POLITECNICO DI TORINO

Corso di Laurea in Ingegneria Informatica

Tesi di Laurea

SEtelegram: Hardware-based Security for Messaging Applications

Relatore

prof. Paolo Prinetto

Candidato

Caterina Oppici

Dicembre 2020

Ringraziamenti

Giunta alla fine di questo percorso, vorrei dedicare questo spazio per ringraziare tutte le persone che hanno contribuito alla realizzazione di questo lavoro di tesi e a tutti coloro che mi sono stati vicini, supportandomi e accompagnandomi in questo periodo ricco di cambiamenti.

In primo luogo ringrazio il Professore Paolo Prinetto per l'opportunità offertami e Nicolò Maunero e Matteo Fornero per avermi guidato ed aiutato con pazienza nello sviluppo del progetto di tesi, oltre che nella stesura di questo elaborato.

Ringrazio la mia famiglia, in particolare i miei genitori, Fiore, Robert, i miei zii e nonna Gabriella, per avermi sempre incoraggiato e supportato in tutti questi anni, accompagnandomi nelle difficoltà incontrate.

Un grazie speciale a Giuseppe, Andrea, Dalila, Francesca, Fabrizio e Francesco per aver condiviso con me le gioie e le fatiche di questi anni, rendendo migliori anche i momenti più bui.

In particolare ringrazio Irma per avermi sempre fatto sentire a casa ed essere stata la migliore coinquilina che si possa desiderare.

Infine, un immenso grazie a Nico per essere sempre stato al mio fianco, nonostante tutte le difficoltà e la distanza, a Paola per avermi ascoltato e supportato da quando ho memoria, e a Ste per esserci sempre stato, anche da lontano. Senza di voi tutto questo non sarebbe stato possibile.

Indice

Elenco delle figure

Capitolo 1

Introduzione

Al giorno d'oggi, con il continuo sviluppo di internet e l'incessante diffondersi di supporti informatici in ambito lavorativo e privato, la protezione dei dati diventa sempre più fondamentale.

Buona parte del nostro vivere quotidiano si basa su dispositivi elettronici e sulla rete: ogni giorno tutto il mondo pubblica online informazioni sulla propria vita e intrattiene conversazioni private attraverso internet e applicazioni di messaggistica istantanea; gran parte delle imprese e delle aziende, inoltre, affida i propri dati a datacenter situati ovunque nel mondo ed interconnessi fra loro.

Ciò comporta una crescente presenza di informazioni sensibili memorizzate online e su sistemi informatici distribuiti, potenzialmente accessibili da utenti malevoli. Dato che tali dati costituiscono non solo informazioni private di singoli utenti, ma anche una parte importante del patrimonio di aziende e governi, risulta fondamentale proteggerne la memorizzazione e il trasferimento sui sistemi informatici attraverso tecniche e hardware specifici.

Negli ultimi anni, in risposta al crescente bisogno di mezzi per comunicare e condividere informazioni anche private e sensibili in modo sicuro ed affidabile, tutti i servizi di messaggistica più utilizzati e conosciuti, fra cui Telegram e WhatsApp, si sono dotati di sistemi di crittografia end-to-end, in modo da rendere intellegibili

i messaggi in transito sulla rete agli utenti esterni alla comunicazione.

Tali applicazioni presentano però un'importante problematica: attraverso la crittografia end-to-end le chiavi di cifratura vengono gestite in locale, quindi memorizzate direttamente sul disco dei dispositivi degli utenti finali. Ciò le rende soggette a possibili attacchi e fruibili da terze parti nel caso in cui il dispositivo fosse accessibile fisicamente o in remoto da un utente malevolo.

Inoltre in alcuni ambiti in cui la sicurezza diventa particolarmente cruciale, come nella gestione dei dati di grandi compagnie, si potrebbe aver necessità di un livello di robustezza addizionale nella crittografia dei dati, oltre ad avere la possibilità di scegliere gli algoritmi da utilizzare per ottenerla.

Lo scopo di questa tesi consiste nello sviluppo di SEtelegram, un'applicazione di messaggistica sicura che punti a colmare le mancanze delle piattaforme attualmente presenti sul mercato. Per farlo è stata utilizzata l'infrastruttura di API open source di Telegram e si è sfruttata la piattaforma chiamata SEcube™, un modulo di sicurezza hardware le cui API permettono di cifrare e decifrare dati nella sua memoria interna.

Usando il SEcube™ per cifrare i messaggi in uscita e decifrare quelli in entrata si ottiene un livello di sicurezza aggiuntivo rispetto a quello offerto dalle applicazioni di messaggistica mainstream già presenti sul mercato, e si sfrutta la maggiore velocità e resistenza agli attacchi dei sistemi di crittografia hardware. Il SEcube™ corrisponde a una black box a cui vengono forniti i dati da cifrare o decifrare e dal quale non è possibile ricavare altra informazione al di fuori dei dati in tal modo modificati: infatti le chiavi utilizzate nelle operazioni di codifica vengono gestite e custodite al suo interno dal Key Management System (KMS) del SEcube™, chiamato SEkey, e non memorizzate all'interno del device dell'utente. Risulta dunque impossibile ricevere in chiaro i messaggi di un'applicazione così integrata senza possedere il SEcube™ degli utenti coinvolti nella conversazione. Il SEcube™

è un dispositivo di crittografia a più fattori, poiché le chiavi utilizzate per cifrare i messaggi sono protette da una robusta autenticazione a due fattori: bisogna sia possedere fisicamente il modulo SEcube™, sia conoscere il pin per effettuare il login a tale dispositivo. Inoltre, anche i metadati relativi agli utenti e alle chiavi, utilizzati negli algoritmi di crittografia interni al SEcube™, sono crittografati.

L'interfaccia grafica di SEtelegram è stata implementata utilizzando il framework cross-platform Qt, ed è costituita da quattro diversi thread che interagiscono fra loro:

- **GUI thread**: il thread principale che si occupa di costruire e disegnare i diversi elementi grafici all'interno delle finestre dell'applicazione; gestisce l'interazione dell'utente con i widget che compongono l'interfaccia grafica (per esempio bottoni e campi di inserimento testo), chiamando di conseguenza i rispettivi handler, funzioni che principalmente richiedono agli altri thread dell'applicazione i dati da visualizzare sull'interfaccia.
- **Telegram thread**: il thread che si occupa della ricezione e dell'invio dei messaggi delle diverse chat, interagendo con l'infrastruttura di API di Telegram, chiamata TDLib (Telegram Database Library).
- **Update thread**: il thread che ogni minuto richiede al thread di Telegram di controllare se sono arrivati nuovi messaggi e di aggiornare i contatori dei messaggi non letti per ogni chat.
- **SEcube™ thread**: il thread che interagisce con il SEcube™ per ottenere l'elenco delle chat i cui messaggi devono essere decriptati in ricezione e criptati in invio e per effettuare le operazioni di cifratura sui messaggi.

L'applicazione gestisce sia le chat di Telegram comuni, cioè quelle che non prevedono un'ulteriore cifratura da parte del SEcube™, sia quelle che invece vengono

gestite da quest'ultimo. Con SEtelegram possono essere ricevuti e inviati messaggi di testo, immagini e documenti.

Nel capitolo 2 verranno brevemente descritte le principali applicazioni di messaggistica attualmente presenti sul mercato e il modo in cui esse gestiscono la cifratura dei messaggi.

Il capitolo 3 si occuperà di descrivere la piattaforma SEcube™ e la sua architettura interna, costituita da SEfile, SElink e SEkey.

Nel capitolo 4 verranno analizzate nel dettaglio la struttura e le caratteristiche dell'applicazione di messaggistica che è stata sviluppata grazie al SEcube™, SEtelegram.

Infine nel capitolo 5 verranno considerati gli sviluppi e i miglioramenti che in futuro potranno essere integrati a questa prima versione dell'applicazione.

Capitolo 2

Stato dell'Arte

Le applicazioni di messaggistica istantanea più utilizzate al giorno d'oggi, con alcune eccezioni, si affidano alla crittografia end-to-end (E2EE)[8] per proteggere i dati in transito degli utenti.

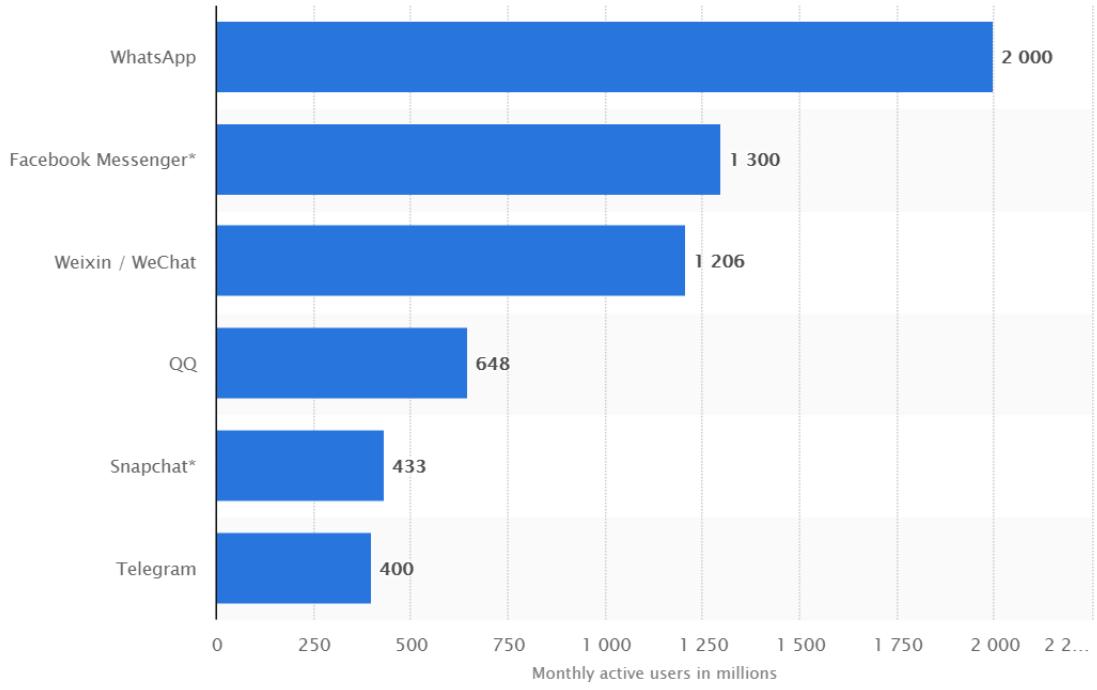

Figura 2.1. Le applicazioni di messaggistica più popolari al mondo ad Ottobre 2020 secondo, basato sul numero di utenti attivi mensilmente (in milioni) [2].

La crittografia end-to-end permette di proteggere i dati da possibili attacchi di tipo "man in the middle" (MITM)[10], in cui terze parti riescono a intercettare le informazioni in transito sulla rete per leggerle o alterarle. Solo il destinatario dei dati avrà in suo possesso la chiave per decifrare il messaggio ricevuto, che quindi rimane cifrato anche nel caso debba passare attraverso server non sicuri.

Figura 2.2. Le applicazioni di messaggistica più diffuse per paese/territorio a Gennaio 2019 [9].

I dati vengono cifrati attraverso la crittografia asimmetrica: questo tipo di codifica prevede l'uso, da parte di ogni utente, di una coppia di chiavi, una pubblica e una privata, generate in modo che risulti impossibile ricavare la chiave privata dalla corrispondente chiave pubblica. Per criptare i messaggi nelle applicazioni di messaggistica istantanea viene utilizzata la chiave pubblica del destinatario, che in seguito decripterà i dati ricevuti con la propria chiave privata.

Con questo tipo di crittografia i dati sono al sicuro da eventuali attacchi da terze parti mentre si trovano in transito nella rete, ma i dispositivi degli utenti rimangono comunque vulnerabili, dato che possono essere rubati o esposti a spyware e simili.

In questo capitolo verranno analizzate le principali applicazioni di messaggistica sicura diffuse attualmente, in particolare verrà descritto il modo in cui ognuna di esse gestisce la cifratura dei messaggi.

2.1 WhatsApp

Attualmente WhatsApp[1] è l'applicazione di messaggistica istantanea più utilizzata in Italia e nel mondo, con circa 2 miliardi di utenti connessi mensilmente e con maggiore diffusione nei paesi fuori dagli Stati Uniti [2], come mostrato nelle figure 2.1 e 2.2.

Creata nel 2009 e successivamente acquisita dal gruppo Facebook Inc. nel febbraio 2019, WhatsApp permette di mandare messaggi di testo ma anche immagini, documenti, messaggi vocali, di effettuare chiamate vocali e video, e di pubblicare storie. Come nella maggior parte delle applicazioni di messaggistica attuali, è possibile comunicare sia singolarmente con un altro utente sia in gruppi di più utenti.

Per criptare i messaggi, WhatsApp sfrutta il protocollo end-to-end Signal, sviluppato da Open Whisper Systems e utilizzato nell'omonima applicazione open source. Il processo prevede più fasi, in ognuna delle quali vengono utilizzate chiavi diverse [28]:

[1] https://www.whatsapp.com/

1. **Registrazione dell'utente**: quando il client di WhatsApp viene istallato, vengono generate alcune coppie di chiavi, fra cui le *Identity Keys*, che identificano l'utente e che successivamente verranno utilizzate per generare altre chiavi per criptare i messaggi; le chiavi pubbliche di ognuna di queste coppie vengono mandate al server di WhatsApp, che le memorizza associandole all'utente.

2. **Impostazione della sessione**: fra i client di WhatsApp deve essere stabilita una sessione criptata; per farlo, l'utente iniziatore della sessione genera una coppia di chiavi effimere; poi, attraverso l'algoritmo di Diffie-Hellman su curva ellittica (ECDH), calcola un valore, il *master secret*, a partire dalla sua *Identity Key* privata, dalla chiave effimera privata e dalle chiavi pubbliche dell'altro utente, che richiede al server; il *master secret* così calcolato viene utilizzato per creare una *Root Key*, a partire dalla quale viene generata una serie di *Chain Keys*.

3. **Ricezione della sessione**: a questo punto l'iniziatore manda già i primi messaggi al ricevente, e con essi include le informazioni per permettergli di creare una sessione corrispondente; il ricevente può così calcolare a sua volta il *master secret*, utilizzando le sue chiavi private e le chiavi pubbliche ricevute dall'iniziatore (cioè la *Identity Key* pubblica e la chiave effimera pubblica); anche il ricevente genera, a partire dal *master secret*, una *Root Key*, e da quest'ultima una serie di *Chain Keys*.

4. **Scambio di messaggi**: stabilita la sessione, quando un utente vuole mandare un messaggio genera ogni volta una nuova chiave effimera, chiamata *Message Key*, a partire dalla propria serie di *Chain Keys*; quest'ultima viene di volta in volta fatta "scattare in avanti" e aggiornata con la creazione di una nuova *Chain Key*, in modo che risulti impossibile ricostruire la *Message Key* corrente o una di quelle precedentemente utilizzate.

Per i messaggi contenenti media o altri allegati il processo è leggermente diverso: l'allegato viene criptato e caricato in uno spazio di archiviazione apposito; in seguito il ricevente lo potrà scaricare e decriptare usando le informazioni inviategli dal mittente attraverso un normale messaggio, criptato come precedentemente descritto.

Per quanto riguarda la comunicazione nei gruppi, WhatsApp prevede un procedimento iniziale da seguire la prima volta che si manda un messaggio in un determinato gruppo, basato sulle *Sender Keys* del protocollo Signal: il mittente genera una *Chain Key* e una coppia di *Signature Keys*; dalla combinazione della *Chain Key* e della *Signature Key* pubblica, crea un'ulteriore chiave, la *Sender Key*, che invia individualmente a ciascun membro del gruppo, secondo il procedimento usato per la comunicazione individuale fra utenti descritto precedentemente.

I messaggi successivi vengono criptati con la *Message Key* come descritto in precedenza e firmati con la *Signature Key* privata, poi inviati singolarmente al server, che si occupa di distribuirli a ogni membro del gruppo.

Le restanti features dell'applicazione, come le chiamate vocali, la condivisione dello stato e della posizione, utilizzano procedimenti per criptare i dati in transito simili a quelli già descritti.

Negli anni WhatsApp non è stata esente da critiche e perplessità da parte di studiosi e ricercatori, oltre ad aver subito numerosi attacchi.

Per esempio nel 2017 il ricercatore Tobias Boelter ha affermato che in WhatsApp esisteva un processo che creava una potenziale vulnerabilità [7]: i messaggi non ancora recapitati al destinatario potevano essere nuovamente criptati con una chiave diversa dalla precedente e rispediti, senza che né il mittente né il ricevente ricevessero notifica del cambiamento avvenuto. Questo processo avveniva in automatico quando le chiavi del destinatario venivano rigenerate, per esempio in seguito a una nuova installazione dell'applicazione, per evitare la perdita dei messaggi. Secondo Boelter, tuttavia, tale approccio rendeva le chat vulnerabili ad attacchi di tipo

"man-in-the-middle", quindi accessibili in chiaro da terze parti o da WhatsApp stesso. Successivamente WhatsApp ha introdotto la possibilità da parte di un utente di verificare le chiavi (quindi l'identità) dei propri interlocutori[28].

Nel 2019, a causa di un bug nella gestione delle chiamate WhatsApp (l'overflow di un buffer), era possibile installare lo spyware Pegasus nel dispositivo di un utente semplicemente chiamandolo: il malware aveva completo accesso al dispositivo, compreso lo schermo e la tastiera, e quindi alle conversazioni degli utenti in chiaro, rendendo la crittografia end-to-end dell'applicazione totalmente inutile[19][24][5].

