SEtelegram

Project Documentation

Release: February 2021

Proprietary Notice

The present document offers information subject to the terms and conditions described hereinafter. The authors reserve the possibility to change the content and information described in this document and to update such information at any time, without notice. Despite the attention that has been taken in preparing this document, typographical errors or omissions may have occurred.

Authors

Matteo FORNERO *(Researcher, CINI Cybersecurity National Lab)* matteo.fornero@consorzio-cini.it
Nicoló MAUNERO *(PhD candidate, Politecnico di Torino)* nicolo.maunero@polito.it
Caterina OPPICI *(Student at Politecnico di Torino)*
Paolo PRINETTO *(Director, CINI Cybersecurity National Lab)* paolo.prinetto@polito.it
Gianluca ROASCIO *(PhD candidate, Politecnico di Torino)* gianluca.roascio@polito.it
Antonio VARRIALE *(Managing Director, Blu5 Labs Ltd)* av@blu5labs.eu

Trademarks

Words and logos marked with ® or ™ are registered trademarks or trademarks owned by Blu5 View Pte Ltd. Other brands and names mentioned herein may be the trademarks of their respective owners. No use of these may be made for any purpose whatsoever without the prior written authorization of the owner company.

Disclaimer

THIS DOCUMENT AND THE INFORMATION CONTAINED HEREIN IS PROVIDED ON AN “AS IS” BASIS AND ITS AUTHORS DISCLAIM ALL WARRANTIES, EXPRESS, OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY RIGHTS OR ANY IMPLIED WARRANTIES OR MERCHANTABILITY OR FITNESS FOR A PURPOSE. THE SOFTWARE IS PROVIDED TO YOU “AS IS” AND WE MAKE NO EXPRESS OR IMPLIED WARRANTIES WHATSOEVER WITH RESPECT TO ITS FUNCTIONALITY, OPERABILITY, OR USE, INCLUDING, WITHOUT LIMITATION, ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PURPOSE, OR INFRINGEMENT. WE EXPRESSLY DISCLAIM ANY LIABILITY WHATSOEVER FOR ANY DIRECT, INDIRECT, CONSEQUENTIAL, INCIDENTAL OR SPECIAL DAMAGES, INCLUDING, WITHOUT LIMITATION, LOSS REVENUES, LOST PROFITS, LOSSES RESULTING FROM BUSINESS INTERRUPTION OR LOSS OF DATA, REGARDLESS OF THE FORM OF ACTION OR LEGAL THEREUNDER WHICH THE LIABILITY MAY BE ASSERTED, EVEN IF ADVISED OF THE POSSIBILITY LIKELIHOOD OF SUCH DAMAGES.

Contents

1 Introduction

SEtelegram is a proof of concept of a secure instant messaging application implemented resorting to the open source TDlib[1] Telegram APIs and to the security capabilities of the **SE***cube*™ device.
Being a proof of concept, it is not the aim of this project to implement a daily usable application. The target, instead, is to show what can be achieved thanks to the **SE***cube*™ Open Source Security Platform. Because of this, SEtelegram lacks most of the features nowadays available in common messaging apps; moreover, no optimization has been carried out in terms of GUI features and overall performance.

2 Overview

SEtelegram uses the Telegram APIs to give you access to your Telegram contacts and exchange encrypted messages using a secure end-to-end communication channel.
The standard Telegram client already gives you the possibility to enable the end-to-end encryption, but what if you do not trust the encryption layer provided by the application? You need to provide your own encryption, and this is done in SEtelegram thanks to the **SE***cube*™ device.
The cryptographic keys used to encrypt the messages of SEtelegram are stored exclusively on the **SE***cube*™ devices belonging to the people who are communicating, they are managed by the **SE***key*™ Key Management System and they are used by the **SE***link*™ library for encrypting *data in motion*.

Important notice: SEtelegram is heavily based on other **SE***cube*™ libraries. We strongly suggest you to download and read carefully the entire documentation about the **SE***cube*™ and **SE***key*™ before going on with this project (the documentation is included with the source code of SEtelegram).

