Semester Project Report

AN OPEN SOURCE TOOLKIT PROVIDED BY:

Fast prototyping of secure applications:

Modeling and creating secure chat applications

Vikram KATURI

INTRODUCTION

A quick look into

- ➔ TTool
- ➔ Secure Symmetric Encryption
- ➔ Chat App

TTool

A free and open-source environment for modeling embedded systems - hardware or software elements -, and performing really easily simulations and mathematical proofs directly from UML/SysML models.

1. **DIPLODOCUS**: Partitioning of embedded systems
2. **AVATAR**: Modeling/design and verification of embedded software
3. **SysML-Sec**: Design safe and secure embedded system
4. **TTool-AMS:** Design of Mix analog/digital systems

TTool

We will concentrate on :

C code generator of the AVATAR feature.

Secure Symmetric Encryption

Secure Key Exchange:

Diffie-Hellman

Symmetric Encryption/Decryption:

AES

Easy to implement, less memory needs.

Previous work done in this area is reused.

Chat App

- ➔ What?
  - ◆ To build a secure chat application.
- ➔ Why?
  - ◆ Previous work did not achieve a full fledged chat application.
- ➔ How?
  - ◆ Modularize heavily and enable plug and play.

Building Chat Application

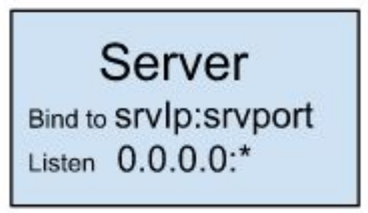
Architecture and design:

- Central Server
- Acts as relay between clients
- Clients only talk to server

Building Chat Application

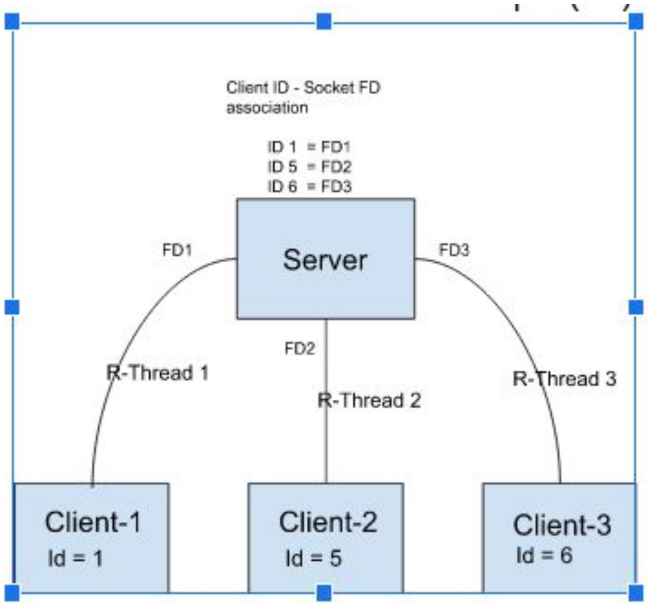
Architecture and design:

Central server, listening for client connections



Building Chat Application

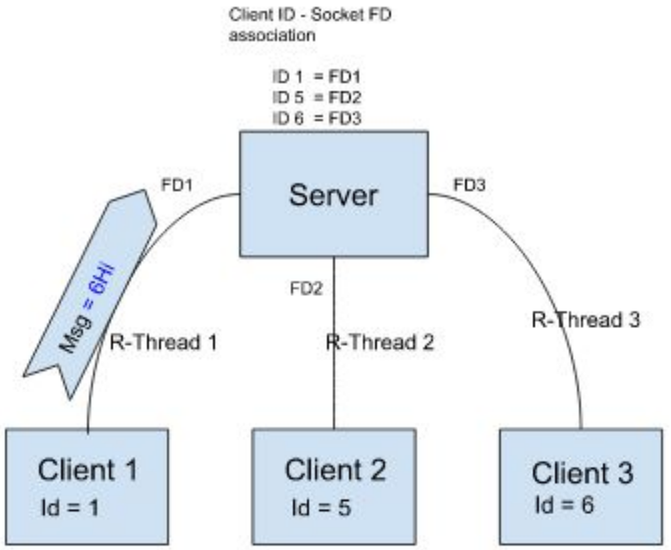
Architecture and design:

