception will generally involve encryption by a participating principal, which may be rank-increasing, since the intruder may learn the result of encrypting a message (so that should be of positive rank), while the message itself should not be learned by the intruder. The rank of the result of concatenation is assumed to be the minimum rank of the two terms concatenated. Some messages that are known to be generated by the protocol are also assigned rank one.

There are also a few exceptions to the general strategy. Many of these arise from the fact that Schneider's earlier model, like the NRL Protocol Analyzer, models encryption and decryption in terms of algebraic identities. However, unlike the NRL Protocol Analyzer, no a priori restriction on the types of identities is made. Thus rank functions must be defined so that all members of an equivalent class have the same rank. In particular, if p_j denotes encryption with a public key belonging to j and s_j denotes encryption with a corresponding private key s_j, then $s_j(p_j(m))$ should have the same rank as m.

Given the exceptions resulting from the necessity of remaining consistent with algebraic identities, we see the messages of rank zero or less, at least in the Needham-Schroeder-Lowe analysis have a structure very similar to that of ideals. They are preserved under encryption by most keys, and they are preserved under concatenation with any other term. The similarity of their structure to that of ideals and languages is of course not surprising, since like them they are used to characterize terms that can never be members of the intruder's knowledge set.

The messages of positive rank, which characterizes the messages that can be sent during a protocol execution, also have a structure of their own. They tend to be closed under encryption by most public and private keys, and under concatenation with other messages of positive rank. More recent work of Heather and Schneider [7] has tended to concentrate more on the structure of the messages of positive rank than on those of rank zero or less. We will say more about this in Section 4.

3.2 Using Invariants

3.2.1 Secrecy Results

Once we have defined our invariants, we need to figure out how to use them. Basically, they are used to prove two sorts of things. The first, and simplest, is to prove that the intruder can never learn a particular message or set of messages. This type of result is usually used to prove general secrecy theorems, e.g. that the intruder can never add the master keys used by honest principals to its knowledge set. The second is to prove that the intruder can only learn a particular message if it was sent under certain circumstances, e.g. by a particular principal at a particular point in its execution of the protocol.

The techniques for using the invariants to prove secrecy employed by the three systems do not vary much. However, the NRL Protocol Analyzer does offer a somewhat different approach by allowing the state of the intruder's knowledge set to be used in the definition of a language. A user can define a seedword and specify that a subterm of that seedword not belong to the intruder's knowledge set. The property that the language generated by the NRl Protocol Analyzer does not contain any elements of the intruder's knowledge set is then proved to be invariant under all transitions that preserve the property that the subterm is not in the intruder's knowledge set.

This definition of seedwords in terms of intruder knowledge comes from the way in which the NRL Protocol Analyzer uses backwards search. For example, if it is trying to find out how the intruder could learn a message M, it will be told that M can be found if the intruder knows $e_K(M)$ and K. Since we are not interested in the case in which the intruder already knows M, we need to be able to characterize the cases in which the intruder knows $e_K(M)$ but not necessarily M. Thus the property to be proved invariant becomes the property that the intruder does not know any member of the language, and the intruder does not know M, where $e_K(M)$ is the seedword generating the language.

As for authentication, each of the three systems uses a somewhat different approach. However, as we shall see below, they are closely related.

3.2.2 Strand Space Ideals and Authentication

In the strand space model, an ideal is said to have an *entry point* if at some point in an execution a member of that ideal is sent as a message by one of the principals (including, possibly, the intruder). Thus, if we are able to prove that under no circumstances does an ideal have an entry point, we can show that under no circumstances can the intruder add any member of the ideal to its knowledge set.

In order to address the problem of authentication, the strand space model allows us to include the notion of how a term was originated. An entry point is a *regular entry point* if it was originated by an honest principal, and a *penetrator entry point* if it was produced by the penetrator. Thus one can prove an authentication result guaranteeing that a message M could only have been sent by a principal A executing step N of the protocol in the following way:

1. Construct an ideal containing M. Usually, this will be $I_{\mathbf{K}}(M)$ for some $\mathbf{K}$.
2. Show that the ideal has only regular entry points.
3. Identify which regular nodes (that is, which principals at which points of the protocol) could have sent message M.

3.2.3 NRL Protocol Analyzer Languages and Authentication

The NRL Protocol Analyzer does not allow us to identify explicitly the origins of the terms in languages. However, it provides some tools that indirectly give the user the same capability.

