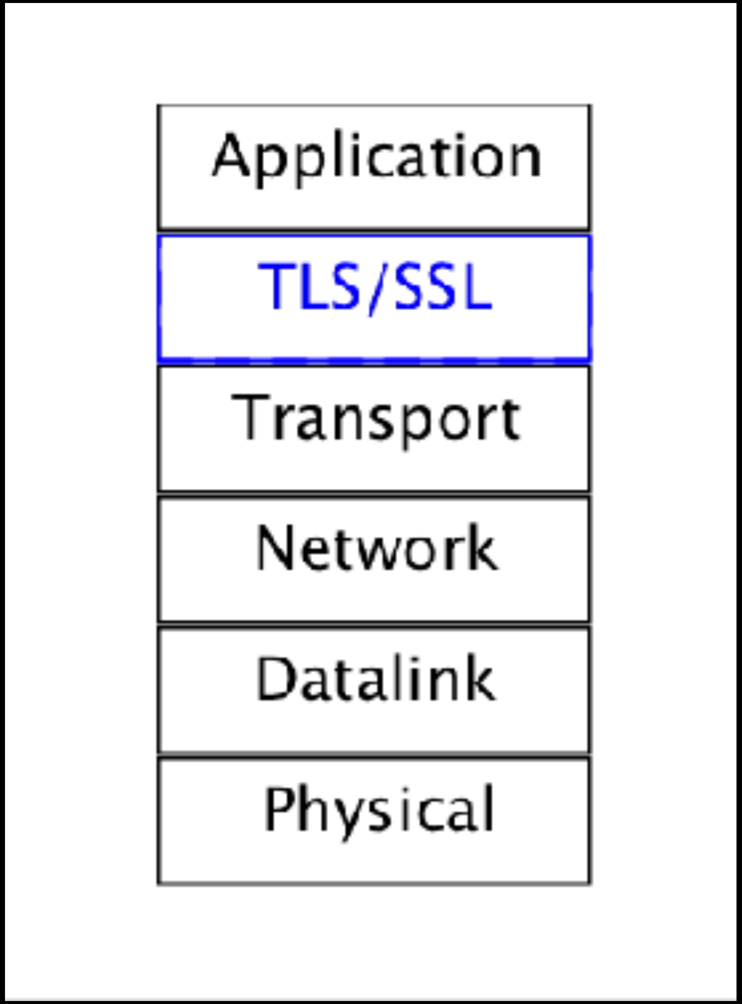
SSL Handshake Project

Prepared by Meng Yang

Description

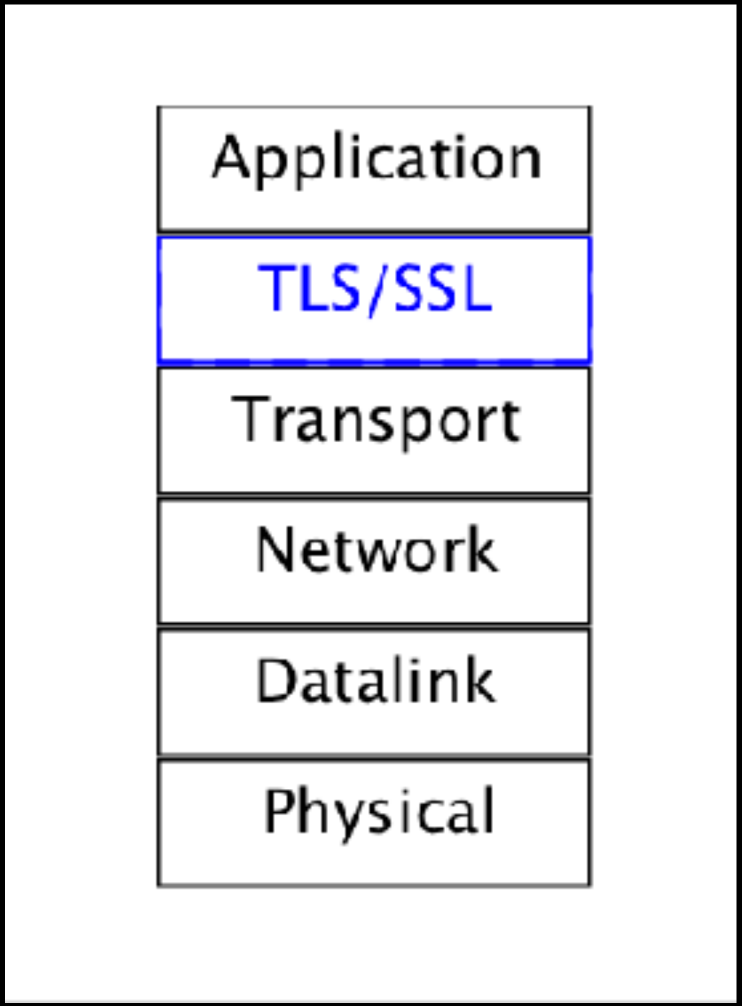
Network Layers (1/2)

- Communication done using this simplified layered structure
- Application Layer: where server and client are exchanging data
- Transport Layer: make sure messages is sent/received. But message content from application is visible.



