Second Pull Request on Telegram

Team Foobar: Yijia Zhang, Bingfei Zhang, Xiling Li

In Telegram, there is a chat mode called secret chat. The secret chat helps users protect their privacy. One function in secret chat mode is the self-destruct mode. In self-destruct mode, the user can set a timer and the message will be self destroyed when time out. In the current version of Telegram, the timer is started when the receiver reads the message. However, considering that the receiver may not read the message for a long time, it is not safe to store the message in the sender's phone. Thus, we planned to modify the logic of self-destruct, make the timer start after the message is sent in the sender's phone.

To achieve this goal, we need to find out where the timer is set, how the message is shown on the phone view, how to remove the timed out message from the view and how to remove the message stored in the database. Then we should modify these methods to make the message self-destruct after the sender sends it out, not after the receiver reads it. Some of the keywords that we were searching for included “Timer”, “Secret+Chat”, “Destroy” and “Unread”, as they were most likely to be included in activities that constructs the logic of how secret chat messages were destroyed. Digging deeper into the system, keywords like “Encrypted”, “secretMedia” and “SQLite” were added to our target list, as we found them in some ways related to the topic. While some components were easier to identify than others as their names are intuitive and thus searchable, some other methods and variables were hidden in extremely large files. An example of such would be the int variable “ttl”. It is crucial in the process of determining the life span of the secret message before it is destroyed. However, it is named counterintuitively.

Below are some examples where the keywords frequently appeared.

```
public static AlertDialog.Builder createTTLAlert(final Context context, final TLRPC.EncryptedChat encryptedChat) {
    AlertDialog.Builder builder = new AlertDialog.Builder(context);
    builder.setTitle("Self-Destruct Timer");
    final NumberPicker numberPicker = new NumberPicker(context);
    numberPicker.setMinValue(0);
    numberPicker.setMaxValue(20);
    if (encryptedChat.ttl > 0 && encryptedChat.ttl < 16) {...} else if (encryptedChat.ttl == 30) {
        numberPicker.setValue(16);
    } else if (encryptedChat.ttl == 60) {...} else if (encryptedChat.ttl == 60 * 60) {...} else if (encryptedChat.ttl == 60 * 60 * 2
        numberPicker.setValue(20);
    } else if (encryptedChat.ttl == 0) {...}
    numberPicker.setFormatter(value -> {...});
    builder.setView(numberPicker);
    builder.setNegativeButton("Done", (dialog, which) -> {
        int oldValue = encryptedChat.ttl;
        which = numberPicker.getValue();
        if (which >= 0 && which < 16) {
            encryptedChat.ttl = which;
        } else if (which == 16) {
            encryptedChat.ttl = 30;
        } else if (which == 17) {
            encryptedChat.ttl = 60;
        } else if (which == 18) {
            encryptedChat.ttl = 60 * 60;
        } else if (which == 19) {
            encryptedChat.ttl = 60 * 60 * 24;
        } else if (which == 20) {
            encryptedChat.ttl = 60 * 60 * 24 * 7;
        }
        if (oldValue != encryptedChat.ttl) {
            SecretChatHelper.getInstance(UserConfig.selectedAccount).sendTTLMessage(encryptedChat, resendMessage: null);
            MessagesStorage.getInstance(UserConfig.selectedAccount).updateEncryptedChatTTL(encryptedChat);
        }
    });
    return builder;
}
```

This piece of code in AlertsCreator.java sets the self-destruct time of the message. When the user select the self-destruct time, the attribute of encrytedChat.ttl is set to be the self-destruct time.

```
}
if (ttl != 0) {
    newMsg.ttl = ttl;
} else {
    newMsg.ttl = encryptedChat.ttl;
    if (newMsg.ttl != 0 && newMsg.media != null) {
        newMsg.media.ttl_seconds = newMsg.ttl;
        newMsg.media.flags |= 4;
    }
}
```

