

UNIVERSITÀ DEGLI STUDI DI SALERNO

Dipartimento di Ingegneria dell'Informazione ed
Elettrica e Matematica Applicata



Progetto di Cybersecurity

Ammendola Giovanni:	0622701167
Mignone Giovanni:	0622701104
Petrone Vincenzo:	0622701134
Terrone Luigi:	0622701071
Tramuto Luca Giuseppe:	0622701170

Anno Accademico 2019-2020

INDICE

PROPOSTA DI PROGETTO	1
TITOLO PROGETTO	1
TEMA DEL PROGETTO	1
ENTITÀ COINVOLTE	1
OBIETTIVI DEL PROGETTO	1
TASK PREVISTI	2
PROPRIETÀ E AVVERSARI	3
PROPRIETÀ DA GESTIRE	3
<i>Confidenzialità</i>	3
<i>Integrità (autenticità)</i>	3
<i>Consistenza</i>	3
DESCRIZIONE DEGLI AVVERSARI	3
PROGETTAZIONE DEL SISTEMA	5
INTRODUZIONE	5
DATI COINVOLTI	5
GENERAZIONE DELLE CHIAVI E DEGLI <i>EphID</i>	6
GESTIONE DEGLI INFETTI DA PARTE DELL'AUTORITÀ MEDICA	7
PROTOCOLLO DI COMUNICAZIONE	9
ANALISI DEL SISTEMA	11
PROPRIETÀ	11
AVVERSARI	12
IMPLEMENTAZIONE	16
ESECUZIONE	16
<i>Requisiti preliminari</i>	16
<i>Comandi di esecuzione</i>	16
IMPLEMENTAZIONE SERVER	19
<i>File di configurazione per openssl</i>	19
<i>scriptGenerazione.sh</i>	20
<i>server.py</i>	20
IMPLEMENTAZIONE CLIENT	20
<i>script_sender.py</i>	21
<i>script_receiver.py</i>	21
<i>client.py</i>	22

Proposta di progetto

Cognome	Nome	Matricola
Ammendola	Giovanni	0622701167
Mignone	Giovanni	0622701104
Petrone	Vincenzo	0622701134
Terrone	Luigi	0622701071
Tramuto	Luca Giuseppe	0622701170

Titolo Progetto

Quarantine Observance Checking via Privacy-Preserving Contact Tracing

Tema del progetto

Realizzare un sistema, basato sulla tecnologia Bluetooth Low Energy, che monitori in maniera anonima, mediante pseudonimi su smartphone, i contatti tra le persone e verifichi il rispetto delle condizioni di isolamento domiciliare da parte degli individui infetti.

Entità coinvolte

- **Cittadino non infetto:** è un cittadino che non ha ancora ricevuto alcuna comunicazione di essere infetto; viene notificato quando si stabilisce che è recentemente entrato in contatto con un cittadino infetto.
- **Cittadino infetto:** è un cittadino al quale è già arrivata la comunicazione di essere infetto. Ha l'obbligo di rispettare le condizioni di isolamento domiciliare. D'ora in avanti ci si riferirà alle persone che condividono spazi abitativi con tali cittadini come "conviventi".
- **Laboratorio analisi:** è il posto in cui può recarsi un cittadino per effettuare dei test e verificare la positività al virus.
- **Governo:** gestisce un server che permette di coordinare le informazioni collezionate dagli smartphone dei cittadini, con particolare attenzione alle attività di quelli infetti. Coordina anche i laboratori per assistere i pazienti nelle loro operazioni di comunicazione al sistema della loro positività.

Obiettivi del progetto

Il sistema sarà strutturato affinché si possa:

1. Monitorare i contatti tra cittadini, in modo tale che il governo possa notificare questi ultimi, tramite i dati raccolti nel server e le analisi di laboratorio, che si è entrati in contatto con un cittadino risultato infetto.
2. Monitorare in particolare i contatti dei cittadini infetti durante il periodo di

Proposta di progetto

isolamento domiciliare, in modo da verificare che questi ultimi non violino la quarantena tutelando la salute degli altri cittadini, senza l'utilizzo di tracciamento tramite geolocalizzazione. Il sistema dovrà inoltre evitare di segnalare "falsi allarmi" causati da contatti con eventuali familiari residenti nella stessa abitazione.

3. Evitare che individui non autorizzati possano accedere ai dati sensibili di tutti gli altri cittadini, siano essi infetti o meno.

Task previsti

1. Descrizione dettagliata (ma non completamente formale) delle proprietà di completezza, sicurezza e privacy desiderate.
2. Progettazione di un protocollo per la comunicazione tra smartphone, evidenziando le differenze tra:
 - a. comunicazione tra un cittadino infetto e i suoi eventuali familiari.
 - b. comunicazione tra un cittadino infetto ed estranei.
3. Progettazione di un protocollo per la comunicazione tra smartphone e server, evidenziando le differenze in caso di contatto con individui infetti.
4. Analisi euristica ma sufficientemente ricca di dettagli che validi la bontà della progettazione rispetto alle proprietà elencate nel punto 1.
5. Implementazione del meccanismo di generazione degli pseudonimi.
6. Implementazione del meccanismo di notifica al server governativo di avvenuto contatto con un cittadino infetto.

Proprietà e avversari

Proprietà da gestire

Confidenzialità

- Un avversario non deve essere in grado di risalire alle identità degli individui.
- I messaggi scambiati verso altri dispositivi non devono contenere informazioni confidenziali circa gli utenti.
- Un avversario non deve essere in grado di leggere i dati in chiaro conservati dal backend circa i pazienti infetti.

Integrità (autenticità)

- Un avversario non deve essere in grado di fingersi il backend, in quanto questo causerebbe la segnalazione di falsi individui infetti verso gli altri utenti.
- Un avversario non deve essere in grado di fingersi un infetto, in quanto questo causerebbe una falsa segnalazione di quarantena violata.
- Un avversario non infetto si assume non possa segnalare una violazione della quarantena inesistente.

Consistenza

- I dispositivi di tutti gli utenti del sistema devono essere in possesso delle stesse informazioni circa i pazienti infetti.