2.2 Facebook Messenger

Facebook Messenger[2] è la seconda applicazione di messaggistica più utilizzata al mondo (Figura 2.1), diffusa soprattutto negli Stati Uniti, in Australia e Nord Africa (Figura 2.2). Inizialmente integrata in Facebook, è stata poi rilasciata come applicazione stand-alone, dal 2020 anche per desktop.

Similmente a WhatsApp, permette di mandare messaggi di testo ma anche immagini, documenti e altri media e di effettuare chiamate vocali e video.

Per quanto riguarda la protezione dei messaggi, invece, la crittografia end-to-end per ora non è attivata di default, ma può essere abilitata se si avvia una cosiddetta conversazione "segreta" [6].

Le conversazioni segrete vengono memorizzate lato device, a differenza di quelle normali (salvate lato server), quindi risultano accessibili solo dai dispositivi su cui l'utente le ha abilitate. I messaggi vengono protetti all'interno del dispositivo attraverso la crittografia simmetrica, in modo da non risultare accessibili mentre l'utente non sta utilizzando l'applicazione.

[2] https://www.messenger.com/

Dato che Messenger permette l'utilizzo di più account sullo stesso device, per mantenere la protezione dei dati di tutti gli account presenti vengono utilizzate due chiavi di crittografia:

- una locale con cui vengono criptati i messaggi memorizzati nel dispositivo;
- una remota, specifica per l'utente, che viene inviata dal server all'applicazione ogni volta che viene effettuata l'autenticazione, e con cui viene cifrata la chiave locale.

Ogni volta che un utente effettua il login sul dispositivo, la chiave remota corrente, cioè quella dell'utente che era precedentemente autenticato, viene eliminata; successivamente l'utente riceve dal server la chiave remota a lui legata, con cui decripta la chiave locale; quest'ultima viene infine utilizzata per decriptare i messaggi memorizzati in locale.

Messenger prevede l'invio di una notifica all'utente quando cambia la lista dei dispositivi su cui quest'ultimo ha attivato le conversazioni segrete, oltre che quando cambia quella di uno dei suoi interlocutori.

Come in WhatsApp, Facebook Inc. ha scelto di seguire il protocollo Signal per criptare i messaggi in transito anche in Messenger: il processo di crittografia end-to-end utilizzato è equivalente a quello descritto nel capitolo precedente.

Nel 2020 Facebook ha annunciato l'introduzione di una nuova feature di sicurezza in Messenger [20], chiamata "App Lock", un'opzione che permette di bloccare l'accesso all'applicazione a chiunque abbia fisicamente in possesso il dispositivo, utilizzando le impostazioni di privacy del device, come il riconoscimento facciale o tramite impronte digitali, e assicurando di non conservare o trasmettere in alcun modo i dati biometrici così ricevuti.

La principale critica attualmente mossa nei confronti di Facebook Messenger è relativa al mancato utilizzo della crittografia end-to-end come approccio di default nello scambio dei messaggi; inoltre tale opzione non risulta attivabile per le conversazioni fra più utenti, ma solo per quelle individuali.

2.3 Signal

Signal[3] è un'applicazione di messaggistica open source della Open Whisper Systems; il suo sviluppo tiene conto fin da subito della necessità di offrire agli utenti un metodo sicuro per scambiare messaggi, a differenza delle altre applicazioni, come WhatsApp e Messenger, che integrano man mano nel codice tali funzionalità, incontrando inevitabilmente molte difficoltà di progettazione. E' per questo motivo che il protocollo di Signal è considerato il migliore per la crittografia end-to-end e la relativa applicazione è apprezzata e utilizzata da molti esperti di sicurezza informatica, nonostante sia ancora relativamente poco diffusa.

WhatsApp utilizza lo stesso protocollo presente in Signal, ma quest'ultima offre alcune funzionalità che nella prima mancano, fra cui l'eliminazione automatica dei messaggi allo scadere di un timeout specifico (una protezione aggiuntiva nel caso in cui il dispositivo venga compromesso in qualche modo), e la possibilità di criptare in remoto tutti i dati legati al profilo dell'utente (come nome, immagine e numero di telefono), sempre attraverso lo stesso protocollo utilizzato per proteggere i messaggi [17]. Grazie a questo approccio, i server di Signal non hanno praticamente accesso in chiaro a nessun tipo di dato legato agli utenti, siano essi messaggi o informazioni sul profilo o sulla lista di contatti.

[3] https://signal.org/

Una delle limitazioni della crittografia end-to-end utilizzata dalle applicazioni precedentemente descritte consiste nel fatto che, nonostante il contenuto dei messaggi venga criptato, i relativi metadati rimangono non cifrati; le informazioni legate ai messaggi stessi, e quindi agli utenti, come la data e l'ora in cui un messaggio è stato inviato oppure i numeri di telefono di mittente e destinatario, in molte applicazioni risultano potenzialmente accessibili a terze parti. In Signal tale pericolo non sussiste, dato che semplicemente i metadati non sono conservati in alcun modo, ma vengono inviati anch'essi criptati insieme al messaggio [18].

Essendo un'applicazione open source, Signal è meno soggetta a possibili vulnerabilità nel codice sorgente, che vengono rilevate molto più velocemente rispetto a quanto accade con le applicazioni proprietarie; in tal modo gran parte dei bug viene trovata e risolta prima che un qualsiasi attacco la possa sfruttare.

La diffusione di Signal è incrementata notevolmente in Cina in agosto, in seguito all'ordine esecutivo del presidente degli Stati Uniti Donald Trump che vieta l'utilizzo dell'applicazione WeChat da parte dei cittadini e delle aziende americane, dato il pesante controllo che il governo cinese mantiene sull'uso di internet da parte dei cittadini e il mancato utilizzo della crittografia end-to-end nella popolare applicazione cinese [3].

2.4 WeChat

WeChat[4] è l'applicazione di messaggistica più diffusa in Cina, con più di un miliardo di utilizzatori, come mostrato nelle figure 2.1 e 2.2. In un articolo del 2016 Amnesty International ha collocato WeChat al fondo di una classifica delle migliori applicazioni di messaggistica in termini di sicurezza offerta agli utenti, assegnandole 0 punti su 100 [1].

WeChat non utilizza alcun tipo di crittografia end-to-end, si occupa solo di cifrare i messaggi in transito dai dispositivi degli utenti ai suoi server, su cui vengono decriptati: i dati risultano protetti solo da attacchi da terze parti mentre si trovano in rete, ma sono accessibili in chiaro dall'applicazione, nonostante la compagnia abbia dichiarato che tali dati vengano memorizzati solo temporaneamente, per poi essere eliminati una volta consegnati al destinatario. Diverso è invece il comportamento dell'applicazione nel momento in cui un utente aggiunge un messaggio nei "Preferiti": in tal caso esso viene conservato permanentemente nei server. WeChat offre, inoltre, funzionalità di traduzione e di localizzazione, utilizzando però servizi esterni, che hanno così completo accesso in chiaro ai messaggi e ai dati degli utenti [26] [27].

Le falle nella sicurezza appena descritte, insieme al fatto che il regime cinese applica una rigida sorveglianza e una pesante censura dei dati degli utenti in rete, porta alla facile conclusione che l'applicazione possa contenere backdoor o addirittura spyware per meglio monitorare le azioni degli utilizzatori e registrarne le informazioni [14]. Ovviamente le autorità cinesi hanno completo accesso ai dati degli utenti di WeChat, alle loro conversazioni private e alle ricerche effettuate durante l'uso dell'applicazione, informazioni che WeChat stessa dichiara di raccogliere [27].

[4] https://www.wechat.com/it/

2.5 Telegram

Telegram[5] è un'applicazione di messaggistica nata nel 2013, utilizzata da circa 400 milioni di utenti (Figura 2.1), diffusa soprattutto in Etiopia, Iran e Uzbekistan, come mostrato in Figura 2.2. La sua principale caratteristica è quella di essere basata su cloud: ciò permette agli utenti di accedere alle proprie conversazioni da più dispositivi, anche contemporaneamente, e di scambiare file di dimensione anche considerevole, fino a 1,5 GB [22].

In Telegram ci sono due possibili modalità di crittografia, entrambe basate sull'algoritmo MTProto (sviluppato da Telegram stesso): la crittografia client-server utilizzata di default, mostrata in Figura 2.3, e la crittografia end-to-end utilizzata nelle chat segrete, mostrata in Figura 2.4.

2.5.1 Chat basate su cloud

All'atto della registrazione viene generata, tramite il protocollo Diffie-Hellman, una chiave condivisa da server e client, la *Authorization Key*, una per ogni sessione generata da quell'utente su un dispositivo diverso [23].

Una parte della *Authorization Key*, insieme all'hash SHA-256 di parte del messaggio originale e del suo header interno, formano una seconda chiave, la *Message Key*, che viene a sua volta utilizzata per generare la chiave e il vettore di inizializzazione che servono per criptare il messaggio tramite AES-256. Il messaggio così criptato viene corredato da un header esterno, composto dalla *Message Key* e dall'*Authorization Key ID* (generato a partire dalla *Authorization Key* per indicare quale chiave è stata usata per cifrare il messaggio), e in seguito trasmesso al destinatario [23].

[5] https://telegram.org/

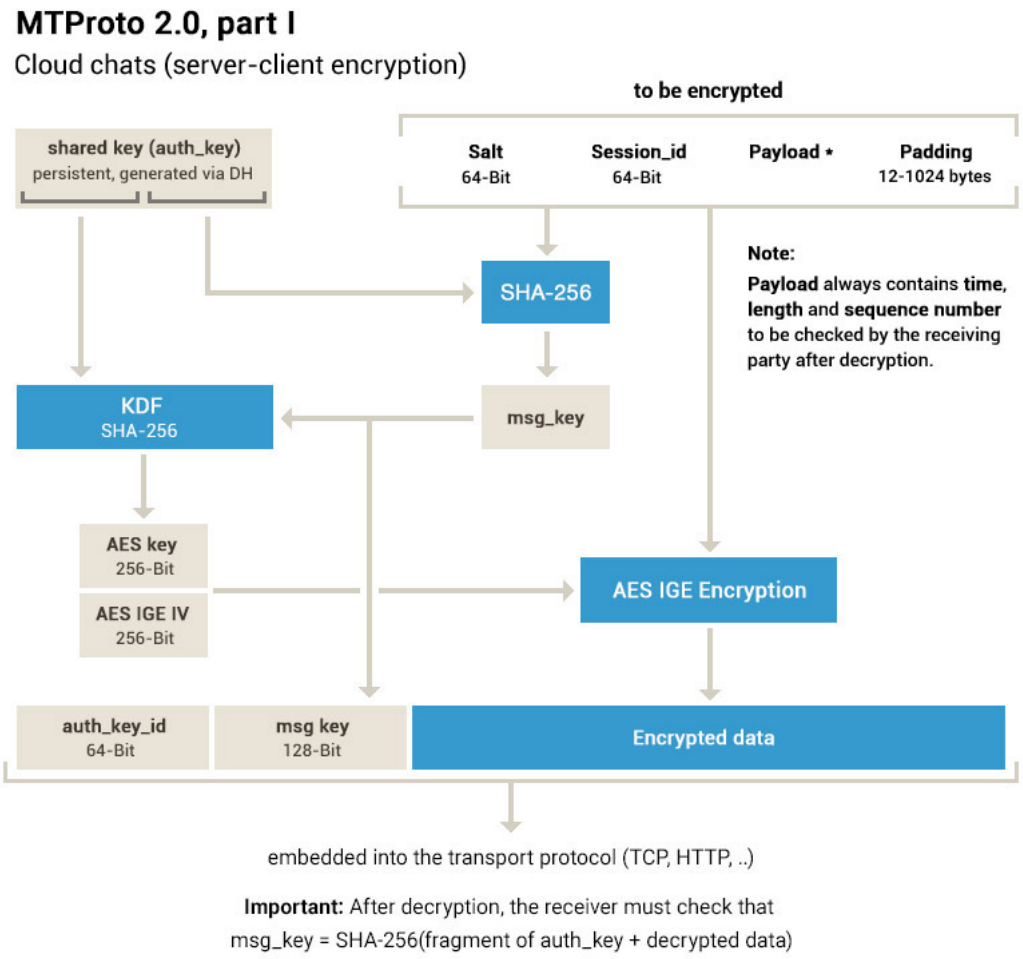

Figura 2.3. La modalità di MTProto utilizzata nelle chat su cloud di Telegram (crittografia client-server) [23].

Nell'header interno, che viene criptato insieme al messaggio, sono presenti informazioni di cui il destinatario dovrà verificare la correttezza, fra cui l'ID della sessione, l'ID del messaggio, il sequence number del messaggio (legato alla sessione) e il "sale" del server (un valore di 64 bit aggiornato periodicamente, usato per scongiurare alcuni tipi di attacco). È necessario controllare anche che la *Message Key* sia uguale all'hash SHA-256 calcolato sul messaggio decriptato insieme a un frammento di 32 bit della *Authorization Key* [23].

2.5.2 Chat segrete

Le conversazioni delle chat segrete non sono memorizzate su cloud, ma custodite nei dispositivi degli utenti, come in WhatsApp.

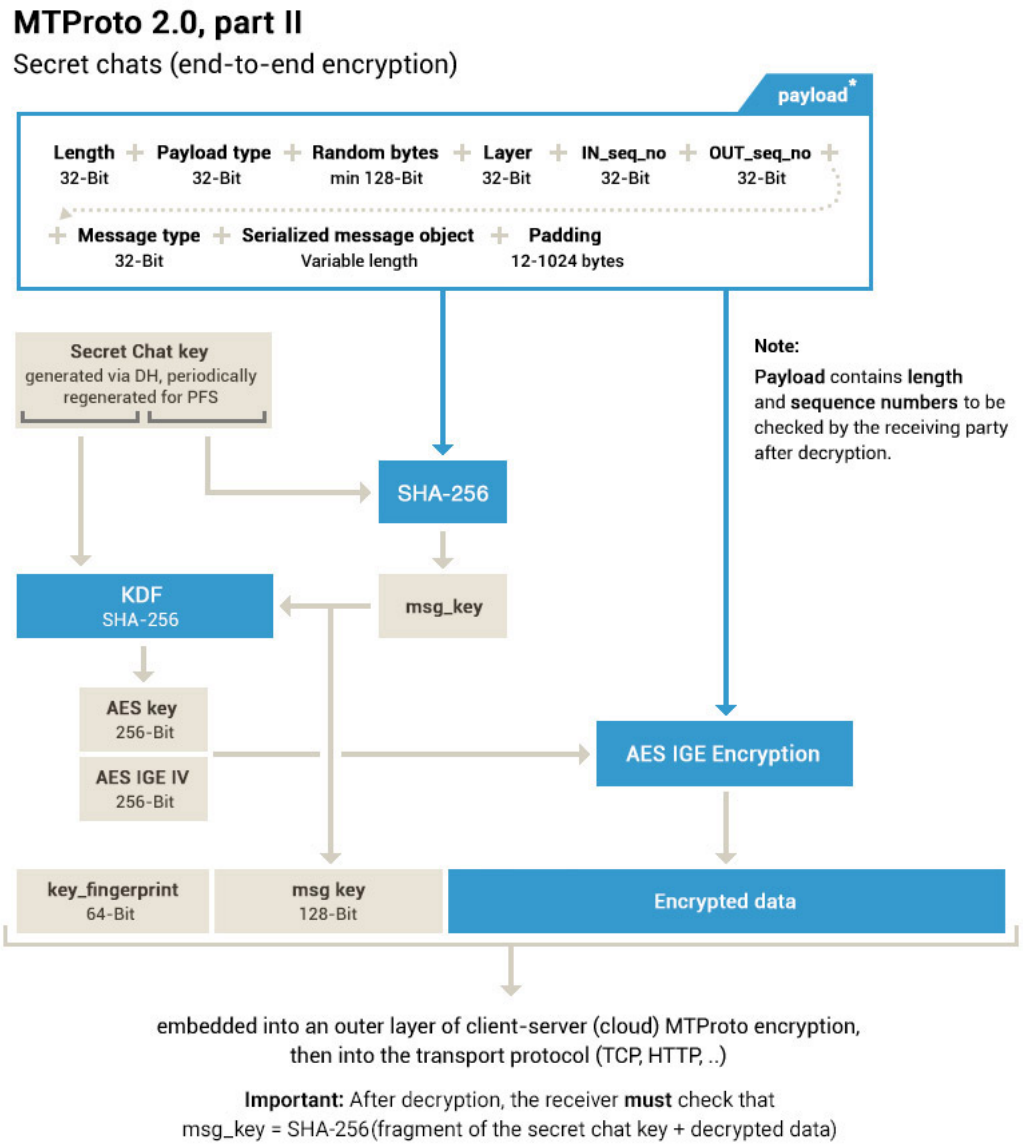

Figura 2.4. La modalità di MTProto utilizzata nelle chat segrete di Telegram (crittografia end-to-end) [21].

Alla creazione di una chat segreta i due utenti creano una chiave condivisa attraverso il protocollo Diffie-Hellman, la *Secret Chat Key*, da cui viene calcolata una *Key Fingerprint* attraverso SHA1, usata per controllare che lo scambio di chiavi sia avvenuto correttamente. La *Secret Chat Key* viene rigenerata periodicamente per ottenere la "perfect forward secrecy": se la chiave di crittografia creata a partire da questa venisse compromessa, non sarebbe comunque possibile risalire ai messaggi criptati con le chiavi generate precedentemente [21].

Un messaggio che deve essere inviato in una chat segreta viene serializzato e integrato con header e padding. Analogamente al procedimento seguito nelle chat su cloud, viene calcolata la *Message Key* a partire dallo SHA-256 dei 128 bit centrali dell'array di byte così ottenuto e da un frammento di 32 bit della *Secret Chat Key*. La *Message Key* e la *Secret Chat Key* vengono poi utilizzate per creare la chiave AES da 256 bit e il vettore di inizializzazione che servono per crittografare il messaggio. Al messaggio così cifrato viene aggiunto un header esterno, composto dalla *Key Fingerprint* calcolata in precedenza e dalla *Message Key*; l'array di byte risultante viene inviato al client destinatario.

