Stream Cipher

Security

For the security of the stream cipher, it is vital to have a key stream that have a large period and it must be impossible to recover the cipher's from the key stream.
In order to use the stream cipher securely, the same key stream should never be used twice, since attackers could easily compute the plain text in this situation.
Usually, the stream cipher could only provide the privacy and cannot provide the authentication, which means the encrypted message may have been modified in transit.

Usage

The speed of the stream cipher is fast compared with others and it is simple to implement.
Besides, the stream cipher is suitable for applications where plaintext comes in quantities of unknowable length, because the stream cipher does not need padding and just operates on smallest unit that can be transmitted.

Man in the Middle Attack

pseudo-code

```
void function start() {

        // Create a server socket to receive the connect from the client
        ServerSocket = new Socket();
        // accept the connect from the client
        socket = serverSocket.accept();
        // create the writer and reader to the client
        writer = socket.DataOutputStream();
        reader = socket.DataInputStream();

        SMsocket  = Socket(ip, port)      //SMsocket is the socket between the server and attacker
        SMwriter = SMsocket.DataOutputStream();
        SMreader = SMsocket.DataInputStream();

        //contact phases
        contactPhase()

        // exchange key phase
        exchangePhase();

        //specification phase
        specificationPhase();

        //communication phase
        communicationPhase();

        //exist
        exit();
}

void function contactPhase() {

        // Read CLIENT_HELLO(studentId) from the client
        message = reader.read();
        // Send the message to the server
        SMwriter.write(message);
```

```
        // Read SERVER_HELLO() from the Server
        message = SMreader.read();
        // Send the message to the client
        writer.write(message);
}

void exchangePhase() {
        // read the CLIENT_DHEX_START() from the client
        message = reader.read();
        // send the message to the server
        SMwriter.write(message);

        //read the SERVER_DHEX(g,p,g^b) from the server
        message = SMreader.read();
        // get the DHExStartResponse from the message
        DHExStartresponse = new DHExStartResponse(message);

        // get the generator, prime, public key of the server from the DHExStartresponse
        generator = response.getGenerator();
        prime = response.getPrime();
        pkServer = response.getPkServer();

        // create the key pair for the client
        BigInteger temKey = DHEx.createPrivateKey(2048);
        BigInteger[] pair = DHEx.createDHPair(generator, prime, temKey);
        skToClient = pair[0];
        pkToClient = pair[1];

        // create new DHExStartresponse with the pkToClient and send it to the client
        writer.write(new DHExStartresponse(generator, prime, pkToClient));

        // get the public key of the client
        message = reader.read();
        dhexRequest = new DHExRequest(message)
        pkClient = dhexRequest.getPkClient();

        // Calculate the shared key between the attacker and client
        MCsharedKey = DHEx.getDHSharedKey(pkClient, skToClient, prime);

        // Create the key pair for the server
        temKey = DHEx.createPrivateKey(2048);
        pair = DHEx.createDHPair(generator, prime, temKey);
        skToServer = pair[0];
        pkToServer = pair[1];

        // create new dhexRequest with pkToServer and send it to the server
        SMwriter.write(new DHExRequest(pkToServer));

        // calculate the shared key between the server and attacker
        SMsharedKey = DHEx.getDHSharedKey(pkServer, skToServer, prime);
```