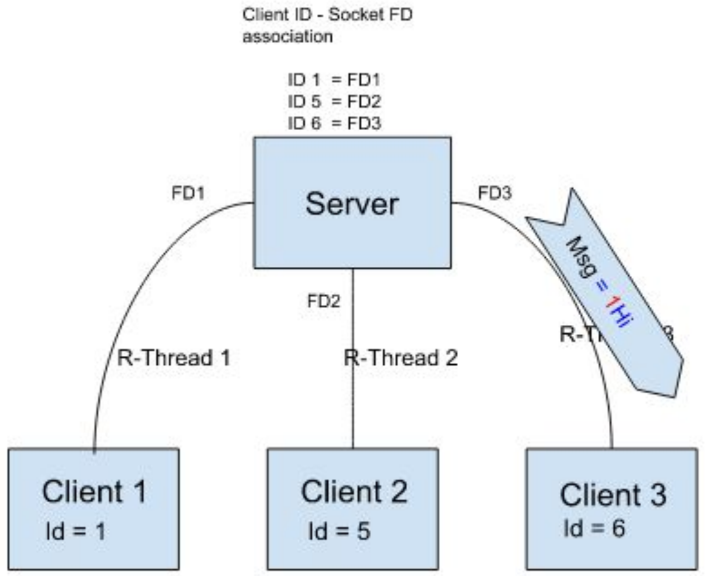


- Dedicates a reader thread per client.
- Maintains clientID to connection FD association.

Building Chat Application

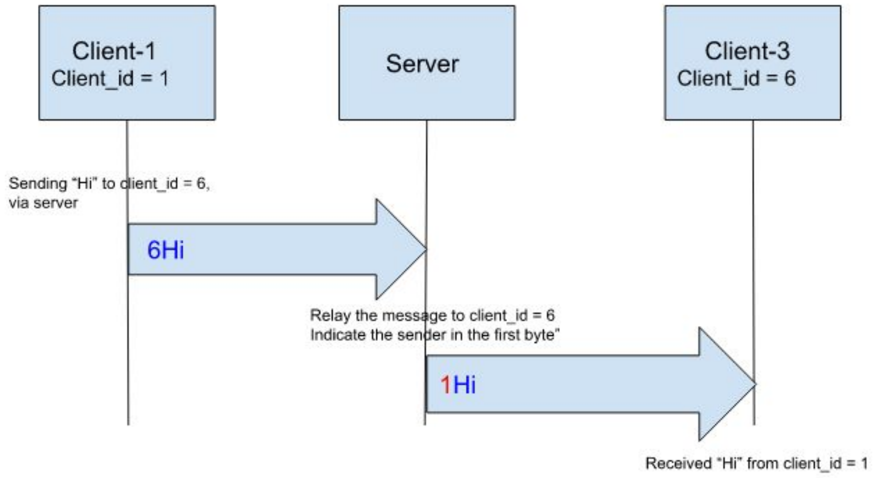
Architecture and design: Server assisted client to client communication