Network Layers (2/2)

- TLS/SSL: sits in between Application and Transport, adds CIA to the messages from application



What is Provided (1/6)

- /src
  - /tcp : TCP implementation
  - /ssl: SSL implementation (handshaking not provided)
  - /cryptopp: crypto primitives

- /tst
  - /ssl: Test Bench (Application Layer)

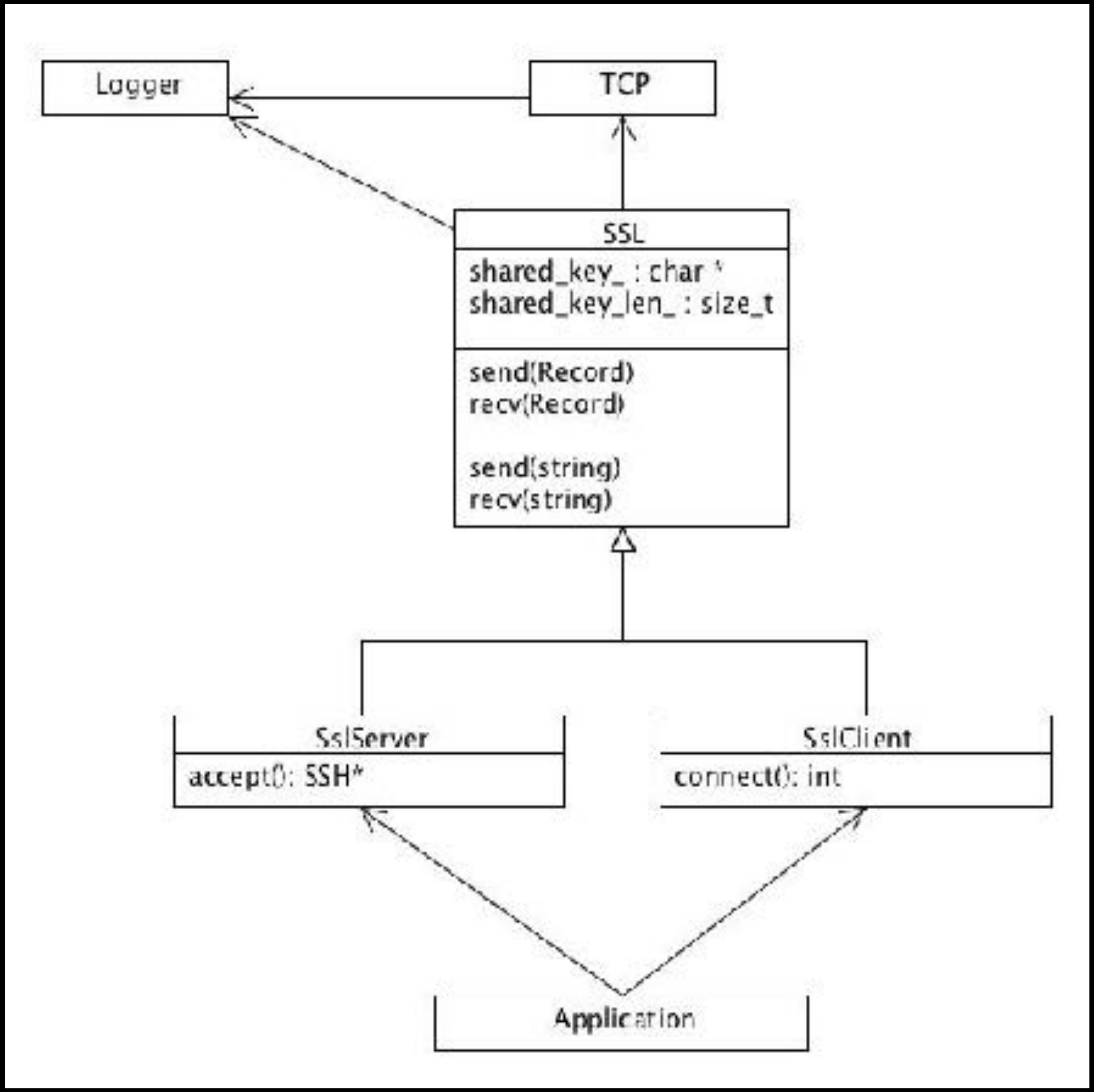
- all makefiles are provided

What is Provided (2/6)

- Test bench simulates a server talking to two clients.
  - ➡Server:
    - Waits for two client connections
    - Once connected, prints out a message from client
    - Waits for two clients to connect, then broadcast a message to both clients
  - ➡Clients:
    - Connects to the server using DHE (test_ssl1.sh) or RSA (test_ssl2.sh)
    - Sends a message to server
    - Waits for server's broadcast