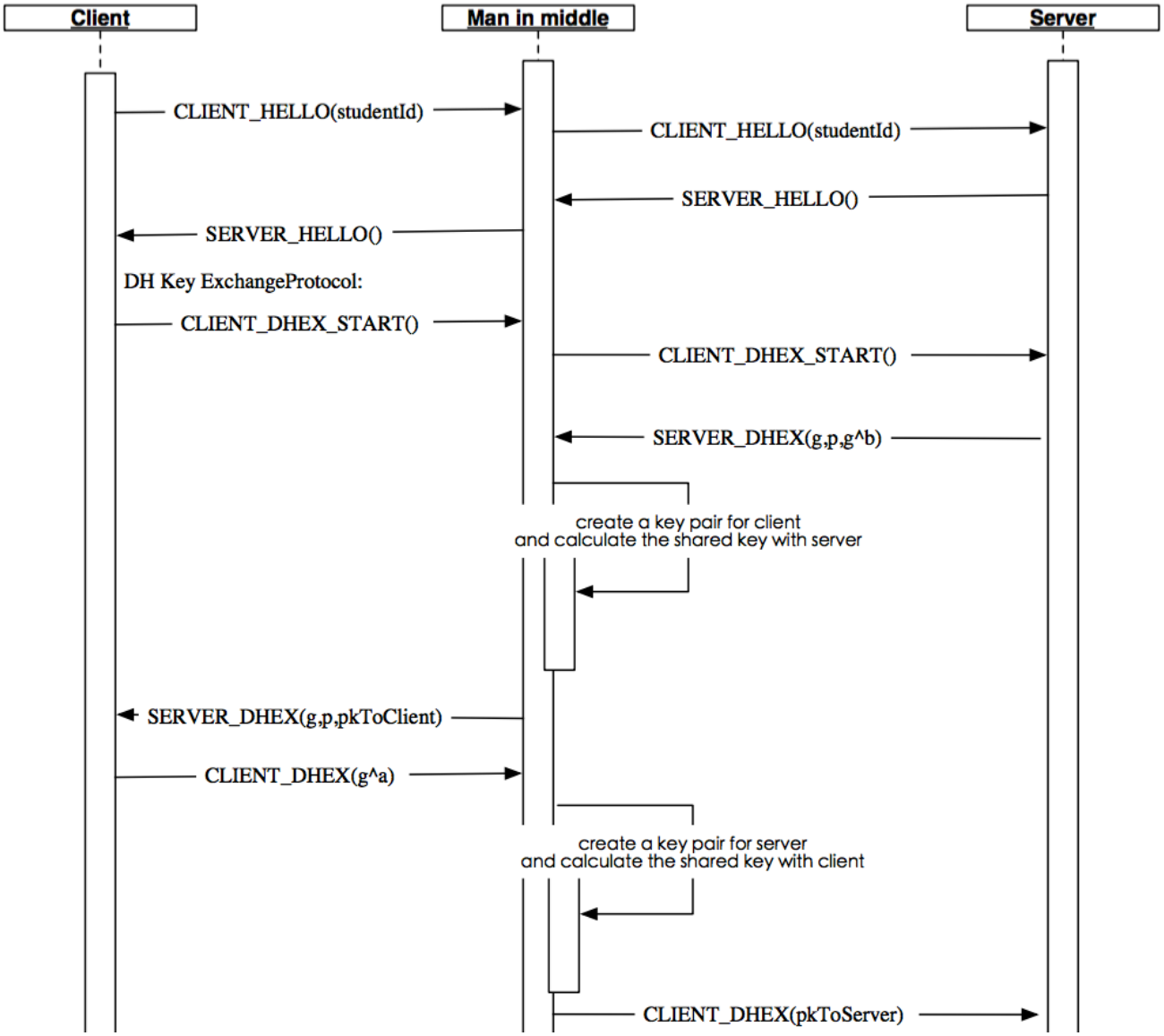
```
        // read the SERVER_DHEX_DONE() from the server
        message = SMreader.read();
        // send the message to the client
        writer.write(message);

        //Receive the CLIENT_DHEX_DONE(g^ab) from the client
        message = reader.read();
        // Create new DHExDoneRequest with the SMsharedKey and send it to the server
        SMwriter.write(new DHExDoneRequest(SMsharedKey));

}

void specificationPhase() {
        // Receive the message from the server
        message = SMreader.read();
        if message.contains("SERVER_DHEX_ERROR") {
                // send it to the client and then throw exception
                writer.write(message)
                throw new ServerDHKeyException()
        } else {
                // get linesToWork, p1, p2 from the message
                specsResponse = new SpecsResponse(message);
                linesToWork = specsResponse.getOutLines();
                p1 = specsResponse.getP1();
                p2 = specsResponse.getP2();
                // send the message to the client
                writer.write(message);
        }
}

void communicationPhase() {
        // Create the sreamCipher for server and client
        SMstreamCipher = new StreamCipher(SMsharedKey, prime, p1, p2);
        MCstreamCipher = new StreamCipher(MCsharedKey, prime, p1, p2);

        // receive the CLIENT_SPEC_DONE() from the client
        message = reader.read();
        // send it to the server
        SMwriter,write(message);

        // Receive all text from Server and send them to the client
        receiveAllLines();

        // Reset the Shift Register prior to sending out CLIENT_TEXT messages
                SMstreamCipher.reset();
                MCstreamCipher.reset();
        // Receive all text from Client and Send them to Server
                sendAllLines();

        // receive the SERVER_COMM_END() from the server and send it to the client
        message = SMreader.read();
```

