Analyzing the Needham-Schroeder Public Key Protocol: A Comparison of Two Approaches

Catherine A. Meadows

Code 5543
Center for High Assurance Computer Systems
Naval Research Laboratory
Washington DC, 20375
meadows@itd.nrl.navy.mil

Abstract. In this paper we contrast the use of the NRL Protocol Analyzer and Gavin Lowe's use of the model checker FDR [8] to analyze the Needham-Schroeder public key protocol. This is used as a basis for comparing and contrasting the two systems and to point out possible future directions for research.

Most early work in the automated analysis of cryptographic protocols concentrated on building special-purpose tools, such as the NRL Protocol Analyzer [5, 10], the Interrogator [11, 5], and Longley and Rigby's protocol analysis tool [6]. Although some work existed on the application of existing tools, such as Kemmerer's use of Ina Jo [5], this was not an approach followed by many. Some of this early concentration on special-purpose tools may have been a result of the belief that cryptographic protocols had certain unique properties that would make them more amenable to analysis by a tool using special-purpose models and algorithms. This was certainly the belief that motivated much of the development of the NRL Protocol Analyzer. But, as research has progressed in this area, and people are becoming more and more comfortable with the techniques and assumptions that are needed to analyze cryptographic protocols, more researchers are exploring ways in which existing formal methods tools can be applied to the problem of either assuring a cryptographic protocol's correctness or finding a flaw if it exists. Thus, for example the theorem prover HOL has been used by Snekkenes in [13] for stating and proving properties of cryptographic protocols.

At this point, it makes sense to ask the question: what comparisons can we make between these two ways of approaching the problem? Clearly, some of the advantages of using an existing system is its greater power and maturity, while the advantages of using one's own is that it can be tailored to the problem at hand, giving better assistance for solving the problems of interest. Thus, we might expect existing systems to provide the most help in solving the parts of the problem that most resemble other problems, and a special purpose system to provide the most help in solving the parts of the problem that are unique or at least unusual. But one cannot tell exactly how these things will balance out until some experience has been gained in applying both types of systems to the problem.

In this paper we try to at least begin to answer these questions by comparing the use of two different tools to analyze the same protocol, the Needham-Schroeder public-key authentication protocol [12]. One of these tools, the NRL Protocol Analyzer, is a special-purpose tool for analyzing cryptographic protocols. The other, FDR [2], is a model checker that tests whether CSP programs satsify their specifications. We will use the results of this analysis to provide a basis for the comparison of these two tools. Although this itself will not be enough to provide a comparison of the use of special-purpose versus general tools, it can be used as a basis for further work by others looking at the same problem.

The history of the use of these two tools to analyze the protocol is as follows. The FDR tool was used first by Lowe to find a triangular attack on the protocol [7, 8]. This attack was new; as a matter of fact the protocol had previously been "proven" correct in [1]. The NRL Protocol Analyzer was then used by us to reproduce the same attack, as well as to find some new attacks. [1]

The remainder of this paper is structured as follows. Section 2 contains a description of the Needham-Schroeder public-key protocol (from now on referred to as the Needham-Schroeder protocol, not to be confused with their more famous private-key protocol), and some of the attacks that were found. In Section 3 we present a brief account of FDR and Lowe's analysis of the Needham-Schroeder protocol; a more complete account can be found in [8]. In Section 4 we describe the NRL Protocol Analyzer and show how it was used to analyze the protocol. In Section 5 we compare the two approaches. Section 6 concludes the paper.

1 The Needham-Schroeder Public-Key Protocol

The Needham-Schroeder protocol uses public keys to achieve authentication between two parties. The protocol involves an initiator A, a responder B, and a server S. It proceeds as follows.

1. $A \rightarrow S : B$
   A requests B's public key from S.
2. $S \rightarrow A : \{K_B, B\}_{K_S^{-1}}$
   S sends B's public key and name to A, signed with its digital signature. A checks the signature and B's name.
3. $A \rightarrow B$: $\{R_A, A\}_{K_B}$
   A sends a nonce R_A, together with A's name to B, encrypted with B's public key. B decrypts to get A's name and nonce.
4. $B \rightarrow S$: A
   B requests A's public key from S.