The fundamental block of information managed by SEtelegram is the chat, each chat is internally characterized by a specific ID. SEtelegram implements a manual mapping between chats and **SE***key*™ users or **SE***key*™ groups.
Let us suppose that Bob, a SEtelegram user, wants to encrypt all the messages that he will exchange with his Telegram contact named Alice. When starting SEtelegram, the application automatically checks whether an encryption rule for the chat between Bob and Alice has already been registered. If such rule exists, then it is loaded and applied. If such rule does not exist, the application asks to Bob to specify whether he wants to encrypt or not the chat with Alice. If Bob actually wants to encrypt the chat, then he must provide the **SE***key*™ user ID of Alice (or the **SE***key*™ group ID of a group of which he and Alice are both members) to SEtelegram. After this, SEtelegram registers the new rule and stores it in an encrypted database on the **SE***cube*™ .
Figure 1 and Figure 2 show how the GUI allows the user to specify whether a chat should be encrypted or not.
Notice that the GUI, being a simple proof of concept, does not implement any feature to show, given a chat, if that chat should be encrypted. These settings can be decided only once, when the GUI finds a new chat for the first time, as described by Figure 1 and Figure 2 .

[1] https://core.telegram.org/tdlib/docs/

Figure 1: Popup asking to the user whether he wants to encrypt or not the chat with Alice.

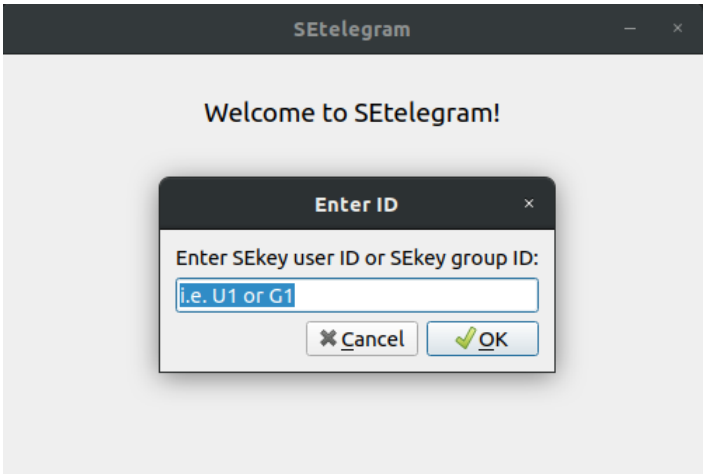

Figure 2: Entering **SE***key*™ user ID of Alice or **SE***key*™ group ID of a group shared with Alice.

3 Requirements

In order to use SEtelegram, you need:

- one **SE***cube*™ device for each person involved;
- one personal computer for each person involved;
- a preexisting **SE***key*™ Key Management System configuration.

If the **SE***key*™ requirement is not satisfied, it will be impossible to encrypt the messages exchanged with SEtelegram. Notice that SEtelegram embeds the same **SE***key*™ APIs that you should use to instantiate the **SE***key*™ KMS configuration. In order to understand how to create a valid **SE***key*™ configuration, please refer to the official **SE***key*™ documentation included with the source code of SEtelegram. The minimal configuration that you should create includes two **SE***key*™ users, one **SE***key*™ group including both users and one active key for that group.

In order to compile and run SEtelegram, you need:

- TDLib version 1.6.0[2]
- Qt5 framework[3]
- CLion IDE[4] (tested version 2020.3)
- Eclipse C++ IDE[5] (tested version 2020-12)
- Linux operating system (tested version Ubuntu 18.04.5 LTS, Kernel Linux 5.4.0-65-generic)

In order to install the TDLib APIs on your computer, please refer to the official documentation[6].

4 SEtelegram setup

Supposing that you have correctly installed all the required dependencies, follow these steps:

1. open the Eclipse IDE setting your workspace to `SEtelegram/SEcube/`;
2. open the **SE*key*™** user guide provided with SEtelegram;
3. setup **SE*key*™** as described in the user guide, **SE*key*™** is located at `SEtelegram/SEcube/SEcubeServer/sekey/`;
4. if you do not have a valid **SE*key*™** configuration, refer to the APIs of **SE*key*™** to create it;
5. compile the `SEcubeServer` project opened in the Eclipse workspace using the 'Release' mode;
6. close the Eclipse IDE and open the Clion IDE;
7. open the `SEtelegram` folder as a project;
8. open the file `CMakeLists.txt` and add the path of the TDLib APIs, for example: `set(Td_DIR "~/Downloads/td/tdlib/lib/cmake/Td");`
9. In **CLion - Settings - Build, Execution, Deployment - CMake - CMake options** add the path to the installation folders of Qt, for example: `–DCMAKE\_PREFIX\_PATH=/home/USERNAME/Qt/5.12.3/gcc\_64/lib/cmake`
10. To run and debug the project, create a new configuration, by clicking on **Add Configuration...**, selecting **CMake Application** from **Templates**, clicking on **Create configuration** and choosing `SEtelegram` for both **Target** and **Executable**, as shown in Figure 3;
11. open `global.cpp` and update the `SEcubeServer_path` variable with the path of the executable file you compiled at step 5, for example: `"~/Desk/SEtelegram/SEcube/SEcubeServer/Release/SEcubeServer.exe"`;
12. still in `global.cpp`, set the `errlog` and `telegramlog` variables with the paths where you want to write the log files (.txt format) of the application;
13. compile SEtelegram selecting **Build -> Build SEtelegram**.

[2] https://github.com/tdlib/td/tree/v1.6.0
[3] https://www.qt.io/
[4] https://www.jetbrains.com/clion/
[5] https://www.eclipse.org/downloads/packages/release/2020-12/r/eclipse-ide-cc-developers
[6] https://core.telegram.org/tdlib/docs/

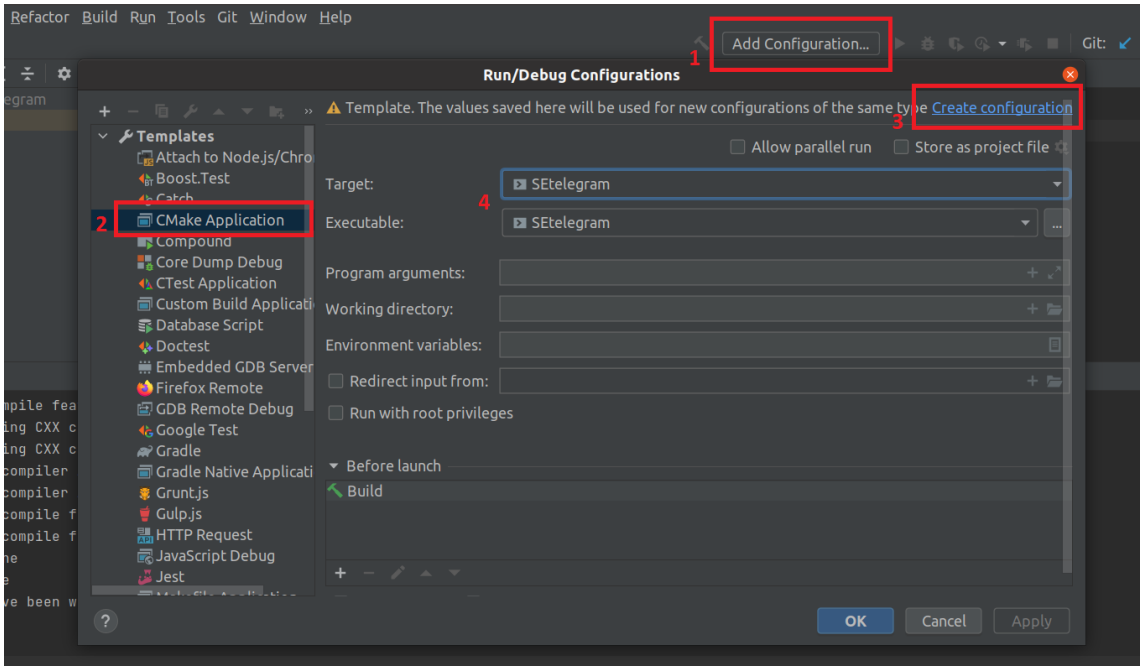

Figure 3: Configuring the project on CLion.

