ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

S. Mödersheim
Ch. Sprenger
AS 2009

Formal Methods for Information Security

Exercise Sheet 1

Hand-in date: Oct 5, 2009

Exercises *do not count* towards the final grade for the course, but you can hand them in to have them corrected in order to get the feedback. This is optional but recommended.

Assignment 1.1: Applying Formal Methods.

In this exercise, assume you've been hired to do some consulting work applying formal methods. The Swiss government wants you to evaluate their new software which is designed to make referenda (*Volksabstimmungen*) automated and online. Their contractors, Spitzenprotokoll AG, have designed the following protocol:

$$\begin{array}{lll} 1. & A \to S: & A \\ 2. & S \to A: & \{Q, N_S\}_{\mathsf{pk}(A)} \\ 3. & A \to S: & \{Ans_Q, N_S\}_{\mathsf{pk}(S)} \end{array}$$

where A is a voter, S is the voting server, N_S is a nonce sent by the server to ensure freshness, Q is a referendum question, and Ans_Q is A's answer to that question. Assume S and A share their respective public keys in advance.

Your job is to compare this protocol with the existing, physical voting method, which we assume is secure. We make the standard Dolev-Yao assumptions presented in the lecture.

(a) Does this protocol provide anonymity? Can an attacker tell who has voted?

(b) Does it provide confidentiality? Can an attacker find out, for a given A, how he or she voted?

(c) Does the protocol provide authentication? Can S be sure that the answer came from A?

(d) Can each voter vote at most once?

(e) Is availability guaranteed? Can A be sure that she can vote if she wants to?

(f) Is integrity provided? Does S know that a given answer hasn't been modified.

Solution

(a) *Anonymity*: An attacker can easily tell who has voted from the first message.

(b) *Confidentiality*: Confidentiality should be provided as long as the server private key $\mathsf{inv}(Ks)$ remains secure.

(c) *Authentication:* It's reasonable for S to assume that the answer Ans_Q came from A, since only A should have been able to read and return the correct N_S. However, A cannot be sure that she is answering the right *question*. Nothing authenticates S to A, so the intruder can pose as S and make A think she's voted when in fact her answers never got to the real server.

(d) *Multiple Votes:* There is no obvious way for a malicious user to vote more than once on the same question without compromising the private keys of someone else.

(e) *Availability:* Availability generally can't be guaranteed in the Dolev-Yao setting. If the intruder is able to block all messages, then he can easily mount a DoS attack.

(f) *Integrity:* Integrity often follows from authentication, so the situation is similar: S is assured that Ans_Q has not been tampered with, but A cannot say anything about what she receives from S.

Assignment 1.2: Attack-Preserving Assumptions.

In protocol analysis, making assumptions or abstracting certain things away can be very helpful. Some assumptions, however, can exclude attacks at analysis time. We call these assumptions *non-attack-preserving*. Using non attack-preserving assumptions is a tradeoff.

(a) What are some arguments for and against the use of non-attack-preserving assumptions or abstractions?

(b) In the lecture (Module 2, Slide 10) we have made the assumption that when A and B receive messages, they "know" what protocol they belong to.

Do you think this assumption is reasonable? Do you think it is attack-preserving?

(c) What common mechanisms do we use in practice to try to realize this assumption?
Hint: On a Linux system, check out the file `/etc/services`.

Solution

(a) Non-attack-preserving assumptions can bring great efficiency gains at analysis time. Sometimes it can be a disadvantage to exclude attacks, because the model becomes less general. But restricting the model is not always a bad thing. Some assumptions that add a lot of generality, like for example assuming the intruder can always compromise private keys (maybe by bribing people), introduce attacks that are uninteresting because you can't really defend against an intruder that powerful anyway.

(b) Usually there is some information like port-numbers as part of the plain-text of transmitted messages that are not being authenticated. They play an important role in practice, namely telling the recipient to which protocol the message should be associated—otherwise the recipient would have to try for each protocol if a received message "fits" to that protocol. In our formal models, where we usually omit things like port-numbers, the agents can indeed associate an incoming message to any protocol that they are currently running. Although this in itself is not realistic, it perfectly models that the intruder can change this un-authenticated, non-confidential piece of information and make a receiver parse a message as being part of any protocol where it fits.

It is different, however, when the information is indeed authenticated in some way (e.g. part of a signed message), which is in general a good idea. Here the agent can check certain properties, e.g., the creator attached information of how to interpret a message that cannot be changed by the intruder. In this case, omitting the information may introduce attacks that are not possible in practice.

(c) `/etc/services` defines the port numbers of many common protocols, e.g., port 80 for `http`.

Assignment 1.3: Diffie-Hellman

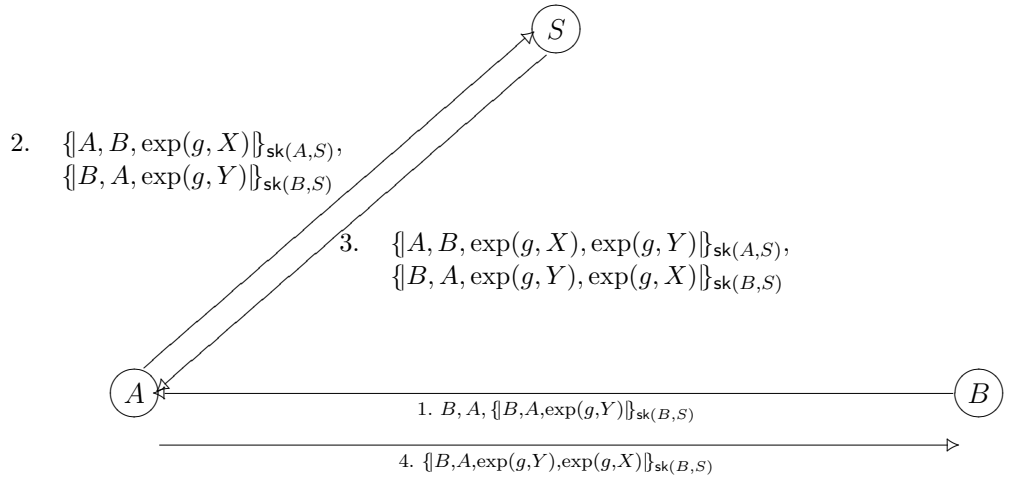
In module 2, we have built key-establishment protocols using an honest key-server S who has a shared key $\mathsf{sk}(A, S)$ with every agent A.

(a) Combine this schema with the Diffie-Hellman key-exchange, using the key-server to authenticate the exchange.
Hint: Use the structure of the protocol on slide 28.

(b) Argue why your Diffie-Hellman based protocol offers stronger security than the key-exchange protocols of module 2 in a certain situation.
Hint: Suppose the intruder is able at some point to compromise the honest key-server and find out all long-term keys $\mathsf{sk}(A, S)$.

Solution

(a) For example:



(b) Note that the server does not learn the negotiated key $\exp(\exp(g, X), Y)$ of any pair of agents A and B. Consider an intruder who has recorded all messages between several honest agents and the server. Suppose the intruder is able to break into the server at some point and see all long-term keys $\mathsf{sk}(A, S)$. What concerns all the old recorded traffic, he is now able to decrypt all messages encrypted with $\mathsf{sk}(A, S)$ and thus see what the server could see: the Diffie-Hellman half keys of honest users. The intruder cannot figure out the full keys $\exp(\exp(g, X), Y)$ from any old session, and thus all messages between honest agents based on these keys remain secret. This is called *perfect forward secrecy*. In contrast, in the classical protocols like NSCK, the intruder can immediately find out all old session keys once he has compromised the server.