Il destinatario effettua tutti i passaggi precedentemente descritti nell'ordine inverso, infine controlla la correttezza del messaggio ricevuto, verificando l'uguaglianza fra la *Message Key* e l'hash SHA-256 calcolato su un frammento della *Secret Chat Key* e sui 128 bit centrali del messaggio decriptato [21].

Per inviare un file in una chat segreta si trasmette un messaggio con un campo "file", contenente l'indirizzo del file criptato presente sul server, e con la chiave per decriptare tale file. La chiave usata per cifrare il documento è di tipo one-time, generata sempre insieme a un vettore di inizializzazione per l'algoritmo AES-256 [21].

2.5.3 Critiche e Vulnerabilità

Anche Telegram è stata soventemente oggetto di critiche. La perplessità maggiore da parte degli esperti riguarda il mancato uso della crittografia end-to-end come impostazione di default, come viene affermato in un articolo di Gizmodo del 2016 [25], che critica anche la scelta di sviluppare un proprio protocollo di crittografia, diversamente da quanto fatto da WhatsApp per esempio. L'anno successivo il portavoce di Telegram Markus Ra e il fondatore Pavel Durov hanno difeso in due articoli [16] [4] l'approccio usato dall'applicazione nella gestione della crittografia dei messaggi. A differenza di WhatsApp, che affida i backup delle conversazioni dei propri utenti a terze parti, cioè Google e Apple, crittografati con una chiave di cui comunque è in possesso, Telegram utilizza la propria infrastruttura di data-center, collocati in parti diverse del mondo: tale aspetto permette di memorizzare i dati degli utenti (ovviamente criptati) e le relative chiavi di crittografia in server appartenenti a giurisdizioni diverse, rendendo più complicato il processo necessario alle autorità per avere accesso a tali informazioni. La possibilità di offrire un servizio di backup delle conversazioni, inoltre, rende l'applicazione più appetibile per la maggior parte degli utenti (una delle cause della poca diffusione di applicazioni estremamente sicure come Signal è la mancanza di questa funzionalità).

Viene anche sottolineato il fatto che, sempre a differenza di altre applicazioni (si fa sempre riferimento a WhatsApp), il codice di Telegram è completamente open source, e con esso è disponibile una dettagliata documentazione e un'ampia descrizione del protocollo di crittografia MTProto, in modo che se ne possa verificare l'efficacia [16].

Nonostante tali rassicurazioni, anche Telegram ha spesso subito attacchi. È possibile, per esempio, sfruttare gli SMS utilizzati all'atto della registrazione di un nuovo dispositivo per accendere agli account degli utenti, vulnerabilità sfruttata nel 2016 da alcuni hacker iraniani (probabilmente legati al governo) [12], dalla polizia

tedesca per tracciare dei sospetti terroristi e probabilmente anche dalla società russa di telecomunicazioni MTS per controllare attivisti e dissidenti politici [9].

Durante le proteste ad Hong Kong nel 2019 molti manifestanti che avevano utilizzato gruppi di Telegram per organizzare le rivolte sono stati identificati dalla polizia attraverso i loro numeri di telefono (Telegram ha successivamente introdotto l'opzione che permette di nasconderli) [9].

Sempre nel 2019 i bot di Telegram, non protetti dal protocollo MTProto come le conversazioni normali, sono stati sfruttati da uno spyware in grado di accedere alle informazioni legate all'utente e di visualizzare l'intera conversazione su cui si trovava [15].

2.5.4 Diffusione

Negli anni Telegram ha spesso visto crescere la sua diffusione in territori protagonisti di rivolte oppure oppressi da regimi dittatoriali, come è capitato ad Hong Kong nel 2019 e in Bielorussia nel 2020. Risulta essere l'applicazione di messaggistica principale in Iran, accessibile tramite VPN, utilizzata per avere accesso a Internet aggirando il controllo governativo. La sua popolarità è aumentata anche in Brasile in seguito ai ban temporanei di WhatsApp (dopo che su quest'ultima era stata integrata la crittografia end-to-end) e in Corea del Sud quando il governo ha annunciato di voler monitorare le conversazioni di Kakao Talk [9].

Telegram è molto popolare anche in Etiopia e in Uzbekistan, dove è utilizzata anche dalle organizzazioni statali; ciò è dovuto probabilmente al fatto che, rispetto alle altre applicazioni di messaggistica più diffuse, il programma in sé occupa meno spazio sul dispositivo [9] e non ha bisogno di salvare le conversazioni e i media localmente, dato che con le chat su cloud i dati risultano salvati in remoto, permettendo, così, di risparmiare molta memoria su disco.

Capitolo 3

SEcube™ Open Security Platform

In questo capitolo verrà descritta la piattaforma SEcube™ (Secure Environment cube) [11], un ambiente di crittografia open source sviluppato nel 2015 dal gruppo Blu5[1] con l'aiuto di molte istituzioni accademiche, primo fra tutte il Politecnico di Torino.

L'approccio adottato dalla piattaforma è stratificato su più livelli: ogni livello di astrazione è basato su meno di 10 API, con lo scopo di nascondere la complessità ed eterogeneità del sistema di crittografia sottostante, in modo da permettere allo sviluppatore che lo utilizza di concentrarsi su aspetti più comuni e basilari legati alla sicurezza, come la gestione di gruppi e policy, piuttosto che doversi occupare di algoritmi e chiavi di crittografia.

Il vantaggio che si ottiene grazie a questa piattaforma è dato dal fatto che la crittografia basata su hardware, nonostante sia meno flessibile di quella software, offre una maggiore robustezza ad eventuali attacchi informatici, oltre ad essere più veloce.

[1] https://www.blu5group.com/

3.1 SEcube™ Hardware Security Module

SEcube™ è un "System-in-Package" composto da tre diversi elementi (Figura 3.1): un processore Cortex-M4 sviluppato dalla ST Microelectronics, una FPGA ad alte prestazioni e una SmartCard EAL5+ [11].

Figura 3.1. I tre componenti di SEcube™ [11].

SEcube™ è integrato in diversi dispositivi, fra cui una chiavetta USB, un telefono e una scheda PCIexpress. Per lo sviluppo di SEtelegram, l'applicazione oggetto di questa tesi, è stata utilizzata la scheda di sviluppo, chiamata *SEcube™ DevKit* (Figura 3.2); il dispositivo destinato a essere distribuito sul mercato per gli utenti finali dell'applicazione, invece, è la chiavetta USB, la *USEcube™ Stick* (Figura 3.3).

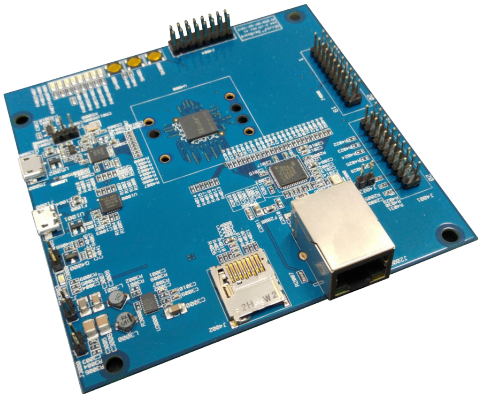

Figura 3.2. Il SEcube™ Development Kit.

Figura 3.3. La USEcube™ Stick.

Il DevKit è stato pensato per lo sviluppo di soluzioni hardware e software che prevedono l'uso del SEcube™; viene alimentato attraverso un connettore USB ed è possibile aggiornarne il firmware programmandolo attraverso un'interfaccia JTAG, collegata a sua volta al debugger ST-LINK/V2 mostrato in Figura 3.4.

Figura 3.4. Il debugger ST-LINK/V2 con i suoi connettori USB e JTAG.

Per quanto riguarda l'architettura software di SEcube™, le API sono organizzate in diversi livelli di astrazione: le funzionalità di ogni livello vengono sfruttate da quello sovrastante. Le API lato SEcube™ vengono eseguite dai core del dispositivo che integra il chip, quelle lato host, invece, dal dispositivo a cui il SEcube™ è connesso come periferica esterna; per queste ultime, quindi, è fondamentale la scalabilità e la portabilità su più sistemi operativi [11].

Le librerie lato device comprendono i driver richiesti per interagire con la scheda SD connessa al dispositivo e col connettore USB, oltre a una serie di funzioni che gestiscono l'inizializzazione del SEcube™, le chiavi di crittografia, le fasi di login e logout, le richieste e le risposte da e per l'host, la crittografia e i relativi algoritmi.

Le librerie lato host, invece, comprendono quattro livelli di API:

- **L0** include tre tipologie di API: quelle di tipo *Provisioning* si occupano dell'inizializzazione del SEcube™, quelle di tipo *Communication* gestiscono la comunicazione fra il SEcube™ e il dispositivo host, quelle di *Commodities* vengono utilizzate per trovare tutti i SEcube™ inizializzati che risultano connessi e per selezionarne uno;

- **L1** sfrutta le funzionalità del livello L0 per gestire le operazioni di login e logout, il contesto di crittografia (cioè gli algoritmi da utilizzare e i relativi parametri), la codifica, decodifica e calcolo del digest di un buffer di byte, la gestione delle chiavi;

- **L2** comprende le API che, sfruttando le funzionalità del livello L1, permettono di sviluppare applicazioni sicure senza dover conoscere in dettaglio i meccanismi di sicurezza sottostanti; tali API includono il SEfile (sezione 3.2) per la protezione dei dati statici, il SElink (sezione 3.3) per la protezione dei dati in transito e il SEkey (sezione 3.4) per la gestione delle chiavi e dei gruppi;

- **L3**, infine, sfrutta le API del livello L2 per lo sviluppo di applicazioni sicure, fra cui SEtelegram (capitolo 4).

Questa organizzazione stratificata delle API permette di avere un approccio olistico alla sicurezza: gli utenti e gli sviluppatori non hanno bisogno di sapere come sono implementati gli algoritmi di crittografia e in che modo vengono gestite le chiavi, né di preoccuparsi della complessità dei relativi concetti matematici e statistici; possono, invece, focalizzarsi su aspetti legati al servizio finale a cui deve essere applicata la sicurezza. Lo sviluppatore, quindi, si occupa solo di scegliere la libreria di API da utilizzare in base al tipo di dati che vuole proteggere (statici o in transito) e di gestire i gruppi di utenti e le relative policy di sicurezza.

3.2 SEfile

SEfile è una libreria di API utilizzata per proteggere i dati di tipo "at-rest", cioè statici, sfruttando principalmente le librerie del livello L1, che gestiscono il processo di crittografia e le chiavi a livello hardware; fornisce riservatezza, integrità ed autenticazione ai dati che elabora ed è portabile su diversi sistemi operativi. Grazie al SEfile è possibile immagazzinare ed accedere in modo sicuro ai dati, diversamente da quanto si può ottenere dal normale file system di un computer o da un cloud; questo perché, in un normale sistema operativo, eventuali falle presenti a livello applicazione possono essere sfruttate da malware o utenti malintenzionati per accedere al *secure layer* del sistema, cioè la parte del file system che si occupa di proteggere i dati e che è accessibile dal livello user attraverso il Virtual File System. Con l'utilizzo di un device come SEcube™ la protezione dei file è garantita anche nel caso in cui il dispositivo host venga compromesso, poiché quest'ultimo non ha alcun accesso ai dati che il device codifica [11].

Un file criptato utilizzando SEfile è composto da più blocchi criptati e firmati, come mostrato in Figura 3.5: il primo consiste di un header contenente i metadati del file stesso e di un padding (randomico, in modo da evitare gli attacchi di tipo "known plaintext"); tutti i blocchi successivi contengono i dati del file criptati.

Figura 3.5. La struttura di un file protetto da SEfile.

Tale struttura permette di velocizzare i tempi di codifica e decodifica, specialmente nel caso in cui sia stata modificata solo una parte limitata del file.

SEfile deve accedere al blocco del file richiesto dall'applicazione, mandare i dati da criptare o decriptare alle API di L1, infine memorizzare il risultato dell'operazione nella memoria RAM dell'host, in modo che l'applicazione possa accedervi. I blocchi contenenti i dati del file vengono criptati con AES-256-CTR ed autenticati con HMAC-SHA256.

3.3 SElink

SElink è la libreria di API della piattaforma SEcube™ utilizzata per proteggere i dati di tipo "in-motion", cioè in transito sulla rete, sfruttando le librerie del livello L1, che gestiscono il processo di crittografia a livello hardware. SElink permette di creare un canale di comunicazione criptato fra due dispositivi connessi ai rispettivi SEcube™, creando un sistema di protezione simile a una VPN e ai meccanismi di crittografia end-to-end utilizzati, per esempio, dalle applicazioni di messaggistica descritte nel capitolo 2.

SElink garantisce la riservatezza, l'integrità e l'autenticazione dei dati da esso manipolati; il suo punto forte è la semplicità di utilizzo e il numero ristretto delle sue API, che sono solo cinque [11]:

- **`selink_encrypt_manual`** e **`selink_encrypt_auto`**: API per la codifica dei dati; la prima prevede che il chiamante le fornisca manualmente l'ID di una chiave di crittografia contenuta all'interno del SEcube™, mentre la seconda richiede solo una lista di riceventi (singoli utenti o gruppi) appartenenti al sistema SEkey, in modo che il SElink possa sfruttare quest'ultimo per ricavare la chiave più sicura da utilizzare. Il chiamante deve passare per parametro anche un vettore di byte contenente i dati da cifrare e la sua lunghezza. I dati vengono criptati e memorizzati all'interno di un oggetto di tipo

`selink_ciphertext` insieme ai parametri di crittografia.

- **`serialize`**: metodo della classe `selink_ciphertext` per serializzare i dati in esso contenuti (cioè un testo criptato e i parametri di crittografia utilizzati) in un vettore di byte in base64, in modo da poterli poi inviare al destinatario attraverso, per esempio, un socket TCP. Grazie a questa funzionalità, SElink è completamente indipendente dal protocollo utilizzato per trasmettere i dati da un dispositivo a un altro.

- **`deserialize`**: metodo della classe `selink_ciphertext` per deserializzare un array di byte in base64 e copiare i dati così ottenuti (cioè un testo criptato e i relativi parametri di crittografia) all'interno dell'oggetto `selink_ciphertext` su cui è stato chiamato.

- **`selink_decrypt`**: API per decriptare i dati precedentemente codificati da `selink_encrypt_manual` o da `selink_encrypt_auto`. Il testo cifrato gli viene passato per parametro all'interno di un oggetto `selink_ciphertext` creato dal metodo `deserialize`, insieme ai parametri di crittografia, attraverso i quali può decriptare i dati e verificarne la firma, garantendone così l'integrità e l'autenticazione. Ritorna un errore nel caso in cui la decodifica non vada a buon fine o le firme non coincidano, altrimenti copia i dati decriptati all'interno del vettore di byte passatogli per reference.

Le applicazioni che vogliono sfruttare le funzionalità offerte dal SElink e criptare i propri canali di comunicazione necessitano di essere modificate per utilizzare le API di questa libreria, poiché essa non intercetta automaticamente il traffico in uscita o in entrata dal dispositivo. Nonostante ciò, gli aspetti di basso livello riguardanti il processo di crittografia, come il calcolo dei padding e la verifica delle firme, rimangono completamente trasparenti a livello applicazione, poiché vengono gestiti automaticamente da SElink, sfruttando le librerie del SEcube™ sottostanti L0 e L1, descritte nella sezione 3.1.

Lo scenario di utilizzo di SElink, descritto in Figura 3.6, prevede che ogni interlocutore abbia accesso al proprio SEcube™, sia possibilmente registrato nel sistema di gestione delle chiavi SEkey (per aumentare la protezione e l'automazione del processo) e utilizzi un'applicazione di messaggistica customizzata per SElink.

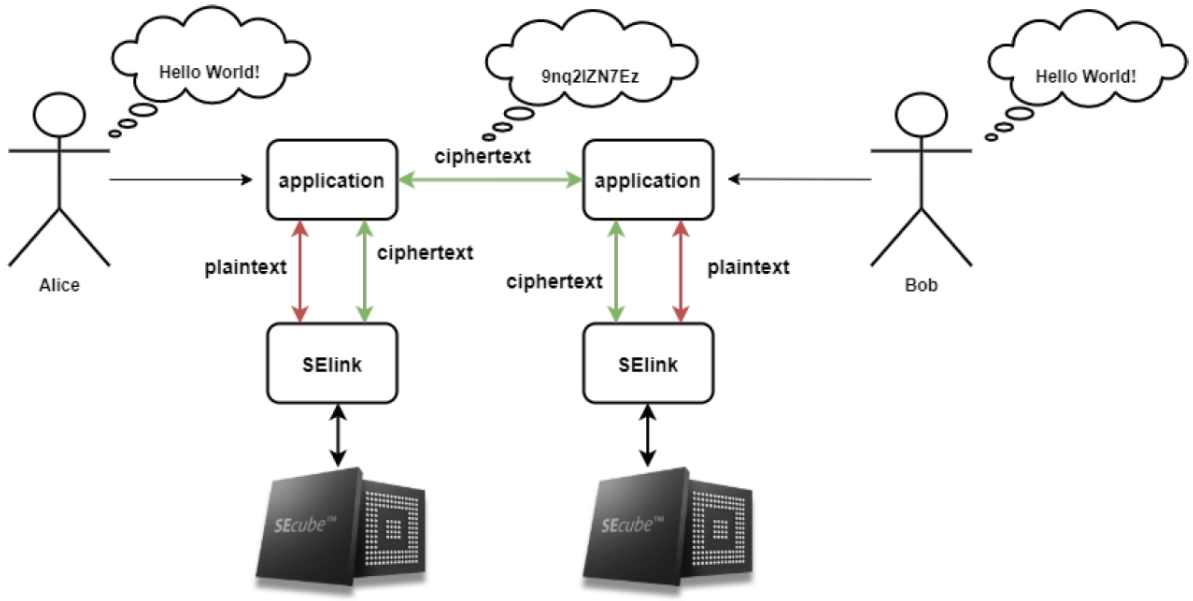

Figura 3.6. Scenario di utilizzo di SElink [11].

