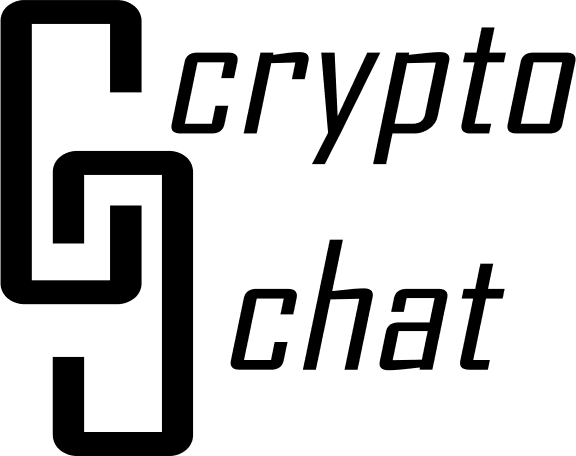
******

A Project by

Simon Hoyos Cadavid (2319104)

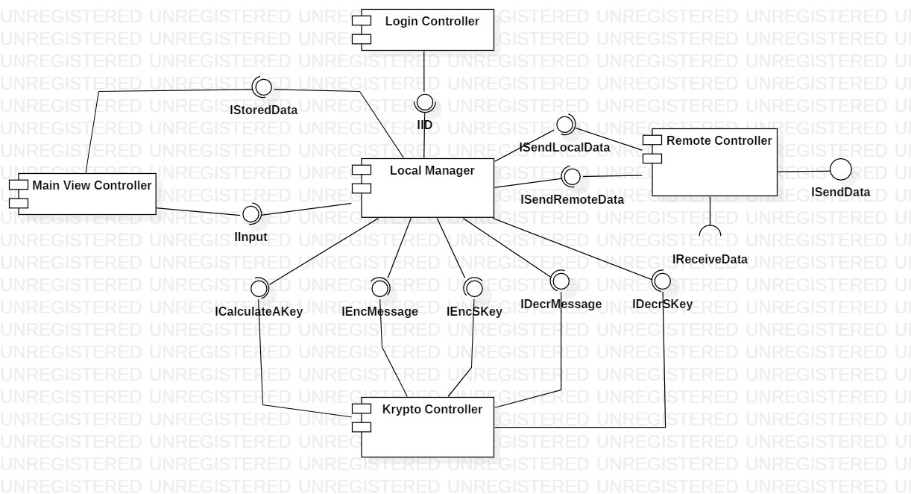
Max Bauer (2323129)

Matthias Pawlitzek (2315565)***Introduction:***

One of the greatest challenges in undergraduate life is trying to overcome the issues that come together with working as a team. During the university, it is most common for professors to make students create free topic projects in which they have to learn by themselves how to manage their time, organize and distribute work, and more importantly, trust in their teammates; because it is vital for them to learn that it is impossible to do everything by oneself in a working environment. Taking this into account, the Media Systems students of the University of Applied sciences in Hamburg, were tasked with developing a project that made usage of the different skills practiced in previous semesters. This report will shortly explain the making of CryptoChat: an end to end encrypted chat program, with emphasis in the teaching of cryptography, while at the same time showing the practical application of this same premise.

***Concept:***

The first clue for the development of CryptoChat was going to be born by the students’ curiosity towards cryptography. Not knowing what exactly they wanted to do, but at the same time craving to improve their knowledge in the fields of cryptography, they were set up to create something that was based on these principles. The aforementioned idea was, for the most part, the core of the whole project, and was always to be maintained as such, realizing the project around it. Because of this, the concept ended up being inspired by the modern-day chat software, which already utilize different types of encryption, but having on the counter side the big question of: What if you could decide your own encryption, while at the same time understanding what is happening behind the curtains?

In order to begin with the implementation, one of the first questions in a developer mind has to be the programming language in which a program is going to be written, but that was something that the students settled easily: the answer was Java. The decision for this was the ease of access and experience the developers had with this specific language, making it a clear cut in comparison with other languages like C++. Moreover, it was set that the program was not going to be a web application, which in turn got rid of options like JavaScript or HTML5. It is important to explain, that the creators wanted to use the least amount of external libraries as possible, meaning that everything they programmed was, for the most part, “*from scratch*”.

Looking at the component diagram to the right, it is possible to understand the initial planning for the program. It was supposed to have a local manager component which was going to be in charge of the supervision of all the data that came into a client, while at the same time distributing tasks for the rest of the local components to process. This local manager controller was also thought to be able to connect to a remote controller component, which would be obviously resting on the server, and was in turn there for the successful communication between clients. This aforementioned controller was then also made to be the universal storage for the public keys of each and every one of the connected clients, in order for other clients to have an ease of access to these when needing to encrypt a message.

1. Initial Component Diagram

Having the main components for the communication and management both in the servers and clients, only two big pieces of a functional program were missing: the encryption and the GUI. The latter was left to a Main View controller component, which as stated in its name, was going to be in charge of drawing the whole user interface. On the other hand, the Krypto Controller was made to include all of the cryptographic algorithms for the successful encryption of the messages. It is important to clarify, that these were going to be all programmed by the students themselves, meaning they were not going to be taken out of any preexisting library.

***Implementation:***

The final product of CryptoChat was for the most part a big success, in that it accomplished most of the original ideas specified in the concept. The program has a server which can be accessed from anywhere with an internet connection, allowing an easy communication via encrypted messages through these channels. Each time a new client is connected, the program will calculate the corresponding asymmetric keys; afterwards sharing them with the previously mentioned server, which stores them and in turn directs them to all the other clients connected at the time of the query being sent. Thereafter, one is welcomed to a main screen in which one can select with whom to chat with, from an array of all the other connected clients. This allows for the decision of which kind of, both symmetric and asymmetric, encryption is going to be used for the session – which can of course be changed at any time. The types of encryption supported by CryptoChat are the following:

***Asymmetric:***

* RSA.
* ElGamal (Diffie – Hellmann).

***Symmetric:***

* Affine cipher.
* Vigenère cipher.
* Hill cipher.
* Rivest Cipher 4 (RC4).
* Data Encryption Standard (DES).

As an important note, all of the steps needed for the encryption and decryption of data are always shown to the user in the log to the right of the window, which includes the public and private keys, as well as the encrypted message. Moreover, all of the algorithms are used in a hybrid way, meaning the symmetric key is asymmetrically encrypted, while the messages themselves are encrypted symmetrically with the aforementioned key. This practice is done because the asymmetric encryption algorithms are more demanding than the symmetric, therefore taking more time and computing power. It is also the closest to reality, where both symmetric and asymmetric algorithms are used in conjunction in order to create the best security, while maintaining a certain feeling of instantaneousness.

***How to use:***

The student’s initial thought was to make this application available to the world and thus public to the world wide web. Due to some limitations and security concerns on one of the students private Servers, we decided not to host the server-side implementation on our own. A user wanting to use the final program must follow two simple steps:

1. Run ServerMain.jar
2. Run ClientMain.jar

And that’s it! The user will probably want to run two instances of the ClientMain program, since we are talking about a messaging application. Therefore, everything happens locally on the user’s computer.