First of all, the generation algorithm allows for the notion of an *exception*. Seedwords are typically defined using variable arguments, and thus it is likely that for certain substitutions to the variables, the seedword may be learnable by the intruder. In that case, an exception rule is generated that excludes those substitutions from the language. For example, suppose that we start with a seedword $s_A(M)$ where s_A indicates signing by A's private key, where A and M are variables, and that the NRL Protocol Analyzer finds out that the term $s_A((X, A))$ can be sent by any principal A. Then the definition of the seedword is modified to read all $s_A(M)$ *except* where $s_A(M) = s_A((X, A))$.

Information about which principal generated the message and under what circumstances is not saved by the NRL

Protocol Analyzer. Instead, it is possible to reconstitute this information under backwards search. Suppose, for example, in a backwards search the NRL Protocol Analyzer finds a state in which a principal receives a message of the form $s_A(M)$, where A and M are variables. According to the definition of the language, this message cannot be sent unless it is of the form $s_A((X,A))$. Thus the NRL Protocol Analyzer makes the appropriate substitution to the variables, replacing M by (X,A). One can then continue the search to find the exact circumstances under which the term was generated.

3.2.4 Rank Functions and Authentication

Schneider's approach to authentication is to define it in terms of a condition that a given action **A** must be preceded by another action **B**, where **A** and **B** usually denote the sending of particular messages by particular honest principals. This is done by constructing a specification S of the protocol that is identical to the original specification in all respects except that **B** is blocked from occurring, and then showing that **A** cannot occur in S either.

What is usually discovered is that **A** cannot occur unless some message M is received by the principal executing **A**. Thus it is enough to show that M can't be sent in S, that is, that the intruder can never learn M. This, of course, is a secrecy theorem, and can be proved using rank functions in the same way that Schneider proves his secrecy results. Thus this technique has the advantage that no new methods have to be introduced to handle authentication.

4 Subalgebras and Sets of Terms Known by the Intruder

4.1 Definitions and Motivation

In this section we will examine crypto protocol analysis systems that use subalgebras or similar constructs to characterize the sets of terms that can be learned by an intruder. Such a subalgebra is typically defined by a set **S** that is closed under any operation op($x_1, ..., x_n$) for all operations op that are defined by the system, and all terms x_i belonging to **S**. Clearly, the set of all terms known by the intruder defines a subalgebra under this definition: in particular, if the intruder knows a set of terms, it should be able to discover the results of performing any operation on those terms. What we will often be interested in here is the ability to find a set of generators of a subalgebra **S** of terms known by the intruder; this will allow us to characterize the subalgebra in a useful way.

We define a subalgebra more formally as follows.

Definition 3 *Let S be a set, and let O = { op_1,..., op_n } be a set of function symbols that may or may not obey some algebraic identities. Let O' be a subset of O. We say that a subset G of S is a* subalgebra *of S with respect of O' if for each $op_i \in O'$, if $x_1, ..., x_{k_i} \in G$, where k_i is the arity of op_i, then $op_i(x_1, ..., x_{k_i}) \in G$. We say that G is* generated *by a set S' if G is the smallest subalgebra containing S'; in this case we say that $G = G(S', O)$.*

In this section we will consider four different systems that use some versions of subalgebras. The first we will look at is Rankanalyzer [7], Heather's and Schneider's automation of analysis by rank functions, which uses the generators of subalgebras to determine which terms can be learned by intruder. The next we will look at is Paulson's inductive method [14], which uses subalgebras in a limited way. We will then look at Millen and Ruess' use of a combination of strand space ideals and of concepts from Paulson's inductive message to prove a general secrecy theorem [12]. Finally, we consider Cohen's TAPS [4], which takes a similar approach to the inductive method, but, unlike the inductive method, in which each invariant is defined individually, offers a general set of invariants which can be defined and reasoned about mostly automatically. TAPS invariants bear some resemblance to subalgebras, but unlike the other invariants discussed in this paper, which are static, the makeup of TAPS invariants evolve with each state transition.