5 Running SEtelegram

Before running SEtelegram, connect your **SE***cube*™ to your PC. Remember that your **SE***cube*™ must have already been initialized and must be part of a valid **SE***key*™ configuration. SEtelegram will initially ask you for the PIN code to login on your **SE***cube*™ device, as shown in Figure 4. You must use your PIN for user privilege level (which, by the way, is the only PIN given to any **SE***key*™ user). After clicking the 'OK' button, the application will proceed with the login on the **SE***cube*™ and

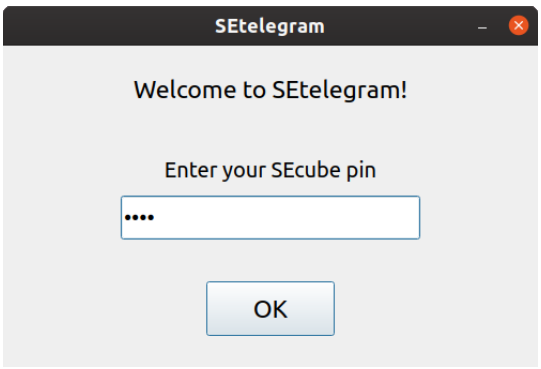

Figure 4: Window for inserting the **SE***cube*™ PIN.

with the boot of the **SE***key*™ system. This operation might take up to 30/40 seconds (especially if you are running the application on a virtual machine) because the boot of **SE***key*™ includes a full decryption of the encrypted **SE***key*™ metadata database in order to check for its integrity.
If no errors occur, a second window appears (Figure 5). Here you have to enter a password that will be used to encrypt the local database created by the TDLib APIs, containing data (i.e., your phone number) related to the authentication process of Telegram.
If you are launching SEtelegram for the first time, you have to establish your password right now.

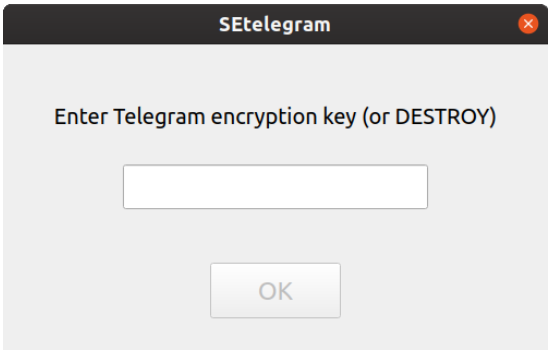

Figure 5: Window for inserting the TDLib database password.

If this is not the first time that you launch SEtelegram but you do not remember your password, enter the keyword 'DESTROY' to start over with a new authentication procedure.
After you enter your password for the first time or the 'DESTROY' keyword, the application also asks you for your phone number (with prefix) and an authorization code that you will receive by message on Telegram.
After clicking the 'OK' button, SEtelegram starts retrieving your Telegram chat list (the application is configured to retrieve up to 50 different conversations). This process can take a while (up to a few minutes) because, for each chat, SEtelegram will look into the **SE***cube*™ device searching for a valid encryption rule for that chat. If no rule is found, you will be asked to create one, as shown in Figure 1 and Figure 2.
Finally, when the whole setup process is completed, the main application window appears. It shows the list your Telegram chats on the left and the currently selected chat on the right (Figure 6 and Figure 7).

