Modeling and Automated Verification of Authentication Protocols

Parosh Aziz Abdulla, Bengt Jonsson, Aletta Nylén
Dept. of Computer Systems
P.O. Box 325
S-751 05 Uppsala, Sweden
{parosh,bengt,d93ame}@docs.uu.se

Abstract

In this paper we present a general framework for modeling infinite-state authentication protocols that allows an unbounded number of protocol participants. We present a method for automated verification by performing a backwards reachability analysis where sets of insecure states are specified using constraints. We illustrate our method by showing how to model the Needham-Schroeder public-key protocol.

1 Introduction

The increasing popularity of distributed systems and the emergence of new technologies, such as electronic commerce, demands new security solutions. The cornerstone of security is often authentication, therefore easy-to-use methods and tools for modeling and verification of authentication protocols are needed.

Authentication protocols are designed for networks of protocol participants which are able to communicate asynchronously. The communication medium is essentially passive and can be manipulated by intruders that can intercept or interfere with network traffic. These protocols typically have few messages that may be encrypted with symmetric or asymmetric cryptographic keys.

Formal modeling and verification of cryptographic protocols has received much attention in recent years. Several frameworks for modeling authentication protocols and associated correctness properties have been presented. Burrows, Abadi and Needham [BAN89] have developed a belief-based logic for expressing properties of authentication protocols together with inference rules for reasoning about such properties. It is far from obvious how proofs in this framework could be automated. A simpler approach to modeling, which is not based on special purpose logic has been presented by Bolignano [Bol96] and by Schneider [Sch96a, Sch96b]. These approaches model protocols in rather standard transition-system based formalisms, and investigate how authentication can be expressed and analyzed. In general, the mechanization of proofs requires the use of interactive theorem provers. However, by employing carefully chosen abstractions, a finite-state model of the protocol can be created and subsequently analyzed in a finite-state automatic verifier. Manual intervention is still necessary for supplying and verifying the abstraction. In this work, we intend to improve on these approaches by attempting to automate as much as possible the verification of authentication protocols.

The main problems of verifying security protocols are that they are intended to run on networks of arbitrary size, i.e. networks with an unbounded number of protocol participants, and that the participants operate on data such as keys and nonces with values that are hard to bound. We intend to use and further develop recent advances in the analysis of infinite-state systems, which have been able to automate the analysis of certain classes of parameterized systems and systems operating on infinite-valued data structures. In particular, some of our earlier work in [AJ97] have concentrated on proving safety properties: this is applicable since most security properties of interest are safety properties [Bol97].

We present a network model based on the ideas of [Bol96]. Our goals in designing this model was to simplify the task of modeling authentication protocols, and to model authentication in a way that can be used with mechanized techniques of proving invariants of parameterized systems.

Security properties are generally safety properties that can be decided by analyzing reachability of network configurations. We present a constraint system that can be used to specify insecure network configurations.

In the next section, we introduce the basics of our protocol model. In Section 3 the network model is presented. In Section 4 we show how insecure network configurations can be specified using constraints and Section 7 explains how we perform verification using this model. A model of the Needham-Schroeder public-key protocol is presented in Section 8.

2 Basics

Authentication protocols usually run on networks of protocol participants which are able to communicate asynchronously over a medium. The medium is essentially passive and can be manipulated by malicious agents that can intercept or interfere with network traffic. The network consists of an arbitrary number of *principals*. Some of these principals will be considered to be trustworthy, they always act according to the protocol. The non-trustworthy principals may act according to the protocol, but they may also perform malicious actions, such as interfering with network traffic, impersonating other principals and perhaps even cooperating with each other. Since the non-trustworthy principals can manipulate the medium we will model these principals together with the medium as a single process, the *enemy*. The enemy can store and (if possible) decrypt the messages circulating on the medium. It can also encrypt data to create new messages and send messages in order to mislead trustworthy principals. Similar approaches have been used in [Sch96b, Bol96, Mea96].

The trustworthy principals will be modeled as a set of identical processes called *users*. The users can play different roles in different protocol sessions, e.g. the user that initiates a session acts as an initiator and the user that gets contacted by the initiator acts as a responder. In general, a user may be engaged in several sessions simultaneously.

In our framework, we model a protocol as a transition system, with states (called *configurations*), and transitions among configurations. A *configuration* is built from (typed) objects and their attributes. Thus, for each protocol we must first decide on a finite set of types of objects. This choice can vary between protocols. A typical choice could be to let objects be of the following types.

