Finite-State Analysis of Security Protocols

John C Mitchell

Dept. Computer Science, Stanford University, Stanford, CA, 94305
mitchell@cs.stanford.edu http://www.stanford.edu/~jcm

Abstract. Several approaches have been developed for analyzing security protocols. These include specialized logics that formalize notions such as secrecy and belief, special-purpose automated tools for cryptographic protocol analysis, and methods that apply general theorem-proving or model-checking tools to security protocols. This short document, written to accompany the author's invited lecture, provide background information and references on finite-state methods that use standard model-checking tools.

1 Introduction

A variety of protocols based on cryptographic primitives have been used to protect access to computer systems and to protect transactions over the internet. Two well-known examples are the Kerberos authentication scheme [KNT94,KN93], used to manage encrypted passwords on clusters of interconnected computers, and the Secure Sockets Layer [FKK96], used by internet browsers and servers to carry out secure internet transactions. Handshake or initialization protocols are often used to establish secret session keys for subsequent encrypted communication. These protocols also generally try to establish or confirm the identity of participants. Since a protocol used to establish secure communication may transmit secret keys across the network, correctness of such a protocol is essential for system security.

Over the past decade or two, a variety of methods have developed for analyzing and reasoning about protocols that use cryptographic primitives. The recognized approaches include specialized logics such as BAN logic [BAN89], special-purpose tools designed for cryptographic protocol analysis [KMM94], and theorem proving [Pau97] and model-checking methods using general purpose tools [Low96,Mea96,MMS97,Ros95,Sch96]. While specialized logics and methods have proven successful in many situations, the focus of this paper is the use of conventional logics and analysis methods. One reason is that a summary of methods based on conventional logics and formalisms necessarily involves explicit discussion of the modeling assumptions used to treat cryptographic primitives and security goals. Although modeling assumptions that are common to theorem proving and model checking will be discussed, finite-state methods will be emphasized since another CAV '98 lecture is expected to focus on theorem-proving methods.

In general, software and hardware verification involve establishing that a system design meets some specification. Depending on the level of verification, the specification may be a detailed description of precise inputs and expected resulting behavior, or may simply require that certain undesirable conditions do not occur. The main challenge, and the main property distinguishing verification from "checked execution," lies in some form of quantification over possible inputs and possible internal choices. We naturally want a system to perform correctly for any reasonable or anticipated input, and in spite of any variations in factors such as network transfer rates or the order in which tasks or processes might be scheduled. In many contexts, factors outside of our control are modeled nondeterministically, meaning that we want the specification to be satisfied even when the system operates in the most unlucky manner imaginable.

In security analysis, a system must not only behave correctly when internal events occur in an unlucky order, but also when a malicious adversary interferes with the operation of the system. From a general point of view, this might be viewed only as a matter of degree: a concurrent system must behave well when the scheduler is "malicious" and a cryptographic protocol must behave correctly when messages may be "maliciously" blocked, stored, altered and replayed. However, in practice, this is a significant matter of degree. A simple authentication protocol with only three steps leads to a nondeterministic system with over 500,000 states when a malicious intruder is added and two sessions are allowed to be carried out simultaneously [MMS97].

From a scientific point of view, the most significant problem in analyzing security protocols is accurate modeling of the malicious adversary (also referred to as the *intruder*). Generally speaking, the more detailed model, the more difficult the analysis and the more useful the results. Most current approaches, however, use essentially the same basic model of network communication and adversary capabilities. This model, largely derived from [DY81], provides a reasonable balance between tractability and insight into the behavior of protocols. However, since many reasonable attacks are not modeled within this common framework, it is likely that at least one branch of future research will focus on more detailed intruder models. The eventual goal would be to bring protocol analysis closer to the common intruder assumptions used in cryptography, which allow probabilistic attacks and incorporate complexity-theoretic assumptions (see, e.g., [Lub96,MvV97]). Other directions for further research involve improved proof methods, more efficient and expressive tools that take advantage of particular properties of cryptographic protocols, and methods that combine theorem proving with finite-state methods or specialized decision procedures.

2 Outline of a general approach

We have analyzed a range of protocols [MMS97,MSS98] using the verification tool Murφ [DDHY92], which does an optimized form of exhaustive search of finite-state nondeterministic systems. This work follows a general methodology that is similar to the approach used in CSP model checking [Low96,Sch96] of

cryptographic protocols. In outline, this approach uses the following sequence of steps:

1. *Formulate the protocol.* If the protocol is described in detail, then this generally involves simplifying the protocol by identifying the key steps and primitives. On the other hand, formal analysis requires a more detailed description of the behavior of protocol participants than the high-level descriptions sometime seen in the literature. For example, the common notation

$$A \rightarrow B : \{n\}_K$$
$$B \rightarrow A : \{n+1\}_K$$

   specifies that Alice send a number n to Bob, under encryption by key K, and Bob returns $n+1$ encrypted in the same way. However, this notation does not clearly specify what Bob will do in response to a message other than $\{n\}_K$. In order to analyze the effects of a malicious adversary, we need to understand the way that each participant will respond to any possible message.
2. *Add an adversary to the system.* It is commonly assumed that the network is under control of the adversary and that the adversary has the following capabilities:
   - overhear any message and prevent a message from reaching the intended recipient if desired,
   - decompose a message comprised of several fields into parts and decrypt any field for which the intruder has already obtained the encryption key,
   - remember all parts of all messages that have been intercepted or overheard,
   - generate messages and send them to any other protocol participant. Generated messages may be composed of any combination of initial intruder knowledge about the system and data obtained by overhearing or intercepting messages.

   The simplest way to model an adversary of this form is by a process that maintains a set of "known" data and chooses nondeterministically between possible actions at each step. While this provides an extremely limited range of possible attacks (with virtually no computation of attack messages), many protocol errors can be identified in this way. However, since there may be many possible intruder messages at each point, the state space may become very large. One relevant optimization is to eliminate useless messages from the adversary description (as discussed below).
3. *State the desired correctness condition.* Typical correctness conditions are related to authentication and secrecy. For example, if a server S enters a state in which it commits to open a session with client C, then C must have previously requested a session with S. We also specify that each datum that is initially secret remains secret. We have generally found it easy to state correctness conditions, but we have no reason to believe that there are no protocols where this step could prove subtle.

4. *Run the protocol* for some specific choice of system size parameters. Speaking very loosely, systems with 4 or 5 participants (including the adversary) and 3 to 5 intended steps in the original protocol (without adversary) are easily analyzed in minutes of computation time using a modest workstation. Doubling or tripling these numbers, however, may cause the system to run for many hours, or terminate inconclusively by exceeding available memory.
5. *Experiment with alternate formulations and repeat.* In examples which are found to be incorrect, it can be informative to repair the detected errors, either by strengthening the protocol or redirecting the efforts of the adversary. In cases where a protocol appears correct, it may be useful to investigate the consequences of strengthening the adversary, possibly by revealing some of the "secret" information to see how robust the protocol is to partial breaches of security.

In the general approach used by our group and other researchers, a protocol adversary is allowed to nondeterministically choose among possible actions. This is a convenient idealization, intended to give the adversary a chance to find an attack if there is one. At the same time, the set of messages an adversary may use to interfere with a protocol is severely limited. For example, an adversary cannot send a message with random data, for example, even though it is certainly possible for an attacker to send randomly chosen data in practice. Another limitation is that a nondeterministic setting does not allow us to analyze probabilistic protocols. Some of the advantages of probabilistic encryption, for example, are enumerated in [GM84].

3 Discussion

General-purpose finite-state analysis tools have proven useful for analyzing cryptographic or security-related protocols. The main challenges that arise are:

- State-space explosion, as with other tools,
- Subtleties involving formalization of the adversary or adversaries, and
- Subtleties involving properties of the encryption primitives, which may be modeled as completely secure black-box primitives, or primitives with other algebraic or "malleability" [DDN91] properties.

Formulating the adversary can be complex and can consume close to half of the time required to prepare a Murφ description of a protocol. There are two main challenges in formulating an adversary to a protocol:

- Formalize the "knowledge" of the adversary, as a function of some initial conditions and the set of messages an adversary has observed up to that point in the run of the protocol. By "knowledge," we mean set of possible entries that an adversary may insert in each message field.
- Select a finite set of possible adversary actions at any point in the run of the protocol, using the knowledge the adversary has at that point.

The first activity is more conceptual; the later involves pragmatic considerations.

For the protocols we have examined, we have characterized the knowledge of the adversary as simply the union of some set of initial data, such as public keys and the names of participants, and the data obtained by overhearing any message sent from one participant to another. This is relatively straightforward, given the message format and the assumption that an encrypted message may be decrypted only if the adversary knows the associated key. In particular, the initial data is finite (and small) and the data remembered from previous communication is also finite (and small), keeping us within the domain of finite-state systems. For adversaries of this form, we believe it will be possible to generate some or all of the adversary description automatically from formulations of other parts of the system.

With more subtle cryptographic assumptions, it becomes more difficult to model the adversary accurately. For example, if we want to allow the adversary to transmit $\{n{\cdot}i\}_K$, on the basis of seeing only $\{n\}_K$, for small integers $i = 1, 2, 3$, say, then we currently have to specify this explicitly in our intruder description. (Using the terminology of [DDN91], this change in the cryptographic assumption gives us a *malleable* encryption function, of which RSA is a specific example.) Moreover, we either have to model the message content as an integer and perform multiplication, or model the message content as a pair and put both n and i into the message. In any case, we must keep the state space small if the analysis is to remain practical.

