# OOP3 Final Assignment – 2021-2022

## Context

Email has become one of the critical means of communication. With that some serious issues have arisen, such as spam and security.

Clients like MS Outlook try filtering spam, but that only works partially. Furthermore, the place of filtering is not ideal; bandwidth has been spent delivering the message anyway, and the technologies used still have you end up with the emails, just placed in a separate folder.

Another issue is security. The commonly used email protocols do not use any encryption, mainly because it would prevent users not know to the recipient from sending a message. Email clients usually have the option to install a plugin to do encryption, but they are not easy to use and offer limited functionality and strength of the encryption. And without encryption the email can be easily retrieved from the server and analyzed.

Last but not least there is the issue of confidentiality. Even if your emails are encrypted, they will be (temporarily) stored on some server, making it possible to trace one’s connections. Especially in unfree countries this is something people might want to avoid.

This assignment aims to create a new email client with built-in strong encryption and the option not to use a server as intermediary between two parties communicating.

### Some notes on the implementation

As this is just a first version of the application, not all functionality will be implemented, just the ones critical to being able to use it. This specifically means:

* The preferred way of communication between to clients is without using a server, instead using a direct networking connection with networking sockets.
* If this is not available (the receiver being offline or not having a known IP address) an email server will be used. IMAP will not be implemented in this first version, only POP3 and SMTP.
* As all encryption is being done in the application itself, no SSL/ TSL layer will be used.
* All messages will be stored in their encrypted formats. Decryption and encryption of messages is done in the background; the only place a decrypted message resides is in memory, never on permanent storage.
* In this first version, only messages received will be stored, not sent messages.
* As computer technology is advancing rapidly, our current encryption might not be good enough in only a few years. The chosen architecture needs to be adaptable to use newer techniques.

### Dealing with customers

The customer creating this idea did (obviously) not program it before describing this. Things might be unclear, requirements can be vague or contradict themselves, a seemingly simple functionality might turn out to be much harder than imagined. Do not just assume it’s you who should understand better! Of course, first think about it yourself, but if that does not give clarity, ask the customer! In this case the customer sometimes also is a technical consultant.

## Functional description

### Domain Model



This is not a definite description of the classes needed, but shows the four most important ones from the domain view.

A Contact is any person or organization the user wants to exchange messages with (including the user himself/ herself.) Needed to know are at least an email address (functioning as a username as well,) a name and a public key, which is given to the user in the form of a certificate file (see the description of the encryption to use.) The IP address and port are needed when exchanging messages using sockets, the SMTP address is needed when using SMTP, so at least one of those is required.

User is a Singleton; the software is written for single users. For a user the password and private key are also needed.

Key represents a key for asymmetric encryption, with a public key in a certificate (which can be shared with others), a private key, a password and if needed a salt.

Message will in this first version only contain text, other mime types can be added later. A message has a sender and a receiver (we will ignore multiple receivers in this version), a subject (or title), a timestamp when it has been sent, and a body with the text. In the class diagram six methods have been given, of which four are private and are used by the other two.

* When sending a message, it is first signed using the private key of the sender, then encrypted using the public key of the receiver.
* When receiving a message, it is stored on permanent storage (hard disk) in its encrypted form. When reading it the application first decrypts it using the private key of the user, then decrypted using the public key of the sender. This both verifies confidentiality as well as ensures the sender indeed was the contact (if not the message will be garbled.)

### Use Case Diagram – Contact Management



The application has the following menu options:

* Editing the user’s profile. When no profile has been made this becomes creating a profile. In the profile the user sets his email address, name, IP Address and port, SMTP server and creates a certificate with password and private key.
* Showing a list of contacts. No editing functionality is needed here, only the option to delete a contact.
* Create a new contact, giving the name, either the IP Address and port, the SMTP server or both, and giving the location of the certificate file. You can either have a file picker here or just have the user enter the directory as a string.

### Use Case Diagram - Reading and sending messages



The application has a list of messages showing only title and timestamp. When selecting a message in the list, details are shown on the right.

When creating a message the user sets the title and text himself. The application adds the timestamp when sending. Before sending the message is encrypted as follows:

* Sign the message by encrypting it with the private key of the sender
* Prepend this message by the name of the sender. This is necessary to tell the receiver whose key to use in decrypting.
* Encrypt the message using the public key of the receiver.

The following part happens in a background thread. First try setting up a connection with the receiver using network sockets. If that works send the message over this connection, making sure not to block him for multiple senders. If there is no connection possible, send the message using SMTP.