Descrizione degli avversari

1. Gli attacchi che si basano su vulnerabilità note dei protocolli di trasmissione e cifratura si suppongono essere mitigati dagli strumenti e librerie utilizzati, quali, ad esempio, il timing attack.
2. Un paziente risultato infetto che deve rispettare l'isolamento domiciliare, potrebbe violare tale imposizione senza essere scoperto, spegnendo o lasciando il suo smartphone lontano da sé.
3. Un paziente infetto potrebbe simulare un contatto con un convivente quando invece è entrato in contatto con estranei (dunque violando la quarantena).
4. **Replay attack:** Un attaccante potrebbe fingersi un paziente infetto e simulare la violazione della quarantena da parte sua.
5. **False report attack:** Un attaccante potrebbe segnalare false violazioni della quarantena.
6. **Spam attack o DDoS:** un avversario potrebbe inviare un numero elevato di falsi match con lo scopo di mandare down il server.
7. **Relay attack:** in quanto sono necessarie delle precisazioni tecniche circa il funzionamento del sistema, questo attacco sarà descritto nei [paragrafi successivi](#).

8. **Sybil attack:** poiché il sistema si basa su generazione di pseudonimi, un avversario potrebbe simulare contatti con infetti generando un numero di pseudonimi a piacere.
9. **Ghost attack:** un infetto potrebbe usare una versione parallela dell'applicazione in modo tale da non comportarsi nella maniera attesa.

Progettazione del sistema

Introduzione

Per effettuare il tracciamento della prossimità, gli smartphone generano localmente identificatori effimeri (*EphID*), che cambiano frequentemente durante la giornata, e li trasmettono tramite tecnologia Bluetooth Low Energy. Gli altri smartphone osservano questi *EphID* e, se l'intensità del segnale supera una certa soglia, li memorizzano insieme con la durata e un'indicazione approssimativa del tempo in cui sono stati ricevuti. Il processo di tracciamento della prossimità è supportato da un server back-end che condivide informazioni di contatto anonime con l'app in esecuzione su ciascun telefono. Gli altri smartphone interrogano periodicamente il back-end per recuperare queste informazioni e ricostruiscono localmente gli *EphID* corrispondenti dei pazienti infetti. Se lo smartphone ha archiviato un record di uno di questi *EphID* infetti, ciò vuol dire che l'utente dello smartphone è stato in contatto con una persona infetta.

Per effettuare la verifica del rispetto della quarantena da parte dei pazienti infetti, ogni smartphone memorizza localmente le chiavi dei pazienti infetti (ottenute tramite interrogazioni al back-end come precedentemente spiegato). Inoltre, memorizza le chiavi dei familiari infetti, vale a dire le persone con le quali si trova a stretto contatto. In questo modo, ogni smartphone verifica se è capace di ricostruire un *EphID* corrispondente a una certa chiave di un paziente infetto non familiare: in questo caso, può comunicare al back-end che tale chiave corrisponde a una violazione della quarantena.

Oltre al back-end del sistema, si suppone che esista un server gestito dall'autorità medica che memorizza in un database locale le associazioni tra le chiavi dei pazienti infetti e l'identità dei pazienti stessi, al fine di poter gestire eventuali segnalazioni, da parte degli utenti, di violazione della quarantena da parte di un infetto. Si supponga che tale server, in quanto gestito dall'autorità medica governativa, sia una root CA che possa certificare il back-end e i pazienti infetti.

Dati coinvolti

- SK_t
 - chiave relativa al giorno t
- $n = 256$
 - numero di bit della chiave SK
- $L = 10$
 - numero di minuti dopo i quali cambia l' $EphID_i$
- $N = 24 \cdot 60 / L$
 - numero di *EphID* generati durante una giornata (in 24 ore)
- $EphID_i$

- l' i -esimo $EphID$ generato durante il giorno corrente ($i = 1, 2, \dots, N$)
- $TH = -52 \text{ dB}$
 - valore dell' $RSSI$ (intensità di un segnale BLE) oltre il quale un $EphID$ è definito sufficientemente prossimo da determinare un contatto (circa 1 metro)
- $BroadcastKey$
 - una stringa pubblica fissata dal protocollo di $N \cdot 128$ bit
- $h = 4$
 - numero di ore per cui uno smartphone di un cittadino non infetto verifica se ha incontrato un cittadino infetto

Generazione delle chiavi e degli $EphID$

1. Sia SK_0 la chiave relativa al giorno 0, inteso come il primo giorno di utilizzo del sistema. Essa viene generata casualmente come stringa di n bit da ogni smartphone che utilizza il sistema.
2. Sia la chiave SK_t relativa al giorno t , calcolabile a partire da quella relativa al giorno precedente, come $SK_t = H(SK_{t-1})$.
 - a. H è una CRHF, nello specifico SHA-256.
3. All'inizio di ogni giorno t , ogni smartphone genera localmente una lista di N nuovi $EphID$ da trasmettere durante il giorno t . Data la chiave SK_t , ogni dispositivo calcola $EphID_1 || EphID_2 || \dots || EphID_N = PRG(PRF(SK_t, BroadcastKey))$ e trasmetterà $EphID_1$ durante i primi L minuti della giornata, $EphID_2$ durante i successivi L minuti e così via per il resto del giorno t .
 - a. PRG è un cifrario a blocchi, nello specifico AES-256-CBC.
 - i. Dunque ogni $EphID_i$ sarà un blocco di 128 bit (la lunghezza dell'output di tale cifrario); questo giustifica perché la taglia $BroadcastKey$ è $N \cdot 128$ bit.
 - ii. Le specifiche di AES-256-CBC sono consultabili nel capitolo 5 del seguente documento: [FIPS 197: Advanced Encryption Standard \(AES\)](#).
 - b. PRF è una funzione pseudo-casuale, che può essere modellata secondo un Random Oracle attraverso una Keyed Hash Function H su chiave SK_t e input $BroadcastKey$: $H_{SK_t}(BroadcastKey) = H(SK_t || BroadcastKey)$.
 - c. La taglia di ogni $EphID_i$ è 16 byte.
4. Per mitigare un replay attack, un dispositivo di un cittadino dichiarato infetto trasmetterà non solo $NEphID_i$, ma accompagnerà a ciascuno di essi una firma che certificherà che il messaggio provenga effettivamente da lui.
 - a. Invece, un dispositivo di un cittadino non ancora dichiarato infetto trasmetterà una firma contenente soli 0, in quanto non è strettamente necessario, nel contesto del controllo della quarantena, che gli utenti non infetti certifichino l'autenticità dei propri messaggi.