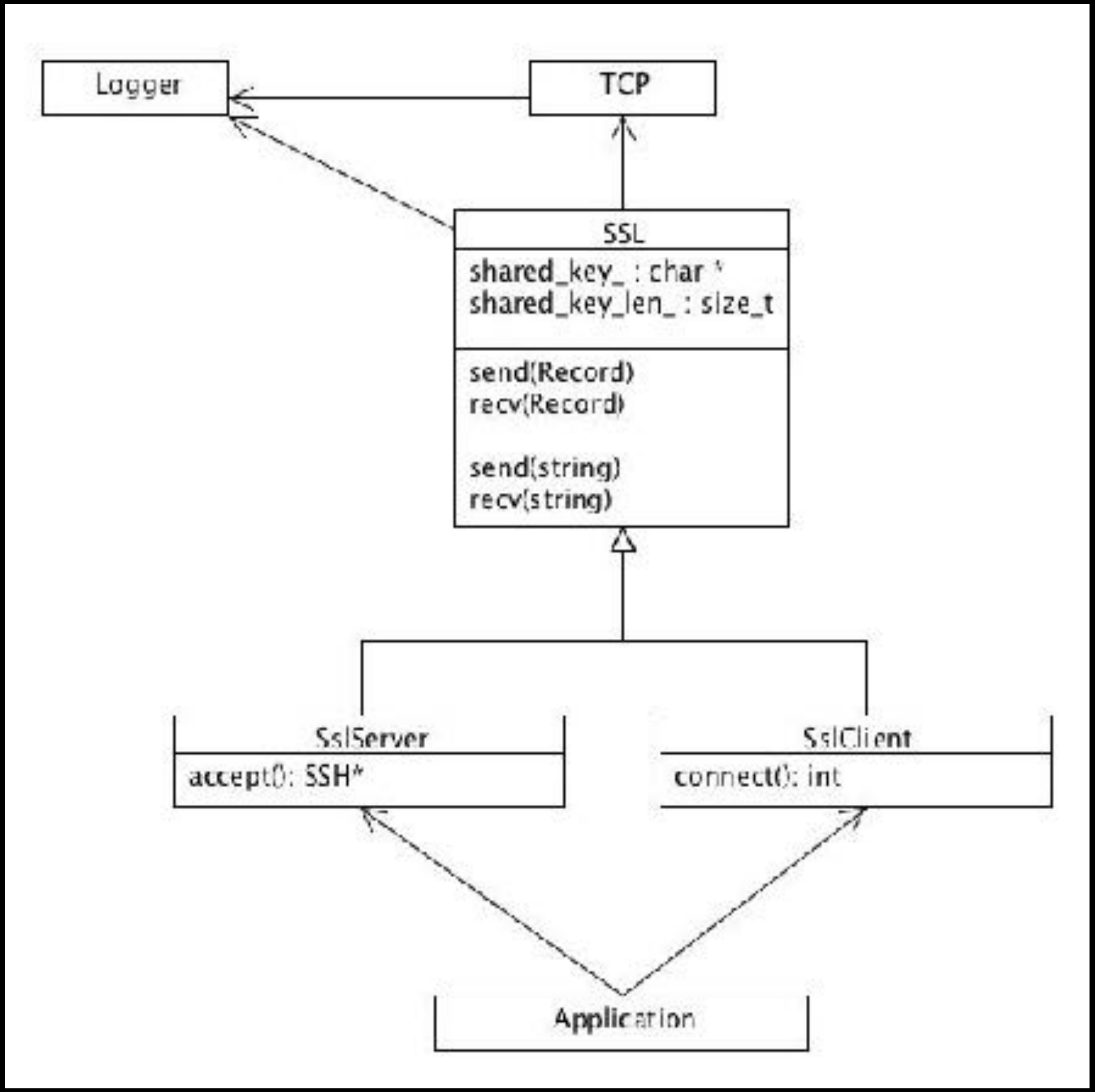
What is Provided (3/6)

- Server application has one instance of SslServer
- Client application has one instance of SslClient
- Once SslServer or SslClient is created, a Logger will be created to log TCP messages.



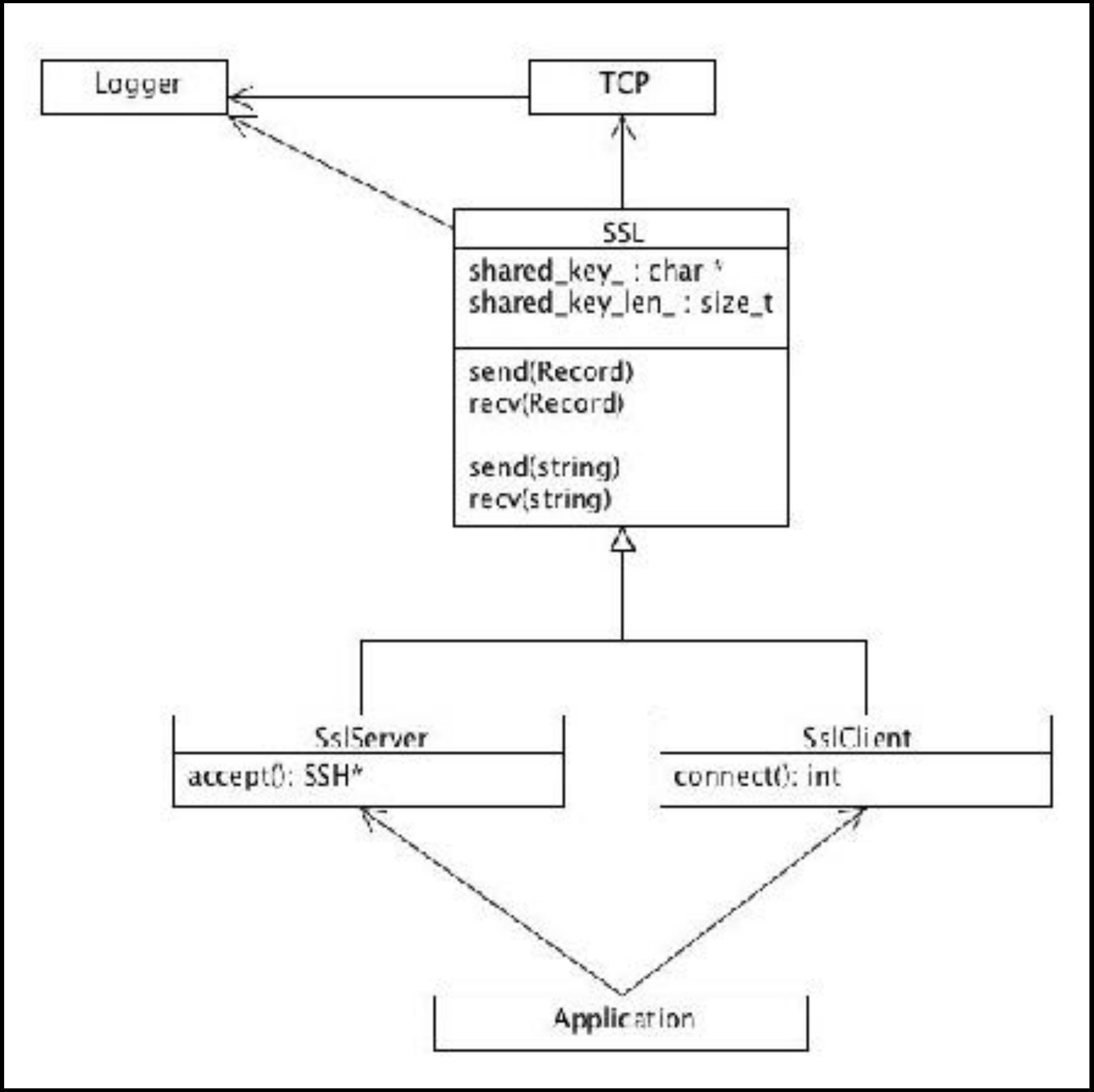
What is Provided (4/6)