- *principals*, ranged over by A, B, C.
- A set of *session local states*, ranged over by over by $\mathcal{S}$. A session local state is used by a principal to keep track of the current state of the session. Each time a principal engages in a new protocol session, a new local session state is created.
- A set of *keys*, ranged over by $\mathcal{K}$.
- A set of *nonces*, ranged over by $\mathcal{N}$. A nonce is a random number intended for use in a *single* protocol run. The purpose of using a nonce is to ensure that a message is recent and not a replay of a previously sent message.
- A set of *control states*, ranged over by Q, used to represent states of processes during the sessions.
- A set of *rôles* which typically will consist of the two elements: `initiator` and `responder`. The set is used to define whether a certain process acts as an initiator or a responder during a given session.

We will use typewriter font to designate specific objects. Thus we use e.g. `A`, `B` to describe specific principals, and `S` to describe a specific session local state, etc.

The sets of principals, session local states, keys, and nonces are all infinite, while the set of control states is finite. A key is said to be symmetric if it is used both for encryption and decryption, otherwise it is asymmetric. An asymmetric encryption (decryption) key K has a corresponding decryption (encryption) key K^{-1}, which is said to be the inverse key of K. The inverse key of a symmetric key is the key itself.

In the sequel, we make the assumption that encrypted messages are not altered, e.g. decomposed into encrypted parts, except by decryption and re-encryption. In other words, messages cannot be altered by a principal that does not know the decryption and encryption keys.

In the description of a specific protocol, we will assume a finite set of *attributes*. An attribute is associated with objects of a specific type. The value of an attribute is an object. A typical use of attributes is to describe different properties of sessions, or to specify the keys belonging to different principals, etc. We use the notation $object_1.attr = object_2$ to denote that the attribute $attr$ of $object_1$ has value $object_2$.

As an example, in the Needham-Schroeder Protocol (which we shall consider in Section 8), a session local state `S` has the following attributes, with corresponding types.

$$\begin{array}{l} S.\texttt{self} : \text{principal} \\ S.\texttt{peer} : \text{principal} \\ S.\texttt{initiator\_nonce} : \text{nonce} \\ S.\texttt{responder\_nonce} : \text{nonce} \\ S.\texttt{control} : \text{control state} \end{array}$$

Messages which are exchanged during a session, are formed according to the following rules.

- An object (e.g., a principal, key, or nonce) is a message.
- A tuple of messages is a message.
- A key applied to a message is a message.

In a particular protocol configuration, the enemy knows particular messages. These messages can be used to create new messages, which can then be sent to principals. Initially, the enemy knows some messages. He can deduce new messages from given ones. This deduction relation is usually generated by the operations:

- *composition:* If M_1 and M_2 are known messages, then the composite message $\langle M_1, M_2\rangle$ can be deduced.
- *decomposition:* If a composite message $\langle M_1, M_2\rangle$ is known, then the messages M_1 and M_2 can be deduced.
- *encryption:* If a message M and a key K are known, then the encrypted message $[M]_K$ can be deduced[1].
- *decryption:* If an encrypted message $[M]_K$, and the inverse K^{-1} of K are known, then the message M can be deduced.

If M is a set of messages, then M^* denotes the union of M and the set of messages that can be deduced from M by the operations listed above.

3 The Network Model

We assume that we work with infinite sets $\mathcal{A}$ of principals, $\mathcal{S}$ of session local states, $\mathcal{K}$ of keys, $\mathcal{N}$ of nonces.

A protocol P is a triple $\langle Q, Decl, R\rangle$, where

- Q is a finite set of control states of principals;
- $Decl$ is a declaration of relevant attributes of the different object types; and
- R is a finite set of *rules*, which guide the transitions performed by the protocol. The exact definition of rules follows below.

A *configuration* of P is defined by giving (i) a set of active objects, (ii) the values of attributes of these objects, and (iii) the state of the intruder. The state of the intruder is represented by the set of messages known to him. Formally, a *configuration* γ is a triple $\langle X, V, known\rangle$, where

- X is a a finite set of objects (which are active at γ);

[1] Sometimes, we use $[M_1, M_2]_K$ to denote $[\langle M_1, M_2\rangle]_K$.

- V is a partial function which assigns values to attributes of objects in X; and
- *known* is the set of messages known by the intruder.

Observe that $known^*$ is the set of messages which can be deduced from *known*.

Rules The dynamic behavior of P is defined by the finite set R of *rules*. A rule $r \in R$ has a finite set of parameters, i.e., free variables. Each free variable ranges over objects of a given type. We use lowercase letters to range over variables: thus a, b, c range over free variables of type principal, and similarly for other types. We use x, y to range over variables of all types.