- b. La firma avviene tramite lo schema di firma digitale Elliptic Curve Digital Signature Algorithm (ECDSA), il cui il messaggio da firmare è un $EphID_i$.
 - i. In questo modo viene seguito l'approccio *encrypt-then-authenticate* per la cifratura autenticata.
 - ii. Le specifiche di ECDSA sono consultabili nel capitolo 6 del seguente documento: [FIPS 186-4: Digital Signature Standard \(DSS\)](#).
 - iii. Le motivazioni per cui è stato scelto l'algoritmo ECDSA sono la conseguenza del confronto con due possibili alternative: Rivest–Shamir–Adleman (RSA) e Digital Signature Algorithm (DSA). Lo svantaggio di RSA è la produzione di una firma di 256 byte, incompatibile con le dimensioni supportate dallo standard BLE, mentre lo svantaggio di DSA è la produzione di una chiave pubblica di 1184 byte (parametri p , q , g e y), a fronte della lunghezza della chiave pubblica per ECDSA che è di 64 byte (coordinate x e y).
- c. La taglia della firma è 64 byte.
 - i. Dunque il messaggio contenuto nel pacchetto BLE ha una taglia totale di 96 byte, in quanto tale pacchetto è composto dall' IV del cifrario AES-256-CBC (16 byte), dall' $EphID$ (16 byte) e dalla firma (64 byte). Nonostante la massima dimensione di un pacchetto BLE sia di 20 byte, questo può essere risolto a livello applicazione; si consulti, ad esempio, la documentazione della classe [android.bluetooth.BluetoothGatt](#). In più, lo standard Bluetooth 4.2 supporta un payload nei pacchetti BLE di 251 byte (si consulti, ad esempio, [questo link](#)).
- 5. Ogni smartphone che riceve un $EphID$ con $RSSI$ superiore a TH , lo memorizza in un database locale.

Gestione degli infetti da parte dell'autorità medica

Segue una descrizione di come un paziente, una volta dichiarato infetto, possa comunicare ai relativi conviventi la propria chiave; questi passi fungono da impostazione preliminare per la specializzazione del protocollo per distinguere tra i contatti con infetti conviventi e quelli con infetti estranei.

1. Il paziente P_i viene dichiarato infetto dall'autorità medica il giorno t .
2. Il paziente P_i richiede il rilascio di un certificato digitale in modo da poter associare alla sua identità una coppia di chiavi (pubblica e privata) da infetto; sia la chiave pubblica $PubK_i$ e sia la chiave privata $PrivK_i$:
 - a. Tale coppia di chiavi viene generata seguendo l'algoritmo di generazione delle chiavi ECC (Elliptic Curve Cryptography), adatte ad essere utilizzate per l'algoritmo di firma ECDSA.
 - b. Poiché la chiave pubblica $PubK_i$ è caratterizzata dalle coordinate x e y

- (dalla taglia di 32 byte ciascuna), dunque è rappresentabile attraverso 64 byte, ma il cifrario utilizzato per generare $EphID$ richiede una chiave dalla taglia di 32 byte, la relativa nuova SK_i viene generata come $H(x || y)$.
- c. L'autorità medica firma un nuovo certificato digitale per P_i in modo da riconoscere la sua identità da infetto.
 - i. Sia tale certificato CP_i .
 - ii. All'interno di CP_i la chiave $PubK_i$ verrà salvata come chiave pubblica di P_i , in quanto verrà comunicata a tutti gli altri utenti del sistema, come specificato più avanti al punto 7.
 - d. Si noti che, tuttavia, SK_i verrà utilizzata da P_i solo per generare nuovi $EphID$, mentre per la firma specificata nel [paragrafo precedente](#) al punto 4 verrà utilizzata $PrivK_i$, tenuta segreta da P_i .
3. L'autorità medica chiede a P_i di dichiarare con quante persone condivide la sua abitazione; sia tale numero N_i .
 4. L'autorità medica salva nel proprio database l'associazione tra $PubK_i$, CP_i e N_i .
 - a. $PubK_i$ viene salvata in chiaro, in quanto nota a tutti gli utenti e non contenente informazioni sensibili circa P_i .
 - b. Invece, CP_i e N_i vengono salvati in maniera crittografata in quanto sono informazioni confidenziali.
 5. L'autorità medica crea un codice QR, sia esso QR_i , scansionabile al massimo N_i volte che, una volta scannerizzato, permette di ottenere $PubK_i$.
 - a. QR_i viene generato a partire dalla concatenazione di $PubK_i$ e una stringa nota unicamente al server (sia essa $salt_i$), ovvero $H(salt_i || PubK_i)$.
 - i. Seguendo la logica dell'utilizzo comune della stringa "salt" in crittografia, tale stringa è diversa per ogni utente.
 - b. Il server salva l'associazione tra $PubK_i$ e $H(salt_i || PubK_i)$.
 6. P_i fa scannerizzare QR_i ai suoi N_i conviventi.
 - a. Alla scansione di QR_i da parte di uno dei conviventi, il server back-end riceve una richiesta di ottenere $PubK_i$, richiesta resa univoca appunto dal calcolo di $H(salt_i || PubK_i)$.
 - i. Contestualmente, il server back-end inoltra la richiesta al server autoritativo, il quale a sua volta decrementa N_i di un'unità.
 - ii. Quando N_i raggiunge 0, il server back-end non accetterà più messaggi di richiesta con contenuto $H(salt_i || PubK_i)$.
 - b. Dunque, ciascuno degli N_i conviventi otterrà $PubK_i$ in modo da non segnalare violazioni della quarantena da parte del familiare infetto.
 7. Ogni volta che un nuovo paziente viene dichiarato infetto, il server dell'autorità medica invia al server dell'applicazione la chiave pubblica $PubK_i$ del paziente infetto. A questo punto il server dell'applicazione invia a tutti i dispositivi, previa interrogazione come accennato [nell'introduzione](#), $PubK_i$ tramite protocollo HTTPS, insieme al giorno t in cui P_i è stato dichiarato infetto. In più, il server backend richiede da P_i la sua chiave SK_{t-14} , vale a dire la chiave utilizzata per

generare pacchetti 14 giorni prima del giorno in cui è stato dichiarato infetto.