- Applications talk to each other using these two functions:
  - SSL::send(string): encrypts string then sends
  - SSL::recv(string&): receives encrypted string then decrypts
- AES in CBC mode with the shared key (see following slides)



What is Provided (5/6)

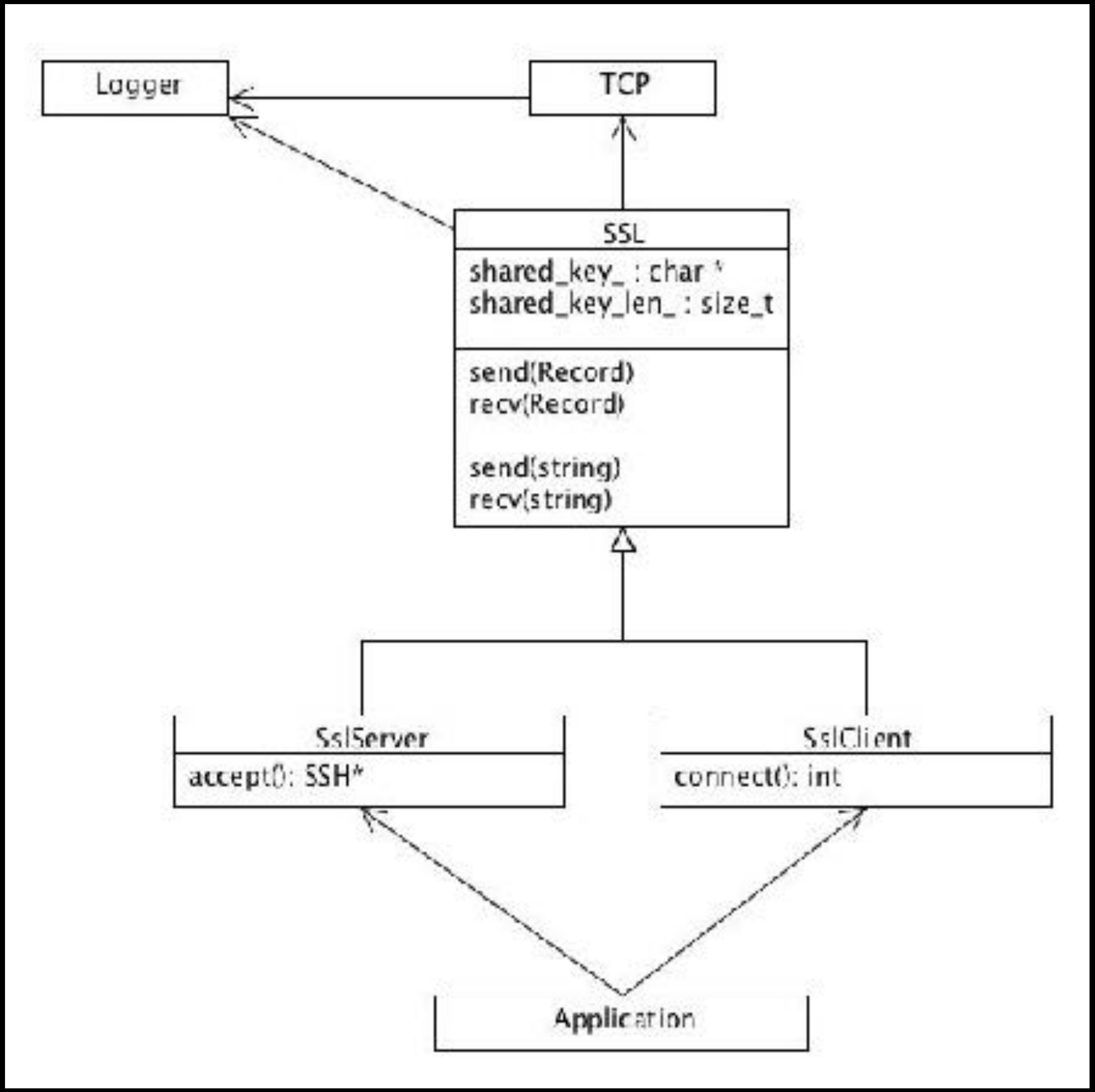
- SSL client and server must talk to each other using records (SSL::Record).
  - SSL::send(Record): flattens Record to a string, and sends it using TCP
  - SSL::recv(Record&): receives a flattened Record using tcp, then reshapes it