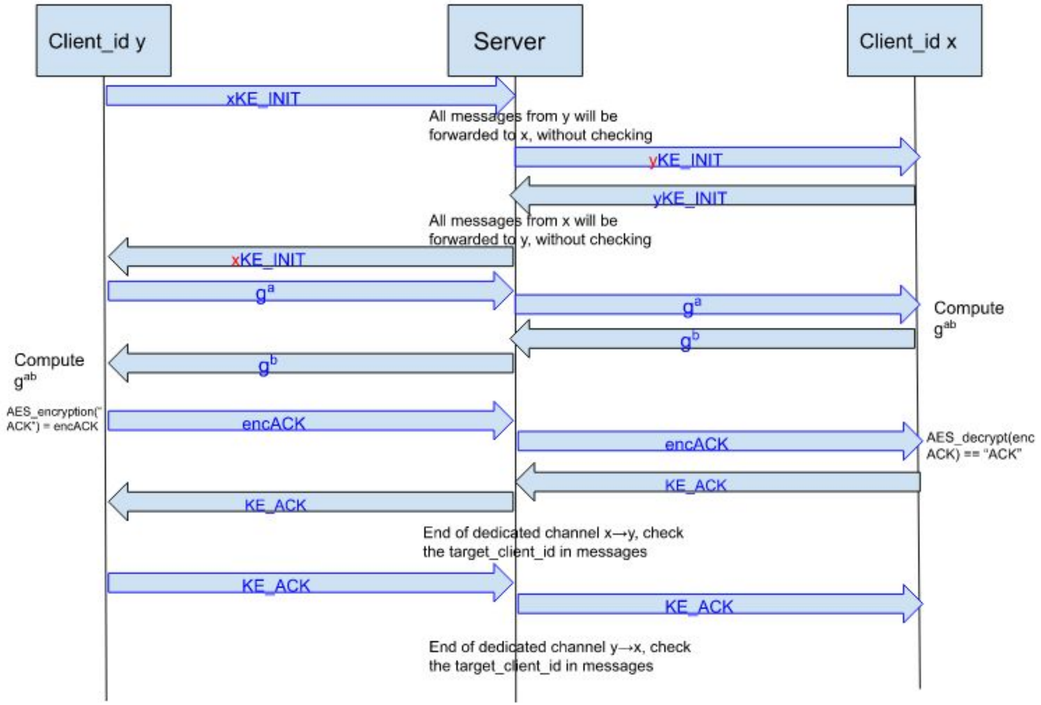
Building Chat Application

Architecture and design: Server assisted client to client communication


Client-1
Client_id = 1
Server
Client-3
Client_id = 6
Sending "Hi" to client_id = 6,
via server
6Hi
Relay the message to client_id = 6
Indicate the sender in the first byte"
1Hi
Received "Hi" from client_id = 1

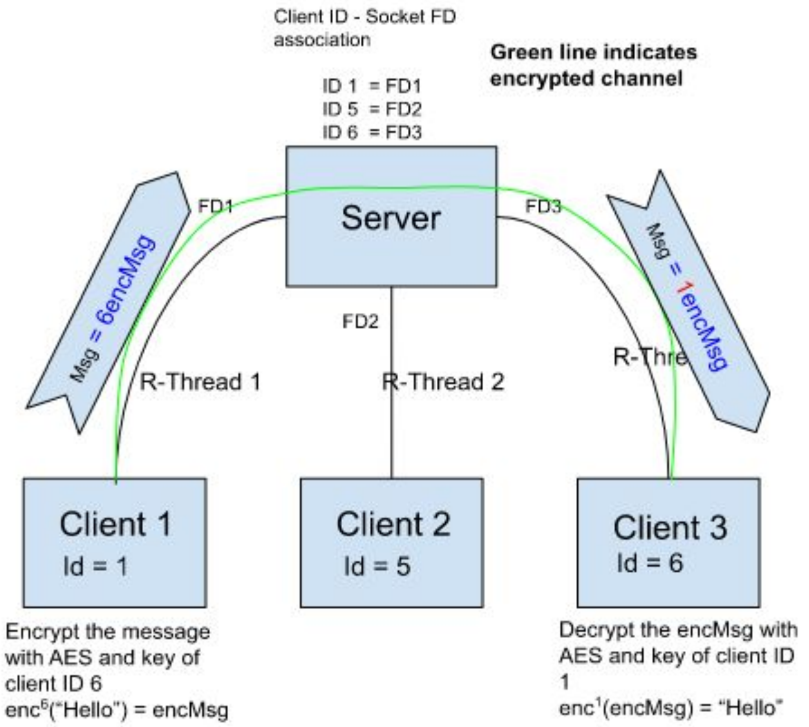
Building Chat Application

Architecture and design: Server assisted client to client Key-exchange


Client_id y
Server
Client_id x
xKE_INIT
All messages from y will be forwarded to x, without checking
yKE_INIT
yKE_INIT
All messages from x will be forwarded to y, without checking
xKE_INIT
g^a
g^a
Compute g^{ab}
g^b
Compute g^{ab}
g^b
AES_encryption("ACK") = encACK
encACK
encACK
AES_decrypt(encACK) == "ACK"
KE_ACK
KE_ACK
End of dedicated channel x→y, check the target_client_id in messages
KE_ACK
KE_ACK
End of dedicated channel y→x, check the target_client_id in messages