A *meta-message* is a "message" built only from variables, i.e., it is formed according to the following rules.

- A variable is a meta-message.
- A tuple of meta-messages is a meta-message.
- A variable of type key applied to a meta-message is a meta-message.

A rule consists of a guard and a body.

- the guard is a conjunction of conditions of form
  - $fresh(x)$, where x is a variable. This type of conditions is used to model generation of fresh sessions, nonces, and keys.
  - $msg \in known^*$, where msg is a meta-message consisting only of parameters of the rule.
  - $x.attr = y$ where $attr$ is an attribute, x is a parameter of the rule and y is a parameter of the rule or a constant, such as a control state.
- the body consists of a collection of the following kinds of statements.
  - $x.attr := y$ where $attr$ is an attribute, x is a parameter of the rule and y is a parameter of the rule or a constant, such as a control state.
  - $known := known \cup \{msg\}$, where msg is a meta-message consisting only of parameters of the rule.

We now define the effect of executing a rule in a given protocol configuration. An *instance* of a rule is obtained by replacing the parameters of a rule by objects. The rule is enabled if its guard is true (in the obvious way), where the conjunct $fresh(x)$ requires that the object X is not present in the configuration. An enabled rule is executed by possibly adding to the configuration those objects in the instantiated rule which were not present before, and then executing the statements in the body.

For configurations γ_1, γ_2, and a rule r, we use $\gamma_1 \longrightarrow_r \gamma_2$ to denote that we can reach γ_2 from γ_1 through the execution of r. We define the *discrete transition relation* $\longrightarrow$ by $\longrightarrow = \cup_{r \in R} \longrightarrow_r$. We let $\stackrel{*}{\longrightarrow}$ denote the reflexive transitive closure of $\longrightarrow$.

4 Constraints

A constraint is a formula of form

$$\exists x_1, \ldots, x_n \phi(x_1, \ldots, x_n)$$

where $x_1, \ldots, x_n$ are variables as usual, and where ϕ is the conjunction of atomic formulas of the following form

- $x_i.attr = x_j$.
- $msg \in known^*$, where msg is a meta-message.

Since all variables are always implicitly existentially quantified, we will sometimes for brevity drop the quantification, and simply write $\phi(x_1, \ldots, x_n)$.

Intuitively, a constraint ϕ asserts the existence of a set of objects and known messages, whose attributes (parameters) are in some cases identified. Fields are identified if the variables that occupy them are identical. Different variables may or may not represent the same objects in a given configuration which satisfies the particular constraint.

A configuration γ *satisfies* a constraint $\exists x_1, \ldots, x_n \phi(x_1, \ldots, x_n)$ if for each x_i there is an object in the configuration, such that the $\phi(x_1, \ldots, x_n)$ is satisfied. Note that different variables can be mapped to the same object. We let $[\![\phi]\!]$ denote the set of configurations satisfying ϕ.

We define an ordering $\preceq$ on constraints by $\phi_1 \preceq \phi_2$ if and only if $[\![\phi_2]\!] \subseteq [\![\phi_1]\!]$. The ordering can be characterized as follows. We have

$$\phi_1(x_1, \ldots, x_n) \preceq \phi_2(y_1, \ldots, y_m)$$

if for each x_i we can substitute a corresponding y_j such that each conjunct in $\phi_1(x_1, \ldots, x_m)$ occurs in $\phi_2(y_1, \ldots, y_n)$, or can be deduced by decryption.

5 Reachability

A configuration γ_0 is said to be *initial* if there are no active objects at γ_0. We say that a configuration γ is *reachable* if $\gamma_0 \stackrel{*}{\longrightarrow} \gamma$, for some initial configuration γ_0. For a finite set of constraints Φ, we say that Φ is *reachable* if there is a $\phi \in \Phi$ and a reachable $\gamma \in [\![\phi]\!]$.

The reachability problem is defined as follows

Instance: A finite set of constraints Φ.

Question: Is Φ reachable?