What is Provided (6/6)

- After handshake, the shared key and its length are saved inside SSL:
  char* shared_key_
  size_t shared_key_len_

- SSL::send/recv(string) will use this key



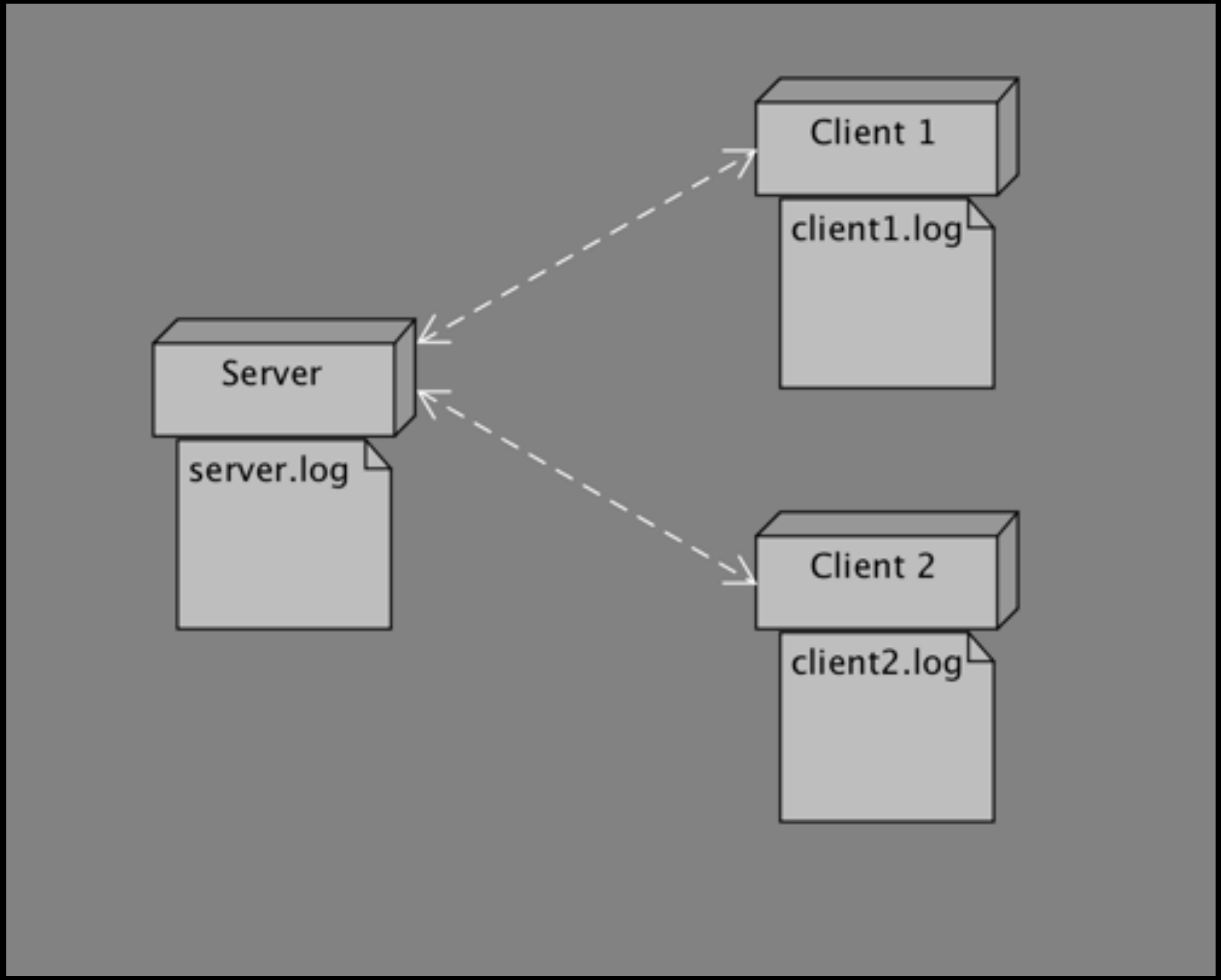
What To Implement

- 2 functions for SSL handshake:
  - (in *ssl_client.cc)* SslClient::connect(ip, port, mode);
    - connect to server using tcp with IP and port
    - do handshake + establish a shared key using DHE / RSA
    - return 0 on success, -1 otherwise
  - (in *ssl_server.cc)* SslServer::accept();
    - waits for client's TCP connection
    - do handshake + establish a shared key using DHE / RSA
    - returns a new SSL instance on success, NULL otherwise