- a. Ogni utente, nota $PubK_i$ (dunque noti i relativi parametri x e y), può dunque calcolare $SK_i = H(x || y)$.
8. Ogni giorno il back-end aggiorna le chiavi di tutti gli infetti come descritto nel [paragrafo precedente](#) (punto 2).
9. Una volta che termina il periodo di quarantena da parte di un certo individuo infetto P_i , l'autorità medica comunica al server applicativo la rimozione della chiave $PubK_i$. Il server applicativo, dopo aver effettuato tale modifica sul suo database locale, comunica in broadcast a tutti i dispositivi di effettuare un aggiornamento locale dei loro database e rimuovere la chiave $PubK_i$.

Protocollo di comunicazione

A partire dalla comunicazione degli $EphID$ tra smartphone e dalla comunicazione degli infetti da parte dell'autorità medica descritte nei paragrafi precedenti, segue una descrizione della specializzazione del protocollo per distinguere i contatti tra pazienti infetti e non, e quelli tra infetti conviventi e infetti estranei.

1. Ogni dispositivo ha memorizzato localmente in file distinti:
 - a. le chiavi di tutti gli infetti del Paese di residenza; sia l'insieme di queste chiavi Z .
 - b. le chiavi dei propri conviventi infetti, ottenute tramite la scansione dei rispettivi codici QR; sia C l'insieme di queste chiavi.
 - c. gli $EphID$ degli smartphone che hanno dimostrato un $RSSI$ superiore a TH .
2. Con cadenza periodica, ogni h ore, ogni dispositivo verifica se ha memorizzato un $EphID$ generabile da una chiave SK_i in $Z \setminus C$.
 - a. Come esempio, si supponga che un dispositivo abbia memorizzato $EphID_a$, $EphID_b$ e $EphID_c$, e che tra questi solo $EphID_c$ sia generabile dalla chiave di un infetto (sia essa SK_c), dunque l'unica chiave presente in Z . Il dispositivo calcolerà che $EphID_a$ e $EphID_b$ non sono generabili da nessuna chiave in Z , quindi non leggerà mai la relativa firma (che, come descritto nel paragrafo precedente, non si tratta in realtà di una stringa ben formata); invece, verificherà che $EphID_c$ è effettivamente calcolabile da $SK_c \in Z$, quindi eseguirà la verifica della relativa firma (che in questo caso sarà una firma valida).
3. Nel caso in cui un dispositivo riconosca che un pacchetto $EphID_i$ è generabile dalla chiave di un infetto SK_i , ciò vuol dire che il paziente P_i ha violato la quarantena: quindi il dispositivo, per notificare tale evento, invia al server, tramite HTTPS:
 - a. La chiave SK_i ;
 - b. Il pacchetto $EphID_i$;
 - c. La relativa firma ricevuta con il pacchetto $EphID_i$.

4. Il server back-end, alla ricezione di tali dati da parte di un utente, esegue la verifica dell'autenticità di $EphID_i$, dopodiché:
 - a. in caso positivo, invia SK_i al server autoritativo.
 - i. L'autorità medica procede all'identificazione dell'infecto P_i e alla segnalazione all'autorità giudiziaria.
 - ii. Sarà compito dell'autorità giudiziaria effettuare i dovuti controlli e, confermata la violazione, procedere secondo le normative vigenti.
 - b. in caso negativo, ignora i dati ricevuti.
5. Ogni giorno il dispositivo di ogni cittadino aggiorna la sua chiave e tutte le chiavi in Z come descritto nel paragrafo [Generazione delle chiavi e degli EphID](#) (punto 2).

Analisi del sistema

Proprietà

- **Confidenzialità circa l'identità degli infetti:** Un avversario non deve essere in grado di leggere i dati in chiaro conservati dal server autoritativo circa i pazienti infetti.
 - Ciò è garantito dal fatto che il database crea una corrispondenza tra la chiave di ogni cittadino infetto (si nota che queste chiavi sono note a tutti) e la cifratura dei dati confidenziali del paziente. Dunque i dati confidenziali del paziente sono ricavabili rompendo solo lo schema di cifratura usato dal server dell'autorità medica, o ad eseguire un attacco di forza bruta della complessità computazionale nell'ordine di 2^n , con n la lunghezza in bit della chiave di cifratura utilizzata dal server autoritativo.
Si richiama, a tal proposito, al paragrafo [Progettazione del sistema: Gestione degli infetti da parte dell'autorità medica](#) (Punto 4).
- **Integrità dei messaggi da/verso il server autoritativo:**
 - Tale proprietà è garantita dal fatto che il server autoritativo svolge il ruolo di root CA in grado di fornire certificati. La connessione al server back-end avviene tramite protocollo HTTPS e lo scambio di messaggi è vincolato all'utilizzo di un certificato rilasciato dall'autorità medica al back-end stesso. Si richiama, a tal proposito, al paragrafo [Progettazione del sistema: Introduzione](#).
- **Confidenzialità dei messaggi tra gli utenti:**
 - Tale proprietà è garantita dal fatto che ogni messaggio è calcolato come output di un cifrario a blocchi CPA-sicuro (sia esso c), accompagnato da una firma calcolata a partire da c . Questa costruzione è una funzione di cifratura CCA-sicura, che mitiga anche attacchi di malleabilità sugli *EphID* generabili da pazienti infetti.
 - Tuttavia, la sicurezza CCA è relativa unicamente ai messaggi provenienti da cittadini infetti, in quanto la verifica della firma ha effetto solo in questo caso; infatti, i messaggi provenienti da cittadini non infetti, in quanto non accompagnati da una firma ben formata, sono l'output di una funzione di cifratura CPA-sicura, ma non CCA-sicura.
 - Si richiama, a tal proposito, al paragrafo [Progettazione del sistema: Generazione delle chiavi e degli EphID](#) (Punto 3).
 - Inoltre, si fa notare come l'algoritmo, basandosi su pacchetti generabili tramite pseudonimi, non prevede la trasmissione di dati confidenziali.
- **Consistenza:**

- I dati condivisi dagli utenti consistono principalmente nelle chiavi SK degli infetti, che vengono scaricate ogni giorno. Nonostante sia previsto che l'operazione di aggiornamento venga eseguita da tutti gli utenti del sistema alla stessa ora, non si deve escludere che qualche utente possa lasciare il dispositivo spento o disconnesso, ritardando l'aggiornamento delle nuove chiavi infette.