4.2 Rankanalyzer

Rankanalyzer uses the same general framework as Schneider's original system (with the minor difference that is uses a free algebra for its crypto-algebra). That is, one constructs a specification of a cryptographic protocol for each result to be proved. For secrecy results, this is the specification of the original protocol. For authentication results, this is the specification with a key event blocked. One then attempts to construct a rank function and to show that the message of interest is of rank less than one and so can never by generated by the specification. However, the strategy for constructing rank functions is much different. Rankanalyzer constructs a rank function that is one on the set X of all messages that could be generated from the initially known set of messages by the protocol rules. This X is clearly a subalgebra under any operation performable by the intruder (in Rankanalyzer's case, these are encryption and concatenation), since whenever a key and a term are produced as the result of applying sequences of protocol rules, the intruder can learn the encryption of the term with the key by the application of a protocol rule, and whenever two terms are produced by a sequence of protocol rules, then the intruder can learn the concatenation of the two terms by the application of a protocol rule.

Although it is easy to define X, in order to make it useful it is necessary to have an effective procedure for determining whether or not a term is a member of X. This is done by finding a set of generators of X in two steps as follows.

First, let D be the set of all messages that could ever appear in a protocol run, that is, all messages that could be sent or accepted in R by legitimate principals. Heather and Schneider restrict themselves to protocols in which legitimate principals will not generate messages with arbitrarily large numbers of concatenations and encryptions, so this removes one level of complexity. They then let $\mathcal{M}^0$ be the set of all subterms of all messages appearing in D, together with all k^{-1} such that k appears in $\mathcal{M}^0$. The terms that Heather and Schneider are trying to prove unlearnable always reside in $\mathcal{M}^0$, so all that is necessary is a way of determining whether or not a term is in $X \cap \mathcal{M}^0$. This is provided by giving an algorithm for computing $X \cap \mathcal{M}^0$ and another for determining a finite representation; details are given in [7]. We note that, although Heather and Schneider don't use this fact (they don't need to, since the terms they are trying to prove unreachable always reside in $\mathcal{M}^0$), that $X \cap \mathcal{M}^0$ is a set of generators of X over encryption and concatenation.

4.2.1 Paulson's Inductive Method

Paulson uses the Isabelle theorem prover and induction on protocol execution to prove security properties of cryptographic protocols. This is done by stating each lemma and theorem in terms of an invariant that is preserved by the execution of each and every protocol rule. This results in a large number of proof obligations, but in practice most are trivial, so this approach has been effective in proving results about some specifications of considerable complexity.

Paulson does not attempt to define any general automatable procedure for proving properties of his protocol specifications, so subalgebras and ideals do not play as prominent a role here as in the other systems we have described. However, there is one place in which they do make an appearance. Unlike the other models we have described here, Paulson's does not model intruder operations as messages sent in the protocol from the intruder to itself. Instead, Paulson describes the execution of the protocol in terms of messages sent and/or received by legitimate participants. He then defines, for any set of messages X two sets $\mathbf{analyz}(X)$ and $\mathbf{synth}(X)$, where $\mathbf{analz}(X) = G(X \cup \mathbf{K}, O_1)$ where O_1 consists of deconcatenation and decryption and $\mathbf{K}$ is the set of keys known by the intruder, and $\mathbf{synth}(X) = G(X \cup \mathbf{Y}, O_2)$, where $\mathbf{Y}$ is the set of agent names and guessable nonces, and O_2 consists of concatenation, encryption, and other relevant cryptographic functions such as hashing.

Although Paulson does not describe any general procedure for using these constructs, he does prove a number of results about the relationships between **analyz**, **synth** and a related construct **parts**(X) that is to be defined to be the set of all subterms of the terms in X (over 110 results, according to [14]), and he uses them extensively to prove results about the obtainability of messages by the intruder.

4.2.2 Co-Ideals and Protocol-Independent Secrecy

Millen and Ruess [12] use both the concept of an ideal from strand spaces and Paulson's notions of **analyz**, **synth**, **parts**, to develop a theorem that describe when a secret can not be disclosed by a protocol. They first define the concept of a co-ideal, which is simply the set complement of an ideal. They point out that co-ideals are closed under **analyz** and **synth**, thus making them subalgebras under our definition. Of particular interest to them is the co-ideal of $\mathcal{I}(S) = I_k[S]$, where k is the set of keys whose inverses are not in S.

Millen and Ruess then use co-ideals to prove results about the security of keys. In particular, they show that, under certain conditions, that, if S is a set of secrets together with the set of all encryption keys whose inverses are known by principals who are *not* intended to know the secrets in S, and whenever all previous messages sent by an honest principal are contained in the co-ideal of $\mathcal{I}(S)$, so is any next message sent by an honest principal, then it is the case that none of the secrets are ever compromised by the protocol.

