# Logbook

# 10/07/2021

I’ve decided to continue the development of the messaging application. The first goal is to re-work the message exchange to use symmetric (AES) encryption. This will pave the way for the implementation of file exchange, although how the file exchange will be implemented is yet to be decided.

Created new branch: Symmetric block

Current objective: implement symmetrically encrypted messages

## Later on

I decided that the client and server components were messy and would benefit from a refactor.

All networking will be moved away from client and server components and instead will reside in NetworkUtility. NetUtil is static and stateless, it is passed every object it needs in order handle transmission. This allows client and server components to focus on handling state rather than the details of how transmission are done.

Transmissions have been split into three categories (with send and receive for each one): plaintext, RSA and AES. It would be best to make all of these asynchronous, and now that they have been abstracted away it is much more feasible, I’ll spend some time experimenting, but for now I want to make move all networking functions as they are to NetworkingUtility, then I’ll consider making them asynchronous.

## Changes

Created NetworkUtility class  
Created NetWorkUtilityException (extends Exception)  
Moved networking code from ClientComp to NetworkUtility

# 11/07/2021

All Networking functions have been moved to NetworkUtility.   
Instead of functions depending on the cryptography, instead there will be functions for synchronous/asynchronous sending/receiving. It will be the responsibility of the client/server to build the transmission formats/interpret what they receive. NetworkUtility will only be concerned with how something is sent, not the content that is sent.  
An asynchronous socket connect method has been implemented, this way when trying to send a message to an offline contact the user is not blocked while waiting for the connection to fail.

Yet another class may be implemented, TransmissionFormatter. This class could be responsible for taking information and returning a formatted byte array ready for transmission, the array can then be passed to NetworkUtility for either synchronous or asynchronous transmission.  
It can also be used for decoding a transmission (e.g. with RSA it can extract and return a tuple containing a byte array with the data and a byte array with the signature, which the server will then worry about validating).

Implemented Asynchronous socket connection in NetworkUtility.

## Later on

A new concern has been realised, when an exception occurs in a thread (e.g. sending/receiving data) the exception is *not* thrown to the invoker if it is in another thread. As a result one of two things must happen, either each method is provided with a callback to use for exceptions, or I need to find something that allows that to happen, from superficial research I’ve seen that Tasks might allow that.

Research Tasks

# 14/07/2021

I think I understand enough about tasks to try to migrate every async function to a task. A test using tasks for the initial connection was successful.

New GIT branch: TaskMigration

TODO – reduce time trying to connect port – make sure that the client still waits for receive confirmation

## Later On

I may have to make some adjustments later on, but now I have successfully implemented the following methods for NetworkUtility: connectSync/Async, transmitSync/Async, receiveSync/Async.

Right now the code is a mess, filled with old unused code, client and server still have lots of parts that do not conform to the new pattern that have been implemented. Before merging task migration I am going to clean up the code and restructure client/server to use the new pattern entirely.

Finally today I will also begin implementing the TransmissionFormatter class that was postulated on 11/07/2021.

NetWorkUtility synchronous and asynchronous methods finished.

ToDo: clean up code and ensure client/server conform to the new pattern

Todo: begin implementation of TransmissionFormatter