L'applicazione di messaggistica, prima di inviare un messaggio in rete, chiama `selink_encrypt_manual` o `selink_encrypt_auto` per criptarlo, scegliendo attraverso SEkey, se possibile, la chiave più sicura da utilizzare; poi serializza con il metodo `serialize` i dati criptati e li trasmette al destinatario.

Quando invece l'applicazione riceve un messaggio, prima lo deserializza utilizzando il metodo `deserialize`, poi chiama `selink_decrypt` per decriptarlo e verificarne la firma, in modo da controllarne l'integrità e l'autenticazione; a questo punto, se non si sono verificati errori, può mostrarne il contenuto all'utente.

3.4 SEkey

SEkey è l'insieme delle API che costituiscono il Key Management System (KMS) delle applicazioni basate sul SEcube™. Il compito di un KMS consiste nella creazione, nell'uso, nella conservazione e nell'eliminazione delle chiavi di crittografia, ma soprattutto nella loro stessa protezione, poiché una chiave non adeguatamente custodita risulta inutile [11]. SEkey permette agli utenti possessori di un SEcube™ e appartenenti a un sistema comune di scambiare informazioni in modo sicuro.

Le chiavi gestite da SEkey sono a lungo termine e simmetriche (quindi utilizzate sia in fase di codifica sia in fase di decodifica); vengono descritte da alcuni attributi, fra cui il nome, l'ID, l'ID del loro proprietario (cioè un gruppo di utenti), lo stato del loro ciclo di vita, l'algoritmo di crittografia in cui vengono utilizzate, lunghezza, le date relative al loro ciclo di vita e il loro "cryptoperiod", cioè il periodo di tempo in cui possono essere utilizzate. Tale periodo varia a seconda del tipo di chiave: la *Master Key*, specifica per ogni SEcube™, viene utilizzata solo per gli aggiornamenti, quindi può avere un periodo di attività di qualche anno, così come quelle personali del dispositivo, usate solo in locale per criptare le informazioni all'interno del SEcube™; invece le chiavi che proteggono i dati scambiati dai membri di un gruppo vengono utilizzate molto spesso e son condivise da più utenti, quindi necessitano di essere cambiate nel giro di pochi giorni o settimane.

Durante il suo ciclo di vita, una chiave di crittografia passa attraverso diversi stati, descritti in Figura 3.7.

Il primo stato in cui la chiave entra appena viene generata è quello chiamato *Pre-Active*, in cui non è ancora utilizzabile per criptare o decriptare i dati; in alcuni sistemi questo stadio viene saltato e la chiave passa direttamente a quello successivo, in cui è *Active*, cioè utilizzabile nei processi di crittografia.

Una chiave attiva può venire temporaneamente sospesa per non permettere il suo utilizzo nella codifica di nuovi dati; essa però può ancora decriptare i dati che

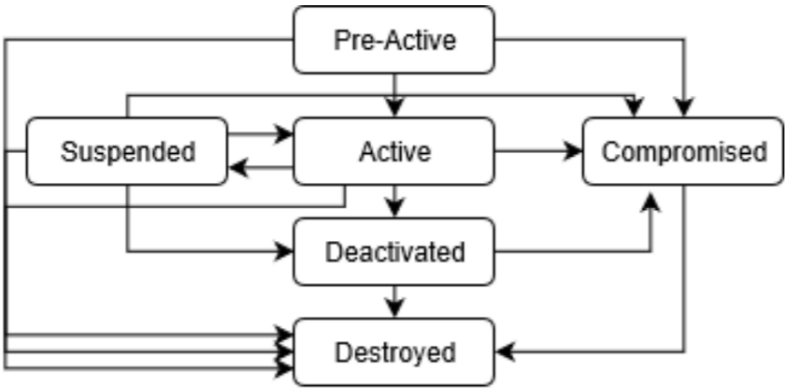

Figura 3.7. Diagramma di stato di una chiave secondo le raccomandazioni NIST per la gestione delle chiavi [13].

aveva criptato prima della sua sospensione, e può ancora essere attivata, a differenza di una chiave disattivata, che potrà solo decriptare i dati e mai tornare attiva.

Infine esistono due stadi a cui la chiave può arrivare in qualsiasi momento del suo ciclo di vita, a partire da qualsiasi stadio, cioè lo stato *Compromised* e lo stato *Destroyed*: una chiave può essere compromessa in un qualsiasi momento e perdere la propria segretezza o integrità, quindi non deve più essere utilizzata per criptare i dati (tranne che in specifici casi in cui l'utente ha un controllo elevato del sistema ed è consapevole dei possibili rischi); una chiave, inoltre, può essere eliminata in qualsiasi momento se non viene più considerata necessaria.

Gli utenti possessori di un SEcube™ e appartenenti a uno stesso gruppo utilizzano una chiave simmetrica comune per proteggere i dati che condividono fra loro. Due utenti, quindi, hanno bisogno di essere parte di almeno un gruppo in comune per poter comunicare in modo sicuro; nel caso condividano più gruppi, il SEcube™ sceglie automaticamente la chiave appartenente al gruppo meno numeroso.

A ogni gruppo sono associate delle policy di sicurezza, cioè dei parametri, fra cui il massimo numero di chiavi a cui il gruppo può essere associato, il loro "cryptoperiod" predefinito e l'algoritmo di crittografia utilizzato.

L'architettura di SEkey prevede un amministratore di sistema, che, attraverso il proprio SEcube™, gestisce la creazione, la modifica e l'eliminazione dei gruppi, delle relative chiavi di crittografia e degli utenti; l'amministratore si occupa anche di aggiornare lo stato delle chiavi e il loro ciclo di vita.

Il SEkey ha un sistema distribuito su più SEcube™, come mostrato in Figura 3.8: il SEcube™ dell'amministratore copia i metadati relativi a utenti, gruppi e chiavi in una posizione predefinita, in modo che i SEcube™ degli utenti vi possano accedere per recuperare eventuali aggiornamenti riguardanti la loro configurazione.

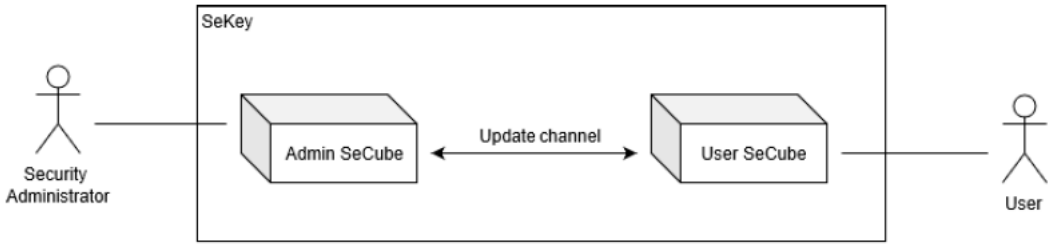

Figura 3.8. L'architettura di SEkey [11].

L'amministratore ha accesso alle informazioni di tutto il sistema di SEcube™, mentre fornisce ai dispositivi degli utenti solo i dati e gli aggiornamenti riguardanti loro stessi e i gruppi di cui fanno parte.

I metadati del sistema SEkey vengono conservati nella memoria SD del SEcube™, all'interno di un database SQL criptato con SEfile (sezione 3.2).

Capitolo 4

SEtelegram

Le funzionalità e i vantaggi offerti dalla piattaforma SEcube™, descritti nel capitolo 3, insieme al sempre crescente bisogno di ovviare alle mancanze nella gestione della sicurezza presenti nelle attuali applicazioni di messaggistica, hanno portato alla creazione di SEtelegram (la sua finestra principale è mostrata in Figura 4.1).

SEtelegram è stata sviluppata su Linux in C++, utilizzando Qt, la libreria cross-splatform per la creazione di programmi con interfaccia grafica[1], e l'infrastruttura di API open source di Telegram, chiamata TDLib (Telegram Database Library)[2].

Inizialmente si è tentato di sfruttare il codice sorgente di Telegram disponibile online[3], in modo da integrare le funzionalità di sicurezza del SEcube™ direttamente nell'applicazione originale, ma tale codice è risultato difficile sia da compilare sia da modificare: la procedura per effettuare la build del progetto presenta molte difficoltà ed è strettamente dipendete dal sistema operativo, anche perché viene richiesta l'installazione a parte di molte librerie di terze parti; il codice stesso, inoltre, è estremamente complesso, praticamente privo di commenti, molto dispersivo

[1] https://www.qt.io/

[2] https://core.telegram.org/tdlib

[3] https://github.com/telegramdesktop/tdesktop

e poco documentato. Al contrario le API di TDLib sono crossplatform, ben documentate[4] e semplici da utilizzare, oltre ad essere disponibili per diversi linguaggi di programmazione.

In questo capitolo SEtelegram verrà esaminata nel dettaglio: nella sezione 4.1 verrà analizzata la struttura generale del suo codice, nella sezione 4.2, invece, saranno mostrate le funzionalità offerte dall'applicazione e verrà descritto il modo in cui sono state implementate.

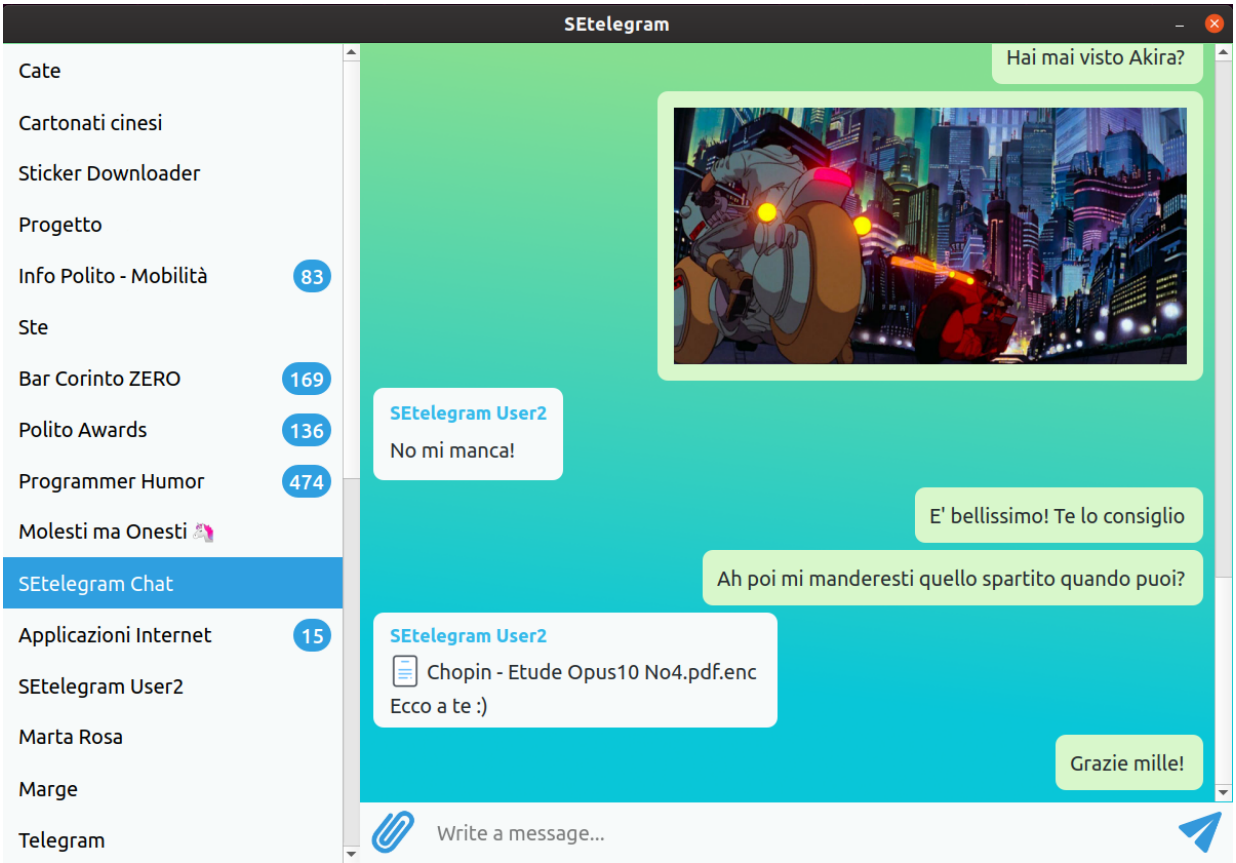

Figura 4.1. Esempio di una chat di SEtelegram.

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

4.1 Struttura del Codice

Il codice di SEtelegram è basato su quattro diversi thread sincronizzati fra loro attraverso l'uso di `mutex` e `condition_variable`:

- **GUI thread**: il thread principale che si occupa di costruire e disegnare i diversi elementi grafici all'interno delle finestre dell'applicazione; gestisce l'interazione dell'utente con i widget che compongono l'interfaccia grafica (per esempio bottoni e campi di inserimento testo), chiamando di conseguenza i rispettivi handler, funzioni che principalmente richiedono agli altri thread dell'applicazione i dati da visualizzare sull'interfaccia.

- **Telegram thread**: il thread che si occupa della ricezione e dell'invio dei messaggi delle diverse chat, interagendo con l'infrastruttura di API di Telegram, chiamata TDLib (Telegram Database Library).

- **Update thread**: il thread che ogni minuto richiede al thread di Telegram di controllare se sono arrivati nuovi messaggi e di aggiornare i contatori dei messaggi non letti per ogni chat.

- **SEcube™ thread**: il thread che interagisce con il SEcube™ per ottenere l'elenco delle chat i cui messaggi devono essere decriptati in ricezione e criptati in invio e per effettuare le operazioni di cifratura sui messaggi.

Il thread principale avvia i restanti thread man mano che viene effettuato il processo di avvio descritto nella sezione 4.2.1.

4.1.1 SEcube™ Thread

La prima operazione eseguita dal `GUI thread` è la creazione del `SEcube thread`: si è scelto di separare il codice legato all'interazione con il SEcube™ dal resto dell'applicazione, creando una struttura di tipo client/server, in cui l'applicazione

Qt rappresenta il client che richiede le operazioni di crittografia per i messaggi ad un server C++, chiamato SEcubeServer; il server ascolta le connessioni provenienti solo da localhost e su una porta specificata dall'applicazione. Questo approccio è stato adottato per permettere a una qualsiasi applicazione (quindi anche ad altre eventuali applicazioni che potranno essere sviluppate in futuro, dopo SEtelegram) di utilizzare le funzionalità di crittografia offerte dal SEcube™ senza dover includere nient'altro all'interno del proprio codice, ma semplicemente inviando richieste e ricevendo le relative risposte da un server esterno; in tal modo viene applicato il principio di sicurezza olistica alla base della piattaforma SEcube™ descritto nel capitolo 3.

Il SEcubeServer è costituito da un file sorgente principale che include al suo interno alcune delle librerie del SEcube™ analizzate nel capitolo 3, fra cui le API di livello L0 e L1 per effettuare il login e il logout alla piattaforma SEcube™. Include, inoltre, una libreria aggiuntiva, chiamata anch'essa SEtelegram e specifica per l'applicazione: gestisce la crittografia dei messaggi delle chat, utilizzando le funzionalità di SElink e SEkey, e crea un database criptato con SEfile nella memoria del SEcube™; tale database contiene la lista degli ID delle chat di SEtelegram che sono protette dal SEcube™, associate agli ID dei corrispettivi gruppi SEkey. Grazie a questo database l'applicazione sceglie quali sono le chat da criptare.

La prima operazione effettuata dal SEcubeServer è la creazione di un socket in ascolto su localhost e sulla porta passata come parametro (settata all'interno dell'applicazione), in attesa che il client, cioè SEtelegram, richieda una connessione.

Non appena la richiesta di connessione da parte dell'applicazione viene accettata, il SEcubeServer entra in un ciclo `while` in attesa di richieste da parte di SEtelegram; tali richieste sono a limitate a una lista di stringhe ben definita, quindi qualsiasi richiesta in arrivo che abbia un formato o un contenuto diverso da quello atteso viene scartata. All'interno del ciclo `while`, in questa fase iniziale, le

uniche stringhe che il server si aspetta sono una richiesta di login, concatenata al pin per accedere al SEcube™ (inserito dall'utente nell'applicazione), o la richiesta di terminare l'esecuzione, in arrivo da SEtelegram nel caso in cui l'utente avesse chiuso l'applicazione prima di provare ad effettuare il login: in quest'ultimo caso il server interrompe direttamente il ciclo, chiude il socket su cui era stata accettata la connessione con l'applicazione e termina l'esecuzione del programma; nel caso in cui, invece, l'utente avesse richiesto il login al SEcube™, il server interagisce con le librerie L0 e L1 per localizzare il dispositivo ed accedervi con il pin fornito, poi avvia il SEkey e la libreria specifica per SEtelegram precedentemente descritta, in modo da creare una connessione al database locale delle chat protette e poter successivamente applicare la crittografia di SElink ai loro messaggi.

Dopo questa prima fase di inizializzazione, il server entra in un secondo ciclo, una seconda fase in cui potrà gestire le successive richieste da parte dell'applicazione: in particolare, potrà indicare se una chat prevede la protezione del SEcube™ o no, cifrare e decifrare testi o file generici e gestire la chiusura dell'applicazione, fermando SEkey, effettuando il logout dal SEcube™ e chiudendo il socket della connessione con l'interfaccia di SEtelegram.