Il mancato aggiornamento delle chiavi degli infetti potrebbe causare due problemi per un cittadino non infetto:

- Nel caso in cui il dispositivo rilevi un $EphID$ generabile da una nuova chiave SK , ma l'elenco dei nuovi infetti non è stato aggiornato (per via di una disconnessione dalla rete), il cittadino non verrà immediatamente notificato del possibile contagio, ma tale notifica arriverà non appena l'elenco delle SK degli infetti sarà aggiornato.
 - Dunque, questo scenario non rappresenta un problema di inconsistenza in quanto il contatto, seppur con ritardo, verrà notificato.
- Restando nell'ipotesi che l'elenco delle chiavi degli infetti non sia stato aggiornato per alcuni giorni, un dispositivo potrebbe rilevare un $EphID$ generabile da un SK , la quale però è stata rimossa dall'elenco degli infetti. Tale problematica viene risolta dal server, che controllerà che la chiave non appartiene più ad un infetto.

Avversari

- **Replay attack:** Un attaccante potrebbe fingersi un paziente infetto e simulare la violazione della quarantena da parte sua.
 - Tale attacco è mitigato dal fatto che un dispositivo di un cittadino infetto trasmetterà non solo $NEphID_i$, ma accompagnerà a ciascun $EphID_i$ una firma per certificare la propria identità.
 - Dunque un avversario che tenta un replay attack (sia esso A_{RA}), come tutti gli utenti è in possesso della chiave SK degli infetti, dunque può generare $EphID$ validi.
 - Tuttavia, A_{RA} dovrebbe anche generare una firma valida, il che equivale a rompere lo schema di firma digitale ECDSA o ad eseguire un attacco di forza bruta della complessità computazionale nell'ordine di 2^{512} , essendo la firma lunga 64 byte (512 bit).
 - Si richiama, a tal proposito, al paragrafo [Progettazione del sistema: Generazione delle chiavi e degli \$EphID\$](#) (Punto 4).
- **False report attack:** Un attaccante potrebbe segnalare false violazioni della quarantena.

- Tale attacco è mitigato dal fatto che un dispositivo di un cittadino che vuole segnalare che l'infecto P_i abbia violato la quarantena trasmetterà non solo la chiave SK_i , ma anche l' $EphID_i$ e la relativa firma.
- Dunque un avversario che tenta un false report attack (sia esso A_{FRA}), come tutti gli utenti è in possesso della chiave SK degli infecti, dunque può generare $EphID$ validi.
- Tuttavia, per effettuare una segnalazione valida al server, A_{FRA} dovrebbe anche generare una firma valida, il che equivale a rompere lo schema di firma digitale ECDSA o ad eseguire un attacco di forza bruta della complessità computazionale nell'ordine di 2^{512} , essendo la firma lunga 64 byte (512 bit).
- Si richiama, a tal proposito, al paragrafo [Progettazione del sistema: Protocollo di comunicazione](#) (Punto 3).
- **Spam attack o DDoS:** un avversario potrebbe inviare un numero elevato di falsi match solo per lo scopo di mandare down il server.
 - Tale attacco può essere mitigato impostando una soglia di messaggi massimi che un utente può inviare al server in un certo intervallo di tempo oppure fare in modo che la procedura di invio messaggi al server da parte del dispositivo non sia possibile in modo arbitrario da parte dell'utente.
- **Utente che spegne il proprio dispositivo:**
 - Purtroppo tale comportamento da parte dei cittadini è impossibile da prevenire senza limitare le libertà personali degli individui. L'applicazione si affida al buon senso da parte degli utenti.
- **Utente infecto che fa scannerizzare il proprio codice QR a persone non conviventi:**
 - Scansionare il codice QR di un infecto implica che la persona che ha effettuato la scansione non notificherà al server la violazione della quarantena e soprattutto non riceverà alcuna notifica di un possibile contagio. Tale aspetto rende immediato intuire che la scansione di un codice QR di una persona non convivente, vada contro i propri interessi, mettendo a rischio la propria salute.
 - Nonostante tali considerazioni, questo comportamento viene mitigato tramite una convalida del codice appena scansionato da parte del server. Completata la scansione, viene inviata una richiesta al server per verificare che non sia stato superato il numero massimo di utilizzi del codice.
- **Relay attack:** Il suddetto attacco è realizzabile da parte di un possibile avversario A che, avendo avuto o meno contatti con una persona infecta M , può intercettare i pacchetti BLE inviati dal dispositivo di M (contenenti $EphID$ e firma di M) e re-indirizzare questi pacchetti ad altri utenti facendo risultare che M abbia violato la quarantena.

- Nell'ipotesi che tale intercettazione sia la conseguenza di un effettivo contatto con M , tale attacco produce come risultato la segnalazione da parte di un altro utente del sistema; segnalazione che è ritenuta valida in quanto M è effettivamente entrato in contatto con un estraneo.
- Nell'ipotesi invece che tale intercettazione sia la conseguenza di una modifica della soglia TH o dell'utilizzo di strumenti non previsti dal sistema in grado di rilevare segnali sopra la soglia minima, tale attacco è di fatto efficace.
 - L'attacco potrebbe essere mitigato modificando il protocollo di comunicazione tra gli utenti, includendo nel pacchetto BLE la codifica della propria posizione tramite GPS, e prevedendo che chi riceve il pacchetto salvi anche la posizione in cui l'ha ricevuto. Un esempio di questa implementazione è disponibile a [questo link](#) (capitolo 1 per una descrizione informale, capitolo 3 per i dettagli tecnici).
- **Sybil attack:** poiché il sistema si basa su generazione di pseudonimi, un avversario potrebbe simulare contatti con infetti generando un numero di pseudonimi a piacere.
 - Un attacco che consiste nel generare *EphID* casuali nel tentativo di simulare il comportamento di un infetto non ha senso, in quanto questi sono calcolabili deterministicamente nota una SK pubblica. In questo contesto, ha più senso tentare un Replay Attack.
 - Un attacco del genere potrebbe risultare problematico nel contesto del calcolo del rischio di infezione da parte di pazienti non infetti. Si supponga che un utente x riceva un *EphID* prodotto da un avversario che tenta un Sybil Attack; si supponga che uno di questi *EphID* sia generabile da una chiave SK_a appartenente a un utente P_a non ancora dichiarato infetto. Nell'ipotesi in cui P_a dovesse essere dichiarato infetto, in questo scenario x calcolerà che l'*EphID* è generabile a partire dalla chiave di P_a (si ricorda a tal proposito che quando un paziente viene dichiarato infetto nel giorno t , viene comunicata a tutti gli utenti anche la chiave SK_{t-14} utilizzata a partire da 14 giorni precedenti t , come specificato nel paragrafo [Gestione degli infetti da parte dell'autorità medica](#)). Dunque x risulterà un paziente potenzialmente a rischio di contagio.
 - In questo caso, l'attacco risulterà efficace.
 - Tuttavia, la generazione di un *EphID* generabile a partire da una particolare SK richiede anche la generazione di un IV valido. Essendo le taglie di *EphID* e IV di 16 byte (128 bit) ciascuno, un attacco del genere ha successo con probabilità $\frac{\#Infetti}{2^{256}}$.
 - Anche supponendo che il numero degli infetti sia l'attuale popolazione mondiale (arrotondata per eccesso a 8