Expected Output

Test Bench Setup

- 1 server
- 2 clients
- each of them has their own logs that log messages from TCP



Terminal Output

(what application layer sees)

```
[ecelinux ./tst/ssl]$ ./test_ssl1.sh

( ... build messages ... )

./$SERVER &

sleep 2
s: started on 129.97.56.12 20000

./$CLIENT 1 &
sleep 1
        c[1]: connected
        c[1]: sent
s: accepted 1 client(s)
s: received 'client says hello'0

./$CLIENT 2
            c[2]: connected
            c[2]: sent
s: accepted 2 client(s)
s: received 'client says hello'0
s: broadcasting
s: shutting down
s: free-ing
        c[1]: received
        c[1]: 'Server says "HELLO ALL"'23
        c[1]: closing
        c[1]: closed
        c[1]: exiting
            c[2]: received
            c[2]: 'Server says "HELLO ALL"'23
            c[2]: closing
            c[2]: closed
            c[2]: exiting

sleep 8
s: exiting

pkill $SERVER &
pkill $CLIENT
[ecelinux ./tst/ssl]$
```

call script that does everything

server starts

client 1 starts, connects and sends msg

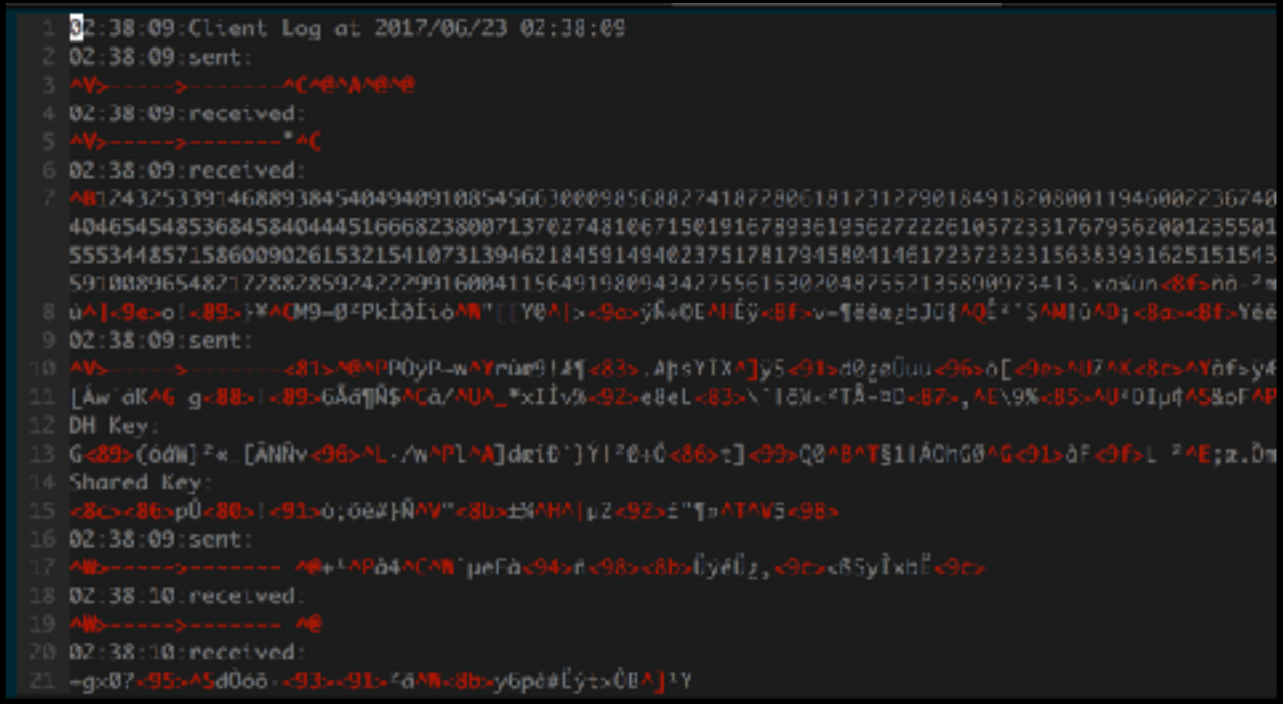
client 2 starts and does same thing

server receives both then broadcasts

both client receives

Client Logs (DHE)