[1] The fact that these attacks were not found by FDR does not necessarily point out a limitation in that system or in Lowe's approach; it is simply the result of the fact that we and Lowe made slightly different assumptions about primitives used by the protocol, and that Lowe analyzed a fragment, not the entire protocol.

5. $S \rightarrow B$: $\{K_A, A\}_{K_S^{-1}}$
   S sends B A's public key. B checks the signature and A's name.
6. $B \rightarrow A$: $\{R_A, R_B\}_{K_A}$
   B generates a nonce R_B and sends it together with R_A to A, encrypted with K_A. A decrypts the message. If it finds R_A, it assumes that this is a message from B in response to its original message.
7. $A \rightarrow B$: $\{R_B\}_{K_B}$
   A encrypts R_B with K_B and sends it to B. B decrypts the message. If it finds R_B, it assumes that this is a message from A in response to its original message.

The reasoning behind this protocol is straightforward. When A gets the message $\{R_A, R_B\}_{K_A}$, it assumes that, since B received the message encrypted under its public key, and B has no reason to reveal R_A, only B knows R_A, and thus the message must be from B in response to the message $\{R_A, A\}_{K_B}$. Likewise, when B receives $\{R_B\}_{K_B}$, it reasons that only A knows R_B, and so A must have sent the message in response to the message $\{R_A, R_B\}_{K_A}$. In other words, R_A and R_B serve not only as nonces, but as authenticators. Indeed, in [1] Burrows, Abadi, and Needham suggest that they be used as authenticators in subsequent communication. [2] As we will see below, this could have dangerous consequences.

The above argument appears in more rigorous form in [1] where Burrows, Abadi, and Needham use the logic of authentication set forth in that paper to prove that the protocol is correct. However, since their logic relies upon the assumption that principals to not divulge secrets, it misses the following attack discovered by Lowe [7], which relies on a participant's willingness to divulge a secret nonce. [3] Following Lowe, we leave off the initial distribution of public keys. This attack involves four parties: A, B, and an intruder I. We use the notation I_X to mean I impersonating X. I without a subscript refers to the intruder acting as itself.

3. $A \rightarrow I$: $\{R_A, A\}_{K_I}$
   A initiates communication with I.
3'. $I_A \rightarrow B$: $\{R_A, A\}_{K_B}$
   I initiates communication with B, using R_A.
6'. $B \rightarrow A$: $\{R_A, R_B\}_{K_A}$
   B responds to A. A decrypts and finds R_A.
7. $A \rightarrow I$: $\{R_B\}_{K_I}$
   Thinking that the previous message is a response from I, A responds in kind. I decrypts R_B and can now use it to impersonate A to B.
7'. $I_A \rightarrow B$: $\{R_B\}_{K_B}$
   I completes the protocol with B.

If R_A and R_B are used as authenticators subsequent communication, I now has the ability to impersonate A to B for the rest of the session, although I cannot

[2] This is in contrast to the original Needham-Schroeder paper [12], which suggests that authentication be supplied by digital signatures.

[3] For a more complete discussion of the relationship between the attack and Burrows', Abadi's and Needham's proof of correctness, see [3].

read B's messages. Even if digital signatures are used for subsequent authentication instead, and I cannot impersonate A, I has still managed to get A and B in an inconsistent state in which B thinks that A has initiated communication with it when in fact it has not.

Both Burrows, Abadi, and Needham, and Lowe assume that principals can distinguish between types; that it is not possible, for example, to confuse a name with a nonce, or either with a key. Although this is an assumption that is usually considered easy to ensure by proper formatting, it is nevertheless sometimes useful to see what can happen if types are confused. If nothing else, this can at least point out the places in which unambiguous formatting is vital to security.

The following attack, found using the NRL Protocol Analyzer, makes use of the key distribution as well as the authentication phase of the protocol, and relies upon a confusion between nonces and names.

3. $I_A \rightarrow B : \{R_I, A\}_{K_B}$
4. $B \rightarrow S : A$
5. $S \rightarrow A$: $\{K_A, A\}_{K_S{}^{-1}}$
6. $B \rightarrow A$: $\{R_I, R_B\}_{K_A}$

I intecepts this message.

3'. $I_{R_B} \rightarrow A$: $\{R_I, R_B\}_{K_A}$

I sends the intercepted message to A as the initiator of the protocol, with R_B as the name field.