To check the reachability of Φ from γ, we perform a reachability analysis backwards. Let $pre(\phi)$ denote the set $\{\gamma \;:\; \exists \gamma' \in [\![\phi]\!] : \gamma \longrightarrow \gamma'\}$, and $pre(\Phi)$ denote the set $\{\gamma \;:\; \exists \phi \in \Phi : \exists \gamma' \in [\![\phi]\!] : \gamma \longrightarrow \gamma'\}$. Note that $pre(\Phi)$ is equivalent to $\cup_{\phi\in\Phi} pre(\phi)$. Starting from Φ we define the sequence $\Phi_0, \Phi_1, \Phi_2, \ldots$ of finite sets of constraints by $\Phi_0 = \Phi$ and $\Phi_{j+1} = \Phi_j \cup pre(\Phi_j)$. Intuitively, Φ_j denotes the set of configurations from which Φ is reachable by a sequence of at most j transitions. Note that the sequence is increasing i.e., that $[\![\Phi_0]\!] \subseteq [\![\Phi_1]\!] \subseteq [\![\Phi_2]\!] \subseteq \cdots$. If there is a k such that $[\![\Phi_k]\!] = [\![\Phi_{k+1}]\!]$, then $[\![\Phi_k]\!] = [\![\Phi_j]\!]$ for all $j \geq k$. It follows that Φ is reachable if and only if there is an initial configuration γ_0 such that $\gamma_0 \in [\![\Phi_k]\!]$, which is easily checked since Φ_k is a *finite* set of constraints. In general, it cannot be guaranteed that a fixpoint is reached.

6 Computing *pre*

We have already covered the ordering relation between constraints. Let us now turn to calculating *pre* of a constraint. So, assume a constraint and a rule. We will compute $pre(\phi)$ as $\cup_{r\in R} pre(r, \phi)$, where $pre(r, \phi)$ denotes the set $\{\gamma \;:\; \exists \gamma' \in [\![\phi]\!] : \gamma \longrightarrow_r \gamma'\}$. If ϕ is a constraint and r is a rule, then $pre(r, \phi)$ is calculated according to the following steps:

For all substitutions of variables in the constraint and in the rule

1. Whenever $x.attr := y$ occurs in the rule, remove all occurrences of $x.attr = y$ from the constraint.
2. Whenever $known := known \cup \{msg\}$ occurs in the rule, then: For each relevant assumption about the contents of $known$, remove all occurrences of messages m, such that $m \in \{msg\}^*$, from the constraint. Note that $known \cup \{msg\}$ must satisfy ϕ.
3. Check that if a condition $fresh(x)$ occurs in the rule (after substitution), then x no longer occurs in the constraint. If x does occur in the constraint, abort this substitution.
4. Add the guard to the constraint.

We observe that in general, the smallest constraint is obtained by choosing those parameters of the rule that occur only in the guard distinct from all other variables.

7 Authentication

We verify authentication by checking whether insecure states, i.e. states where violation of authentication has occured, can be reached from an initial state. We model sets of insecure states by constraints. Usually, violation of authentication can be described a conjunction of conditions on local states and on the set of messages that the intruder knows. For example the following constraint characterizes an insecure set of states, where a nonce used in a certain session is also known to the intruder.

$$\exists s, n, a, b \left[\begin{array}{ll} & s.\mathtt{self} = a \\ \wedge & s.\mathtt{peer} = b \\ \wedge & s.\mathtt{initiator\_nonce} = n \\ \wedge & s.\mathtt{control} = \mathtt{connected} \\ \wedge & n \in known^* \end{array} \right]$$

8 Example: The Needham-Schroeder Public-Key Protocol

The Needham-Schroeder public-key protocol ([NS78]) is based on public-key cryptography. Each principal A possesses a public key, denoted K_A, which any other principal can obtain from a key server. It also possesses a secret key, K_A^{-1}, which is the inverse of K_A.

The messages exchanged consists of agent ids and nonces, random numbers intended to be used in a *single* run of the protocol. The goal of the protocol is to establish mutual authentication between an initiating principal A and a responding principal B. The protocol described and modeled here is a reduced version of the protocol. The reduced version omits the steps where the agents request and receive each others public keys from the key server and instead assumes that the participants initially have knowledge of all public keys.

The reduced protocol can be described as:

$$\begin{array}{l} A \rightarrow B : [N_A, A]_{K_B} \\ B \rightarrow A : [N_A, N_B]_{K_A} \\ A \rightarrow B : [N_B]_{K_B} \end{array}$$

The first message, which can only be understood by responder B, tells B that initiator A wants to establish communication and secretly communicates a nonce identifier N_A. In the second message A receives a proof of Bs presence in the form of the same nonce identifier encrypted with As public key. The second message also contains a nonce challenge which if properly responded to, in message 3, proves As presence to B. Thus after a protocol run the secrets N_A and N_B will have been established and the presence of both A and B will have been proved.