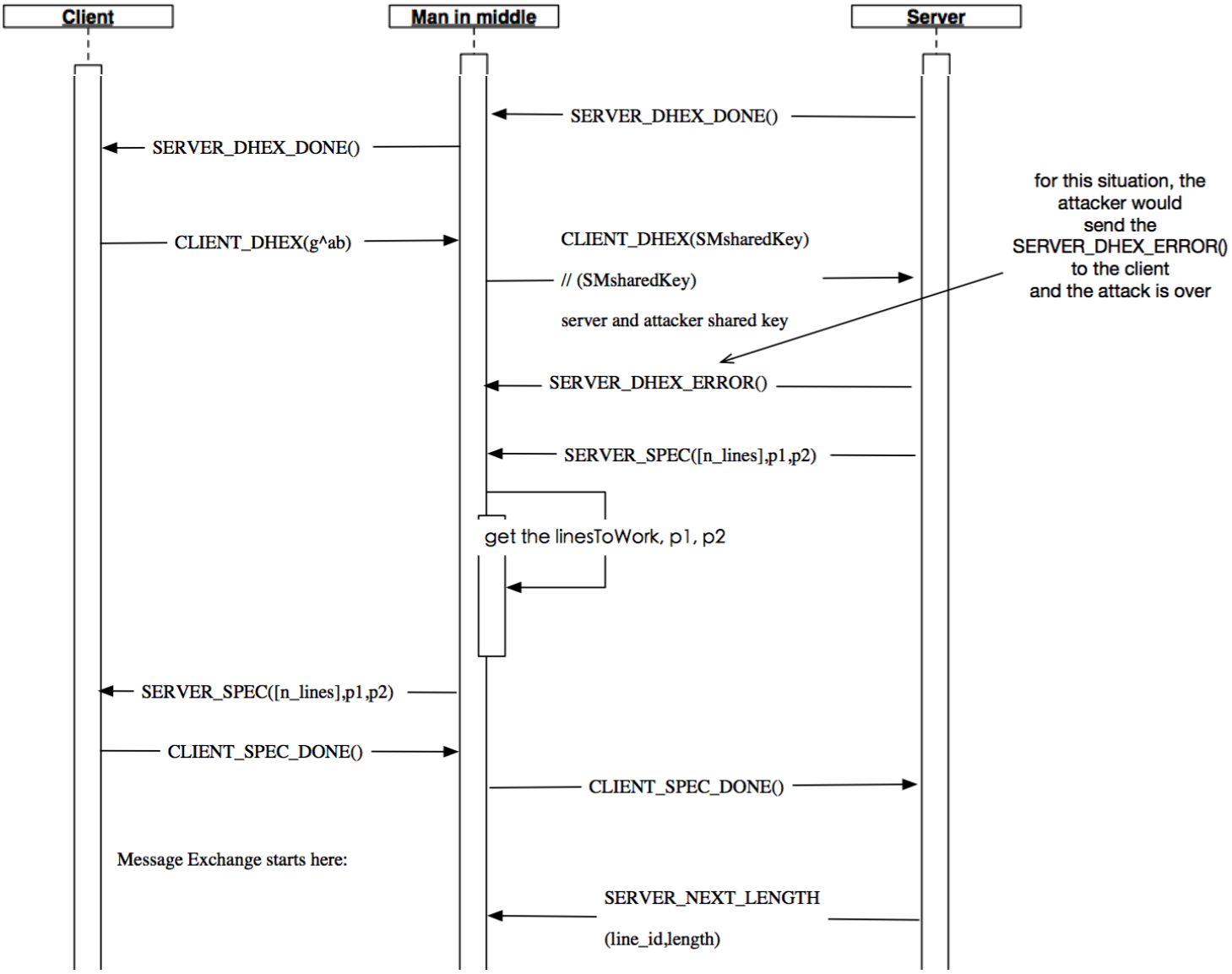
```
        writer.write(message);

        // receive the CLIENT_COMM_END() from the client and send it to the server

        message = reader.read();

        SMwriter.write(message);



}

void receiveAllLines() {
        // let's do this until no more lines are received from the server!
        while (true) {
                        if (receiveLine())
                                break;
                }
}


boolean receiveLine() {
        // read the message from the server
        message = SMreader.read();
        if (message.contains(SERVER_TEXT_DONE)) {
                // send the message to the client
                writer.write(message);
                return true;
        } else {
                // get the messageLength
                response = new NextLengthResponse(message);
                messageLength =  response.getLength();

                // Since we will change the content of the message, we need to identify the real
messageLength of the modified message. In order to realize this, firstly, we need to get the original message.
                // Send the CLIENT_NEXT_LENGTH_RECV() to the server
                SMwriter.write(new MessageLengthReceivedRequest());

                //get the original message from the server, decrypt with the SMsharedKey, modify it
and encrypt it with MCsharedKey

                message = SMreader.read();

                textResponse = new TextResponse(message);

                decryptedBody = decrypt(textResponse.getBody(), SMstreamCipher);

                encryptedMsg = MCstreamCipher.encrypt(decryptedBody);

                // change the textResponse with the modified encrpytedMsg

                textResponse.setBody(encryptedMsg)

                // send the modified next_length message to the client
```