4'. $A \rightarrow S : R_B$

A sends the "name" R_B to S in order to get its public key.

7. $I_A \rightarrow B : \{R_B\}_{K_B}$

I now has the information it needs to impersonate A to B. It encrypts R_B with K_B and sends it to B.

If nonces are used in authenticators during the session, then I can impersonate A to B throughout the session as well.

2 Lowe's Analysis of the Needham-Schroeder Protocol

Lowe specified the Needham-Schroeder protocol in CSP [4], and then used the FDR model checker to search for attacks. FDR works by checking whether or not one CSP specification is a refinement of another. There are several different notions of CSP refinement, of increasing order of complexity. The simplest notion of refinement, usually used for proving safety properties, is the traces model, which is the one used by Lowe. This says that process A refines process B if the traces of A are a subset of the traces of B. If B is a specification of a safety property describing what traces of a system are allowable, and A is a specification of a program, then clearly A satisfies the safety property B if A refines B. FDR checks whether or not one CSP process is a refinement of another in this sense by building up sequences of possible traces for both processes and checking at each stage whether or not the subset property is violated. If it is, it returns the first trace it finds (which will also be the shortest) that violates this property.

Briefly, the specification of the protocol itself consists of three communication events that describe the three authentication messages between A and B. The existence of sets of initiators, responders, public keys, and nonces are assumed. Three channels were specified, the standard *comm*, and two new channels, *fake* and *intercept*, which reflect the intruder's ability to both produce and read messages. Processes are defined for initiators and responders, that describe how they send messages and how they react to received messages. The initiator process begins with a user requesting that process to connect with a responder. Renamings can be applied to the processes to reflect the fact that messages can be intercepted or faked. Thus, if *Msg* is a message sent by the process, then *comm.Msg* can be replaced by *intercept.Msg*; if it is a message received, then *comm.Msg* can be replaced by *fake.Msg*.

Each initiator and responder process is indexed by a name and a nonce. Thus we can think of an initiator or reponder process as corresponding to a single local execution of the protocol. Multiple interleaved executions of a protocol can be represented by interleaving multiple initiator and responder processes. Thus the number of executions in a simulation of the protocol can be limited by limiting the number and type of responder and initiator process.

Finally, a definition of the intruder is given, which describes how it can intercept and fake messages, and how it builds up its knowledge. This is the most complex part of the specification, since each action available to the intruder, who is assumed to be capable of any operation, must be specified separately. The state of the intruder is parametrized by the messages it has been unable to decrypt, and the set of nonces that it has learned. Since the nonces are the only secret information passed in the protocol, this set can be thought of as corresponding to the set of decrypted messages plus any nonces the intruder knows initially (that is, those it generates itself).

After this protocol was specified, the FDR model checker was used on a specification consisting of one initiator process, one responder process, and one intruder process. This meant that FDR only generated traces consisting of at most one local execution of the protocol for initiator and responder. This was sufficient to find the triangular attack on the Needham-Schroeder protocol, however, since this requires only one initiator process and one responder process; the initiator process interacting with the intruder as respondent, and the respondent process interacting with the intruder impersonating the initiator process.

Two specifications of desirable properties of the protocol, one from the point of view of the initiator, and one from the point of view of the responder, were given. The specification $AUTH\_RESP$ says that an initiator should only accept a responder as authenticated if the responder is attempting to respond to the initiator. The specification $AUTH\_INIT$ says that a responder should only accept an initiator as authenticated only the user playing the role of the initiator is actually trying to communicate with the responder. FDR was used to determine whether all the traces of the protocol satisfied these two properties. It found no counterexamples to $AUTH\_RESP$, but it found the triangular attack when it checked $AUTH\_INIT$.

When a flaw is found in a protocol, the next thing to do is fix it. Lowe's suggested fix was to replace the message $\{R_A, R_B\}_{K_A}$ with $\{R_A, R_B, B\}_{K_A}$, so that the originator of the message was not ambiguous. In this way an intruder could not replay B's message as his own. FDR was run on the new protocol using the *AUTH_INIT* and *AUTH_RESP* specification, and no attacks were found.