miliardi), la probabilità che una coppia casuale ($EphID, IV$) sia generabile da una delle chiavi degli infetti è 7×10^{-68} .

- **Ghost attack:** un infetto potrebbe usare una versione parallela dell'applicazione in modo tale da non comportarsi nella maniera attesa, cambiando il comportamento dell'applicazione stessa.
 - Nello specifico quest'ultimo può fare in modo che la firma associata al suo $EphID$ non venga inviata, riuscendo quindi a replicare il comportamento da utente non infetto, evitando di essere rilevato nonostante sia stata violata la quarantena.
 - Questo tipo di attacco non è mitigabile, ma in ogni caso produce il medesimo risultato di un avversario che spegne il proprio telefono o disattiva l'utilizzo del Bluetooth, risultando come un utente "fantasma" che riesce a violare le condizioni imposte di quarantena.

Implementazione

L'implementazione client-server si compone di due parti:

1. Lato client: a sua volta ha due componenti, uno script da eseguire per inviare pacchetti e uno da eseguire per verificare i pacchetti ricevuti, al fine di trovare un paziente che abbia violato la quarantena:
 - a. Script *sender*: genera un $EphID_i$ e, se infetto, genera anche la chiave pubblica.
 - b. Script *receiver*: per ogni chiave pubblica SK_i , verifica quale pacchetto è generabile a partire da SK_i ; in caso di riscontro positivo, dopo aver verificato il certificato del server, invia SK_i , $EphID_i$ e relativa firma al server stesso per notificare che il paziente con chiave pubblica SK_i ha violato la quarantena.
2. Lato server: si occupa della ricezione dei dati da parte dei client e verifica che la firma corrisponda all' $EphID_i$ generato da una SK_i .

Esecuzione

Requisiti preliminari

Per simulare correttamente il sistema, occorre utilizzare un sistema Linux; inoltre è necessario che siano installati OpenSSL e la versione di Python 3.8.2.

Comandi di esecuzione

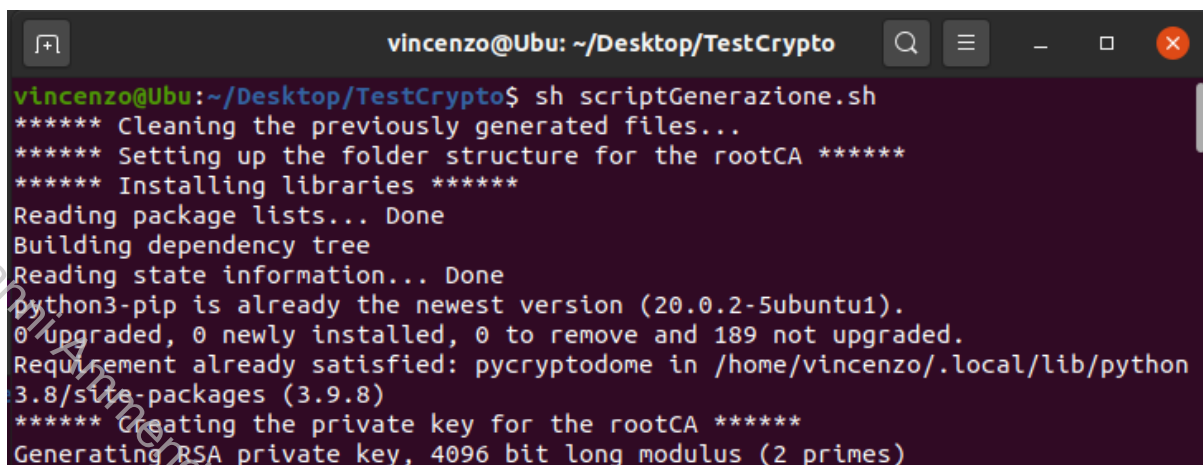
Per simulare correttamente il sistema, eseguire i seguenti passi:

1. Aprire un terminale nella cartella base del progetto (dove è presente il file *scriptGenerazione.sh*).
2. Eseguire lo script bash di generazione con il comando `sh scriptGenerazione.sh`. Durante l'esecuzione dello script verrà richiesta la password dell'utente per l'installazione delle librerie necessarie. Dopo l'installazione delle librerie, verranno generati i certificati delle varie entità coinvolte (Vedere Figura1).
3. Chiudere il terminale.
4. Aprire un nuovo terminale nella sottocartella *server*.
5. Aprire un nuovo terminale nella sottocartella *sender*.
6. Aprire un nuovo terminale nella sottocartella *receiver*.
7. Nel terminale dedicato al server, eseguire lo script python con il comando `python3 server.py`. Questo avvierà il server, che sarà in attesa di comunicazioni da parte del client (Vedere Figura2).
8. Nel terminale dedicato al sender è possibile simulare il comportamento di un cittadino infetto o non infetto:
 - a. per simulare il comportamento di un cittadino non infetto eseguire lo script python con il comando `python3 sender.py 0`. Questo avvierà

la routine eseguita da un cittadino non infetto: vale a dire che il receiver riceverà un $EphID_i$ non firmato (Vedere Figura3).