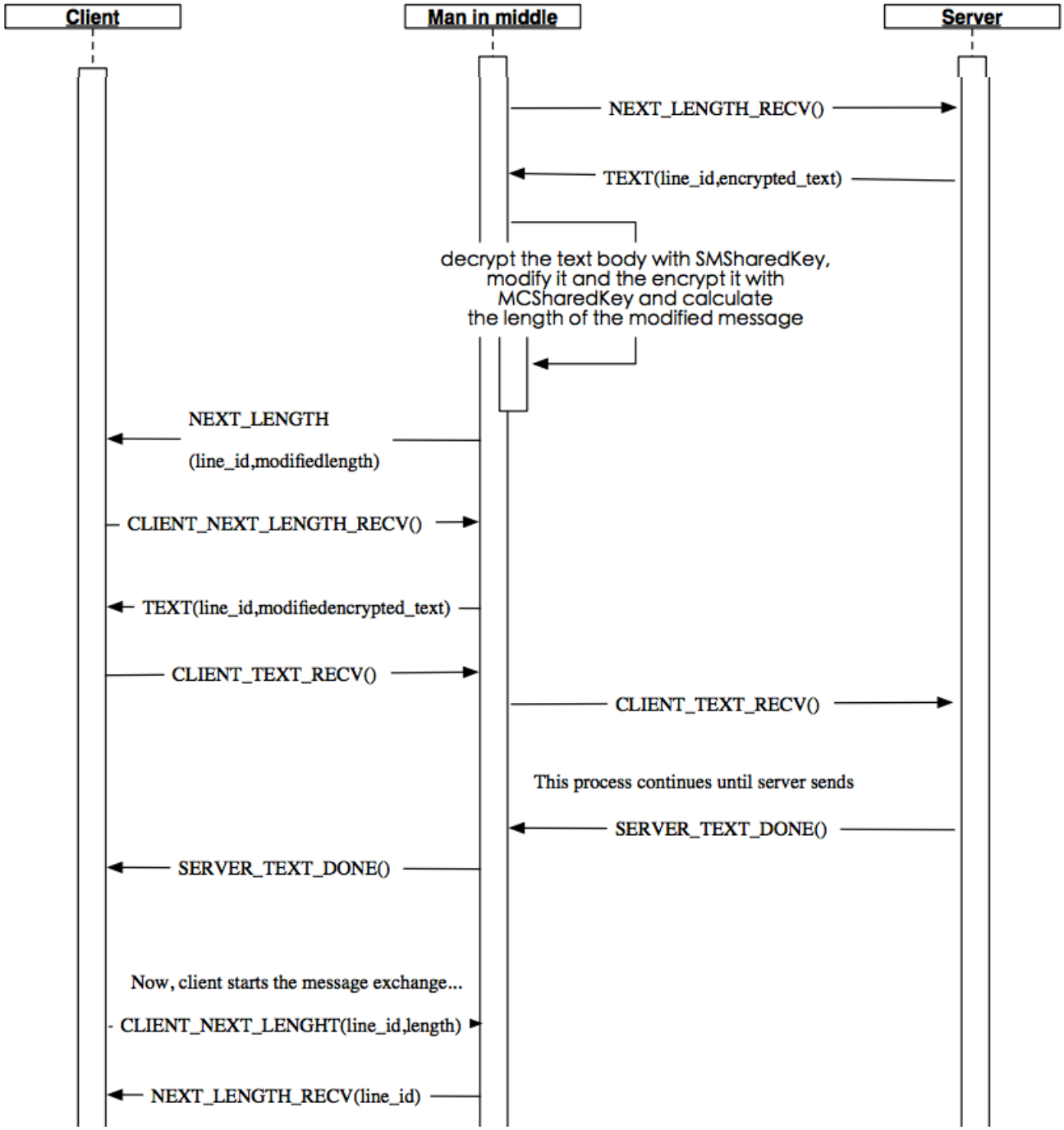
```
                response.setLength(textResponse.length);

                writer.write(response)

                // Receive Acknowledgement of Message Length from the client

                message = reader.read();

                // send the modified message to the client
                writer.write(textResponse);

                // receive the inform from the client and send it to server
                message = reader.read();
                SMwriter.write(message);
        }
        return false;
}

void sendAllLines() {
        For (i = 0; i <linesToWork.length; i++) {
                // receive the message and send it to server after alteration
                sendLine();
        }

        // receive the CLIENT_TEXT_DONE() from the client and send it to the server
        message = reader.read();
        SMwriter.write(message);


}

void sendLine() {
        // receive the CLIENT_NEXT_LENGHT(line_id,'length) from the client
        message = reader.read();

        // get the messageLength of the original message
        lenRequest = new LenRequest(message);
        messageLength = lenRequest.getLength();

        // send the MessageLengthReceivedResponse to the client to get the real message
        writer.write(new MessageLengthReceivedRequest());

        // receive the real message from the client, decrypt it with MCSharedKey, modify it and encrypt it
with the SMSharedKey

        message = reader.read();
        textRequest = new TextRequest(message);
        decryptedBody = decrypt(textRequest.getBody(), MCstreamCipher);
        decryptedBody = decryptedBody + modifyContent;
        encryptedMsg = SMstreamCipher.encrypt(decryptedBody);

        // set the textRequest with the modified encryptedMsg
        textRequest.setBody(encrpytedMsg);

        // send the length of the modified textRequest to the server
```