But the failure of FDR to find attacks does not mean that the protocol is proven secure, since the protocol specification involved a very limited number of protocol executions. Thus there is still the possibility that there could exist attacks that depend on the interleaving of more executions. Since a model checker cannot verify any properties involving an unbounded number of executions, it alone cannot guarantee security of a protocol. Thus Lowe also performed a hand proof is of a theorem that states, in the case of the fixed Needham-Schroeder protocol, that if no attack can be found involving one initiator and one responder, then the protocol is secure. We will not present this proof in detail, but we note that it hinges upon some key lemmas concerning the circumstances under which the intruder can produce and learn words. The first concerns conditions under which messages containing a nonce can be produced by the intruder, and the second concerns the conditions under which the intruder can respond to a nonce challenge by a principal.

3 The NRL Protocol Analyzer Analysis of the Needham-Schroeder Protocol

3.1 Overview of the NRL Protocol Analyzer

As in Lowe's CSP specification of the Needham-Schroeder protocol, the NRL Protocol Analyzer makes the assumption that principals communicate over a network controlled by a hostile intruder who can read, modify, and destroy traffic, and also perform some operations, such as encryption, that are available to legitimate participants in the protocol. The means by which this network model is realized differ in some aspects, however.

In the NRL Protocol Analyzer, actions of legitimate principals are specified by the user as state transitions. Input to the transitions are values of local state variables and messages received by the principal, the latter assumed to have been generated or passed on by the intruder, and output are the new values of local state variables and messages sent by the principal, the latter which are subject to interception and modification by the intruder. The means by which the intruder can modify messages are specified by having the specification writer indicate which operations are performable by the intruder, and what words the intruder may be assumed to know initially. Some operations, such as list concatenation and deconcatenation, are always assumed to be performable by the intruder.

In NRL Protocol Analyzer specifications, local state variables and transitions are indexed by four things: the name of the principal involved, an identifier for the local execution, the step of the protocol it corresponds to, and the principal's local time. The last we have found redundant, and may eliminate in future versions. Principal names, local execution identifiers, and times are all indicated by

variables, which may have an unlimited, even infinite, number of instantiations. If we choose a particular instantiation of principal name and local execution identifier, we see that the set of transitions relevant to a particular role in the protocol corresponds to one of Lowe's CSP processes, where for Lowe the local execution identifier is the nonce generated by a principal for that local execution. The main difference between the Protocol Analyzer and FDR specifications, however, is that for FDR one specifies a finite number of such processes, while the Protocol Analyzer assumes that an unbounded number of runs and principals are possible, since variables in a specification can have an infinite number of possible instantiations. It also allows for odd boundary conditions such as the case in which the same principal plays the role of both initiator and responder; this can be done by instantiating the names of the principals playing these roles to the same term. This possibility of attacks under such conditions will be explored automatically by the Protocol Analyzer unless the user explicitly states that they should not be considered.

The Protocol Analyzer also differs from FDR and other model checkers in that, instead of working forwards from an initial state, it works backwords from a final state. The user of the Analyzer uses it to prove a security property by specifying an insecure state in terms of words known by the intruder, values of local state variables, and sequences of events that have or have not occurred. The Analyzer gives a complete description of all states that can immediately precede that state, followed by a complete description of all states that can immediately precede those, and so forth. Since the search space the Analyzer deals with is infinite, the user is given a number of means of pruning the search space to a manageable size. These include:

1. **Inductive proof of unreachability of infinite classes of states by use of formal languages.** The Analyzer can be used to prove that if the intruder learns a member of the language, then it must have already known a word from that language. The Analyzer in its most recent form can be used to generate languages as well as prove them unreachable.
2. **Remembering conditions on reachability of states.** If a state description has been proved unreachable, or only reachable if the variables in that state description have been instantiated to certain values, the Analyzer can remember that fact and apply it the next time that state description is encountered in a search.
3. **Querying subsets of state descriptions.** When a state description is found by the Analyzer, the user has the option of telling the Analyzer to look for some subset of that state description. For example, given a state description in which the intruder knows two words W and V, the user can ask the Analyzer how to find the state in which the intruder knows V. This is a good strategy, for example, if the user suspects that it is easy to show that the state in which the intruder knows V is unreachable. The Analyzer also has several different sets of search-pruning heuristics built in which the user can specify; these will then be applied automatically. For example, it can be directed to choose to query a state variable or word only if it

contains a term that appeared in the original top-level query. Thus, if the user begins by asking how to find a state in which the intruder knows V, and the Analyzer finds that this can be done if the intruder knows K_A and $\{V\}_{K_A}$, the Analyzer will only look for $\{V\}_{K_A}$, since K_A does not appear in the original query.

