

Husky Instant Messenger



By

Jain Richin

Shivdasani Chander

Table of Contents

1:	Introduction	3
2:	Design Goals	3
3:	Architecture	4
4:	Password Storage	5
5:	Protocol	
5.1	User –Server Authentication	8
5.2	Request for Online Users	8
5.3	Chat Request	9
5.4	User –User Authentication	9
5.5	Message Transfer between users	10
5.6	Logout	10
6:	Answers to Problem Set Questions	11

1: Introduction

This document proposes a design for a Secure Instant messenger. It is intended to be a demonstration of how one may achieve Secure Communication in an Unsecured Network. We begin by specifying the Design Goals, later we move to the architecture and how passwords are stored at the server, finally we conclude by specifying the Protocols that should be used to achieve such a communication.

2: Design Goals

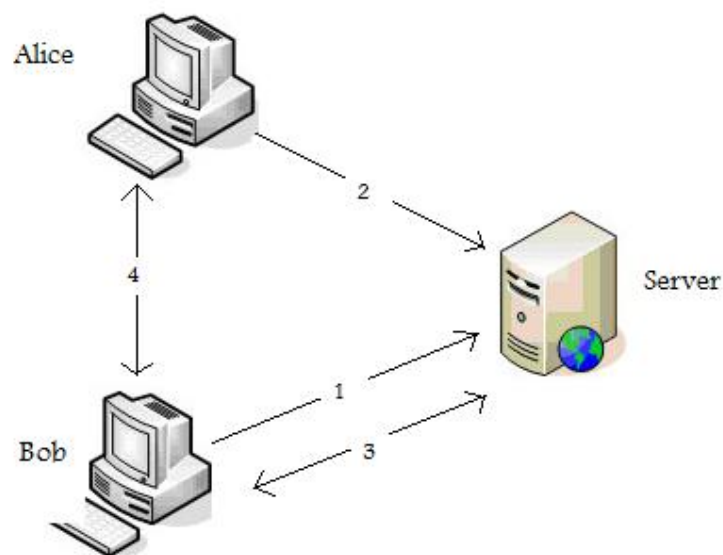
The following security issues have been kept in mind while designing the messenger

- Online/Offline dictionary attacks
- Denial of Service attacks
- Perfect Forward Secrecy
- Endpoint Hiding
- Trust issues with the server
- Replay Attack
- Reflection Attack
- Man in the middle attack
- Mutual Authentication between client and server
- Mutual Authentication between clients
- Session Key establishment for communication between Client-Client and Server – Client.

3: Architecture

The model of our system can be well understood by referring the below diagram. Server will authenticate all the users and will allow only those users to use its resource. Once a user is authenticated to the server, it will share a unique secret key which will ensure secure communication with the server.

To establish a connection with another user, both the users need to be authenticated to the server first. The user, who initiates the conversation with another user, gets a TICKET from the server. He then uses this ticket to authenticate himself to another user. Both the users establish a session key to secure their conversation



1, 2 - Mutual Authentication with Server

3 - Receives Ticket to communicate with another user

4 - Mutual authentication between clients

4: Password Storage

Server uses flat file (passwd.txt) to store username, salt and hash of the passwords. The hashing algorithm used is SHA-1. This file is encrypted using Server's public key, which provides protection from an unauthorized access. An entry in passwd.txt file will look like.

Userid: Username: SHA1 (pw + Userid)

Where, Userid is a 32-bit salt.

The username is a variable length up to 32 characters. The allowable characters for username will come from the set: A-Z, a-z, 0-9.

The password is at least 8 characters long and can go up to maximum of 100 characters.

Online and Offline attacks on Password

The system is resilient to online attack using two mechanisms:

- The minimum password length is 8 characters and each character can have 94 different values. So, it's a total of 94^8 values, which is extremely unlikely to guess.
- The user is given 3 trials to enter correct password. After which server will prevent log in for 5 minutes. This particular mechanism was added to slow down password guessing.

At the server side, the hash of the password is stored along with the salt. Adding of the salt makes rainbow table attacks more difficult. This security mechanism is closely related to the Shadow Password mechanism deployed by Linux systems.

< Aside>:

For added security we can implement two more things:

- If a user doesn't enter correct password in 3 trials, he will have to solve a CAPTCHA before trying for the fourth time.
- If the user doesn't enter a correct password in 5 trials, his account will be blocked and an email would be sent to him for verification and reactivation of account.
- In addition, server can also run a dictionary attack to check the weakness of a password and if found weak, notify the user to change it.

However, we are not going to implement this in our system.

<End of Aside>

5: Protocol

This section describes the protocols that will be used for communication between client-server and client-client. Before we begin, let's look at some of the abbreviations that will be used throughout the document. All other notations have their usual meaning, unless otherwise stated.

Abbreviations:

C: Challenge

T: Time stamp

H (pwd): Hashed Password using SHA-1

PK: Public Key

U: Username

K: Session Key

IPa: IP Address of A

Assumptions:

- Users are already registered with the Server
- Users remember just the username and password
- For each session, clients generate a unique Public/Private key pair. They delete these keys once the session terminates.
- Clients and Servers clock are loosely synchronized with a time skew of 90 seconds.