- b. per simulare il comportamento di un cittadino infetto eseguire lo script python con il comando `python3 sender.py 1`. Questo avvierà la routine eseguita da un cittadino infetto: vale a dire che il receiver riceverà un $EphID_i$ firmato (Vedere Figura4).
9. Nel terminale dedicato al receiver è possibile simulare il comportamento di un utente onesto o di un avversario:
- a. per simulare il comportamento di un utente onesto, eseguire lo script python con il comando `python3 script_receiver.py 0`. Questo avvierà la routine eseguita da un utente onesto: vale a dire che il receiver, avendo ricevuto un $EphID_i$ firmato da una chiave pubblica SK_i , verificherà la firma e invierà SK_i al server. Viene anche mostrato a video il numero di $EphID$ ricevuti, il numero di chiavi pubbliche ricevute, l' $EphID$ generato da un paziente infetto, il risultato della verifica della firma (`True` o `False`) e la risposta del server: in tal caso la risposta è positiva (Vedere Figura5).
 - b. per simulare il comportamento di un avversario, eseguire lo script python con il comando `python3 script_receiver.py 1`. Questo avvierà la routine eseguita da un avversario: vale a dire che il receiver, avendo ricevuto un $EphID_i$ non firmato da una chiave pubblica SK_i , tenterà di verificare una firma contraffatta, e verrà inviata SK_i al server. Viene anche mostrato a video il numero di $EphID$ ricevuti (in questo caso l' $EphID$ non è stato effettivamente ricevuto, ma nella simulazione si può ipotizzare che un avversario lo possa generare, nota SK_i), il numero di chiavi pubbliche ricevute, l' $EphID$ generato da un paziente infetto, il risultato della verifica della firma (`True` o `False`) e la risposta del server: in tal caso la risposta è negativa (Vedere Figura6).
10. Il server, dopo che riceve dei messaggi da parte di un receiver, mostra a video alcune informazioni sulla connessione, quali indirizzo IP e numero di porta utilizzati per la connessione, e informazioni circa il protocollo sicuro TLS utilizzato (Vedere Figura7).

Figure

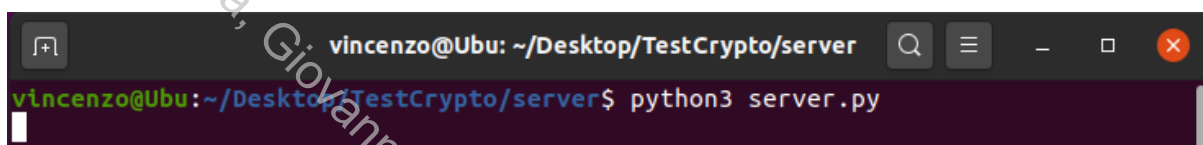


```

vincenzo@Ubu: ~/Desktop/TestCrypto
vincenzo@Ubu:~/Desktop/TestCrypto$ sh scriptGenerazione.sh
***** Cleaning the previously generated files...
***** Setting up the folder structure for the rootCA *****
***** Installing libraries *****
Reading package lists... Done
Building dependency tree
Reading state information... Done
python3-pip is already the newest version (20.0.2-5ubuntu1).
0 upgraded, 0 newly installed, 0 to remove and 189 not upgraded.
Requirement already satisfied: pycryptodome in /home/vincenzo/.local/lib/python
3.8/site-packages (3.9.8)
***** Creating the private key for the rootCA *****
Generating RSA private key, 4096 bit long modulus (2 primes)

```

Figura1

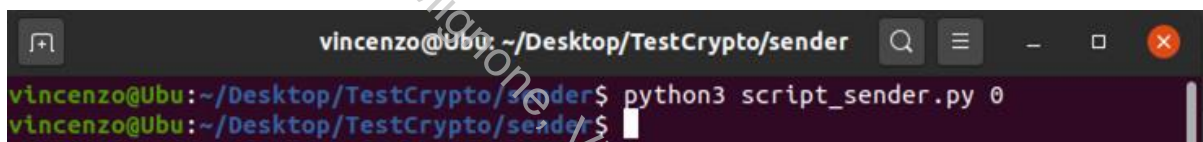


```

vincenzo@Ubu: ~/Desktop/TestCrypto/server
vincenzo@Ubu:~/Desktop/TestCrypto/server$ python3 server.py

```

Figura2

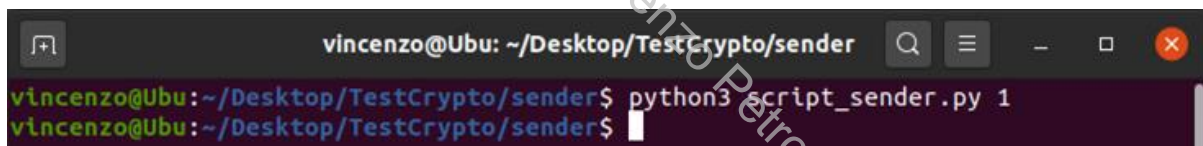


```

vincenzo@Ubu: ~/Desktop/TestCrypto/sender
vincenzo@Ubu:~/Desktop/TestCrypto/sender$ python3 script_sender.py 0
vincenzo@Ubu:~/Desktop/TestCrypto/sender$

```

Figura3

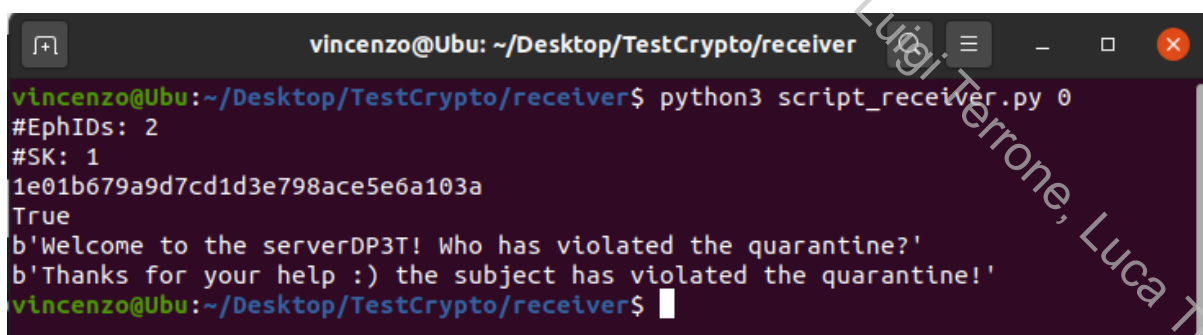


```

vincenzo@Ubu: ~/Desktop/TestCrypto/sender
vincenzo@Ubu:~/Desktop/TestCrypto/sender$ python3 script_sender.py 1
vincenzo@Ubu:~/Desktop/TestCrypto/sender$