Note that these techniques take the place of the theorems Lowe used to narrow his search space to that generated by two processes. Note also that the ability to query a subset of a state description may generate a false attack; the portion of the description that is queried may be reachable, while the entire state is not. Thus any attack produced by the Analyzer when this technique is used must be handchecked to determine whether or not it is a valid one.

Once the various lemmas have been proved, it is possible to use the Analyzer to perform a search. Each time a state is generated in its backwards search, it is checked against each of the lemmas to determine whether or not it is unreachable. If it is, the state is discarded. If it is not, the state is kept, and the Analyzer tries to determine how that state could have been reached. The Analyzer keeps a record of all paths generated, and lets the user know which paths begin in unreachable or initial states. The user can ask the Analyzer to display any path generated.

3.2 The NRL Protocol Analyzer Specification of the Needham-Schroeder Protocol

Our specification of the Needham-Schroeder protocol was written before we had seen Lowe's specification, so there are some important differences. First of all, we specified the request for and distribution of public keys. Secondly, we considered the possibility that old nonces might be compromised. To this end we included a transition that states how a nonce can be compromised any time after it is generated; we specified a "dummy" principal whose sole occupation is to deliver nonces to the intruder.

Finally, we made somewhat different assumptions about principals' ability to distinguish between different types of messages. The Protocol Analyzer always makes the assumption that principals, when asked to retrieve the head of a concatenated list, will always choose the correct word; that is, they will not pick a smaller chunk of the word, or the first two words in the list, for example. However, it makes no other assumptions, except for what the specification writer tells it to assume. Thus we can specify cases in which principals can or cannot tell different types of words from each other.

In our specification we originally started out by making no assumptions about recognizability except for those that are built into the Analyzer. This resulted in searches that failed to terminate, since the Analyzer kept on generating paths that contained local state variables containing longer and longer words. Thus we gave principals the ability to recognize when a message contained the appropriate amount of words. We also gave principals the ability to recognize whether a word was a public key. This seemed reasonable since some formatting of a public key

is necessary in order for it to be usable; for RSA, for example, a user needs to know which is the modulus and which is the exponent. Finally, we gave the server the ability to recognize names, since the server could tell whether or not something was a name by determining whether or not there was a public key associated with it, and we gave the initiator the ability to recognize the name of the principal it was trying to initiate a conversation with. However, when a principal was expecting a nonce, or the responder received what purported to be a name of an initiator, we specified no ability to recognize that a word of this type was what was actually received.

3.3 The Analysis

We analyzed four different versions of the Needham-Schroeder protocol. The first was the original version of the protocol, with the assumptions about typing that we mentioned in the previous section. The second was Lowe's fix of the protocol. The third was the fix with the assumption added that the responder could recognize when a word was not a name. The last was a specification as close to Lowe's specification of the fixed protocol as possible, for purposes of comparison. In this specification we left out the key distribution phase, and we also left out the assumption that nonces could be compromised. All our analyses took place using SWIProlog 2.1.14 running on a Sparc 20 running SunOS 5.4.

In all our analyses except one, we kept track of two statistics: the amount of time a search took, and the number of states that were generated during the search. By "state generated" we mean a state the Analyzer generated in the course of a search that was not immediately rejected as unreachable. This means that the correlation between the time a search took and states generated was not very tight, since some searches may generate more unreachable states than others, even if the number of states that had not been proved unreachable was the same. We also note that times could vary considerably given the load on the system; however we still include them because they help give a relative idea of how long the Protocol Analyzer performed on various problems.

We began by asking the Analyzer how to reach two final states: one in which the initiator of a protocol had accepted a nonce as coming from an honest responder, and one in which a responder had accepted a nonce as coming from an honest initiator. These did generate the attacks described at the beginning of this paper, as well as legitimate runs of the protocol. We also found the following attack:

The protocol proceeds normally in the first six steps:

1. $A \rightarrow S$: B
2. $S \rightarrow A$: $\{K_B, B\}_{K_S^{-1}}$
3. $A \rightarrow B$: $\{R_A, A\}_{K_B}$
4. $B \rightarrow S$: A
5. $S \rightarrow B$: $\{K_A, A\}_{K_S^{-1}}$
6. $B \rightarrow A$: $\{R_A, R_B\}_{K_A}$

At this point, I intercepts the message and sends it to A, as the first message in the authentication coming to A from some other party.