4.1.2 Telegram Thread

In seguito alla fase di login al SEcube™ e di inizializzazione delle librerie ad esso legate, l'applicazione avvia un ulteriore thread, quello che si occupa dell'interazione con le API di TDLib. Il codice eseguito da questo thread è racchiuso in una classe C++ pubblica, chiamata `Td` e basata sull'esempio di utilizzo di TDLib in C++ fornito da Telegram all'interno della documentazione della libreria[5]: prevede anch'essa l'uso di un ciclo `while` in cui vengono gestite le diverse possibili richieste che l'applicazione può voler inviare ai server di Telegram, fra cui il login al servizio, la

[5] https://github.com/tdlib/td/blob/master/example/cpp/td_example.cpp

lista delle chat, l'invio di un messaggio e il download di un file. Le risposte in arrivo dal server vengono ricevute solo in presenza di una richiesta esplicita di ricezione da parte dell'applicazione, motivo per cui ad ogni richiesta è associato un metodo, `waitForServerResponse()`, contenente un ciclo di richieste di aggiornamento che viene interrotto solo quando la risposta viene recapitata.

La classe `Td` comprende: il metodo `send_query`, che si occupa di mandare le richieste al server e di registrare la funzione handler (passatagli come parametro) che verrà eseguita in seguito alla ricezione della risposta; il metodo `process_response`, che gestisce le risposte ricevute, avviando l'esecuzione dei relativi handler oppure chiamando il metodo `process_update` per gli aggiornamenti; le funzioni utilizzate nella fase iniziale per l'autenticazione dell'utente.

Il `Telegram thread` e il `GUI thread` interagiscono scambiandosi dati attraverso una serie di variabili globali: nella fase di avvio dell'applicazione, il `Telegram thread` richiede al `GUI thread` alcuni dati riguardanti l'utente, in modo da poter effettuare il login al servizio di Telegram; in seguito, invece, sarà sempre il `GUI thread` ad inviare richieste al `Telegram thread`. Tale comportamento richiede che i due thread siano sempre sincronizzati e che l'accesso alle variabili globali condivise sia sempre protetto attraverso l'uso di `mutex` e `condition_variable`. Il procedimento utilizzato per ottenere tale sincronizzazione, mostrato in Figura 4.2 per la fase di avvio e in Figura 4.3 per il resto del ciclo di vita dell'applicazione, è il seguente:

Siano "A" il thread che richiede i dati e "B" il thread che li fornisce

1. Il thread A attende, attraverso la `condition_variable "cv_ready"`, che B sia pronto per ricevere richieste, cioè che la variabile globale `thread_ready` sia settata a true;

2. il thread B acquisisce il lock sul `mutex "mutex_ready"` e setta `thread_ready` a true; successivamente manda una notifica al thread A, che si trova in attesa

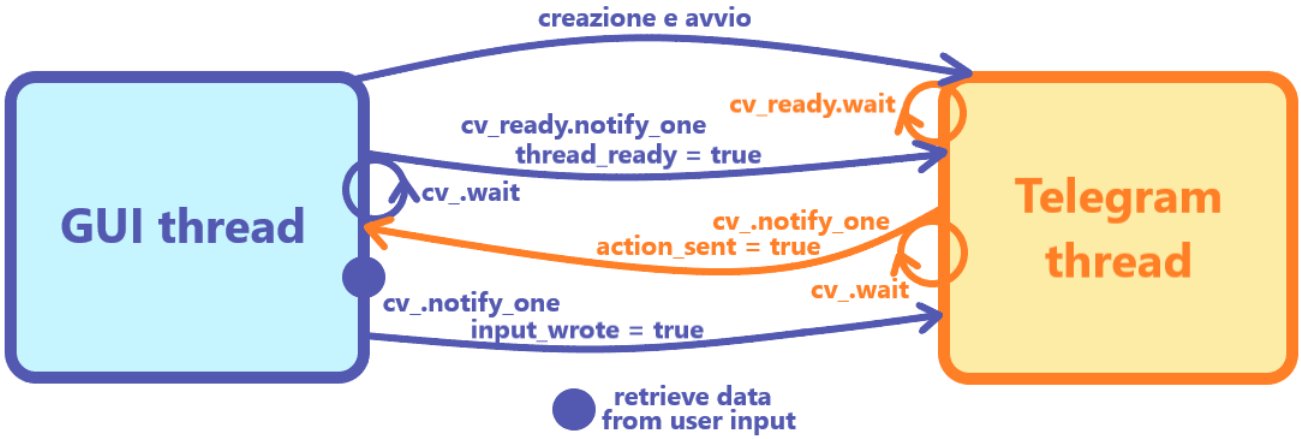

Figura 4.2. Il processo di sincronizzazione fra il `GUI thread` e il `Telegram thread` nella fase di avvio dell'applicazione.

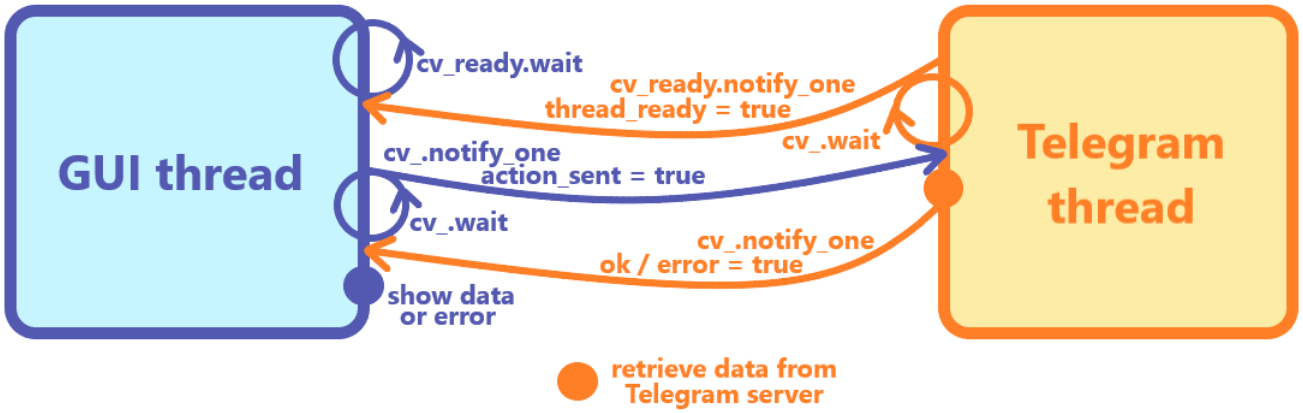

Figura 4.3. Il processo di sincronizzazione fra il `GUI thread` e il `Telegram thread` durante il normale funzionamento dell'applicazione.

sulla `condition_variable "cv_ready"`, per riattivarlo; infine attende sulla `condition_variable "cv_"` che il thread A gli mandi una richiesta all'interno della variabile globale `action` e che setti la variabile `action_sent` a true;

3. il thread A viene riattivato: a questo punto inizializza `action` con la stringa corrispondente alla richiesta che vuole effettuare; poi acquisisce il lock sul `mutex "mutex_"` per settare a true il booleano `action_sent`, invia una notifica al thread B su `cv_` e si mette a sua volta in attesa dei dati su quest'ultimo; verrà riattivato in seguito a una notifica su `cv_`;

4. il thread B viene riattivato: legge la richiesta all'interno di `action`, in base ad essa ricava i dati richiesti e li memorizza nella rispettiva variabile globale, in modo che siano successivamente accessibili dal thread A; poi setta un'ultima variabile booleana per permettere al thread A di riattivarsi, manda la notifica sulla `condition_variable "cv_"` e torna in attesa di nuove richieste.

Nella fase successiva all'avvio dell'applicazione, cioè quando il thread A è il `GUI thread` e il thread B è il `Telegram thread`, l'ultimo booleano settato a true dal `Telegram thread` per riattivare il `GUI thread` può essere la variabile `"ok"` oppure la variabile `"error"`: il `Telegram thread` setta a true `ok` se ha ricevuto correttamente dal server i dati richiesti dal `GUI thread`, `error` in caso contrario; in questo modo il `GUI thread` saprà se visualizzare normalmente i dati o se mostrare un messaggio di errore all'utente.

4.1.3 Update Thread

L'ultimo thread ad essere avviato è quello di update, il cui compito consiste nell'emettere ogni minuto un segnale, chiamato `checkNewMessages`, che porta a sua volta il thread dell'applicazione a richiedere al `Telegram thread` gli aggiornamenti in quel momento disponibili, in modo da ottenere eventuali nuovi messaggi ricevuti nella chat visualizzata e aggiornare i contatori dei messaggi non letti mostrati sui bottoni delle chat.

Il codice eseguito dall'`Update thread` è costituito da un ciclo `while`, la cui condizione dipende dal valore di ritorno di un altro metodo: all'interno di quest'ultimo

la `condition_variable "update_stop_cv"` rimane in attesa per un minuto, per poi ritornare il valore inverso a quello della variabile booleana `update_stop`, che viene settata a true dal `GUI thread` quando l'applicazione viene chiusa, in modo da poter terminare anche l'esecuzione dell'`Update thread`.

Il processo di comunicazione fra il `GUI thread` e l'`Update thread` è mostrato in Figura 4.4:

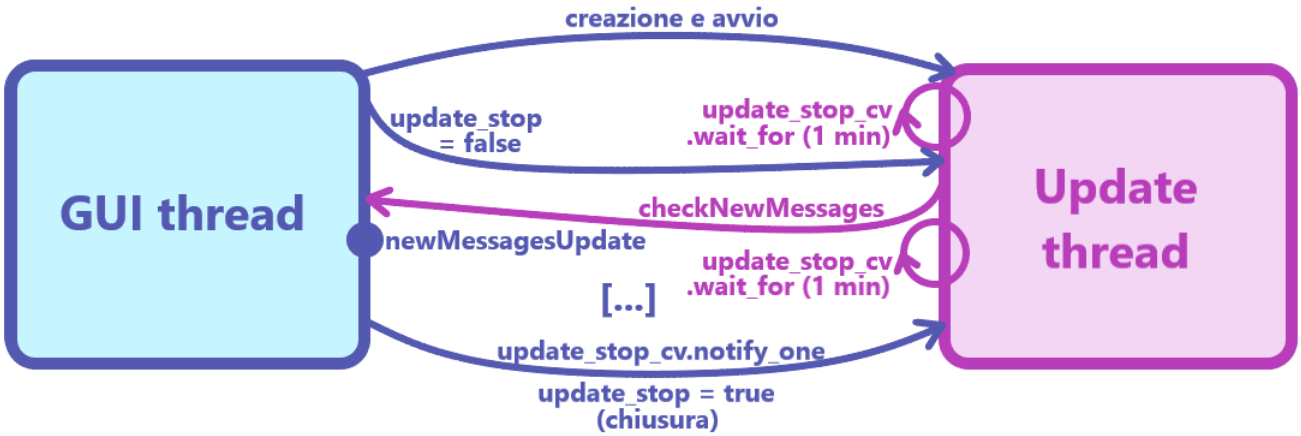

Figura 4.4. Il processo di comunicazione fra il `GUI thread` e l'`Update thread`, dal suo avvio fino alla chiusura.

- Finché `update_stop` rimane settata a false, l'`Update thread`, allo scadere di ogni minuto a partire dal suo avvio, entra nel ciclo `while` ed emette il segnale `checkNewMessages`; in risposta a tale segnale il `GUI thread` chiama il metodo `newMessagesUpdate` (descritto nel capitolo 4.2.6), in cui richiede al `Telegram thread` gli aggiornamenti disponibili, riguardanti eventuali nuovi messaggi ricevuti, e ne gestisce la visualizzazione.

- Non appena l'utente chiude l'applicazione, il `GUI thread` setta a true la variabile `update_stop`: la condizione per l'esecuzione dell'`Update thread` diventa falsa e il thread viene terminato.

4.2 Implementazione

In questa sezione verranno descritte nel dettaglio le funzionalità di SEtelegram e in che modo i thread e i metodi delle varie classi comunicano fra loro per implementarle. In Figura 4.5 sono rappresentate le principali interazioni fra i thread e i metodi dell'applicazione, elencate e descritte nelle Figure 4.6 e 4.7.

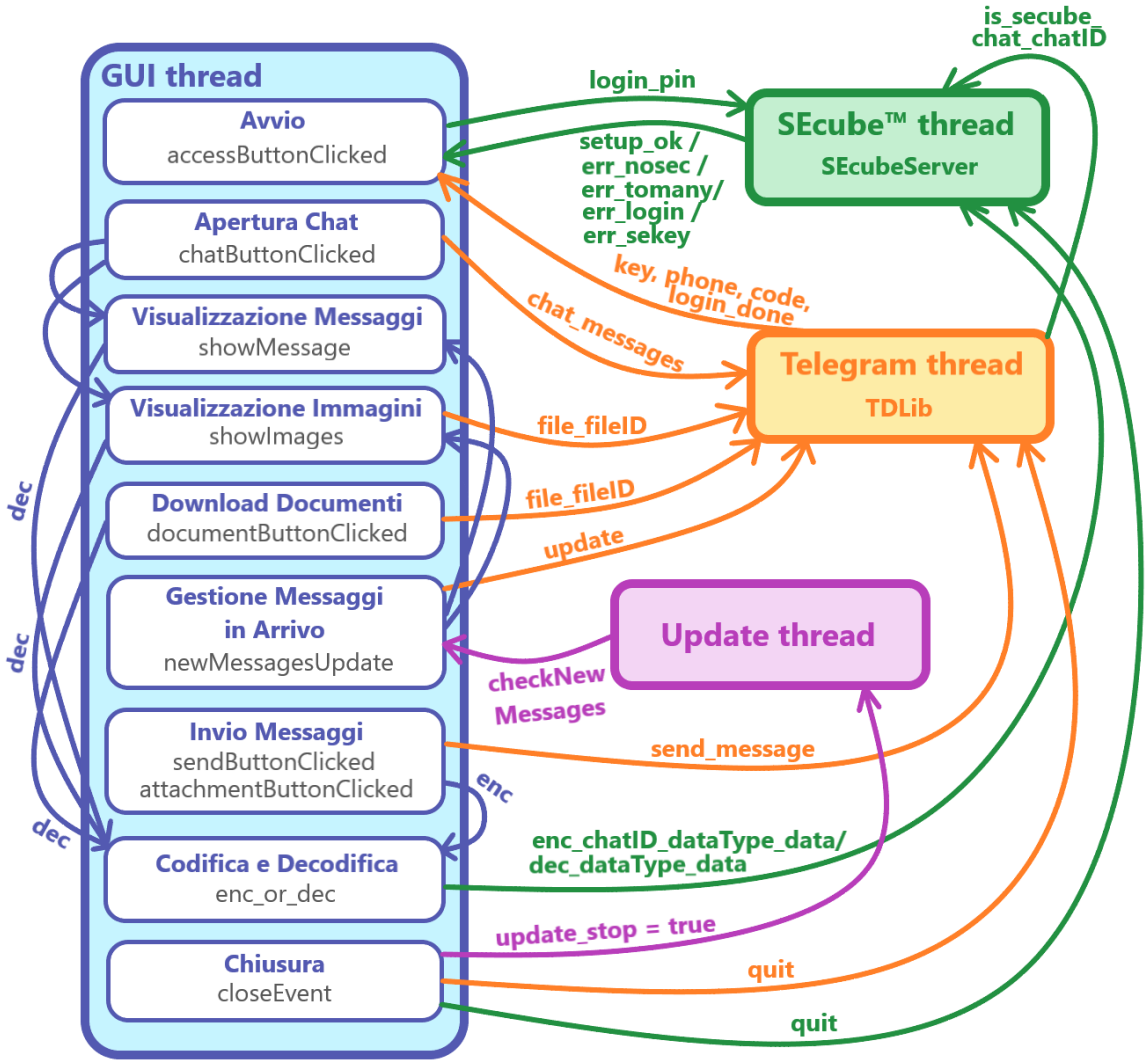

Figura 4.5. Schema delle principali richieste e risposte scambiate fra i thread dell'applicazione.

Richiesta / Risposta	Mezzo	Thread Mittente	Thread Ricevente	Descrizione
login_<pin>	socket	GUI thread	SEcube™ thread	Richiesta di login al SEcube™ e di avvio delle librerie SEkey e SEtelegram
setup_ok	socket	SEcube™ thread	GUI thread	Notifica che il processo di login al SEcube™ e di avvio delle sue librerie è avvenuto con successo
err_nosec	socket	SEcube™ thread	GUI thread	Notifica che non è stato rilevato alcun SEcube™ connesso al dispositivo host
err_tomany	socket	SEcube™ thread	GUI thread	Notifica che sono stati rilevati più SEcube™ connessi al dispositivo host
err_login	socket	SEcube™ thread	GUI thread	Notifica che è avvenuto un errore nel processo di login o che è stato inserito un pin errato
err_sekey	socket	SEcube™ thread	GUI thread	Notifica che è avvenuto un errore nel processo di avvio della libreria SEkey
key	variabile “action”	Telegram thread	GUI thread	Richiede l’inserimento da parte dell’utente della chiave utilizzata per criptare il database creato dalle TDLib
phone	variabile “action”	Telegram thread	GUI thread	Richiede l’inserimento da parte dell’utente del numero di telefono collegato all’account Telegram
code	variabile “action”	Telegram thread	GUI thread	Richiede l’inserimento da parte dell’utente del codice di autorizzazione inviato da Telegram per messaggio
login_done	variabile “action”	Telegram thread	GUI thread	Notifica che il processo di login a Telegram è avvenuto con successo
startup	variabile “action”	GUI thread	Telegram thread	Richiede l’ID Telegram dell’utente e la lista delle chat di cui fa parte
is_secube_chat_<chatID>	socket	Telegram thread	SEcube™ thread	Richiede alla libreria SEtelegram se la chat è protetta dal SEcube™

Figura 4.6. Lista delle richieste e delle risposte scambiate fra i thread durante la fase di avvio.