The proof makes extensive use of the closure properties of co-ideals. Note that, although their result is a protocol-independent theorem, not a search strategy, the statement and proof of the result makes use of forward search techniques, that is, the forward extension of traces. This adds support to our hypothesis that the use of subalgebras tends to work most naturally with the use of forward search techniques.

4.2.3 TAPS

Cohen's TAPS takes an approach similar to Paulson, but, like Millen and Ruess uses standard invariants that can be applied to any of a class of protocols. Moreover, he goes them one further by providing automated support for proving that these invariants hold.

Cohen defines several unary predicates involving intruder knowledge. The first, $pub(X)$, holds if the message X has been sent during the execution of a protocol. The second, $prime(X)$, describes certain types of basic messages that it is permissible for the intruder to know, e.g. names of principals, encrypted messages, and subterms of encrypted messages to which the intruder has the key. For simple protocols, $prime$ can be specified automatically; however TAPS also offers the user the option of specifying $prime$ manually for more complex protocols. Finally, the third predicate $ok(X)$ describes all messages that it is permissible for the intruder to know, and is defined be the strongest predicate satisfying the following:

1. $prime(X) => ok(X)$
2. $ok(X) \wedge ok(Y) => ok((X, Y))$
3. $pub(X) \wedge pub(Y) => ok(e_X(Y))$

Note that the definition of ok differs from the other types of invariants we have discussed in that its makeup is state-dependent, since its definition depends on $pub(X)$, which is also state-dependent.

Cohen then uses these predicates to prove that $pub(X) => ok(X)$ is invariant under state transition. This is done by breaking the invariant down into a set of proof obligations based on the actions that could be taken on a message X. Some of these depend upon the particular rules of the protocol; others are based on intruder actions and are the same from protocol to protocol.

Note that the set of terms satisfying $ok(X)$ is already a subalgebra under concatenation. Note also that the set of terms satisfying $pub(X)$ is *almost* a subalgebra over concatenation and encryption generated by the set of terms satisfying $prime(Y)$, and could be made into one by substituting ok for pub in the last definition. However, this would destroy the state-dependence of the definition of ok.

As it turns out, the subalgebra properties of ok are mainly the result of a compromise for efficiency. A more natural choice would be to include the second part of the definition of ok under the definition of prime, but, for protocols that string a long sequence of elements together in a message, this would require a possibly unmanageably large number of primality cases [3]. However, the use of a subalgebra-like construct at this point shows a possible way in which subalgebras could be traded off with state-dependent invariants.

5 Conclusion

We have given an overview of seven different techniques for generating invariants for cryptographic protocol analysis. We have found considerable areas of commonality between them, especially for techniques for proving results about intruder unlearnability of messages. This suggests some further areas of investigation.

First of all, it would be useful to investigate to what extent it would be possible to mix and match the various techniques? For example, would any of the techniques Thayer, Guttman, and Herzog have developed for reasoning about ideals be helpful in analyzing Schneider-style specifications in which authentication is proved by blocking the authenticating message from occurring? Likewise, it would be helpful to know how any of these techniques would be helpful in augmenting Paulson's inductive method, and if so, what would be the best way of integrating them. We note that Millen and Ruess have showed one way in which these techniques could be integrated; we suspect that there are also many others.

Secondly, it would be useful to look beyond the use of tools and techniques for analysis and see if any of these techniques would be helpful in proving decidability results. We note that both Heather and Schneider's and Cohen's techniques give decision procedures that imply correctness of the protocol when the procedure succeeds; however, when they fail it is not yet known under what circumstances that means that the protocol is insecure. Since security has been shown to be undecidable for even simple protocols in the Dolev-Yao model used by these systems ([6, 8, 5, 2]) it is of course not possible for these techniques to provide a decision procedure for security for the general case. However, there have been classes of protocols, e.g. those discussed by Lowe [9] and Stoller [17], as well as several classes discussed by Durgin et al. in [5] for which it is possible to develop decision procedures for security. A closer examination of the techniques surveyed in this paper might show how they could help in simplifying proofs and extending results.

