International Institute of Information Technology Hyderabad

System and Network Security (CS5470)

Lab Assignment 1: Formal Security Verification of Security Protocols using the AVISPA backends: OFMC and CL-AtSe

Deadline: February 10, 2017 (23:59 P.M.)

Total Marks: 100 [Implementation (Coding + correct results): 75, Vice-voce: 25]

Note:- *It is strongly recommended that no student is allowed to copy programs from others. Hence, if there is any duplicate in the assignment, simply both the parties will be given zero marks without any compromise. Rest of assignments will not be evaluated further and assignment marks will not be considered towards final grading in the course. No assignment will be taken after deadline. Please upload in the HLPSL code along with a README file in the course moodle portal through a ZIP file (Lab1-RollNumber.zip).*

Consider the following protocol. Implement this protocol using HLPSL language of AVISPA specifying clearly the basic roles for the parties involved in the network, and mandatory roles for session, goal and environment. Your code must be well-commented and easy-to understand. You must also specify the secrecy and authentication goals in the implementation. Simulation the protocol using the OFMC and CL-AtSe backends.

Description of the Protocol

This protocol is about the remote user authenticated key agreement scheme using smart cards. In the protocol, a legal user U_i wants to access services from the server S provided that mutual authentication happens between them. Once the mutual authentication is done between U_i and S, both of them establish a session key for their future secure communications.

The server S chooses a long secret key $k_s \in Z_p^*$ randomly and a secure cryptographic hash function $H(\cdot)$, where $Z_p^* = \{1, 2, \cdots, p-1\}$. In addition, S maintains its public key $P_s = g^{k_s} \pmod p$, where $g \in Z_p^*$ is a generator of the cyclic group Z_p^*.

The protocol has the following phases:

Registration phase

In this phase, a user U_i needs to register with the server S using his/her chosen unique identity ID_i and obtain the smart card SC_i. The detailed description of this phase is given below.

R1. $U_i \to S : \langle Reg = \{ID_i\} \rangle$
U_i first chooses his/her unique identity ID_i and sends it to S securely.

R2. $S \to U_i : \langle SC_i \rangle$

Upon receiving the request Reg, S verifies ID_i. If the hash value $H(ID_i||k_s)$ is found in its database, S asks U_i to send another request with a new identity. Otherwise, S selects a new smartcard SC_i and retrieves SC_i's identity SID_i. S computes $C_i = H(ID_i||\ k_s||\ SID_i)$ and stores $\{C_i,\ P_s,\ g,\ p,\ H(\cdot)\}$ into the smartcard SC_i. Finally, S sends the smartcard SC_i to U_i securely, and stores $[H(ID_i||k_s),\ SID_i]$ in the database corresponding to U_i's entry.

R3. After receiving SC_i from S, U_i inputs his/her chosen password pw_i into the smartcard SC_i. Then, SC_i computes $B_i = C_i \oplus\ H(pw_i||\ ID_i) = H(ID_i||\ k_s||\ SID_i) \oplus\ H(pw_i||\ ID_i)$ and $A_i = H(C_i||\ pw_i||\ ID_i)$. Finally, SC_i replaces C_i by B_i and stores A_i in its memory. Hence, SC_i finally contains the information $\{A_i,\ B_i,\ P_s,\ g,\ p,\ H(\cdot)\}$.

The summary of registration phase is given in Table 1.

Table 1: The registration phase

User (U_i)/Smartcard (SC_i)	Server S
Chooses ID_i. $\xrightarrow{Reg = \{ID_i\}}$ (via a secure channel)	
	Checks validity of Reg. Chooses new SC_i and retrieves its SID_i. Computes $C_i = H(ID_i\|\|k_s\|\|SID_i)$, Stores $\{C_i, P_s, g, p, H(\cdot)\}$ into SC_i. Stores $\{H(ID_i\|\|k_S), SID_i\}$ into its database. $\xleftarrow{\langle SC_i \rangle}$ (via a secure channel)
Inputs pw_i into SC_i. SC_i computes $B_i = C_i \oplus H(pw_i\|\|ID_i)$, $A_i = H(C_i\|\|pw_i\|\|ID_i)$. SC_i replaces C_i by B_i and stores A_i.	

Login phase

In this phase, if a legal user U_i wants to login to the server S using his/her credentials, the following steps are executed:

L1. $U_i \rightarrow S : \langle Req = \{TID_i, V_i\}\rangle$

L1.1. U_i inserts his/her smartcard SC_i into a card reader and supplies the identity-password pair (ID'_i, pw'_i) into SC_i.

L1.2. SC_i computes $C'_i = B_i \oplus\ H(pw'_i\ ||ID'_i)$ and $A'_i = H(C'_i||\ pw'_i||\ ID'_i)$, and checks if $A'_i = A_i$ holds. If it holds, SC_i executes the next step. Otherwise, SC_i rejects the entered credentials.

L1.3. SC_i randomly chooses $\alpha \in Z_p^*$ and a random nonce n_1, and then computes encryption key $K_1 = P_s^{H(\alpha||C_i)} \pmod p$, $TID_i = (ID_i||n_1) \oplus H(K_1)$ and $V_i = g^{H(\alpha||C_i)} \pmod p$.