3'. $I \rightarrow A : \{R_A, R_B\}_{K_A}$

A decrypts the message, and, thinking that R_B is some party initiating communications, sends off a request to S for its public key.

4'. $A \rightarrow S$: R_B

Now I can learn R_B and impersonate A to B, causing B to think A has successfully responded to it:

7. $I_A \rightarrow B$: $\{R_B\}_{K_B}$

We did not attempt to perform an exhaustive search in this case.

Then, in order to better compare our work with Lowe's we tried asking the Analyzer the same or similar questions that Lowe asked FDR. Note that the attack we just mentioned above did not appear in these cases, since although B accepted R_B as from A when it had not been sent by A, A had in fact initiated the conversation.

In each case, we asked the Analyzer how two states could be found. The first, corresponding to Lowe's $AUTH\_RESP$, was a state in which an initiator A would accept a nonce R_B as coming from an honest responder B in response to a request containing R_A when:

1. B had not sent R_B to A in response to R_A, and;
2. R_A had not been compromised.

The second condition was necessary because the Needham-Schroeder protocol (or any other) could trivially be compromised if a secret was compromised during the session in which was generated.

The other state, corresponding to Lowe's $AUTH\_INIT$, was a state in which a responder B accepted a nonce R_A as coming from an honest initiator A to which it had responded with R_B when:

1. A had not initiated a session with B using R_A, and;
2. R_B had not been compromised.

We began by running the Analyzer on the second state. In this case, we were able to generate Lowe's attack and another. This attack, like the one on the initiator we presented earlier, relies upon a confusion between names and nonces.

We also found a number of minor variations on Lowe's and the above attack.

Unfortunately, we were not able to complete an exhaustive search in this case. The Analyzer state space exploded when it was well into its search, and this overwhelmed the resources of our system. However, this had an interesting result. After we finished the analysis of the other protocols, we returned to this case and attempted to complete the search by examining the unreachable states produced in the original search and using the results to suggest new lemmas to be proved about unreachability of states. We were still not successful in completing the search, but we proved similar lemmas for the other versions of the protocol, and found that we had reduced the number of states produced considerably. In most cases we reduced the amount of time taken by a proof too, although the reduction was not as dramatic. This is probably because the time taken up in

checking each of the lemmas made up for some of the time saved in checking fewer states.

Our search on the state in which the initiator is fooled went much more smoothly, and completed in about half an hour with no manual intervention, generating 155 states. The Analyzer found the following rather quaint attack, in which the initiator can be fooled if it is trying to talk to itself:

1. $A \rightarrow S : A$
2. $S \rightarrow A : \{K_A, A\}_{K_S^{-1}}$
3. $A \rightarrow A : \{R_A, A\}_{K_A}$

This is intercepted by I, who sends the following to A as responder, impersonating A as initiator.

6. $I_A \rightarrow A$: $\{R_A, A\}_{K_A}$

A as initiator checks for R_A, and believes that it has successfully responded to itself. It will now assume that the second field is a nonce, encrypt that field under K_A, and send the result to itself.

Again, this "attack" depends upon A's confusing messages containing names with messages containing nonces.

We next ran the Protocol Analyzer on Lowe's fix to the Needham-Schroeder protocol, but did not make the assumption that principals could distinguish between names and nonces. Verification of the protocol from the point of view of the responder (Lowe's $AUTH\_INIT$) could now be done completely automatically, and took a little under an hour, generating 689 states. Verification of the protocol from the point of view of the initiator (Lowe's $AUTH\_RESP$) took only about three and a half minutes, generating 66 states. In our second try using the additional lemmas we had proved, $AUTH\_INIT$ took 50 minutes generating 360 states, and $AUTH\_RESP$ took three minutes generating 34 states. Moreover, we were able to prove that the protocol was now sound with respect to these properties. No attacks were found in the exhaustive search. This is not surprising. First of all, Lowe's attack is prevented because the second message can no longer be passed off as coming from another user. Secondly, the various type of confusion attacks are prevented because, even though principals can not distinquish between a name and a nonce, they can distingish between different types of messages. One is an encrypted string containing one word, one contains two, and one contains three.

