MATSE Ausbildung

Project - Step 1: Cybersecurity

1 General Description

In this first assignment, your task is to implement a secure messaging functionality, which will be the basis for later steps in the project.

2 Software Architecture

Your software architecture has to be a *client-server* architecture. The server will be used as a central component that is involved in every communication process and has to be able to handle an arbitrary number of clients.

3 Implementation Details

3.1 Clients

Each client models one *participant* in the communication network. A client is started using a configuration file in *JSON*-format, which looks as follows:

```
{
        "general": {
                "duration": "SECONDS",
                "retries": "NUMBER",
                "timeout": "SECONDS"
        },
        "person": {
                "id": "MY_UNIQE_ID",
                "name": "LASTNAME, FIRSTNAME(S)",
                "keys": {
                "private": "KEY_IN_BASE64_FORMAT",
                "public": "KEY_IN_BASE64-FORMAT"
                }
        },
        "server":
        {
                "ip": "SERVER_IP",
                "port": "PORT"
        },
        "actions": [
                "ACTION 1",
                "ACTION 2",
                "..."
                ]
}
```

Keep in mind that this file may be extended in later steps of the project! As a starting point, two examples for configuration files are provided.

After a client has been started, it registers at the server, providing its *id*, *first name*, *last name*, and *public key*. If the registration was successful, the server sends an acknowledgement. Otherwise, the server returns an error. After the server has acknowledged the registration, the client starts to carry out the defined actions. An action can be one of the following:

- `SEND [NAME] [MESSAGE]`
- `SEND [ID] [MESSAGE]`

Both actions send a message to the specified *name* or *id* via the server. A message is always encrypted using the public key of the recipient, which has to be retrieved from the server beforehand. The server acknowledges the successful transportation of the message or returns an error in case the message could not be delivered. In case of an error, a client retries multiple times to resend the message, as specified in *retries*. Between two retries a specified number of seconds is waited, as specified in *timeout*.

Additionally, a client has to be able to receive a message at any time during its runtime. If a message was received, it shall be printed to a logfile for the client. After the *duration* has expired, the client exits.

3.2 Server

The server has to provide the following methods:

- *Registration* with arguments *id*, *first name*, *last name*, and *public key*
- *Retrieve public key* with arguments *id* OR *first name* and *last name*
- *Send message* with arguments *id* and *encrypted message* OR *first name*, *last name* and *encrypted message*

Each method has to acknowledge the successful operation or return an error with a specific message.

3.3 Protocol

The design of the protocol is up to you! The only requirement is, that the described methods can be carried out.

4 Hints

- An *ID* has to be unique, but a *name* can occur multiple times.
- A private or public key in Base64-Format looks as described here: `https://www.cryptosys.net/pki/rsakeyformats.html`
- Any attribute, except for the *ID* shall be **not** case sensitive!
- This description is a little bit vague sometimes on purpose! Try to find working solutions if things are subject to interpretation!