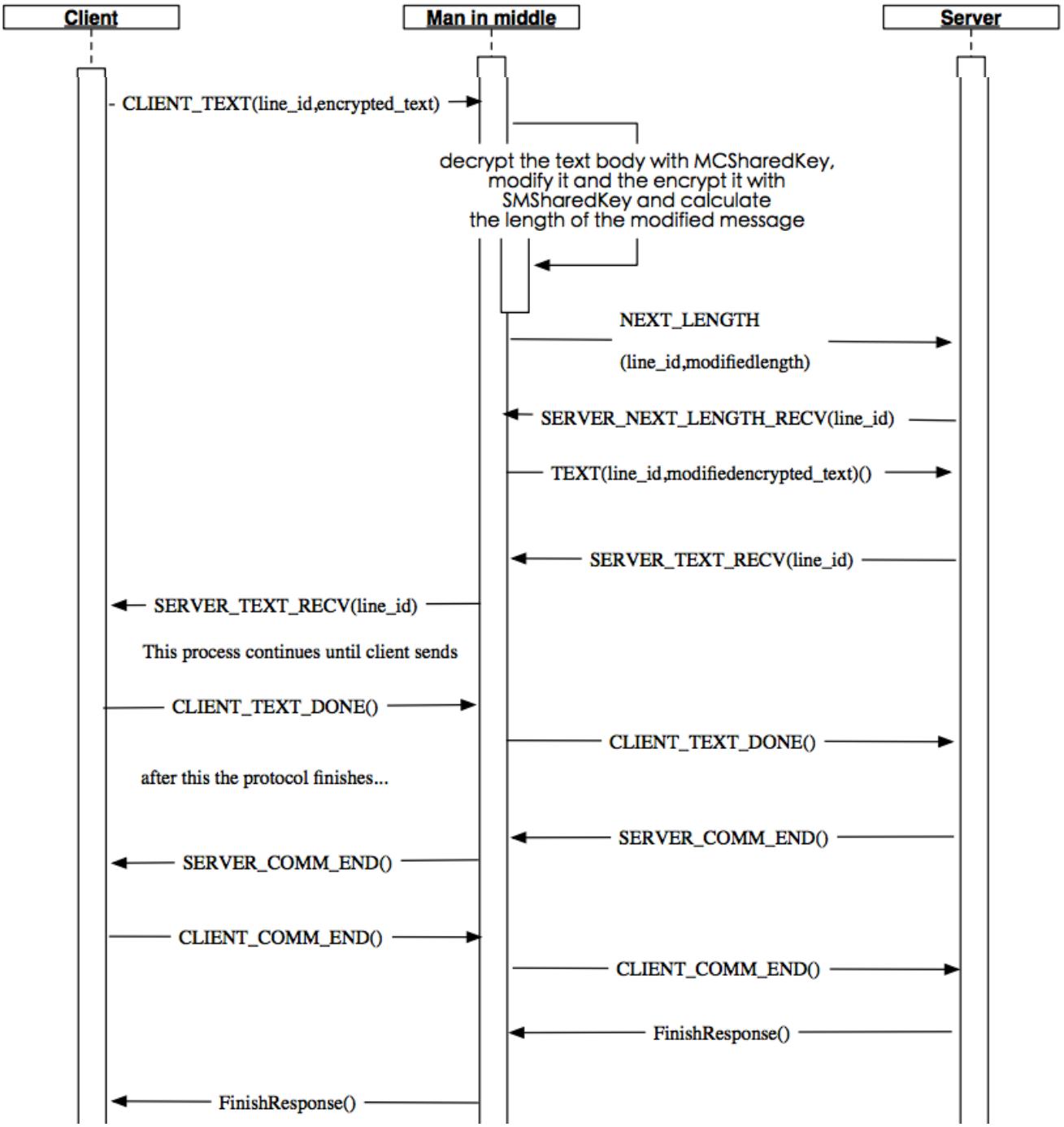
```
        SMwriter(new NextMessageLengthRequest(textRequest.length()));

        // Receive the messageLengthReceivedResponse from the server
        message = SMreader.read();

        // send the modified message to the server
        SMwriter.writer(textRequest);
        //Receive the inform from the Server and send it to the client
        message = SMreader.read();
        writer.write(message);
}

void exit() {
        // process the last message
        message = SMreader.read();
        writer.write(message);
}
```