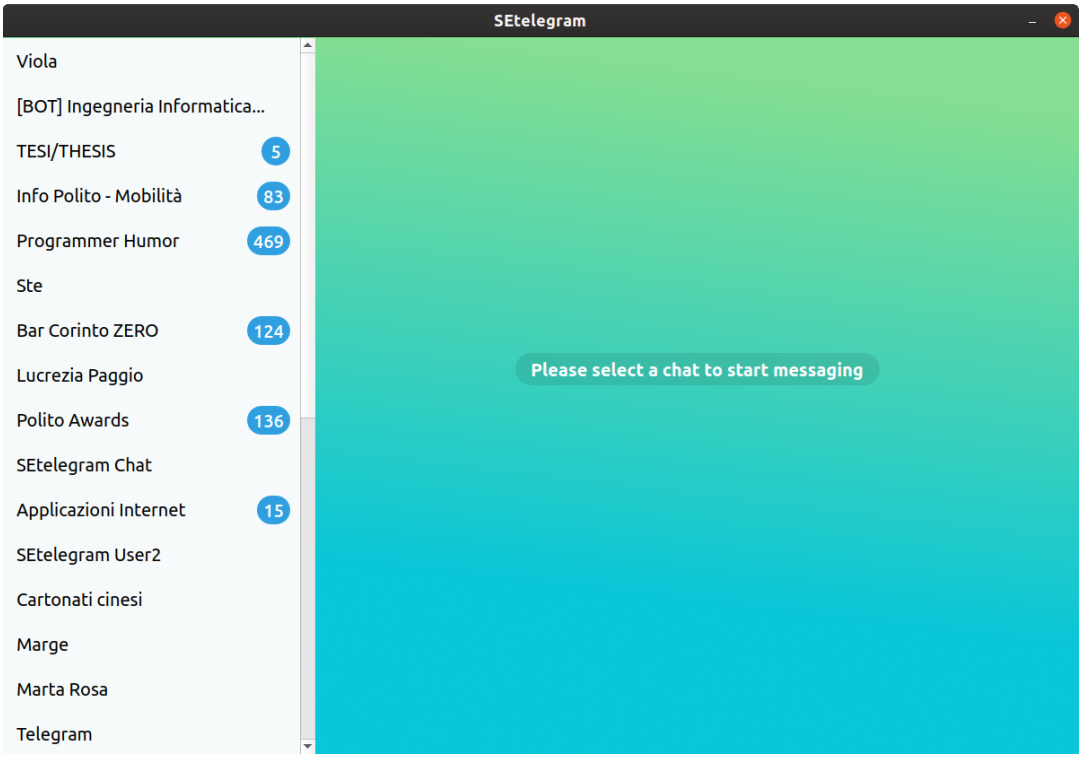

Figure 6: Main SEtelegram window.

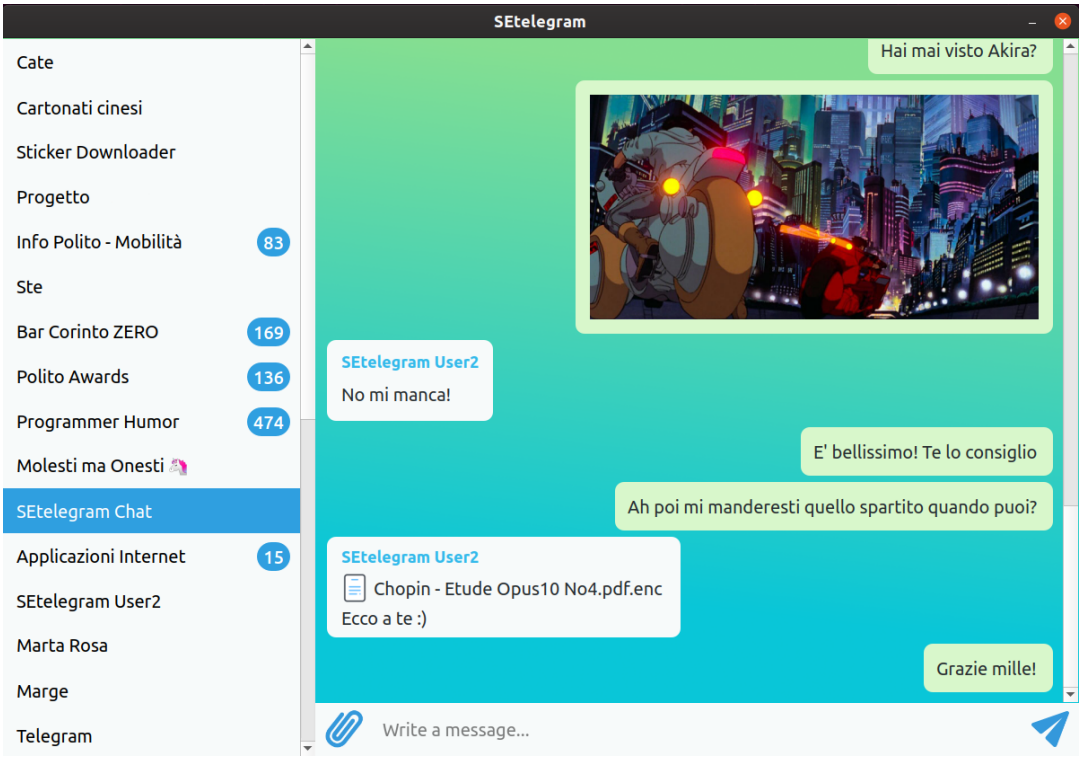

Figure 7: Messages of a chat.