Richiesta / Risposta	Mezzo	Thread Mittente	Thread Ricevente	Descrizione
chat_messages	variabile “action”	GUI thread	Telegram thread	Richiede i messaggi della chat visualizzata
file_<fileID>	variabile “action”	GUI thread	Telegram thread	Richiede il download di un file
update	variabile “action”	GUI thread	Telegram thread	Richiede gli aggiornamenti disponibili (nuovi messaggi in arrivo)
send_message	variabile “action”	GUI thread	Telegram thread	Richiede l’invio di un messaggio nella chat visualizzata
encrypt_<chatID>_ <dataType>_<data>	socket	GUI thread	SEcube™ thread	Richiede la codifica dei dati <data> di tipo <dataType> nella chat con ID <chatID>
decrypt_ <dataType>_<data>	socket	GUI thread	SEcube™ thread	Richiede la decodifica dei dati <data> di tipo <dataType>
quit	variabile “action” e socket	GUI thread	Telegram thread e SEcube™ thread	Richiede la terminazione del thread destinatario

Figura 4.7. Lista delle richieste e delle risposte scambiate fra i thread in seguito alle interazioni dell’utente con l’interfaccia.

4.2.1 Avvio

L’applicazione per prima cosa avvia il `SEcube thread`, poi apre la finestra mostrata in Figura 4.8, in cui l’utente può inserire il pin per effettuare il login al proprio SEcube™.

L’applicazione si connette al SEcubeServer, in attesa che l’utente clicchi su "OK" oppure chiuda la finestra; in quest’ultimo caso viene gestito l’evento di chiusura di quest’ultima, in modo da mandare al SEcubeServer il comando "quit" che permette al `SEcube thread` di terminare.

Nel caso in cui, invece, l’utente accetti di proseguire inserendo il pin e cliccando su "OK", l’applicazione invia al SEcubeServer il comando `login` seguito dal pin, in modo che il server possa tentare di accedere al SEcube™ e completare la fase di inizializzazione descritta nella sezione 4.1.1, per poi inviare l’esito del processo all’applicazione. SEtelegram può ricevere diversi tipi di errore dal SEcubeServer

Figura 4.8. La prima finestra di SEtelegram per l'inserimento del pin del SEcube™.

in questa fase, tra cui la mancata presenza di un SEcube™ collegato, un errore nel processo di login o nell'avvio di SEkey, oppure un errore nella ricezione della richiesta o della risposta. Tali errori vengono segnalati con una finestra di dialogo come quella mostrata in Figura 4.9; chiudendola l'utente può riprovare ad effettuare il login.

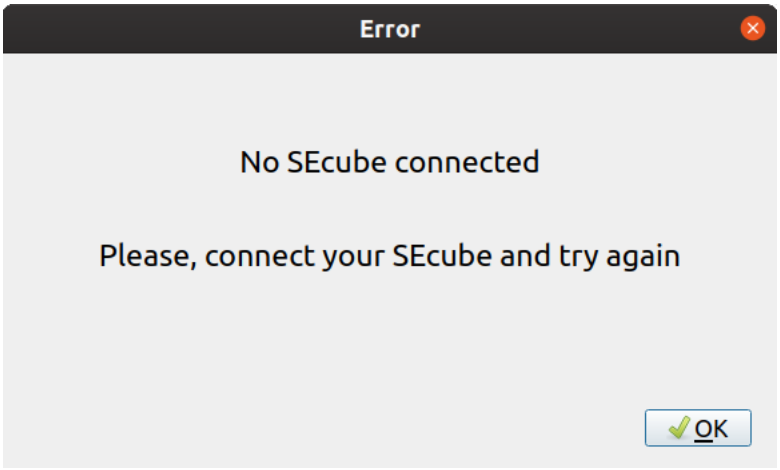

Figura 4.9. Esempio di finestra di dialogo per gli errori in SEtelegram.

Non appena viene ricevuta la stringa `setup_ok` dal SEcubeServer, l'applicazione procede ad avviare il `Telegram thread` per poi sincronizzarsi con esso attraverso

l'uso delle `condition_variable`, in modo da effettuare la procedura per accedere al servizio di Telegram.

Il `Telegram thread`, quando viene avviato, controlla lo stato di autorizzazione dell'utente, in modo da registrarne l'accesso ai server di Telegram: vengono quindi ricevuti degli update dal server di tipo *updateAuthorizationState*[6], che vengono gestiti all'interno del metodo `on_authorization_state_update` della classe `Td` e che indicano in quale stadio del processo di autorizzazione ci si trova.

Dopo il primo stadio, che richiede l'inserimento di alcuni parametri per configurare le TDLib, si passa allo stadio in cui viene richiesto all'utente l'inserimento di una *Encryption Key*, la chiave che viene utilizzata per criptare il database creato in locale dalle TDLib, contenente i dati relativi al processo di autorizzazione che verranno inseriti successivamente dall'utente, fra cui il numero di telefono. A questo punto il `Telegram thread` e il `GUI thread` comunicano seguendo il procedimento descritto nella sezione 4.1.2; in particolare:

- il `Telegram thread` attende con una `condition_variable` che il `GUI thread` sia pronto a ricevere richieste, cioè che setti la variabile globale booleana `thread_ready` a true;

- il `GUI thread` setta `thread_ready` a true e notifica che è pronto, risvegliando il `Telegram thread`, poi si mette in attesa;

- il `Telegram thread` setta `key` come valore della stringa globale `action` e notifica il `GUI thread`, riattivandolo, poi si mette nuovamente in attesa;

- il `GUI thread` mostra all'utente una finestra di dialogo (Figura 4.10) per permettergli di inserire la chiave richiesta; una volta premuto su "OK", il

[6] https://core.telegram.org/tdlib/docs/classtd_1_1td__api_1_1update_authorization_state.html

thread dell'applicazione salva il valore inserito nella variabile globale `key`, notifica il `Telegram thread` e si rimette in attesa di successive richieste;

- a questo punto il `Telegram thread` ha a disposizione il valore della chiave per continuare il processo di autorizzazione dell'utente.

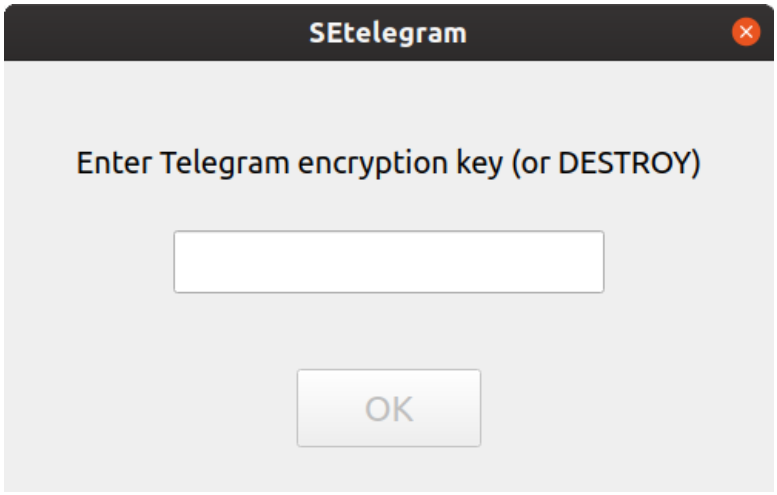

Figura 4.10. Finestra di dialogo per l'inserimento della chiave di crittografia per i dati di autorizzazione di Telegram.

Se il valore inserito per la chiave è pari alla parola chiave "DESTROY", oppure è la prima volta che viene effettuato l'accesso a Telegram mediante le TDLib sul dispositivo in uso, il `Telegram thread` avrà bisogno di altri dati per autenticare l'utente, cioè il suo numero di telefono e un codice di autorizzazione che riceverà per messaggio: l'utente dovrà quindi inserire tali informazioni nelle rispettive finestre di dialogo che il `GUI thread` provvederà a mostrare, seguendo il medesimo procedimento appena descritto.

Il `Telegram thread` gestisce, così, tutti i successivi stadi del processo di autorizzazione, fino a quando non si raggiunge lo stato *authorizationStateReady*; a questo punto il `Telegram thread` comunica al `GUI thread` che il login a Telegram è andato a buon fine (utilizzando sempre la variabile `action` e una notifica sulla

`condition_variable`) e passa finalmente nella parte del suo loop in cui attende richieste di dati Telegram dall'applicazione. Il `GUI thread` chiude la finestra iniziale e apre la finestra principale dell'applicazione, mostrata in Figura 4.11.

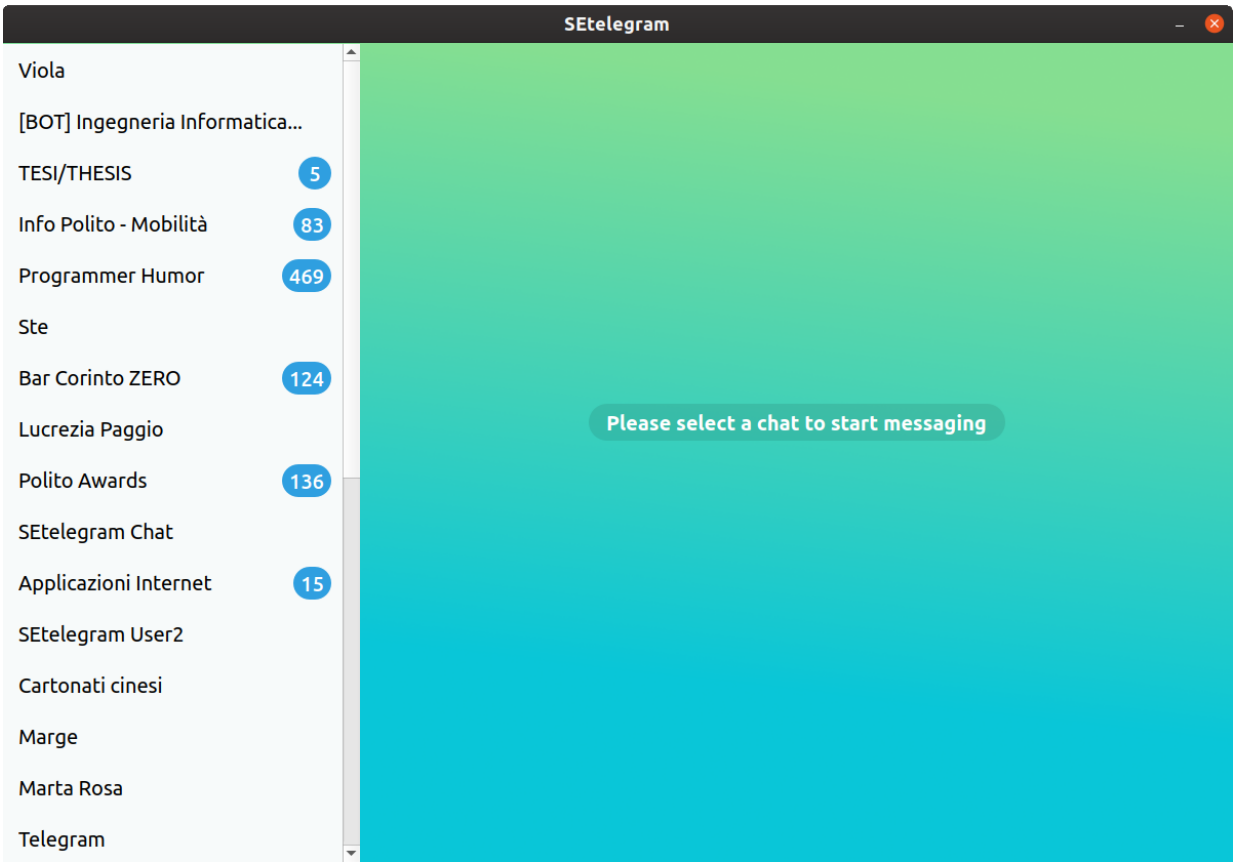

Figura 4.11. La finestra principale di SEtelegram alla fine del processo di avvio.

All'interno del costruttore di questa finestra, chiamata `ChatWindow`, il thread dell'applicazione richiede alcuni dati al `Telegram thread`; per farlo viene seguito un procedimento speculare a quello descritto precedentemente per la chiave di crittografia di TDLib; da qui in poi, però, sarà il `GUI thread` ad aspettare che il `Telegram thread` sia pronto per poi richiedere, attraverso la variabile globale `action`, diversi tipi di operazioni e dati: tutte le successive interazioni fra i due thread avverranno seguendo questo meccanismo di sincronizzazione.

In questa fase il thread dell'applicazione setta la stringa `startup` come valore di `action`, poi notifica il `Telegram thread` e resta in attesa di una risposta. Il `Telegram thread` riparte e controlla il valore di `action`. Nel caso in cui questo sia pari a `startup`, richiede al server di Telegram prima l'ID dell'utente, utilizzando la funzione `getMe` delle TDLib[7], poi la lista delle chat Telegram dell'utente, con la funzione `getChats`[8]. Il risultato della prima richiesta viene gestito direttamente dalla relativa funzione handler, in cui viene semplicemente memorizzato l'ID ricevuto. La richiesta della lista di chat, invece, riceve risposta attraverso una serie di update di tipo `updateNewChat`, gestiti all'interno del metodo `process_update` della classe `Td`: ognuno di questi aggiornamenti corrisponde a una chat, per la quale viene creato un oggetto di tipo `SEchat`, una struttura i cui membri sono:

- **`chat_id`**, cioè l'ID della chat;
- **`chat_name`**, cioè il nome della chat che verrà visualizzato;
- **`chat_type`**, che indica se la chat è di tipo uno a uno, gruppo, canale, ecc.;
- **`unread_count`**, cioè il numero di messaggi di quella chat che non sono ancora stati visualizzati dall'utente;
- **`is_SEcube_chat`**, un valore booleano che indica se la chat ha un corrispettivo gruppo SEkey, cioè se è una chat protetta dal SEcube™.

Per valorizzare l'attributo `is_SEcube_chat`, il `Telegram thread` manda per ogni chat la richiesta `is_secube_chat` al SEcubeServer, corredata con l'ID Telegram di quella chat, in modo che il server cerchi la corrispondenza di tale ID con quello di un gruppo SEkey all'interno del database creato in locale per SEtelegram; tale ricerca viene effettuata dal server chiamando la API `setelegram_find_group`.

[7] https://core.telegram.org/tdlib/docs/classtd_1_1td__api_1_1get_me.html
[8] https://core.telegram.org/tdlib/docs/classtd_1_1td__api_1_1get_chats.html

Il SEcubeServer risponde al `Telegram thread` con il risultato di tale funzione: se questo è pari a zero, l'attributo `is_SEcube_chat` viene settato a true e i messaggi di quella chat verranno criptati dal SEcube™.

Ogni oggetto SEchat creato in questo modo viene man mano aggiunto alla lista `chat_list`, che viene infine consultata dal thread dell'applicazione per creare la lista di bottoni che appare nella parte sinistra della finestra principale (Figura 4.11).

4.2.2 Apertura di una Chat: `chatButtonClicked`

Ora l'utente ha la possibilità di cliccare su uno qualsiasi dei bottoni delle chat creati, per visualizzarne i messaggi. L'evento "clicked" associato a ogni bottone porta all'esecuzione del metodo `chatButtonClicked`. Tale evento può essere scatenato in un qualsiasi momento del ciclo di vita dell'applicazione, motivo per cui il `GUI thread` si accerta che il `Telegram thread` sia pronto a ricevere richieste; poi setta la variabile `action` a `chat_messages` ed attende la risposta. Il `Telegram thread`, a questo punto, deve prima indicare che è stata aperta una chat mandando la richiesta `openChat`[9] al server di Telegram e, nel caso prima ce ne fosse un'altra aperta, mandando anche una richiesta `closeChat`[10] per chiuderla; poi richiede di ricevere i messaggi della chat aperta, attraverso la query `getChatHistory`[11] (utilizzata due volte, prima per ricevere il messaggio più recente, poi per ricevere tutti quelli precedenti); infine, con la query `viewMessages`[12], setta i messaggi ricevuti come visualizzati, in modo da aggiornare l'attributo `unread_count` della chat.

[9] https://core.telegram.org/tdlib/docs/classtd_1_1td__api_1_1open_chat.html

[10] https://core.telegram.org/tdlib/docs/classtd_1_1td__api_1_1close_chat.html

[11] https://core.telegram.org/tdlib/docs/classtd_1_1td__api_1_1get_chat_history.html

[12] https://core.telegram.org/tdlib/docs/classtd_1_1td__api_1_1view_messages.html

Ogni messaggio ricevuto viene memorizzato in una struttura di tipo `message`[13] e inserito all'interno della lista `messages`.

Ovviamente le richieste inviate dall'applicazione ai server di Telegram possono non andare a buon fine: in tal caso l'applicazione, invece di mostrare i dati richiesti, apre una finestra di dialogo simile a quella mostrata in Figura 4.9, contenente il messaggio di errore, in modo da informare l'utente.

4.2.3 Visualizzazione dei Messaggi: `showMessage`

Nel caso in cui i messaggi siano stati ricevuti correttamente e la gestione dell'apertura e chiusura delle chat sia andata a buon fine, viene invocato, per ogni messaggio della lista `messages`, il metodo `showMessage`, utilizzato dall'applicazione per creare un frame personalizzato per ogni messaggio da inserire all'interno della chat.

- Se il messaggio è stato inviato in una chat di gruppo, all'interno del frame del messaggio verrà visualizzato il nome del mittente (a meno che non sia lo stesso utente che sta visualizzando la chat).