Diagram




Client
Man in middle
Server
SERVER_DHEX_DONE()
SERVER_DHEX_DONE()
for this situation, the attacker would send the SERVER_DHEX_ERROR() to the client and the attack is over
CLIENT_DHEX(g^ab)
CLIENT_DHEX(SMsharedKey)
// (SMsharedKey)
server and attacker shared key
SERVER_DHEX_ERROR()
SERVER_SPEC([n_lines],p1,p2)
get the linesToWork, p1, p2
SERVER_SPEC([n_lines],p1,p2)
CLIENT_SPEC_DONE()
CLIENT_SPEC_DONE()
Message Exchange starts here:
SERVER_NEXT_LENGTH
(line_id,length)


Client
Man in middle
Server
NEXT_LENGTH_RECV()
TEXT(line_id,encrypted_text)
decrypt the text body with SMSharedKey,
modify it and the encrypt it with
MCSharedKey and calculate
the length of the modified message
NEXT_LENGTH
(line_id,modifiedlength)
CLIENT_NEXT_LENGTH_RECV()
TEXT(line_id,modifiedencrypted_text)
CLIENT_TEXT_RECV()
CLIENT_TEXT_RECV()
This process continues until server sends
SERVER_TEXT_DONE()
SERVER_TEXT_DONE()
Now, client starts the message exchange...
CLIENT_NEXT_LENGHT(line_id,length)
NEXT_LENGTH_RECV(line_id)


Client
Man in middle
Server
- CLIENT_TEXT(line_id,encrypted_text)
decrypt the text body with MCSharedKey,
modify it and the encrypt it with
SMSharedKey and calculate
the length of the modified message
NEXT_LENGTH
(line_id,modifiedlength)
SERVER_NEXT_LENGTH_RECV(line_id)
TEXT(line_id,modifiedencrypted_text)()
SERVER_TEXT_RECV(line_id)
SERVER_TEXT_RECV(line_id)
This process continues until client sends
CLIENT_TEXT_DONE()
CLIENT_TEXT_DONE()
after this the protocol finishes...
SERVER_COMM_END()
SERVER_COMM_END()
CLIENT_COMM_END()
CLIENT_COMM_END()
FinishResponse()
FinishResponse()