We next ran the Protocol Analyzer on the fixed protocol with typing built in. This did not change the results, but the verification was somewhat faster. Verification of soundness from the responder's point of view ($AUTH\_INIT$) took about forty-five minutes, generating 612 states, while verification from the initiator's point of view ($AUTH\_RESP$) took about two and a half minutes, generating 60 states. Using the additional lemmas, $AUTH\_INIT$ took 38 minutes generating 276 states, while $AUTH\_RESP$ took 2 minutes generating 26 states. Note that the ability to distinguish nonces from names did not give us much a benefit here, probably because principals already had the ability to distinguish between different types of messages.

Finally, for the purposes of comparison, we ran the Protocol Analyzer on a

version of the protocol that corresponded more closely with Lowe's, leaving off request for and distribution of public keys and the compromise of old nonces. For this protocol, verification of *AUTH_RESP* took under two minutes and generated 57 states, while the verification of *AUTH_INIT* showed even more marked improvement, taking about four and a half minutes and generating 128 states. Using the additional lemmas, *AUTH_RESP* took one minute 45 seconds and generated 32 states, while *AUTH_INIT* took six minutes to generate 75 states. In contrast, the FDR analysis took about thirty seconds each running on a Pentium PC for *AUTH_RESP* and *AUTH_INIT*, most of the time being spent on compiling the process definitions, and generated 251 states for each check [9]. In order to get a feel for how the two tools would perform on the same platform, we also used the Analyzer and FDR to analyze their respective protocol specifications on a Sparc 10 on which we had a copy of FDR. In this case the Analyzer took about four times as long as FDR, so FDR's lead was maintained.

4 Comparision Between the NRL Protocol Analyzer and Lowe's Analysis Method

There are a number of similarities between the NRL Protocol Analyzer analysis and Lowe's. Both model the protocol in terms of processes communicating across a channel controlled by a hostile intruder who can modify, intercept and destroy messages. Both deal with a possibly infinite state space by proving results showing that if an attack takes place it must do so within a finite state space, and then searching that space exhaustively. However, the way in both the search and the proofs are conducted are very different.

With the Protocol Analyzer, the user first uses the tool to generate a number of lemmas that identify a number of states as unreachable, and others that are only reachable if certain conditions are satisfied. The user then specifies an insecure final state, and the Analyzer searches backwards. Each time it encounters a state, it tests it against each of the conditions on unreachability stated by the lemmas. If the state satisfies one of these it is discarded. This means, that, every time a state is generated, it must be subjected to a number of tests. This is an inefficient way of searching, especially towards the end of an analysis when the number of lemmas is large. Thus it is not surprising that the Analyzer is much worse at searching through a large number of states than FDR, or any other model checker, for that matter. There is also the drawback that the user never really knows when the Analyzer has proved enough lemmas, except when a search has successfully completed. Thus the usual strategy is to generate a set of standard lemmas, try a search, see where it blows up, generate some more lemmas, and try a search again.

Lowe's analysis proceeded in a very different way. First, it was proved that the fixed Needham-Schroeder protocol was only vulnerable if an attack could be produced within a certain easily defined search space. Then a specification which generated exactly that search space was searched using FDR. This made

for a much for efficient search than that done by the Protocol Analyzer. On the other hand, the proof, which was somewhat complex, was informal and manual. Although the NRL Protocol Analyzer results give independent confirmation that no errors or hidden assumptions were made, this has not always been the case with hand analyses of cryptographic protocols. It is easy to let one's intuition about what should happen get in the way of one's perception of what is actually going on. An automated proof mechanism, which is not burdened by intuition, can often be at an advantage here. Finally, it also appears that the the lemma-proving capability of the Analyzer, if used carefully, can be used to generate a search space that is much smaller than that that would be searched by a model checker. Thus, our analysis of Lowe's fix of the Needham-Schroeder protocol generated 128 and 57 states for the two goals, while Lowe's analysis generated 251 states for each one. The model checker's more efficient search mechanism, however, more than makes up for any potential disadvantages in search time arising from this difference.