- Verrà utilizzato un frame diverso a seconda del tipo di messaggio, in modo che la visualizzazione si adatti ai dati in esso contenuti: un messaggio di testo ha un contenuto di tipo `messageText`[14] e verrà visualizzato con un `TextMessageFrame`, contenente un semplice campo di testo; un messaggio il cui contenuto è di tipo `messagePhoto`[15] verrà mostrato attraverso un `ImageMessageFrame`, che comprende un campo per visualizzare l'immagine e

[13] https://core.telegram.org/tdlib/docs/classtd_1_1td__api_1_1message.html
[14] https://core.telegram.org/tdlib/docs/classtd_1_1td__api_1_1message_text.html
[15] https://core.telegram.org/tdlib/docs/classtd_1_1td__api_1_1message_photo.html

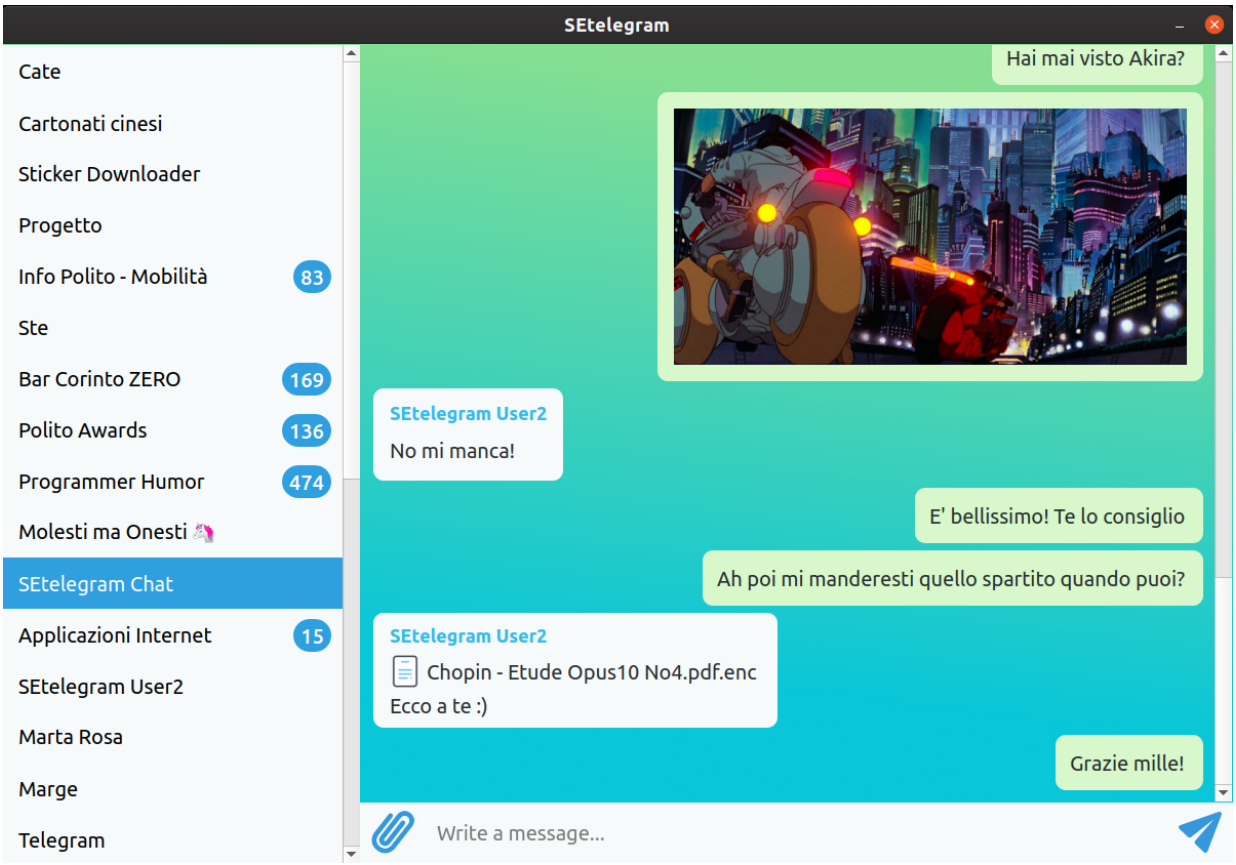

Figura 4.12. Esempio di una chat di SEtelegram, contenente diversi tipi di frame a seconda del messaggio: `TextMessageFrame` per messaggi di testo, `ImageMessageFrame` per le immagini e `DocumentMessageFrame` per altri tipi di file.

uno per l'eventuale didascalia; tutti gli altri tipi di messaggio comprendenti allegati vengono catalogati come di tipo `messageDocument`[16], e vengono visualizzati attraverso un frame di tipo `DocumentMessageFrame`, in cui viene mostrato il nome del file sopra l'eventuale didascalia. I diversi tipi di messaggio e le loro rappresentazioni sono mostrati in Figura 4.12.

- Un messaggio criptato contenente un'immagine viene inviato nello stesso

[16] https://core.telegram.org/tdlib/docs/classtd_1_1td__api_1_1message_document.html

formato usato per i file generici, quindi all'interno di un messaggio di tipo `messageDocument`, poiché un'immagine criptata attraverso le API di SElink diventa un semplice stream di byte e perde la sua struttura originale, non permettendo ai server di Telegram di gestirne upload e download seguendo il procedimento usato per le immagini; quindi il metodo `showMessage`, quando considera un messaggio di tipo `messageDocument` nel contesto di una chat criptata, deve controllare l'estensione del file al suo interno, in modo da distinguere le immagini criptate dai file generici e visualizzare correttamente entrambe le tipologie di messaggio, attraverso un `ImageMessageFrame` in un caso, oppure con un `DocumentMessageFrame` nell'altro.

- Ogni volta che `showMessage` crea un `ImageMessageFrame` per visualizzare un messaggio contenente un'immagine, lo inserisce in un lista, chiamata `image_messages`, che verrà successivamente consultata dal metodo `showImages`, descritto nella sezione 4.2.4: quest'ultimo richiederà al `Telegram thread` di effettuare, per ogni `ImageMessageFrame`, il download della relativa immagine, in modo da mostrarla al suo interno.

- Il download dei file generici viene effettuato solo in seguito al click dell'utente sul `DocumentMessageFrame` corrispondente, evento al quale è associata l'esecuzione del metodo `documentButtonClicked`, descritto nella sezione 4.2.5.

Oltre a considerare il tipo di dati contenuti nel messaggio ricevuto, il metodo `showMessage` controlla se la chat che sta venendo visualizzata ha il campo `is_SEcube_chat` settato a true, cioè se i messaggi in essa contenuti debbano essere decriptati o meno. La gestione dei messaggi criptati varia a seconda del tipo di dato contenuto all'interno del messaggio:

- Il contenuto di un messaggio di testo è una semplice stringa, perciò il metodo `showMessage` potrà chiamare direttamente il metodo `enc_or_dec`, descritto

nella sezione 4.2.8, per richiedere al SEcubeServer di decriptarla, per poi mostrarla in chiaro all'interno di un `TextMessageFrame`. Nel caso la decodifica non andasse a buon fine, al posto del testo verrà mostrato un messaggio di errore.

- Un messaggio di tipo `messageDocument` o `messagePhoto` non contiene al suo interno direttamente il file o l'immagine in allegato, ma comprende un campo con il suo ID Telegram: quest'ultimo può essere utilizzato successivamente per richiederne il download attraverso un'ulteriore query ai server di Telegram. Perciò in questo caso non è il metodo `showMessage` ad occuparsi della decodifica degli allegati, ma il metodo `showImages` (sezione 4.2.4) per le immagini e il metodo `documentButtonClicked` (sezione 4.2.5) per tutti gli altri tipi di file; `showMessage` gestisce solo la decodifica della didascalia testuale, se presente, nello stesso modo con cui gestisce i messaggi di testo.

4.2.4 Visualizzazione delle Immagini: `showImages`

Dopo aver creato un frame per ogni messaggio e averlo inserito nell'area della chat, l'applicazione invoca il metodo `showImages` per scaricare le immagini dei messaggi ricevuti (se presenti). Tali immagini sono identificate da un ID Telegram che viene memorizzato nel corrispettivo `ImageMessageFrame` all'atto della sua creazione: per ogni oggetto presente nella lista `image_messages`, precedentemente popolata dal metodo `showMessage` (4.2.3), il `GUI thread` effettua una richiesta di download al `Telegram thread`, ponendo come valore di `action` la stringa `file` seguita dall'ID dell'immagine.

Il `Telegram thread` manda una query `downloadFile`[17] al server: l'immagine richiesta viene scaricata gradualmente e il progresso del download viene indicato

[17] https://core.telegram.org/tdlib/docs/classtd_1_1td__api_1_1download_file.html

con una serie di update di tipo `updateFile`[18] ricevuti dal server; al completamento dell'operazione, il `Telegram thread` salva l'immagine ricevuta in una variabile globale di tipo `file`[19], in modo che il `GUI thread` possa mostrarla all'interno del proprio `ImageMessageFrame`, eventualmente dopo averla decriptata invocando il metodo `enc_or_dec` (in caso di errori nella decodifica, al suo posto viene visualizzata un'immagine di errore).

A ogni immagine viene associato un evento "clicked", in modo che l'utente possa visualizzarla a schermo intero, utilizzando l'applicazione predefinita del proprio dispositivo.

A questo punto il processo di visualizzazione dei messaggi all'interno della chat selezionata è completo.

4.2.5 Download dei Documenti: `documentButtonClicked`

Diversamente da quanto accade per le immagini, il download dei documenti contenuti nei messaggi avviene in un secondo momento, ovvero quando l'utente clicca su un `DocumentMessageFrame` per visualizzare il contenuto del file al suo interno: tale evento porta all'esecuzione del metodo `documentButtonClicked`.

Il metodo `documentButtonClicked` per prima cosa controlla se il file è già presente in memoria: in tal caso, se necessita di essere decriptato, invoca il metodo `enc_or_dec` (sezione 4.2.8) e apre il documento risultante usando il path da quest'ultimo restituito, altrimenti apre semplicemente il file che era già stato scaricato.

Nel caso in cui il documento non sia già presente in memoria, il `GUI thread` effettua la richiesta di download al `Telegram thread` seguendo la stessa procedura effettuata per le immagini e descritta nella sezione 4.2.4 (utilizzo della query

[18] https://core.telegram.org/tdlib/docs/classtd_1_1td__api_1_1update_file.html
[19] https://core.telegram.org/tdlib/docs/classtd_1_1td__api_1_1file.html

`downloadFile` di TDLib e ricezione dei relativi `updateFile`). Anche in questo caso, se il file necessita di essere decriptato, prima di aprire il file con l'applicazione predefinita del dispositivo, viene invocato il metodo `enc_or_dec` per permettere al SEcube™ di decifrarlo. Se il processo di decodifica non va a buon fine viene visualizzato un messaggio di errore in una finestra di dialogo.

4.2.6 Gestione dei Messaggi in Arrivo: `newMessagesUpdate`

Come descritto nella sezione 4.1.3, allo scoccare di ogni minuto a partire dall'apertura della finestra principale dell'applicazione (Figura 4.11), l'`Update thread` manda il segnale `checkNewMessages` al `GUI thread` affinché quest'ultimo aggiorni i contatori dei messaggi non letti delle chat e visualizzi eventuali messaggi ricevuti nella chat in quel momento aperta. Ciò viene effettuato dal metodo `newMessagesUpdate`, in cui il `GUI thread` richiede gli aggiornamenti al `Telegram thread`, ponendo la stringa `update` come valore di `action`.

Il `Telegram thread` richiede al server gli aggiornamenti disponibili in quel momento e gestisce due tipi di update:

- **`updateNewMessage`**[20] per i nuovi messaggi in arrivo, di cui memorizza in una lista di messaggi a parte, chiamata `newMessages`, solo quelli appartenenti alla chat attualmente aperta;

- **`updateChatReadInbox`**[21] quando cambia il numero di messaggi non letti di una chat: il `Telegram thread` aggiorna il contatore del rispettivo oggetto `SEchat` all'interno della lista `chat_list`.

[20] https://core.telegram.org/tdlib/docs/classtd_1_1td__api_1_1update_new_message.html

[21] https://core.telegram.org/tdlib/docs/classtd_1_1td__api_1_1update_chat_read_inbox.html

Non appena il `Telegram thread` finisce di raccogliere gli aggiornamenti disponibili, viene riattivato il `GUI thread`, che confronta la lista appena popolata, `newMessages`, con la lista dei messaggi già visualizzati, `messages`, in modo da aggiungere in fondo a quest'ultima e alla vista della chat i nuovi messaggi appena ricevuti, utilizzando anche qui i metodi `showMessage` e `showImages`.

4.2.7 Invio di Messaggi

Anche il processo di invio di un messaggio, come quello di visualizzazione, dipende dal tipo di dati che si vogliono trasmettere.

Per inviare un messaggio testuale, l'utente deve semplicemente digitare il testo all'interno della barra di inserimento presente nella zona in basso a destra della finestra principale (Figura 4.12), recante il placeholder "Write a message...", per poi cliccare sul bottone di invio a destra, mostrato in Figura 4.13.

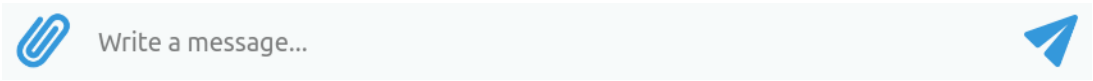

Figura 4.13. Barra per l'inserimento di un messaggio testuale; a destra il bottone di invio, a sinistra il bottone per la creazione di un messaggio con allegato.

L'evento "clicked" legato al bottone di invio invoca l'esecuzione del metodo `sendButtonClicked`, in cui:

1. viene creato un oggetto di tipo `inputMessageText`[22], cioè la struttura del contenuto del messaggio che verrà inviato;

2. se la chat corrente richiede la protezione del SEcube™, il testo inserito dall'utente viene criptato utilizzando il metodo `enc_or_dec`, descritto nella sezione 4.2.8;

[22] https://core.telegram.org/tdlib/docs/classtd_1_1td__api_1_1input_message_text.html

3. il campo `text_` dell'oggetto `inputMessageText`[23] viene valorizzato con il testo del messaggio (in chiaro o criptato a seconda dei casi) e viene inserito come contenuto del messaggio da inviare;

4. viene invocato il metodo `sendMessage`, in cui il `GUI thread` richiede al `Telegram thread` l'invio del messaggio (utilizzando la `sendMessage` delle TDLib[24]), settando la stringa `send_message` come valore di `action`;

5. viene visualizzato il messaggio inviato attraverso un `TextMessageFrame`; in caso di errore nel processo di invio del messaggio viene visualizzata una finestra di dialogo con l'errore avvenuto.

Per l'invio di messaggi con allegati, invece, è previsto che l'utente clicchi sul bottone a "graffetta" presente sulla sinistra del campo per l'inserimento dei messaggi di testo (Figura 4.13); l'evento "clicked" così generato porta all'esecuzione del metodo `attachmentButtonClicked`, in cui:

1. viene aperta una finestra, come mostrato in Figura 4.14, che permette di scegliere il file da allegare (può essere sia un'immagine sia un altro tipo di file);

2. in seguito alla selezione comparirà la finestra di dialogo rappresentata in Figura 4.15, in cui sarà possibile inserire una didascalia, inviare il messaggio o annullare l'operazione;

3. nel caso in cui l'utente abbia scelto il bottone "Send" della finestra di dialogo, il file viene copiato in una delle cartelle contenenti i media di Telegram: in "photos" se è un'immagine oppure in "documents" se è un altro tipo di documento;

[23] https://core.telegram.org/tdlib/docs/classtd_1_1td__api_1_1input_message_text.html

[24] https://core.telegram.org/tdlib/docs/classtd_1_1td__api_1_1send_message.html

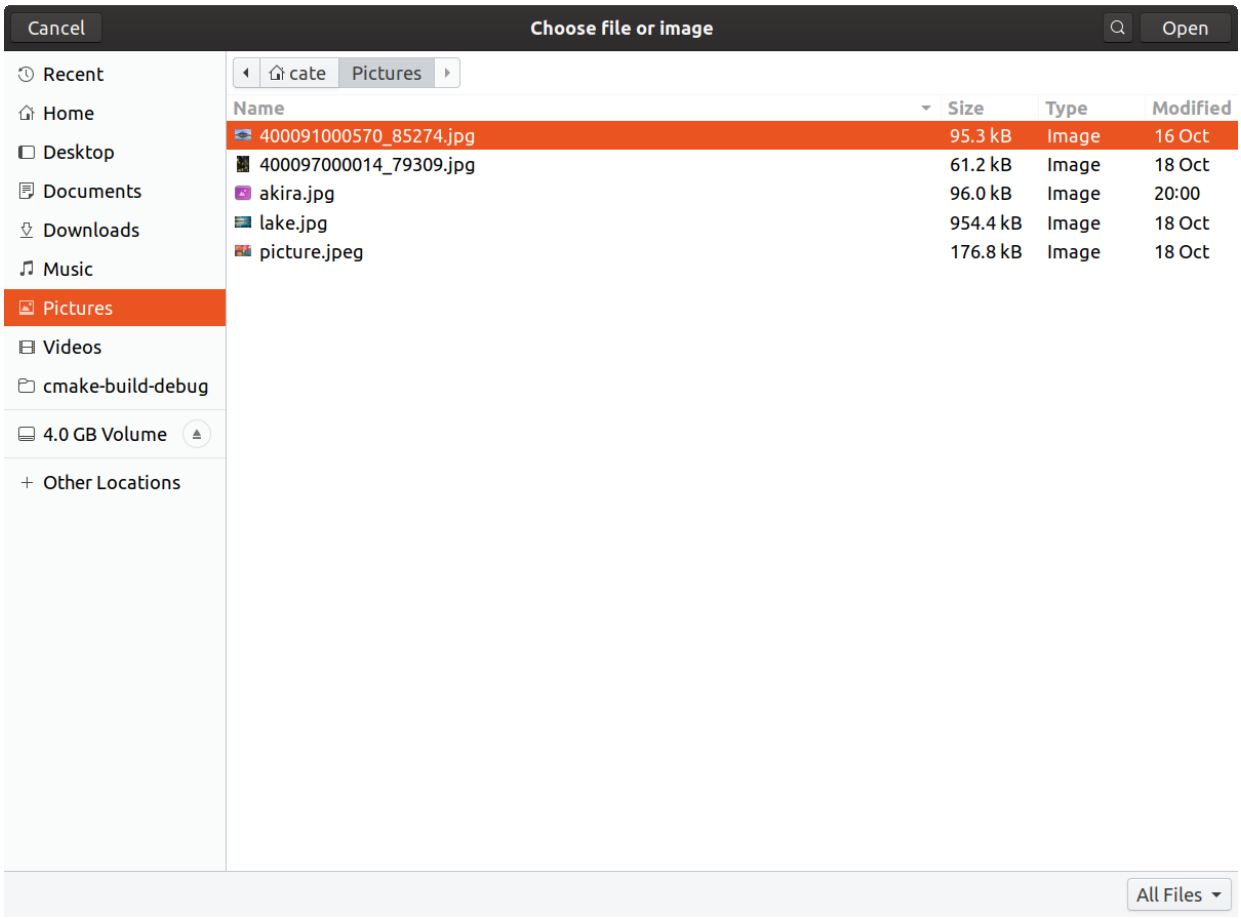

Figura 4.14. Finestra per la selezione di un file o un'immagine da inviare.