(what Eve sees)



```
02:38:09:Client Log at 2017/06/23 02:38:09
02:38:09:sent:
^V>----->-------^C^@^A^@^@
02:38:09:received:
^V>----->-------*^C
02:38:09:received:
^@124325339146889384540494091085456630000985688274187280618173127901849182080011946002236740
404654548536845840444516668238007137027481067150191678936195627222610572331767956200123550l
5553448571586009026153215410731394621845914940237517817945804146172372323156383931625151543
59100896548217288285924222991600411564919809434275561530204875521358909734l3.xoшun<8f>nō-²m
ù^|<9e>o|<89>}¥^CM9—Ø²PkĬðĬió^N"[[Y0^|><9a>ÿŇ+ÕE^HÉÿ<8f>v—¶ëëœ¿bJÜ{^QÊ²"S^N|û^D¡<8a><8f>Yéé
02:38:09:sent:
^V>----->-------<81>^@^PPÖyP—w^Yrùm9!Æ¶<83>.Aþ$YÏX^]ÿ5<91>d0¿øÜuu<96>ò[<9e>^U7^K<8c>^Yôf>y¥
[Áw`áK^G g<88>|<89>GÅã¶Ñ$^Cà/^U^_*xIİv%<92>e8eL<83>\`|õ¥<²TÅ-¤D<87>,^E\9%<85>^U²DIµ¢^5&oF^P
DH Key:
G<89>(óďW]²«_[ĀNÑv<96>^L·/w^Pl^A]dæiÐ`}Y|²0+Ó<86>t]<99>Q0^B^T§1|ÁChG0^G<91>ðF<9f>L ²^E;æ.Ðm
Shared Key:
<8c><86>pÛ<80>|<91>ó;óe#|Ñ^V"<8b>±¾^H^|µZ<92>£"¶»^T^V5<98>
02:38:09:sent:
^W>----->------- ^@+└^Pà4^C^N`µeFà<94>ñ<98><8b>ĺÿéĺ¿,<9c><85yÎxH⁼<9c>
02:38:10:received:
^W>----->------- ^@
02:38:10:received:
-gx07<95>^5dÒöõ·<93><91>²á^N<8b>y6pé#Ľÿ‡>ÛB^]¹Y
```

Client asks for connection

* HANDSHAKING *

(debug: prints shared key to log)

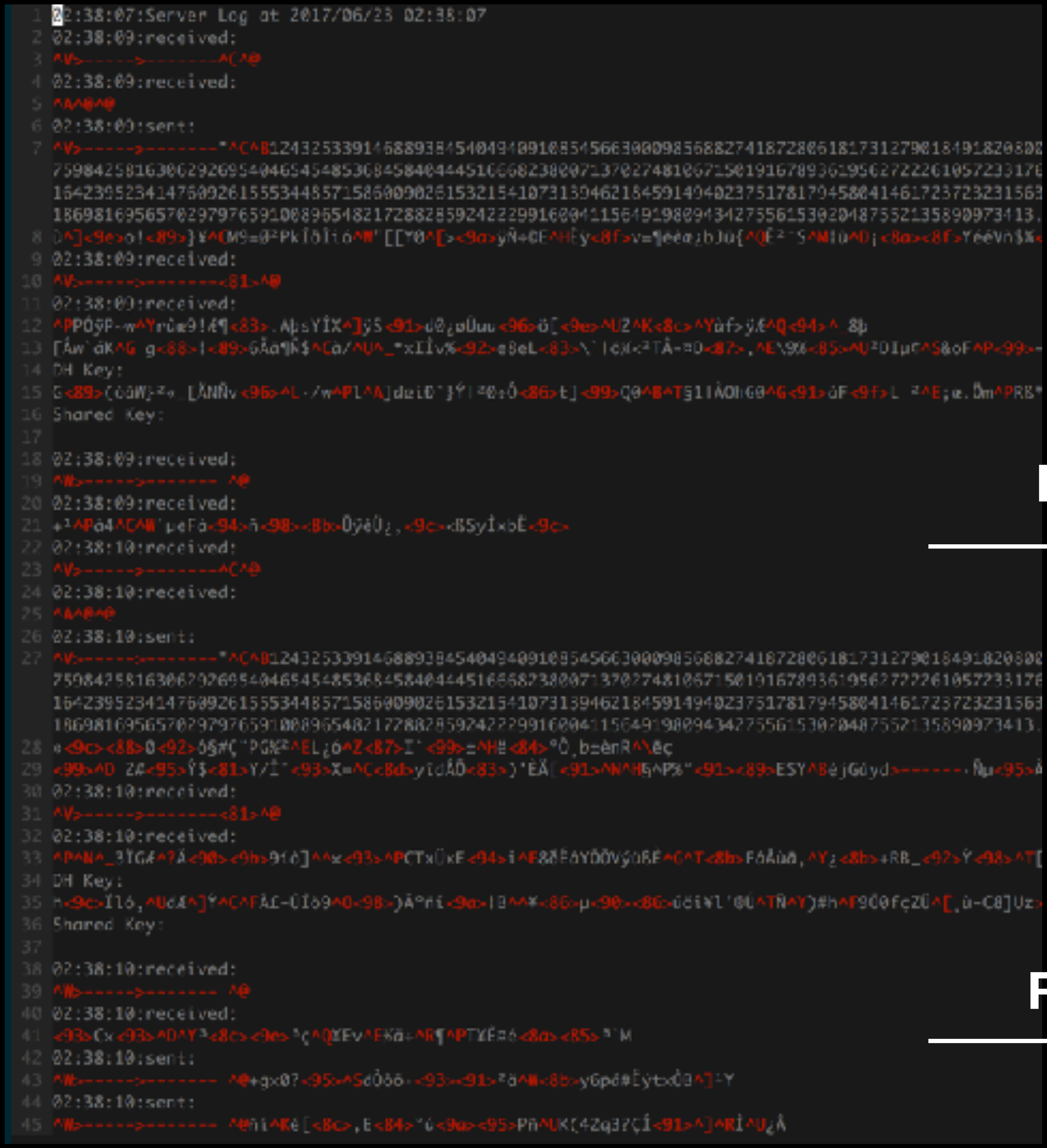
Sends Enc(msg)

Receives broadcast Enc(msg)

Note: SSL::recv(Record) calls TCP::recv twice, thus two receive calls are logged when only one Record is received. This is normal because the first call is only receiving the SSL header to extract the length of the data, and the second call actually receives the data.