In order to send a text message, you simply have to type the text inside the lower input bar, shown in Figure 8, then click the send button on the right.

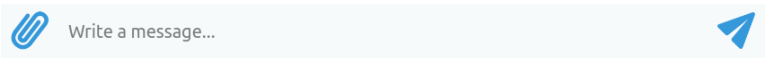

Figure 8: Input bar for sending messages.

For messages with attachments, instead, you have to click on the paperclip button on the left and choose a file from the popup window that appears. You can also enter a caption before sending the message, as shown in Figure 9.

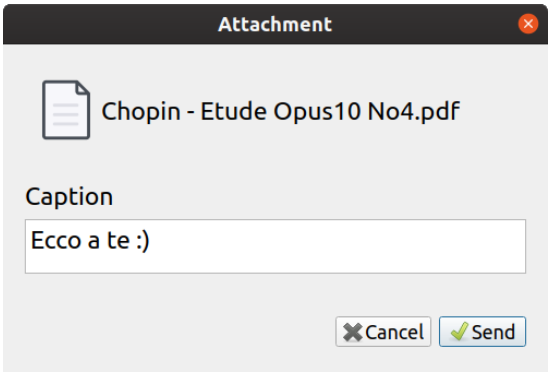

Figure 9: Dialog for entering a caption and sending the message.

Notice that the popup for selecting the attachments might appear a couple of times after sending the actual attachment, this is a known bug.

6 SEtelegram implementation details

SEtelegram is based on four different threads:

- **GUI thread**: the main thread; it manages user interactions with the application calling the respective handlers, functions that mainly ask the other threads for the data to be displayed on the interface.
- **SEcubeServer thread**: a thread that interacts with the **SE***cube*™ to obtain the list of chats whose messages must be decrypted when received and encrypted when sent, and to perform encryption operations on these messages.
- **Telegram thread**: a thread that takes care of receiving and sending messages from the various chats, interacting with the Telegram API infrastructure, called TDLib (Telegram Database Library).
- **Update thread**: a thread that every 30 seconds requests the Telegram thread to check if new messages have arrived and to update the unread message counters for each chat.

6.1 SEcubeServer

The communication between the **SE***cube*™ and SEtelegram follows a client/server pattern; the Qt application (in particular the GUI thread and the Telegram thread) is the client that sends requests to the server named `SEcubeServer`. The server interacts with the **SE***cube*™ , it receives requests coming from the Qt application and uses the APIs of the **SE***cube*™ SDK in order to satisfy the requests. In particular, the `SEcubeServer` uses the **SE***link*™ , **SE***file*™ and **SE***key*™ libraries for several purposes:

- keep track of which Telegram chats should be encrypted and which chats should not be encrypted;
- encrypt outgoing messages;
- decrypt incoming messages.

To keep track of the choices of the user about which chats have to be encrypted, specific metadata are stored in an encrypted SQLite database managed automatically by the `SEcubeServer`. The encrypted database provides security and reliability; however, it also implies a significant overhead in terms of performance. No optimization has been done in order to improve the performance of the `SEcubeServer`. A possible alternative, especially in case of a very limited amount of Telegram chats, it to use an encrypted `.csv` file managed by the **SE***file*™ library. This file would be much faster to read and write, implying a smaller overhead on the system.

6.2 Telegram thread

The Telegram thread uses the TDLib APIs to interact with Telegram servers; its entire code is enclosed in a public C++ class called `Td`. This thread uses a `while` loop for receiving requests coming from the GUI thread and from the Update thread, such as the login to Telegram, the Telegram chat list, sending a message or downloading a file. The class includes the following methods:

```
void waitForServerResponse()
```

Responses arriving from Telegram servers are received only in presence of an explicit request for reception by the application, performed by this method: it contains a cycle of update requests that stops only when the reply is delivered.

```
void send_query(td_api::object_ptr<td_api::Function> f, std::
   function<void(Object)> handler);
```

