Logics in Security **Winter 2014**

Homework for Module 2

Instructor: Deepak Garg TA: Iulia Boloşteanu
`dg@mpi-sws.org` `iulia_mb@mpi-sws.org`

Release date: 01.12.2014 *Due date: 08.12.2014*

General instructions: Attempt all questions. Submit your homework via email to both the instructor and the TA before midnight on the due date. This homework requires you to turn in a written component that must be typeset and two ProVerif source files. The LaTeX source for this homework will be provided to help you typeset. You can also typeset using any other means, including simple ASCII.

Problem 2-1 (4 points)

Meta-logical proofs. Let uppercase letters $A \ldots Z$ denote logical variables and let s and z denote the successor function on natural numbers and zero respectively. Recall the following definition of the predicate $\mathsf{add}(a, b, c)$, which intuitively means that $a + b = c$.

$$\frac{}{\mathsf{add}(\mathsf{z}, N, N)}\,\mathsf{addz} \qquad\qquad \frac{\mathsf{add}(N, M, P)}{\mathsf{add}(\mathsf{s}(N), M, \mathsf{s}(P))}\,\mathsf{adds}$$

Prove the following by induction on a suitable parameter (either one of the terms, or proofs): If $\mathsf{add}(n, \mathsf{s}(m), p)$ has a proof, then there exists a p' such that $\mathsf{add}(n, m, p')$ has a proof and $p = \mathsf{s}(p')$.

Problem 2-2 (6 points)

This problem has two questions. Solve *any one* of them. You may also solve both. In that case, we will score both and consider the maximum of the two scores.

Option 1: More proofs. Recall the following definition of the predicate $\mathsf{even}(a)$, which means that a is even.

$$\frac{}{\mathsf{even}(\mathsf{z})}\,\mathsf{evenz} \qquad\qquad \frac{\mathsf{even}(N)}{\mathsf{even}(\mathsf{s}(\mathsf{s}(N)))}\,\mathsf{evens}$$

Prove that if $\mathsf{add}(n, n, p)$ has a proof, then $\mathsf{even}(p)$ has a proof. [Hint: You will have to use the metatheorem of Problem 2-1 as a Lemma here. You can do that even if you were unable to solve 2-1.]

Option 2: Backwards search. Consider backwards (backchaining) search as discussed in class, without any explicit checking for loops (as in ProVerif). For each of the following goals, state whether or not backward search will terminate and, irrespective of whether or not it terminates, what satisfying substitutions for the variables it will find. Assume that the search always tries the rule `addz` before `adds`. Justify your answers.

a. $\mathsf{add}(n, M, p)$ for a variable M and constants n and p.

b. $\mathsf{add}(N, m, P)$ for variables N, P and a constant m.

c. $\mathsf{add}(N, M, p)$ for variables N, M and a constant p.

Problem 2-3 (5 points)

ProVerif. In the following protocol, Bob (B) wants to send a secret message m to Alice (A). For this, he initiates a session by sending to Alice his public key pk_B. Alice replies with her identity and a nonce[1] n_A encrypted with the key from Bob. Bob generates a nonce n_B and encrypts it together with the nonce from Alice under Alice's public key. Alice verifies that the first nonce in the ciphertext is her nonce and sends back to Bob his nonce encrypted under pk_B.

Bob verifies that the ciphertext from Alice contains the nonce he generated and creates the shared session key $n_A \odot n_B$. Here, $\odot$ is a one-way function. (Think of $n_A \odot n_B$ as a hash function applied to the pair (n_A, n_B). It is impossible to extract either n_A or n_B from $n_A \odot n_B$.) Next, Bob sends to Alice a secret message m symmetrically encrypted under the newly constructed key. Note that Alice can decrypt the ciphertext and recover the message, as she also knows both n_A and n_B and is able to construct the key $n_A \odot n_B$.

$$\begin{array}{ccc}
A & \xleftarrow{\qquad\qquad pk_B \qquad\qquad} & B \\
 & \xrightarrow{\qquad\quad \{A, n_A\}_{pk_B} \quad\qquad} & \\
 & \xleftarrow{\qquad\quad \{n_A, n_B\}_{pk_A} \quad\qquad} & \\
 & \xrightarrow{\qquad\qquad \{n_B\}_{pk_B} \qquad\qquad} & \\
 & \xleftarrow{\qquad\quad \{m\}_{n_A \odot n_B} \quad\qquad} &
\end{array}$$

Your goal is to model this protocol in ProVerif and discover/fix attacks on it. An attack here means that the attacker learns the secret message m. To help you start, we have provided a ProVerif file `secrecy.horn` that contains most of the protocol already. Beginning with this file, solve the following problems:

1. Complete the model of the protocol in `secrecy.horn`. Comments in the file will tell you exactly what you need to fill in and will also explain the rest of the encoding. Turn in the updated `secrecy.horn` file.
2. Run the protocol through ProVerif. If your model is correct, ProVerif will find an attack. Describe this attack in simple words in your solution.
3. Propose (on paper) a fix to the protocol.
4. Encode the fix in ProVerif in a new file `secrecy-fixed.horn` and turn in that file. (Run this file through ProVerif to ensure that there is really no further attack.)

---

[1] Here, a nonce is a random number that the adversary cannot guess.

Instructions on ProVerif: Please use the following link to download and install ProVerif on your machine:

`http://prosecco.gforge.inria.fr/personal/bblanche/proverif/`

Depending on your operating system, you will have to download either a binary package (Windows) or the source code package (Linux, Mac OS). Unzip the package of your choice, go to the unzipped directory and execute the `build` script. Please check the README file for more detailed information on this.
In the downloaded and compiled directory you will find examples (the ones with extension `.horn` should be more relevant for you) and a user manual (`/docs/manual-untyped.pdf`). You can work in this directory or you can add the path where `proverif` lies to your `PATH` environment variable. For verifying the protocols use the following command:

```
./proverif -in horn <file_name.horn>
```

The homework release includes the file `denning-sacco.horn` as an example. This file contains the full Denning-Sacco protocol described in the ProVerif paper. If you encounter any difficulties in installing or using ProVerif, please send an email at `iulia_mb@mpi-sws.org`.