5.1: User-Server Mutual Authentication

FLOW	DETAILS	ENCRYPTION
User – Server	LOGIN	Plaintext
Server – User	C,PKs	Plaintext
User – Server	$C, \{T_a, U_a, PK_a, h(pwd)\}PK_s$	RSA
Server – User	$\{U_a, T_a, T_b, K_a\}PK_a$	RSA
User – Server	$K_a\{T_b\}$	AES

- 1: User sends a LOGIN request to the server.
- 2: Server sends a Challenge C. This challenge is used to prevent DOS attack. Along with the challenge the server also sends his Public key to the user.
- 3: User sends his login credentials along with the Challenge. Login credentials also contain a Timestamp. This timestamp is used to avoid Replay attack and also serves as a challenge for the server.
- 4: Server verifies the timestamp, and creates a new Session key and sends it to the user along with a challenge.
- 5: Client responds to the challenge.

Once the user is authenticated, the server stores his public key.

5.2: Request for the List of Online Users

FLOW	DETAILS	ENCRYPTION
Server-User	$K_a\{T_a, \langle \text{usernames} \rangle\}$	AES

- 1: Once a user is authenticated, the server sends him the list of online users.
- 2: Also, Server notifies all the online users about this new user.

5.3: Chat request

FLOW	DETAILS	ENCRYPTION
User1- Server	{CHAT, Ua, Ka{Ta,Ua,Ub}}PKs	AES
Server-User	Ka{Ta,Ts,Ua,Ub,IPb,Pkb}, Ticket-to-B	AES
	Ticket-to-B = Kb{Ts, Ua, Ub, IPa, PKa}	

- 1: User sends a chat request to the server encrypting it with server's public key.
- 2: Server verifies the user and checks whether Ua and Ub is online. If everything is good, server creates a Ticket-to-B encrypted with User2's private key and sends it to User1. Note that the ticket has a timestamp Ts, User1 stores this and expects B to send it back when User1 and User2 are mutually authenticating them.

5.4: User-User Mutual Authentication

FLOW	DETAILS	ENCRYPTION
User1- User2	Hi, Ticket-to-B	Plaintext
User2-User1	{Ua, Ub, Ts, Tx, Kab}PKa	RSA
User1 – User2	Kab{Tx}	AES

- 1: User1 sends Hi along with Ticket-to-B to User2.
- 2: User2 decrypts the ticket and verifies the timestamp Ts. This timestamp is sent back to User1. This is how User1 authenticates User2. Along with this, User2 also sends a new timestamp which is used by User2 to authenticate User1. Note, User2 selects and new Session key Kab and is sent along with the message.

5.5: Message Transfer between Users

FLOW	DETAILS	ENCRYPTION
User1- User2	{Ua,Kab{Ta,Msg, Hash(Msg)}}PKb	RSA/AES/SHA-1
User2-User1	Kab{Ta}	AES

1: User1 sends a message, its hash and the timestamp to User2 by encrypting it with the session key and encrypting the entire message with user2's public key. Hash of the message is used to verify the integrity of the message. The hash used is HMAC-SHA1.

2: User2, after verifying the message, responds to User1.

5.6: Logout

FLOW	DETAILS	ENCRYPTION
User1- Server	{LOGOUT,Ua,Ka{Ua,Ta}}PKs	AES
Server-User1	Ka{Ta}	AES

1: User sends a LOGOUT message to the server along with the timestamp.

2: Server verifies the user and removes him from the list of online users and forgets the session key. User, on the other end also forgets the session key.

6: Answers to Problem Set Questions

Question 1:

Propose an architecture and design for a secure instant messaging system. The architecture can have a server (but not necessarily, it is up to you to make the decision). If a server is used the messages between the users should not go through the server (with the exception of the first discovery messages).

Solution:

Refer to sections 3, 4 and 5.

Question 2:

Assume that the users only need to remember a single password. Your system should provide mutual authentication (user and server), message integrity and confidentiality.

Solution:

In our case, user just needs to know a single password. Section 5.1 describes how mutual authentication is achieved between user and server.

For message integrity, we send the hash of the message along with the message. This hash is computed using HMAC-SHA-1 hashing function. Section 5.5 describes this in detail.

Confidentiality is achieved by encrypting the message with a session key.

Question 3:

Describe in detail the security protocols that you are proposing for the whole system. Remember that you are not allowed to use SSL or a complete existing protocol. You are allowed to use cryptographic libraries that provide encryption, hashing, etc.

Solution:

Refer Section 5 for a complete description on protocols.

Question 4:**A:**

Does your system protect against the use of weak passwords? Discuss both online and offline dictionary attacks

Solution:

Refer Section 4

B:

Is your design resistant to denial of service attacks?

Solution:

We use cookie mechanism to avoid denial of service attacks. Section 5.1 describes this in detail

C:

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

Solution:

All the messages transferred between client –server or client –client are encrypted either with the session key or public key of the receiver. At no point in the communication we expose the identity of any communicating party.

Perfect forward Secrecy is achieved by generating new Session keys for every communication and forgetting all the previous ones.

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 user trusts (vs. does not trust) the application running on his workstation.

Solution:

In our architecture, when a user A wants to communicate with another user B, server sends both the users a token which is used by them to authenticate each other. After

authenticating both the users establish a new session key, of which server has no knowledge. So, server can never decrypt any communication between clients and hence the trust issue is obviated.