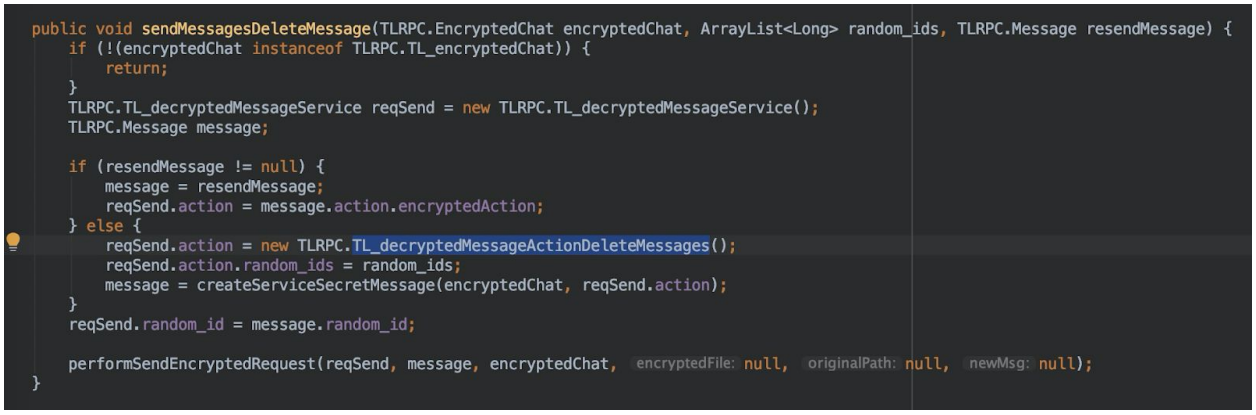
This method retrieves the self-destruct time form encrypted chat context and sets the self-destruct time of a message and then the message is sent out.

One of the most important parts that we were able to identify was the method “sendSecretMessageRead” located in ChatActivity.

```
private boolean sendSecretMessageRead(MessageObject messageObject) {
    if (messageObject == null || messageObject.isOut() || !messageObject.isSecretMedia() || messageObject.messageOwner.destroyTime != 0
        return false;
    }
    if (currentEncryptedChat != null) {
        getMessagesController().markMessageAsRead(dialog_id, messageObject.messageOwner.random_id, messageObject.messageOwner.ttl);
    } else {
        getMessagesController().markMessageAsRead(messageObject.getId(), ChatObject.isChannel(currentChat) ? currentChat.id : 0, inputCh
    }
    messageObject.messageOwner.destroyTime = messageObject.messageOwner.ttl + getConnectionsManager().getCurrentTime();
    return true;
}
```

This method checks if the Message is read by the reader and updates the message status in the controller. Then the application should start the timer and be ready to destroy the message.

Other highly probable methods include “SendMessageDeleteMessage” in the class SecretChatHelper.



```
public void sendMessagesDeleteMessage(TLRPC.EncryptedChat encryptedChat, ArrayList<Long> random_ids, TLRPC.Message resendMessage) {
    if (!(encryptedChat instanceof TLRPC.TL_encryptedChat)) {
        return;
    }
    TLRPC.TL_decryptedMessageService reqSend = new TLRPC.TL_decryptedMessageService();
    TLRPC.Message message;

    if (resendMessage != null) {
        message = resendMessage;
        reqSend.action = message.action.encryptedAction;
    } else {
        reqSend.action = new TLRPC.TL_decryptedMessageActionDeleteMessages();
        reqSend.action.random_ids = random_ids;
        message = createServiceSecretMessage(encryptedChat, reqSend.action);
    }
    reqSend.random_id = message.random_id;

    performSendEncryptedRequest(reqSend, message, encryptedChat,  encryptedFile: null,  originalPath: null,  newMsg: null);
}
```

However, the implementation of decryptedMessageActionDeleteMessages() in TLRPC has stopped us from being able to make further modifications to the code as we could not fully comprehend its mechanism.

Moreover, not only secret texts, but also media can be automatically destroyed, and Telegram has dedicated classes to deal with secret media. That further increases the complexity of our job.

Unfortunately we are not able to implement a functional feature by deadline. We tried numerous modifications and none of them were effective. However we were able to gain a much clearer picture of how secret chat systems worked during the process. We had to go through many files that we at first thought would not be relevant. Meanwhile, it also leads to more uncertainties regarding some implementations. Due to the lack of documentation, it was hard for us to understand the code method by method. Also since most of the modules were interrelated, it was hard for us to make simple changes to observe differences in behaviors. But the task still seems to be doable, and we will keep working on it until we finish it.