One marked difference between the Analzyer and FDR was the fact that the amount of time used to verify $AUTH\_INIT$ and $AUTH\_RESP$ differed greatly for the Analyzer but was the same for FDR. This is because the Analyzer works backwards from a specified final state, and the protocol terminates earlier for the initiator than the receiver. On the other hand, FDR works forwards from an initial state, and thus generates the same state space no matter what property is being verified. We can see how this property of the Analyzer could be useful in analyzing large-scale protocols. Such protocols generally consist of a number of smaller protocols interacting together. If the protocol is well-designed, this interaction should be minimal, and searching backwards from a specified final state should only require examination of a part of the possible state space. Thus, the Analyzer could be well-positioned to explore the security of such protocols.

In summary, it would appear that the Analyzer is better at proving than searching, while Lowe's analysis, at least in its present state, is better at searching that proving. However, Lowe's proofs have the advantage that they are directed to a particular goal, namely, proving that one needs to check only a previously defined finite search space. Thus no trial and error is involved. This suggests a possible direction for Analyzer proofs. Presently lemmas are generated according to rules of thumb, and proved blindly until the user is satisfied, by trial and error, that the Analyzer can complete its search. But it might be possible to indicate a large but finite search space, and see if the Analyzer can used to prove that an exhaustive search of that space is enough to verify a protocol's security.

If it is possible to direct the Analyzer's lemmas in such a way, this suggests a possible way of combining the Analyzer with a model checker. The Analyzer, with its built-in proof techniques specific to cryptographic protocols, could be used to narrow the original infinite search space to a finite one. A model checker could then be used to search the finite space. This would allow us to obtain the benefits of both technologies.

5 Conclusion

In this paper we compared the usefulness of the NRL Protocol Analyzer and model checkers to cryptographic protocol analysis by studying the application of the Analyzer and the model checker FDR to the same protocol. We found the two tools to be somewhat complementary. FDR was very good at exploring a finite state space quickly, but needed outside assistance to prove that exploring a finite state space was sufficient to prove security of a protocol. The Analyzer was considerably slower in exploring state spaces, but could be used to generate a finite state space and prove that exploring it was sufficient for proof of security. This in turn led to considerations of possible directions for future research in which the best parts of the two technologies could be combined.

6 Acknowledgements

I would like to thank Bill Roscoe for suggesting that I apply the NRL Protocol Analyzer to the Needham-Schroeder public key protocol, and Gavin Lowe for useful discussions on his use of FDR to analyze that protocol.

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. Formal Systems (Europe) Ltd. *Failures Divergence Refinement Users Manual and Tutorial, Version 1.4*, January 1994.
3. Dieter Gollmann. What do We Mean by Entity Authentication? In *Proceedings of the 1996 IEEE Computer Society Symposium on Security and Privacy*, pages 55–61. IEEE Computer Society Press, Los Alamitos, California, 1996.
4. C. A. R. Hoare. *Communicating Sequential Processes.* Prentice Hall, 1985.
5. Richard Kemmerer, Catherine Meadows, and Jonathan Millen. Three Systems for Cryptographic Protocol Analysis. *Journal of Cryptology*, 7(2), 1994.
6. D. Longley and S. Rigby. An Automatic Search for Security Flaws in Key Management Schemes. *Computers and Security*, 11(1):75–90, 1992.
7. Gavin Lowe. An attack on the Needham-Schroeder public key protocol. *Information Processing Letters*, 56:131–133, 1995.
8. Gavin Lowe. Breaking and fixing the Needham-Schroeder public-key protocol using CSP and FDR. In *Proceedings of TACAS.* Springer Verlag, 1996.
9. Gavin Lowe. personal communication, Feb. 1996.
10. Catherine Meadows. The NRL Protocol Analyzer: An overview. *Journal of Logic Programming*, 26(2):113–131, February 1996.
11. J. K. Millen, S. C. Clark, and S. B. Freedman. The Interrogator: Protocol Security Analysis. *IEEE Transactions on Software Engineering*, SE-13(2), 1987.
12. R. M. Needham and M. D. Schroeder. Using Encryption for Authentication in Large Networks of Computers. *Communications of the ACM*, 21(12):993–999, December 1978.
13. Einar Snekkenes. *Formal Specification and Analysis of Cryptographic Protocols.* PhD thesis, University of Oslo, May 1995.