Notes on Security Protocols

CSG 399

February 19, 2006

Basis of analyzing security protocols:

- Model the protocol
- Give a specification capturing a property of interest
- Check that the model satisfies the specification

We give a general model for protocols (encompassing all known models).

Assume P a set of plaintext, K a set of keys, $\{m\}_k$: encryption of m with key k, $(m_1, \ldots, m_n)$: message concatenation.

k^{-1}: inverse of key k; message $\{m\}_k$ can be decrypted with key k^{-1}; when $k = k^{-1}$, we have a symmetric key cryptosystem (e.g., DES, AES); when $k \neq k^{-1}$, we have an asymmetric key cryptosystem, also known as a public key cryptosystem (e.g., RSA).

Given a protocol of the form:

$$\begin{aligned} A &\longrightarrow B: \ \{A, n_A\}_{K_B} \\ B &\longrightarrow A: \ \{n_A, n_B\}_{K_A} \\ A &\longrightarrow B: \ \{n_B\}_{K_B} \end{aligned}$$

Note that A and B in the protocol are really variables: they can be instantiated by the actual agents involved in any interaction. These variables are called the roles of the protocol; for a two-party protocol, these are often called the initiator role, and the responder role.

Idea: set a number of agents $A = \{1, 2, \ldots, n\}$. Look at all the possible ways in which these agents can execute the protocol to communicate together. This forms a system. At any point in time, the system is in some global state. Upon actions, the global state of the system changes.

The possible initial states of the system: given a subset of A, the initiators, and who each initiator wants to communicate with.

A *system* is a tuple $S = (G, I, T)$ where S is a set of global states, $I \subseteq G$ is a set of initial states, and T is a (labeled) transition relation, so that $(g, a, g') \in T$ if global state g can become g' after action a is performed.

Often, a global state will have some structure, e.g., it can be represented as a tuple $(s_e, s_1, \ldots, s_n)$, where s_i is the state of each agent in the system (messages received, private keys, public keys of others, etc), and s_e is the state of the environment (messages in transit, what's on the network, etc).

What is the system corresponding to a protocol (and some set of agents)? It depends on the assumptions we make

- Are messages delivered reliably?
- How long do messages take to arrive when they are delivered?
- Can different agents perform an action at the same time?
- Can an agent play two roles at the same time?

Big assumption: agents are honest, that is, they follow their protocols. (This might be hard to make precise.)

Different assumptions will give rise to different systems.

Proving a property of a protocol is really proving a property about a system generated from the protocol. (Thus, we are really proving a property in a particular context that captures a number of assumptions.)

It is often unmanageable to describe this system by hand.

A lot of work focuses on different ways to describe systems, and to prove properties about the system without actually having to generate the whole system.

Dealing with Adversaries

To model the fact that protocols are generally studied in an adversarial setting, we introduce an additional agent in the system, called the *adversary* (or attacker, intruder). Of course, we need to account for the adversary in the system.

Two different kind of adversaries are generally considered

1. Passive adversary: the adversary does not affect the flow of messages in the network (an eavesdropping adversary)
2. Active adversary: the adversary can affect the flow of messages; but how much? Often, active adversaries can intercept messages, add new messages in the system, duplicate messages.

For simplicity, for now, we consider only passive adversaries.

Systems are as before, except with an additional agent, the passive adversary, who does nothing. The state of the adversary is simply the record of all traffic (plus some initial public keys).

Let us look at a typical security property in this setting: message secrecy. Here is one way to translate it into a property of the system: is there a state in the system where the adversary "knows" a particular message (that presumably we want to keep secret)?

What does it mean for an adversary to "know" a message? This is actually a deep question.

The literature has focused on a particular definition, due to Dolev and Yao. It assumes that an adversary cannot crack the encryption (perfect cryptography assumption). But the adversary can take apart unencrypted messages and decrypt messages encrypted with keys he knows. As well, he can concatenate and take part concatenations.) The adversary knows m if m can be derived by the above operations.