L1.4. U_i sends the login request $Req = \{TID_i,\ V_i\}$ to S via a public channel.

Authentication with key agreement phase

In this phase, the server S verifies the validity of the login request Req sent by the user U_i. If it is valid, S sends the response $Resp$ to U_i. U_i then authenticates S by verifying the validity of $Resp$ and replies the

Table 2: The login and authentication with key agreement phases

Login phase	
(a) User credentials verification	
User U_i	Smart card SC_i
Input ID'_i, pw'_i.	Computes $C'_i = B_i \oplus H(pw'_i \| \| ID'_i)$, $A'_i = H(C'_i \| \| pw'_i \| \| ID'_i)$. Checks if $A'_i = A_i$? accept/reject?
(b) Login request	
Smart card SC_i	Server S
Chooses $\alpha \in Z_p^*$ and random nonce n_1. Computes $K_1 = P_s^{H(\alpha \| \| C_i)} \pmod p$, $TID_i = (ID_i \| \| n_1) \oplus H(K_1)$, $V_i = g^{H(\alpha \| \| C_i)} \pmod p$. $\xrightarrow{Req = \{TID_i, V_i\}}$	
Mutual authentication and key agreement phase	
Smart card SC_i	Server S
	Computes $K_2 = V_i^{k_s} \pmod p$, $(ID_i \| \| n_1) = TID_i \oplus H(K_2)$. Checks the validity of ID_i. accept/reject? Chooses $\beta \in Z_p^*$. Computes $C_i = H(ID_i \| \| k_s \| \| SID_i)$, $sk_s = V_i^{H(k_s \| \| \beta)} \pmod p$, $V_s = g^{H(k_s \| \| \beta)} \pmod p$, $M_s = H(V_i \| \| C_i \| \| V_s \| \| sk_s \| \| n_1)$. $\xleftarrow{Resp = \{V_s, M_s\}}$
Computes $sk_i = V_s^{H(\alpha \| \| C_i)} \pmod p$. Checks if $M_s = H(V_i \| \| C_i \| \| V_s \| \| sk_i \| \| n_1)$? accept/reject? Computes $M_i = H(sk_i \| \| V_s \| \| C_i \| \| n_1)$. $\xrightarrow{Conf = \{M_i\}}$	
	Checks if $M_i = H(sk_s \| \| V_s \| \| C_i \| \| n_1)$? accept/reject?
Agree on the session key $sk_i = V_s^{H(\alpha \| \| C_i)} \pmod p = D_i^{H(k_s \| \| \beta)} \pmod p = sk_s$.	

confirmation message $Conf$ to S in order to prove himself/herself at S. Finally, after successful mutual authentication, a session key is established between U_i and S. The detailed process is discussed below.

A1. $S \rightarrow U_i : \langle Resp = \{V_s, M_s\} \rangle$

A1.1. After receiving Req from U_i, S computes the decryption key $K_2 = V_i^{k_s} \pmod p$ and decrypts TID_i using K_2 as $(ID_i||n_1) = TID_i \oplus H(K_2)$. S checks the validity of the derived ID_i by checking the value $H(ID_i||k_s)$ in its database. If it is valid, S executes the next step. Otherwise, S rejects U_i's login request.

A1.2. S randomly chooses $\beta \in Z_p^*$ and computes $C_i = H(ID_i||\, k_s||\, SID_i)$ using its secret key k_s and SID_i corresponding to ID_i available in its database, $sk_s = V_i^{H(k_s||\beta)} \pmod p$, $V_s = g^{H(k_s||\beta)} \pmod p$ and $M_s = H(V_i||\, C_i||\, V_s||\, sk_s\, ||n_1)$.

A1.4. S finally sends the response message $Resp = \{V_s, M_s\}$ to U_i via a public channel.

A2. $U_i \rightarrow S : \langle Conf = \{M_i\}\rangle$

A2.1. After receiving $Resp$ from S, U_i computes $sk_i = V_s^{H(\alpha||C_i)} \pmod p$ and checks whether the condition $M_s = H(V_i||\, C_i||\, V_s||\, sk_i||\, n_1)$ holds or not. If it holds, U_i authenticates S and executes the next step. Otherwise, U_i rejects $Resp$ and terminates the session.

A2.2. U_i computes $M_i = H(sk_i||\, V_s||\, C_i||\, n_1)$ and sends the confirmation message $Conf = \{M_i\}$ to S via a public channel.

A3. Upon receiving $Conf$ from U_i, S checks the validity of $Conf$ by comparing the values M_i and $H(sk_s||\, V_s||\, C_i||\, n_1)$. If both are same, S authenticates U_i and confirms that U_i agrees on the session key $sk_s = sk_i$. Otherwise, S rejects $Conf$ and terminates the session immediately.

The login and authentication with key agreement phases are summarized in Table 2.

Remark: In HLPSL, bitwise XOR operation $A \oplus B$ and exponentiation X^N are represented as xor(A, B) and exp(X, N), respectively. All other descriptions regarding HLPSL implementation are available at http://www.avispa-project.org/.

All the best!