Server Logs (DHE)

(what Eve sees)



```
02:38:07:Server Log at 2017/06/23 02:38:07
02:38:09:received:
^V>----->-------^C^@
02:38:09:received:
^A^@^@
02:38:09:sent:
^V>----->-------"^C^B1243253391468893845404940910854566300098568827418722806181731279018491820808
75984258163062926954046545485368458404445166682380071370274810671501916789361956272226105723317
164239523414760926155534485715860090261532154107313946218459149402375178179458041461723723231563
1869816956570297976591008965482172882859242229916004115649198094342755615302048755213589097341 3.
D^]<9e>o!<89>}¥^CM9=0²PkÎõÎió^W"[[Y0^[><9a>ÿÑ+ŒE^HËy<8f>v=¶ëèæ¿bJü{^QÊ²¨S^M|ü^D¡<8a><8f>YéeVn$%<
02:38:09:received:
^V>----->-------<81>^@
02:38:09:received:
^PPÓÿP-w^YrùæØ!Æ¶<83>.AþsYÎX^]ÿS<91>d0¿øÛuu<96>ö[<9e>^UZ^K<8c>^Yùf>ÿ£^Q<94>^_8þ
[Áw`áK^G g<88>|<89>6Åä¶Ñ$^Ca/^U^_*xIÌv%<92>e8eL<83>\`|cX<²TÂ-ª0<87>,^E\9%<85>^U²DIµ€^S&oF^P<99>-
DH Key:
G<89>{ôäW}²«_[ÀNÑv<96>^L·/w^Pl^A]deiÐ`}Ý|ª0±Ô<86>t]<99>Q0^B^T§1|ÀOhG0^G<91>áF<9f>L ²^E;æ.Ðm^PRB*
Shared Key:

02:38:09:received:
^W>----->------- ^@
02:38:09:received:
+¹^Pá4^C^W`µeFà<94>ñ<98><8b>ÖÿéÛ¿,<9c><ß5yÌxbË<9c>
02:38:10:received:
^V>----->-------^C^@
02:38:10:received:
^A^@^@
02:38:10:sent:
^V>----->-------"^C^B1243253391468893845404940910854566300098568827418722806181731279018491820808
75984258163062926954046545485368458404445166682380071370274810671501916789361956272226105723317
164239523414760926155534485715860090261532154107313946218459149402375178179458041461723723231563
1869816956570297976591008965482172882859242229916004115649198094342755615302048755213589097341 3.
«<9c><88>0<92>ó§#Ç"PG%ª^EL¿ó^Z<87>I`<99>=^H‼<84>°Ô¸b=ènR^\ëç
<99>^D ZÆ<95>Ý$<81>Y/Î`<93>X=^C<8d>yîdÂÕ<83>)'ÈÄ[<91>^N^H§^P%"<91><89>ESY^BèjGöyd>-------Ñµ<95>Å
02:38:10:received:
^V>----->-------<81>^@
02:38:10:received:
^P^N^_3ÏG£^?Ä<90><9b>91ó]^^x<93>^PCTxÜxE<94>i^F8ÃÊáYÐÕVÿöBF^G^T<8b>FóÅùä,^Y¿<8b>+RB_<92>Ý<98>^T[
DH Key:
h<9c>Ïló,^Ud£^]Ý^C^FÂ£-ÜÏó9^O<98>)Ä°ñi<9a>|B^^¥<86>µ<90><86>üčï¥l'0Ú^TÑ^Y)#h^F9C0fçZÜ^[¸ù-C8]Uz<
Shared Key:

02:38:10:received:
^W>----->------- ^@
02:38:10:received:
<93>Cx<93>^D^Y³<8c><9e>³ç^Q¥Ev^EKä÷^R¶^PTŸÉ=6<8a><85>³`M
02:38:10:sent:
^W>----->------- ^@+gx0?<95>^SdÔöö·<93><91>²ä^W<8b>yGpd#Ìÿt×Ó3^]÷Y
02:38:10:sent:
^W>----->------- ^@ñi^Kë[<8c>,E<84>´ú<9a><95>Pñ^UK{4Zq3?ÇÍ<91>^]^RÌ^U¿Å
```

Connection of 1st client

* HANDSHAKING *

(debug: prints shared key to log)

Received 1st client Enc(msg)

Connection of 2nd client

* HANDSHAKING *

(debug: prints shared key to log)

Received 2nd client Enc(msg)

Broadcast Enc(msg) to both

Additional Notes

Suggested Tool For Log Analysis

- In the previous screenshots, vim was used to open up the .log files. This results in many unreadable characters. (vim ssl_server_*.log)

- A better tool to deeply analyze the logs is 'xxd' (piped into 'less'). (xxd ssl_server_*.log | less)

- xxd allows you to view the ASCII hex representation of each unprintable character inside the log file. For example, a backspace character cannot be printed, but it gets displayed as '08' with xxd.

Last But Not Least

- RSA encryption and decryption functions can be found in ./src/ssl/crypto_adaptor.h
- Feel free to create your own server/client test cases inside ./tst/ssl and add them to the makefile