The second general intruder consideration is that while we may want to run the protocol against the most capable, most nondeterministic adversary possible, this may cause the analysis to run more slowly, or consume more space, than desirable. While we have not had to weaken the adversary for complexity reasons, we have found it useful to optimize the adversary using information about the set of messages that other participants will actually accept. For example, suppose the receiver of a message computes a checksum and discards any message without the correct checksum. Then an unoptimized non-deterministic adversary may generate messages that do not have a correct checksum. However, these messages would have no effect on the protocol, if every other participant would reject them. Therefore, we can improve the running time and space of the analysis by rewriting the adversary to avoid generating these useless messages. Depending on the size and complexity of the system, we have found this sort of optimization to alter the size of the state space by one to several orders of magnitude. Therefore, we consider it a useful direction to explore automatic optimization of the adversary.

Acknowledgments: Thanks are due to Martin Abadi, David Dill, Cynthia Dwork, Stephen Freund, Li Gong, Mark Mitchell, John Rushby, Ulrich Stern, Vitaly Shmatikov and many others for helping me learn about security and for their efforts in carrying out work mentioned here.

References

[BAN89] M. Burrows, M. Abadi, and R. Needham. A logic of authentication. *Proceedings of the Royal Society, Series A*, 426(1871):233–271, 1989. Also appeared as SRC Research Report 39 and, in a shortened form, in ACM Transactions on Computer Systems 8, 1 (February 1990), 18-36.

[DDHY92] D. L. Dill, A. J. Drexler, A. J. Hu, and C. H. Yang. Protocol verification as a hardware design aid. In *IEEE International Conference on Computer Design: VLSI in Computers and Processors*, pages 522–5, 1992.

[DDN91] D. Dolev, C. Dwork, and M. Naor. Non-malleable cryptography (extended abstract). In *Proc. 23rd Annual ACM Symposium on the Theory of Computing*, pages 542–552, 1991.

[DY81] D. Dolev and A. Yao. On the security of public-key protocols. In *Proc. 22nd Annual IEEE Symp. Foundations of Computer Science*, pages 350–357, 1981.

[FKK96] A. Freier, P. Karlton, and P. Kocher. The SSL protocol version 3.0. `draft-ietf-tls-ssl-version3-00.txt`, November 18 1996.

[GM84] S. Goldwasser and S. Micali. Probabilistic encryption. *J. Computer and System Sciences*, 28:281–308, 1984.

[KMM94] R. Kemmerer, C. Meadows, and J. Millen. Three systems for cryptographic protocol analysis. *J. Cryptology*, 7(2):79–130, 1994.

[KN93] J.T. Kohl and B.C. Neuman. The Kerberos network authentication service (version 5). Internet Request For Comment RFC-1510, September 1993.

[KNT94] J.T. Kohl, B.C. Neuman, and T.Y. Ts'o. *The evolution of the Kerberos authentication service*, pages 78–94. IEEE Computer Society Press, 1994.

[Low96] G. Lowe. Breaking and fixing the Needham-Schroeder public-key protocol using CSP and FDR. In *2nd International Workshop on Tools and Algorithms for the Construction and Analysis of Systems*. Springer-Verlag, 1996.

[Lub96] M. Luby. *Pseudorandomness and Cryptographic Applications*. Princeton Computer Science Notes, Princeton University Press, 1996.

[Mea96] C. Meadows. Analyzing the Needham-Schroeder public-key protocol: a comparison of two approaches. In *Proc. European Symposium On Research In Computer Security*. Springer Verlag, 1996.

[MMS97] J.C. Mitchell, M. Mitchell, and U. Stern. Automated analysis of cryptographic protocols using Murφ. In *Proc. IEEE Symp. Security and Privacy*, pages 141–151, 1997.

[MSS98] J.C. Mitchell, V. Shmatikov, and U. Stern. Finite-state analysis of SSL 3.0. In *Proc. Seventh USENIX Security Symposium*, pages 201–216, 1998. Preliminary version presented at DIMACS Workshop on Design and Formal Verification of Security Protocols, September 1997; distributed on workshop CD.

[MvV97] A.J. Menzes, P.C. van Oorschot, and S.A. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 1997.

[Pau97] L.C. Paulson. Proving properties of security protocols by induction. In *10th IEEE Computer Security Foundations Workshop*, pages 70–83, 1997.

[Ros95] A. W. Roscoe. Modelling and verifying key-exchange protocols using CSP and FDR. In *CSFW VIII*, page 98. IEEE Computer Soc Press, 1995.

[Sch96] S. Schneider. Security properties and CSP. In *IEEE Symp. Security and Privacy*, 1996.