This can be formalized using a little deductive system. $H \vdash m$, to represent message m can be derived from the messages in H.

$$\frac{m \in H}{H \vdash m} \quad \frac{H \vdash (m_1, \ldots, m_k)}{H \vdash m_i} \quad \frac{H \vdash m_1 \quad H \vdash m_k}{H \vdash (m_1, \ldots, m_k)}$$

$$\frac{H \vdash m \quad H \vdash k}{H \vdash \{m\}_k} \quad \frac{H \vdash \{m\}_k \quad H \vdash k^{-1}}{H \vdash m}$$

As an example, see that $H = \{\{k_1^{-1}\}_{k_2}, \{m\}_{k_1}, k_2^{-1}\} \vdash m$.

$$\dfrac{\dfrac{\{m\}_{k_1} \in H}{H \vdash \{m\}_{k_1}} \quad \dfrac{\dfrac{\{k_1^{-1}\}_{k_2} \in H}{H \vdash \{k_1^{-1}\}_{k_2}} \quad \dfrac{k_2^{-1} \in H}{H \vdash k_2^{-1}}}{H \vdash k_1^{-1}}}{H \vdash m}$$

The property that message m is kept secret from the adversary can be interpreted as: At no state of the system can the adversary derive m from the messages he has intercepted.

Show that NS with a passive adversary and only two agents preserves the secrecy of n_A (and n_B), at least with respect to the adversary.

Active Adversaries

Common assumptions in formal methods about what an active adversary can do: adversary controls the network (intercept any message, forward any message), and adversary can manipulate messages as in the passive case, and send them on the network to any other agent.

Adversary can inject on the network any message he knows. Again, depends on the definition of knowing a message. Thus, in a Dolev-Yao setting, the adversary, having intercepted messages H, can inject any message m with $H \vdash m$. (This is what is typically known as the Dolev-Yao adversary.)

The system corresponding to a protocol in the presence of an active adversary is essentially derived as follows.

- Every agent runs the part of the protocol corresponding to his role

- The adversary intercepts all messages and nondeterministically inject messages into the system

So the system includes, at every state, a transition corresponding to any send of any possible message that the adversary knows at that state.

Of course, this makes the system even bigger.

The property of message secrecy of m remains the same: is there a state of the system that is such that $H \vdash m$, where H is the intercepted messages of the agent at the state?

Challenging question: how do you reason about a system with an active adversary, that is, determine, determine if a property holds in a model? With some luck, we can do it without constructing the whole model.

A variant of the active adversary is to see if the other agents are allowed to start an interaction with the adversary - this is an insider adversary, or a compromised agent.

The formal methods community has tended to analyze protocols in the presence of a Dolev-Yao adversary.

- Quite general adversary
- Provides strong guarantees
- Easy to work with (deductive system $\vdash$ is well behaved)

There are other possibilities. We can add more operations to the algebra of messages, and let the adversary work with them.

Lowe describes an extension of the Dolev-Yao model that can derive guesses that he can validate (off-line guessing). Consider the following protocol, where A and S share a password p_A:

$$\begin{aligned} A &\longrightarrow S: \ A \\ S &\longrightarrow A: \ n_S \\ A &\longrightarrow S: \ \{n_S\}_{p_A} \end{aligned}$$

An adversary can intercept n_S, $\{n_S\}_{p_A}$, go home, and try to quietly guess p_A. Why? Because he has enough information to determine when a guess is correct: he has n_S, so he can try every password p until $\{n_S\}_p = \{n_S\}_{p_A}$, at which point he knows that he has guessed correctly. (It is not sufficient to be able to guess—we need to figure out whether the guess is correct.)

Lowe formalizes this using a relation $H \vdash_L m$, which holds if either $H \vdash m$, or m is a guess that can be validated in H. (The definition of validation is given in Lowe's paper, and is understandably complex.)