### Use Case Diagram – Receiving Messages

### 

The application should listen on the network port configured in the profile. When receiving a message there it:

* Stores the message to disk in its encrypted form
* Starts a background thread to unencrypt
* In this background thread:
  + The message is decrypted using the private key of the user
  + The result starts with the username of the sender. Get his public key from the contact list and use that to decrypt the message
  + Store the resulting message in the list of messages which can be shown.

Once every ten minutes the application connects to the POP server to check for messages there. I there are, use POP3 to retrieve them. Decrypt them the same way as when receiving over sockets.

## Architecture

The encryption used will be provided by the BouncyCastle libraries, however the application needs an architecture which is able to easily change this, even at runtime. Make good use of Design Patterns to achieve this; for this specific requirement a combination of a Factory and a Strategy Patterns can be used.

User is a Singleton, the software is written for single users, so use that pattern.

The list of contacts and the list of emails received are kept in memory. For both having only one unique list makes sense, there should not be copies which might cause inconsistencies.

Message are stored on disk, one file per message, in encrypted form. Use the Apache Commons IO library for file operations.

Keep a single file with the filename, sender, timestamp and title of all messages received (the index file.) Store this as a csv file, using the Apache Commons CSV library for this. This file is not encrypted!

Keep the contact information in another CSV file, also not encrypted.

Do not try to implement encryption from scratch! For this project, the Bouncy Castle libraries are to be used.

When starting the application, it should read the index file for the messages, then create a thread for each message to decrypt it. Make use of a Thread Pool to prevent the computer from being overloaded. Add a Boolean *decrypted* to the Message class, only showing the text in a message when it has been decrypted. For a better application structure, you can use the same Thread Pool to queue incoming messages to.

You will now have a number of threads running decryption. Whenever one is ready, have it put the results in the Collection which in the end is shown on the screen. Use the Observable Design Pattern (or rather the Observable Cascade) to update the GUI anytime a message has been finished.

## Intro to encryption

An introduction to encryption using the BouncyCastle libraries can be found in <https://www.baeldung.com/java-bouncy-castle> .

## Assignment

Create a JavaFX application with a project name containing at least *OOP3* and your name. Implement this design using JavaFX and the given libraries, following good programming and design practice. When finished export this as an Eclipse archive file, put the password *Welkom01* on it and email it to [harald.drillenburg@inholland.nl](mailto:harald.drillenburg@inholland.nl) .

Do not hand in a Maven project, Gradle project, IntelliJ archive or whatever format other than a plain JavaFX project with all necessary libraries in a lib folder; other formats will not be graded or given feedback on, and neither will a folder with Java files the teacher is supposed to create a project from. If it cannot be imported into Eclipse without a lot of effort it will not be looked at. You can use IntelliJ if you prefer, but in the end you hand in an Eclipse project (IntelliJ can export as Eclipse.)

You can do this project with one or two persons. When doing it with two, put a JavaDoc comment on top of each class and of each public method stating what it does and who created that part of the code.

The deadline for handing in this assignment is June 16th, 09:00. When the grade is not a passing one you get feedback by comments in the work you handed in. After that you have one week to improve your application and hand it in again. This also means it is best to hand in what you have, even if you think it will not be passing, as this will still give you valuable feedback.

## Grading

The following criteria determine the final grade:

* Not handed in or not implementing the assignment: 1,0
* Major architectural errors: 4,0. Examples of this are:
  + Not programmed using object orientation
  + No use of Design Patterns leading to unmaintainable code
  + Did not use external libraries where that was explicitly stated in the assignment
  + No or extremely bad use of multithreading
* Significant but not critical architectural issues but the general idea of implementation is correct: 5,5
* Application does implement the assignment and uses the Streams API, no significant architectural errors, but does contain non-trivial errors: 7,0
* Application does implement the assignment, uses the Streams API , no major architectural errors, no non-trivial errors: 9,0
* As with the previous bullet and also optimized for performance and maintainability: 10,0

## Dealing with customers

The customer creating this idea did (obviously) not program it before describing this. Things might be unclear, requirements can be vague or contradict themselves, a seemingly simple functionality might turn out to be much harder than imagined. Do not just assume it’s you who should understand better! Of course, first think about it yourself, but if that does not give clarity, ask the customer! In this case the customer sometimes also is a technical consultant.

Wait, did you read this paragraph before? That’s correct. It’s here again for a reason.