Finally, all of the techniques applied were, at least initially, developed for a somewhat limited model, which assumes that a protocol is designed to exchange information between a bounded number of principals (e.g. a key between an initiator, a responder, and possibly, a key server), employing a fixed set of operations usually restricted to concatenation and public and shared-key encryption, and a fixed set of data types usually restricted to nonce, keys, and names. However, there are increasing number of protocols emerging that go beyond these bounds, in particular group protocols and protocols that make use of other types of cryptographic techniques, e.g. Diffie-Hellman key generation. Several of the systems discussed in this paper have been applied to these types of problems with some success, e.g. the NRL Protocol Analyzer to the Internet Key Exchange Protocol, which used Diffie-Hellman [11], and Paulson's inductive method to the Recursive Authentication Protocol [13], a simple group key distribution protocol. As would be expected, these are two of the systems that offer only partial automated support, and hence allow more flexibility to the user. However, this work does indicate that the general approach is extensible, and it would be interesting to see if, once the analysis of these different types of protocols have been explored more thoroughly, whether or not it would be possible to extend the more automated systems to handle them.

6 Acknowledgments

I would like to thank James Heather, Steve Schneider, Ernie Cohen, and Jon Millen for making preliminary versions of their papers available to me, and for their patience in answering my questions. I also would like to thank Sylvan Pinsky, Al Maneki, Josh Guttman, Jonathan Herzog, Peter Ryan, Paul Syverson, and Iliano Cervesato for helpful discussions. This work was supported by NSA.

References

[1] Michael Burrows, Martín Abadi, and Roger Needham. A Logic of Authentication. *ACM Transactions in Computer Systems*, 8(1):18–36, February 1990.

[2] I. Cervesato, N. A. Durgin, P. D. Lincoln, J. C. Mitchell, and A. Scedrov. A meta-notation for protocol analysis. In *Proceedings of the 12th IEEE Computer Security Foundations Workshop*. IEEE Computer Society Press, June 1999.

[3] E. Cohen. personal communication, Feb. 9 2000.

[4] E. Cohen. Towards automatic verification of cryptographic protocols in natural operational models. preprint, 2000.

[5] N. Durgin, P. Lincoln, J. Mitchell, and A. Scedrov. Undecidability of bounded security protocols. in Electronic Proceedings of the Workshop on Formal Methods and Security Protocols, July 1999. available at http://www.cs.bell-labs.com/who/nch/fmsp99/program.html.

[6] S. Even and O. Goldreich. On the security of multiparty ping pong protocols. Technical Report 285, Technion Israel Institute of Technology. 1983.

[7] J. Heather and S. Schneider. Towards automatic verification of authentication protocols on an unbounded network. Technical Report CSD-TR-00-04, Department of Computer Science, Royal Holloway, University of London, Egham, Surrey TW20 0EX, UK, January 2000.

[8] N. Heintze and J. D. Tygar. A model for secure protocols and their composition. *IEEE Transactions on Software Engineering*, 22(16), 1996.

[9] G. Lowe. Towards a completeness result for model checking of security protocols. In *Proceedings of the 11th IEEE Computer Security Foundations Workshop*, pages 96–105. IEEE Computer Society Press, June 1998.

[10] C. Meadows. Language generation and verification in the NRL Protocol Analyzer. In *Proceedings of the 9th Computer Security Foundations Workshop*. IEEE Computer Society Press, June 1996.

[11] C. Meadows. Analysis of the Internet Key Exchange Protocol using the NRL Protocol Analyzer. In *Proceedings of the 1999 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, May 1999.

[12] J. Millen and H. Ruess. Protocol-independent secrecy. In *Proceedings of the 2000 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, May 2000.

[13] L. C. Paulson. Mechanized proofs for a recursive authentication protocol. In *Proceedings of the 10th Computer Security Foundations Workshop*, pages 84–95. IEEE Computer Society Press, June 1997.

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

[15] S. Schneider. Verifying authentication protocols with CSP. In *Proceedings of the 10th Computer Security Foundations Workshop*. IEEE Computer Society Press, June 1997.

[16] D. X. Song. Athena: A new efficient model checker for security protocol analysis. In *Proceedings of the 12th IEEE Computer Security Foundations Workshop*. IEEE Computer Society Press, July 1999.

[17] S. Stoller. A reduction for automated verification of authentication protocols. Electronic Proceedings of 1999 Workshop on Formal Methods and Security Protocols, http://www.cs.bell-labs.com/who/nch/fmsp99/program.html., July 1999.

[18] J. Thayer, J. Herzog, and J. Guttman. Honest ideals in strand spaces. In *Proceedings of the 11th Computer Security Foundations Workshop*. IEEE Computer Society Press, June 1998.