This method takes care of sending requests to the server and registering the handler function (passed to it as a parameter) which will be executed upon receipt of the response.

```
void process_response(td::Client::Response response);
```

This method manages the responses received, starting the execution of the relative handlers or calling the `process_update` method for updates.

```
void process_update(td_api::object_ptr<td_api::Object> update);
```

This method manages the different types of update that can be received by the Telegram servers, such as when a new message arrives.

```
auto create_authentication_query_handler();
void check_authentication_error(Object object);
void on_authorization_state_update();
```

These methods are used to manage the user authentication and authorization process to the Telegram service. The last one handles the updates, sent by Telegram servers when the user logs in, that indicate which stage of the authorization process he is at.
The Telegram thread and the GUI thread interact by exchanging data through a series of shared global variables, whose access is always protected through the use of multiple mutex and condition variables.

6.3 Update thread

The update thread emits a signal every 30 seconds, the signal triggers a request the Telegram thread to fetch the available updates, such as new messages received in the displayed chat and the unread message counters shown on the chat buttons.

6.4 GUI thread

The GUI thread manages user interactions with the application and deals with the visualization of chats and messages. Its main methods are:

```
void chatButtonClicked(long chat_id);
```

This method is called when the user selects a chat by clicking on its button, the application asks to the Telegram thread for the last messages exchanged in the selected chat.

```
void showMessage(td::tl::unique_ptr<td::td_api::message>&
   message, bool put_at_top);
```

Method invoked for each message received with the `chatButtonClicked` method. Depending on the message type (text, image or document), the application creates a different custom frame in order to show the message.

```
void showImages();
```

Method used to download the images contained in the received messages, if any.

```
void documentButtonClicked(std::int32_t file_id, const std::
   string& file_name);
```

Method invoked when the user clicks on a message containing a document; it asks the Telegram thread to download the file, as in `showImages`.

```
void newMessagesUpdate();
```

Method invoked every 30 seconds in response to the signal emitted by the Update thread. It asks the Telegram thread for updates, in order to show new incoming messages in the currently opened chat and to update the unread message counters.

```
void sendButtonClicked();
void attachmentButtonClicked();
void sendMessage();
```

The first two methods are used to prepare an outgoing message, respectively a text message and a message with an attachment. The last one is used to ask the Telegram thread to send the message.

```
int enc_or_dec(const std::string& e_or_d, const std::string&
   data_type,std::string& path_or_data, std::string& result);
```

This method is executed by the application thread to send an encryption or decryption request to the SEcubeServer. The request must contain the information related to the type of data on which the operation needs to be performed.

```
void waitTelegram();
void requestTelegram(std::string action_str);
```

These methods are used by the application main thread respectively to wait for the Telegram thread to be ready to receive requests and to actually send the request to it.

7 FAQ

- **SEtelegram does not show all my Telegram chats:** the TDLib library allows to retrieve a limited number of chats, this number is currently set to 50. You can try to change the value of the parameter DEFAULT_GETCHATS in `Td.cpp`.
- **SEtelegram frequently shows error, glitches and strange behaviors:** try to delete the `tdlib` folder contained in the directory where SEtelegram is compiled. Deleting this folder, you will have to do again the authentication process (insert password for the first time, insert phone number, insert authentication code).
- **Scrolling up in the GUI looking for old messages does not work:** the GUI retrieves old messages each time you scroll up to the top of the window. Retrieving old messages may take some time because attachments and pictures have to be downloaded. What you may notice is that the scrollbar is not re-positioned correctly when old messages appear on the screen; this is a known problem.
- **I cannot close the authentication window:** once you login on the **SE***cube*™ , you must authenticate to Telegram. There is no way to abort the authentication and close the program. You can solve this by killing the process manually.
- **SEtelegram seems to be stuck at some point, the GUI does not respond to commands:** these problems may occur because of synchronization issues between the threads of the program. Kill the process manually and restart SEtelegram.
- **SEtelegram shows messages whose content does not make any sense:** they may be encrypted messages, check if you have correctly configured the encryption settings for each chat and if you have a valid **SE***key*™ configuration.

- **SEtelegram is very slow when sending messages on encrypted chats:** this is normal because the message must be sent to the **SE***cube*™ through the USB cable, it must be encrypted, signed and sent back to the PC.