Upshot: the adversary knows more messages, and so can mount more attacks, and is more powerful. How reasonable is such an adversary? If we restrict the adversary to guesses he can "reasonably" make (e.g., looking up passwords in a dictionary, i.e. a weak-password attack.), then this makes sense.

Note that in cryptography, the adversary used is very different. Rather than assume that the adversary cannot crack the encryption, they assume that the adversary can perform any probabilistic polynomial-time computation. This gives a class of results orthogonal to formal methods approaches. Proofs are also much much harder to automate (and, in fact, to understand).

Authentication

We saw secrecy of messages as a particular form of security property. Another very common property is authentication. Now, there are many notions of authentication. (Cf. Gollmann paper) We saw that the NS protocol, however, preserves the secrecy of n_A, and this can be helpful for deriving, say, session keys.

The NS protocol was really meant as an authentication protocol: so that at the end of the protocol, A knows that she has been talking to B, and B knows that he has been talking to A.

This is a very "anthropomorphic" way of describing the property. Moreover, recall that we subscribe to the view that properties of protocols are really properties of some underlying system. So what are the properties of systems that correspond to the above?

There are a number of answers in the literature. At one extreme, could just be that after A has finished his part of the interaction, A knows that B was alive during the period where the interaction took place, but not much else. (You can come up with examples where A would be interested in knowing that B is logged in, but nothing else.)

Recall the man-in-the-middle attack against NS; for the attack to be successful, A must have been alive. So after the interaction, despite the attack, B knows that A was there and interacted.

At the other extreme, here is a definition due to Roscoe: If A has finished his part of the interaction (that is, ran his part of the protocol to completion) and believes this was with B, then it must be the case that B has gone through exactly the steps specified by his part of the protocol interacting with A.

This translates into the following properties on traces of the system: for every trace (sequence of states starting at some initial state), if the trace hits a state where A has sent his last message believing he was interacting with B, then preceding states in the trace have all the "right" messages sent by B, in the right order.

This is a (somewhat) precise property of a system, but as a specification, it is problematic, because it explicitly mentions the protocol (the messages); this is not a general definition of authentication, just a specific property of that particular protocol. If you change the protocol, you need to change the specification. Compare this to the message secrecy specification.

Woo and Lam propose another approach:

- Augment the protocol with "logging" events such as "starting the protocol" and "ending the protocol".
- Check that every trace is such that the logged events agree.

For NS:

$$
\begin{aligned}
A &: \text{ logs ``}A\text{ starting interaction with }B\text{''} \\
A \longrightarrow B &: \{A, n_A\}_{K_B} \\
B \longrightarrow A &: \{n_A, n_B\}_{K_A} \\
A \longrightarrow B &: \{n_B\}_{K_B} \\
B &: \text{ logs ``}B\text{ ending interaction with }A\text{''}
\end{aligned}
$$

The log does not exist at runtime in the real protocol. We just use it as a device to analyze the protocol. Only the honest agents can access (write) into the log. This lets us track events performed by the honest agents.

We want a correspondence property: in every trace, if an entry "X ending interaction with Y" appears in the log, then it is preceded by an entry "Y starting interaction with X".

This is not satisfied by NS under Dolev-Yao adversary that is an insider, by the man-in-the-middle attack. Indeed, consider the trace:

$$
\begin{aligned}
A &: \text{ logs ``}A\text{ starting interaction with }X\text{''} \\
A \longrightarrow X &: \{A, n_A\}_{k_X} \\
X \longrightarrow B &: \{A, n_X\}_{k_B} \\
B \longrightarrow A &: \{n_B, n_A\}_{k_A} \\
A \longrightarrow X &: \{n_B\}_{k_X} \\
X \longrightarrow B &: \{n_B\}_{k_B} \\
B &: \text{ logs ``}B\text{ ending interaction with }A\text{''}
\end{aligned}
$$

But this trace fails the correspondence property, since there is no entry corresponding to "B ending interaction with A" in the log.

One question: how do you efficiently check correspondence properties? Another question: do correspondence assertions really capture authentication?