4. se l'allegato scelto è un'immagine e il messaggio non deve essere criptato con il SEcube™, viene creato un oggetto di tipo `inputMessagePhoto`[25]: nel campo `photo_` viene inserita l'immagine copiata al passo precedente all'interno di un oggetto `inputFileLocal`[26] e nel campo `caption` viene settato il testo della didascalia, se presente; se, invece, l'allegato è un'immagine che

[25] https://core.telegram.org/tdlib/docs/classtd_1_1td__api_1_1input_message_photo.html

[26] https://core.telegram.org/tdlib/docs/classtd_1_1td__api_1_1input_file_local.html

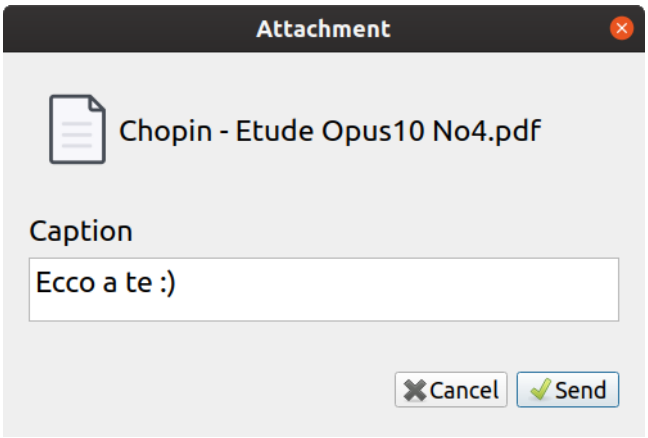

Figura 4.15. Finestra per l'invio di un file o un'immagine e per l'inserimento di una didascalia.

deve essere criptata oppure è un file in un altro formato, viene creato un oggetto di tipo `inputMessageDocument`[27]: nel campo `document_` viene inserita l'immagine criptata oppure il file, in chiaro o criptato (sempre attraverso il metodo `enc_or_dec` descritto nella sezione 4.2.8), all'interno di un oggetto `inputFileLocal`, e nel campo `caption` viene settato il testo della didascalia (eventualmente criptato), se presente;

5. il messaggio così creato viene inviato dal `Telegram thread` nello stesso modo in cui viene inviato un messaggio di testo; se il processo va a buon fine, il messaggio viene visualizzato nella chat, altrimenti viene aperta una finestra di dialogo con un messaggio di errore.

4.2.8 Codifica e Decodifica dei Messaggi: `enc_or_dec`

I metodi appena descritti per l'invio e la visualizzazione dei messaggi, nel caso in cui la chat in cui operano sia protetta dalle funzionalità di crittografia del SEcube™, hanno bisogno, rispettivamente, di criptare e decriptare il contenuto di tali

[27] https://core.telegram.org/tdlib/docs/classtd_1_1td__api_1_1input_message_document.html

messaggi: per far ciò, al loro interno viene invocato il metodo `enc_or_dec`.

Questo metodo viene eseguito dal thread dell'applicazione per mandare una richiesta di codifica o decodifica al SEcubeServer. Le richieste mandate al server son sempre stringhe costruite in modo da fornirgli le informazioni necessarie per eseguire le diverse operazioni di crittografia: la stringa di richiesta comincia con "encrypt" se si vuole criptare o con "decrypt" se si vuole decriptare; nel caso in cui si vogliano criptare i dati, alla stringa "encrypt" deve essere concatenato l'ID Telegram della chat in cui il messaggio verrà inviato; viene poi indicato il tipo di dato che si deve criptare o decriptare, cioè "text" o "file"; infine viene inserito o il testo del messaggio, o il path del file su cui verranno eseguite le operazioni di crittografia del SEcube™. Tutti i diversi campi sono separati fra loro da un "_".

Il SEcubeServer, dopo aver ricevuto la richiesta:

- se si tratta di una richiesta di codifica, crea un vettore di byte a partire dalla stringa di testo o dal file contenuti nella stringa ricevuta, utilizza il metodo `setelegram_encrypt` della libreria SEtelegram di SEcube™ per criptare i dati e serializza il risultato così ottenuto; se i dati erano parte di un documento, viene creato un corrispettivo file ".enc" contenente i dati serializzati; infine, se l'operazione di codifica è andata a buon fine, il SEcubeServer invia all'applicazione uno "0" seguito dal testo criptato o dal path del file criptato, altrimenti invia semplicemente un "1" per indicare che si è verificato un errore;

- se si tratta di una richiesta di decodifica, deserializza i dati ricevuti all'interno di un oggetto di tipo `selink_ciphertext`, da cui può ricavare le informazioni necessarie per decriptarli (fra cui la chiave e l'algoritmo di crittografia utilizzati), poi utilizza il metodo `setelegram_decrypt` della libreria SEtelegram per decodificare i dati e li memorizza in un array di byte; se i dati appartenevano a un documento, crea un corrispettivo file in cui li inserisce; infine, se l'operazione di decodifica è andata a buon fine, invia all'applicazione uno

"0" seguito dal testo decriptato o dal path del file decriptato, altrimenti invia semplicemente un "1" per indicare che si è verificato un errore.

Le funzioni `setelegram_encrypt` e `setelegram_decrypt` sono basate sulle rispettive API di SElink, `selink_encrypt_manual` e `selink_decrypt`, descritte nel capitolo 3.3; nell'esecuzione di `setelegram_encrypt` viene prima consultato il database locale, creato dalla libreria SEtelegram, contenente le associazioni fra chat di Telegram e gruppi di SEkey, in modo da ricavare la chiave di crittografia da utilizzare in `selink_encrypt_manual`.

Se il SEcubeServer è riuscito ad eseguire correttamente l'operazione di codifica o decodifica, il metodo `enc_or_dec` dell'applicazione ritorna "0" e memorizza il risultato nella stringa `result`, altrimenti restituisce un "1", in modo che il metodo che l'ha invocato possa mostrare un messaggio di errore.

4.2.9 Chiusura: `closeEvent`

Nel processo di chiusura dell'applicazione è necessario terminare, oltre al processo principale, i thread che son stati avviati nella procedura di avvio, cioè il `SEcube thread`, il `Telegram thread` e l'`Update thread`. Viene quindi definito il metodo `closeEvent`, in cui viene prima settato a true il booleano `update_stop`, in modo che la condizione del ciclo `while` nel codice dell'`Update thread` risulti falsa e il thread possa terminare (sezione 4.1.3); successivamente il thread dell'applicazione richiede anche al `Telegram thread` di terminare, settando la stringa `quit` come valore della variabile `action`; infine viene mandata la stessa stringa anche al SEcubeServer, in esecuzione sul `SEcube thread`, in modo che possa fermare il SEkey, effettuare il logout dal SEcube™ e uscire da entrambi i suoi cicli `while` per terminare.

A questo punto l'applicazione può eseguire il distruttore della finestra principale e terminare anche la sua esecuzione.

Capitolo 5

Conclusioni e Sviluppi Futuri

In questa tesi è stata sviluppata un'applicazione di messaggistica sicura basata su SEcube™, un modulo hardware orientato alla sicurezza, sfruttando le funzionalità offerte dalle API open source di Telegram, le TDLib.

SEtelegram utilizza la struttura stratificata della piattaforma SEcube™ per criptare a livello hardware i messaggi in arrivo e in uscita dall'applicazione: in tale modo le funzioni di crittografia ed i relativi parametri (chiavi, algoritmi, ecc.) non hanno alcuna influenza a livello applicazione, permettendo all'utente di concentrarsi unicamente sull'organizzazione dei gruppi e sulla gestione delle relative policy di sicurezza.

Rispetto alle applicazioni attualmente diffuse, i vantaggi della soluzione SEtelegram, derivanti dall'utilizzo di un device hardware esterno per la codifica e decodifica dei messaggi, sono principalmente:

- la maggiore velocità con cui l'hardware esegue le operazioni di crittografia rispetto al software;

- la maggiore robustezza dei sistemi hardware agli attacchi informatici;

- il fatto che il device hardware non sia direttamente connesso alla rete, a differenza del software, aspetto che lo rende più difficilmente accessibile dall'esterno;
- la gestione delle chiavi e degli altri parametri di crittografia, oltre che dei metadati relativi agli utenti, che avviene in locale e non coinvolge né la memoria del dispositivo host, né server esterni; tutti i dati memorizzati all'interno del SEcube™, inoltre, sono criptati utilizzando la libreria SEfile;
- l'utilizzo di una piattaforma stratificata come SEcube™, orientata alla sicurezza olistica, che permette di avere un livello aggiuntivo di sicurezza customizzabile, poiché l'utente può gestire le policy di sicurezza di ogni chat, ottenendo un livello di protezione più o meno robusto a piacimento;
- il processo di autenticazione a due fattori ottenuto utilizzando il SEcube™, dato che bisogna sia possedere fisicamente il modulo hardware, sia conoscere il pin per potervi accedere.

Nonostante gli enormi vantaggi offerti dal SEcube™, SEtelegram è ancora un'applicazione al primo stadio del suo sviluppo, a cui possono essere aggiunte molte funzionalità e migliorie:

- la possibilità di richiedere all'amministratore del sistema, direttamente attraverso l'applicazione, la creazione di una nuova chat Telegram collegata ad un corrispettivo gruppo SEkey, specificandone membri e policy di sicurezza;
- criptare con SEcube™ e custodire al suo interno anche i dati relativi all'autenticazione dell'utente necessaria per l'accesso alle API di Telegram, informazioni che nella versione attuale vengono protette da una chiave inserita manualmente dall'utente e memorizzati nella memoria del dispositivo host;
- utilizzare le funzioni di crittografia del SEcube™ per proteggere, oltre ai messaggi, anche altre informazioni che normalmente si trovano all'interno

dei server delle applicazioni di messaggistica, fra cui i nomi delle chat e degli utenti con cui si condivide un gruppo e le informazioni legate al proprio profilo;

- implementare la visualizzazione e l'invio anche di altri tipi di messaggio oltre a quelli testuali, alle immagini e ai documenti, per esempio sviluppando un player per riprodurre messaggi vocali, file audio e video;
- estendere la portabilità dell'applicazione anche ad altri sistemi operativi e sviluppando una versione mobile.

Bibliografia

[1] Amnesty International. How private are your favourite messaging apps? *Amnesty International*, 10 2016. Available online at https://www.amnesty.org/en/latest/campaigns/2016/10/which-messaging-apps-best-protect-your-privacy/.

[2] J. Clement. Most popular global mobile messaging apps 2020. *statista.com*, 10 2020. Available online at https://www.statista.com/statistics/258749/most-popular-global-mobile-messenger-apps/.

[3] Kevin Collier. As trump bans wechat, some in china turn to encrypted messaging app signal. *NBC News*, 8 2020. Available online at https://www.nbcnews.com/tech/security/trump-bans-wechat-some-china-turn-encrypted-messaging-app-signal-n1236184.

[4] Pavel Durov. Why isn't telegram end-to-end encrypted by default? *Telegra.ph*, 8 2017. Available online at https://telegra.ph/Why-Isnt-Telegram-End-to-End-Encrypted-by-Default-08-14.

[5] Facebook Inc. Available online at https://www.facebook.com/security/advisories/cve-2019-3568.

[6] Facebook Inc. Messenger secret conversations technical whitepaper. Technical report, Facebook Inc., 5 2017. Available online at https://fbnewsroomus.files.wordpress.com/2016/07/messenger-secret-conversations-technical-whitepaper.pdf.

[7] Manisha Ganguly. Whatsapp design feature means some encrypted messages could be read by third party. *The Guardian*, 1 2017. Available online at https://www.theguardian.com/technology/2017/jan/13/whatsapp-design-feature-encrypted-messages.

[8] Andy Greenberg. Hacker lexicon: What is end-to-end encryption? *Wired*, 11 2014. Available online at https://www.wired.com/2014/11/hacker-lexicon-end-to-end-encryption/.

[9] Mansoor Iqbal. Telegram revenue and usage statistics (2020). *businessofapps.com*, 10 2020. Available online at https://www.businessofapps.com/data/telegram-statistics/.

[10] Avijit Mallik, Abid Ahsan, Mhia Md. Zaglul Shahadat, and Jia-Chi Tsou. Man-in-the-middle-attack: Understanding in simple words. *International Journal of Data and Network Science*, 2019. Available online at http://www.growingscience.com/ijds/Vol3/ijdns_2019_10.pdf.

[11] Nicolò Maunero, Paolo Prinetto, Gianluca Roascio, and Antonio Varriale. The secube™ open security platform. Technical report, Blue5 Group, 10 2019. Available online at https://www.secube.eu/site/assets/files/1152/secube_sdk_v1_4_1_wiki_rel_009.pdf.

[12] Joseph Menn and Yeganeh Torbati. Exclusive: Hackers accessed telegram messaging accounts in iran - researchers. *Reuters*, 8 2016. Available online at https://www.reuters.com/article/us-iran-cyber-telegram-exclusive-idUSKCN10D1AM.

[13] NIST. Withdrawn nist technical series publication. Technical report, NIST, 1 2016. Available online at https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-57pt1r4.pdf.

[14] NordVPN. Is wechat safe to use? *NordVPN Blog*, 8 2020. Available online at https://nordvpn.com/it/blog/is-wechat-safe/.

[15] Kate O'Flaherty. Telegram bots have got a major problem, security researchers warn. *Forbes*, 1 2019. Available online at https://www.forbes.com/sites/kateoflahertyuk/2019/01/17/watch-out-for-telegram-bots-security-researchers-warn/?sh=475e1fc215db35.

[16] Markus Ra. Should you stop reading gizmodo right now? *Telegra.ph*, 6 2017. Available online at https://telegra.ph/Why-you-should-stop-reading-Gizmodo-right-now-Long.

[17] Signal. Available online at https://support.signal.org/hc/en-us/articles/360007459591-Signal-Profiles-and-Message-Requests.

[18] Signal. Available online at https://signal.org/blog/sealed-sender/.

[19] Mehul Srivastava. Whatsapp voice calls used to inject israeli spyware on phones. *Financial Times*, 5 2019. Available online at https://www.ft.com/content/4da1117e-756c-11e9-be7d-6d846537acab.

[20] Jay Sullivan. Messenger introduces app lock and new privacy settings. *Facebook*, 7 2020. Available online at https://about.fb.com/news/2020/07/messenger-app-lock-and-privacy-settings/.

[21] Telegram LLC. Available online at https://core.telegram.org/api/end-to-end.

[22] Telegram LLC. Available online at https://telegram.org/faq.

[23] Telegram LLC. Mtproto mobile protocol. Technical report, Telegram, 2020. Available online at https://core.telegram.org/mtproto.

[24] Liam Tung. Update whatsapp now: Bug lets snoopers put spyware on your phone with just a call. *ZDNet*, 5 2019. Available online at https://www.zdnet.com/article/update-whatsapp-now-bug-lets-snoopers-put-spyware-on-your-phone-with-just-a-call/.

[25] William Turton. Why you should stop using telegram right now. *Gizmodo*, 6 2016. Available online at https://gizmodo.com/why-you-should-stop-using-telegram-right-now-1782557415.

[26] WeChat. Available online at https://help.wechat.com/cgi-bin/micromsgbin/oshelpcenter?opcode=2&plat=1&lang=en&id=1208117b2mai1410243yyQFZ&Channel=helpcenter.

[27] WeChat. Available online at https://www.wechat.com/it/privacy_policy.html.

[28] WhatsApp Messenger. Whatsapp encryption overview technical whitepaper. Technical report, Facebook Inc., 10 2020. Available online at https://scontent.whatsapp.net/v/t39.8562-34/122249142_469857720642275_2152527586907531259_n.pdf/WA_Security_WhitePaper.pdf?ccb=2&_nc_sid=2fbf2a&_nc_ohc=91J2ix5M5noAX8dyMDT&_nc_ht=scontent.whatsapp.net&oh=9b49bd322db3685f8a3eecd5879e541a&oe=5FC7FE19.