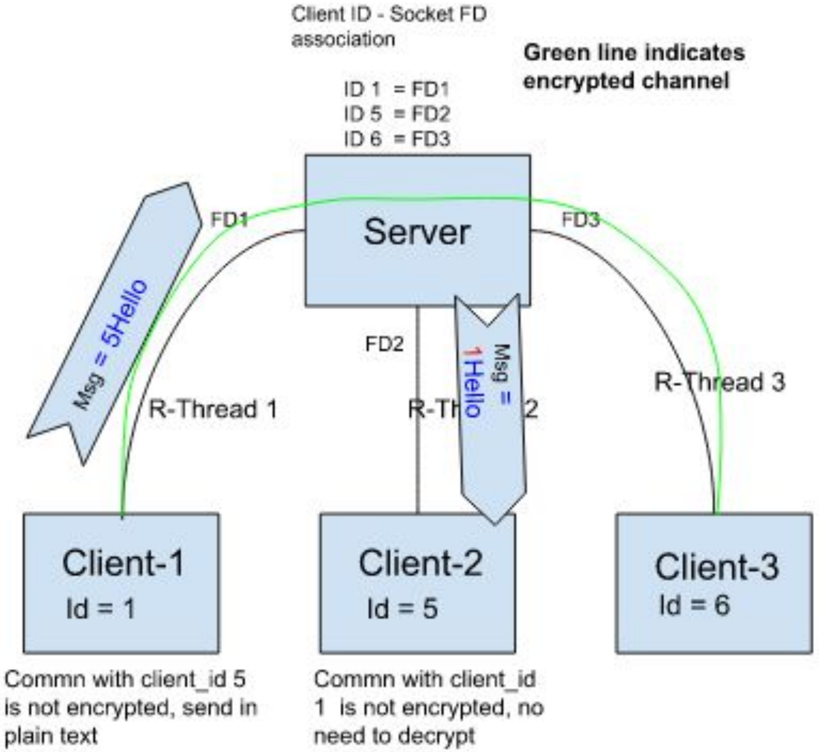
Building Chat Application

Architecture and design: client to client encrypted communication



Building Chat Application

Architecture and design: client to client **unencrypted** communication


Client ID - Socket FD association
ID 1 = FD1
ID 5 = FD2
ID 6 = FD3
Green line indicates encrypted channel
Server
FD1
FD2
FD3
Msg = 5Hello
Msg = 1Hello
R-Thread 1
R-Thread 2
R-Thread 3
Client-1
Id = 1
Client-2
Id = 5
Client-3
Id = 6
Commn with client_id 5 is not encrypted, send in plain text
Commn with client_id 1 is not encrypted, no need to decrypt

Building Chat Application

Implementation: server

Building Chat Application

Implementation: Client

Building Chat Application

Integration to TTool

Files and Locations to patch

seccom.c seccom.h	->	TTool/executablecode/src/	
Makefile Makefile.defs	->	TTool/executablecode/lib/generated_src	
TaskFile.java	->	TTool/src/main/java/avatartranslator/toexecutable	-> Needs a "make ttoolnotest"

Load ChapApps.xml for client and server code:

Generate server app
Generate client app1
Generate client app2

Usage Tutorial

<DEMO>

Results

Successfully generated a flexible chat application.

Further enhancements?

- Try to bring the modularization to the server side and decouple the server functions to have more flexibility on the server side like client side.
- Make the input/output of the clients interface with a GUI for more human friendly usage.
- Could try to implement the same for other security protocols by further decoupling the security protocol from the client application.