CS 4740: Problem Set 4

Secure Instant Messaging System (SIMS)

Eric Chin (chiner) John McGuiness (inkd)

Architecture

Our secure instant messaging system (SIMS) will have the following system architecture:

1. Any number of clients who will be using the SIMS
2. A server that will act as the key distribution center for clients and help establish sessions between the clients

Note that the server will only be used for authentication, key distribution, and client discovery.

Assumptions

- Client knows the public key of the server
- Client and server will share knowledge of password
- Users will only remember one password
- Symmetric keys are at least 128-bit keys
- Asymmetric keys are at least 1024-bit keys
- Symmetric encryption operations use the AES encryption protocol
- Asymmetric encryption operation use the RSA encryption protocol
- Message authentication codes are generated using HMAC
- Each hashing function is a cryptographic hash function in the SHA-2 series

Services

- Mutual authentication between client and server and client and client
- Perfect forward secrecy
- Message integrity
- End-to-end security
- Non-repudiation
- Password protection

The Protocol

The security of this protocol is founded on the security of its key establishment, which is separated into three general categories: client-server authentication and key exchange, server-client key distribution, and client-client session establishment.

K_{AS} is the symmetric key between A and S
P_A is the password shared between A and S
Pu_A is the public key of A which is randomly generated on client login
Pr_A is the private key of A which is randomly generated on client login
h_1, h_2 are two different cryptographically secure hashing functions
s_1, s_2 are two different 8-bit salt
R_S is a random number

Client-Server Authentication / Key Exchange

$A \rightarrow S$: "A", $Pu_S\{K_{AS}\}$, $K_{AS}\{Pu_A, h_i(P_A|s_1), s_1\}$
$A \leftarrow S$: $Pu_A\{R_S\}$, $K_{AS}\{h_2(P_A|s_2), s_2\}$
$A \rightarrow S$: $h_1(R_S)$

Server-Client Key Disribution

$S \rightarrow A$: $K_{AS}\{Pu_B, \text{"B"}\}$, $K_{AS}[Pu_B, \text{"B"}]$
$S \rightarrow B$: $K_{BS}\{Pu_A, \text{"A"}\}$, $K_{BS}[Pu_A, \text{"A"}]$

Client-Client Session Establishment

$A \rightarrow B$: $Pu_B\{K_{AB}\}$, $K_{AB}\{R_A\}$
$A \leftarrow B$: $Pu_A\{R_B\}$, $K_{AB}\{h_1(R_A)\}$
$A \rightarrow B$: $h_2(R_A, R_B)$

These will provide mutual authentication between the *A* and the server and between *A* and *B*.

Message Formats and Protocols

Messages hereafter are of the following form:

Client-Server Requests

$A \rightarrow S$: $K_{AS}\{\text{Client Request}\}$, $K_{AS}[\text{Client Request}]$
$A \leftarrow S$: $K_{AS}\{\text{Server Response}\}$, $K_{AS}[\text{Server Response}]$

Client to server requests include: "list", "logout."

Client-Client Messages

$A \rightarrow B$: $K_{AB}\{message_A\}$, $K_{AB}[message_A]$
$A \leftarrow B$: $K_{AB}\{message_B\}$, $K_{AB}[message_B]$

Discussion of Issues

a. Does your system protect against the use of weak passwords? Discuss both offline and online dictionary attacks.
   Yes. Because we salt and hash each password, adversaries will not be able to perform dictionary attacks because of the added entropy of the random salt.

b. Is your design resistant to denial of service attacks?
   Our design does protect against DoS attacks. In particular, because each client maintains a list of the currently logged on users, a client's maintained communication with other clients is not centralized. Therefore, if the server is DOS'd, the server will be unable to authenticate new clients or distribute client lists, but clients will still be able to continue their current conversations.

c. To what level does your system provide end-point hiding, or perfect forward secrecy?

Our protocol provides perfect forward secrecy. It also provides end-point hiding in that end-points are never explicitly described in the message protocol. This protocol maintains a high level of end-to-end security as well, as the protocol provides for key exchange and establishment between clients, which provides a secure and encrypted channel for client-to-client messages. Additionally, our architecture requires all clients to authenticate first with the server and key distribution center, so a third-party adversary will be unable to obtain a user list.

d. If the users do not trust the server can you devise a scheme that prevents the server from decrypting the communication between the users without requiring the users to remember more than a password? Discuss the cases when the users trusts (vs. does not trust) the application running on his workstation.
In our design, the server is unable to decrypt communication between users because two clients mutually negotiate a session key without server input.

   If the user cannot trust any component in the SIMS architecture, then communication to or from that component cannot be trusted. This is the nature of the chat protocol. In particular, if a user cannot trust the server, then the server can provide the user with illegitimate information of other clients (with whom the user will be able to communicate freely). If the user cannot trust his application, then his application will have access to his communication and session. And if the user cannot trust another client's application, then his communication with that client is compromised. Our architecture assumes that all components are trusted.

   If the application on the client's workstation is untrusted, it would not have a private key matching the public key hosted by the server, and therefore would be unable to decrypt the proposed session key.