In [Low96], Lowe presented an attack upon this protocol, which allows an intruder to impersonate another agent. He also suggests an adaption of the protocol to make it secure, in which the second message has been changed to

$$B \rightarrow A : [N_A, N_B, B]_{K_A}$$

This version of the protocol will be used in our example model.

The Needham-Schroeder Public-Key protocol is defined by the control states `idle`, `init_wait`, `resp_wait`and `connected`, where `idle`is the initial value and the state `connected`denotes that authentication is completed. The attributes of the session local states are:

`control`	the session local state
`self`	a principal
`peer`	a principal
`initiator_nonce`	a nonce created by the initiator
`responder_nonce`	a nonce created by the responder

The rules describing the transitions of the protocol are listed in Figure 1. The first rule corresponds to the initiator sending the first protocol message. The second rule is a combination of the responders receiving the first message and sending the second. The third rule describes the initiators receiving the second message and sending the third message and the fourth rule is where the responder receives the third message and the protocol run is completed.

In an initial configuration, the set *knows* consists of all public keys, the enemy's private key, and the names of all principals in the network.

1.

$$fresh(n) \wedge fresh(s) \rightarrow \begin{array}{l} s.\texttt{control} := \texttt{init\_wait}, \\ s.\texttt{self} := a, \\ s.\texttt{initiator\_nonce} := n, \\ s.\texttt{peer} := b, \\ known := known \cup \{[n,a]_{k_b}\} \end{array}$$

2.

$$fresh(rn) \wedge fresh(s) \wedge [in,a]_{k_b} \in known^* \rightarrow \begin{array}{l} s.\texttt{control} := \texttt{resp\_wait} \\ s.\texttt{self} := b \\ s.\texttt{peer} := a \\ s.\texttt{initiator\_nonce} := in \\ s.\texttt{responder\_nonce} := rn \\ known := known \cup \{[in,rn,b]_{k_a}\} \end{array}$$

3.

$$\begin{array}{l} s.\texttt{control} = \texttt{init\_wait} \wedge \\ s.\texttt{initiator\_nonce} = in \wedge \\ s.\texttt{self} = a \wedge \\ s.\texttt{peer} = b \wedge \\ [in,rn,b]_{k_a} \in known^* \end{array} \rightarrow \begin{array}{l} s.\texttt{control} := \texttt{connected}, \\ s.\texttt{responder\_nonce} := rn, \\ known := known \cup \{[rn]_{k_b}\} \end{array}$$

4.

$$\begin{array}{l} s.\texttt{control} = \texttt{resp\_wait} \wedge \\ s.\texttt{responder\_nonce} = n \wedge \\ s.\texttt{self} = b \wedge \\ [n]_{k_b} \in known^* \end{array} \rightarrow s.\texttt{control} := \texttt{connected}$$

Figure 1: Rules of the Needham-Schroeder Public-Key Protocol.

References

[AJ97] Parosh Aziz Abdulla and Bengt Jonsson. Verifying networks of timed processes, 1997. in preparation.

[BAN89] Michael Burrows, Martín Abadi, and Roger Needham. A logic of authentication. In *Proceedings of the Royal Society of London A*, volume 426, pages 233–271, 1989.

[Bol96] Dominique Bolignano. An approach to the formal verification of cryptographic protocols. In *Proc.* 3^{rd} *ACM Conference on Computer and Communication Security*, March 1996.

[Bol97] Dominique Bolignano. Towards a mechanization of cryptographic verification. In 9^{th} *International Computer-Aided Verification Conference, CAV'97*, June 1997.

[Low96] Gavin Lowe. Breaking and fixing the Needham-Schroeder public-key protocol using FDR. In Margaria and Steffen, editors, *Proceedings of TACAS*, volume 1055 of *Lecture Notes in Computer Science*, pages 147–166. Springer Verlag, 1996.

[Mea96] Cathrine Meadows. The NRL protocol analyser: An overview. *Journal of Logic Programming*, 1996.

[NS78] Roger M. Needham and Michael D. Schroeder. Using encryption for authentication in large networks of computers. *Communications of the ACM*, 21(12):993–999, December 1978.

[Sch96a] Steve Schneider. Security properties and CSP. In *IEEE Computer Society Symposium on Research in Security and Privacy*, Oakland, 1996.

[Sch96b] Steve Schneider. Using CSP for protocol analysis: the Needham-Schroeder public-key protocol. Technical Report CSD-TR-96-14, Royal Holloway, University of London, 1996.