```

Figura4

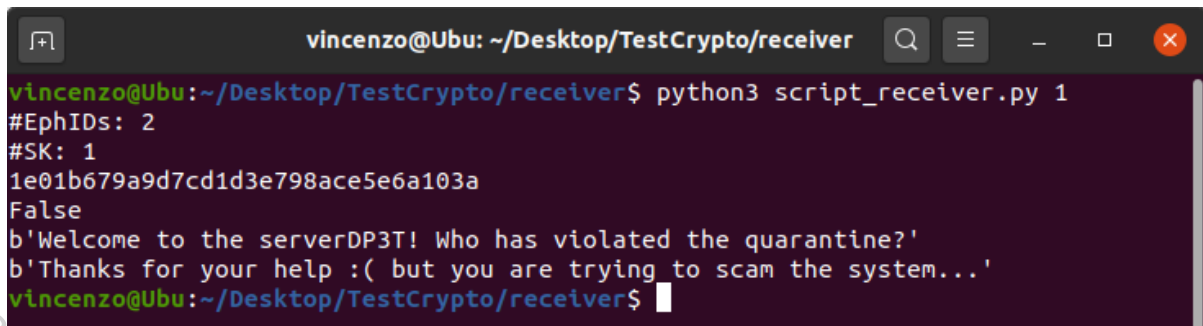


```

vincenzo@Ubu: ~/Desktop/TestCrypto/receiver
vincenzo@Ubu:~/Desktop/TestCrypto/receiver$ python3 script_receiver.py 0
#EphIDs: 2
#SK: 1
1e01b679a9d7cd1d3e798ace5e6a103a
True
b'Welcome to the serverDP3T! Who has violated the quarantine?'
b'Thanks for your help :) the subject has violated the quarantine!'
vincenzo@Ubu:~/Desktop/TestCrypto/receiver$

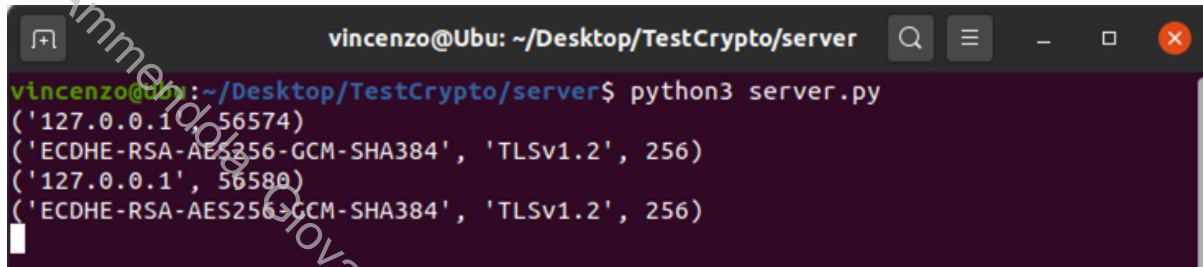
```

Figura5



```
vincenzo@Ubu: ~/Desktop/TestCrypto/receiver
vincenzo@Ubu:~/Desktop/TestCrypto/receiver$ python3 script_receiver.py 1
#EphIDs: 2
#SK: 1
1e01b679a9d7cd1d3e798ace5e6a103a
False
b'Welcome to the serverDP3T! Who has violated the quarantine?'
b'Thanks for your help :( but you are trying to scam the system...'
vincenzo@Ubu:~/Desktop/TestCrypto/receiver$
```

Figura6



```
vincenzo@Ubu: ~/Desktop/TestCrypto/server
vincenzo@Ubu:~/Desktop/TestCrypto/server$ python3 server.py
('127.0.0.1', 56574)
('ECDHE-RSA-AES256-GCM-SHA384', 'TLSv1.2', 256)
('127.0.0.1', 56580)
('ECDHE-RSA-AES256-GCM-SHA384', 'TLSv1.2', 256)
```

Figura7

Implementazione server

Per l'implementazione del lato server, sono state utilizzate le seguenti librerie di python:

- **socket**: per la creazione del socket e per accettare le connessioni;
- **ssl**: wrapper per rendere sicura la connessione del socket;
- **Crypto**: per la verifica delle firme, utilizzando gli strumenti crittografici messi a disposizione dalla libreria.
 - Documentazione: [PyCryptodome>>Docs>>API documentation](#)

Come già accennato, la comunicazione fra client e server avviene tramite una connessione sicura, utilizzando il protocollo TLS. Per la creazione dei certificati è stato creato uno script bash che prima crea una rootCA (che rappresenta il server governativo), e in seguito fornisce un certificato a una intermediateCA, che rappresenta il server applicativo che sarà utilizzato per la connessione sicura con i client, ovvero con gli utenti del sistema.

File di configurazione per openssl

Sono stati creati due file di configurazione per le due CA, basandosi sulla configurazione di default di openssl e modificando alcuni parametri: oltre a quelli relativi all'entità (ad esempio `commonName` e `countryName`) e ai percorsi con i relativi nomi dei certificati, sono stati modificati i parametri di sicurezza, impostando i seguenti valori:

- `default_bit` = 4096
- `default_md` = sha256

Implementazione

- `basicConstraints = CA:`
 - `TRUE`: nel file di configurazione della `rootCA` questo parametro è vero poiché le entità che certifica (nel nostro caso solo il server back-end, vale a dire l'`intermediateCA`) devono a loro volta essere `CA`;
 - `FALSE`: nel file di configurazione dell'`intermediateCA` questo parametro è falso perché non deve certificare nuove `CA`.

`scriptGenerazione.sh`

Lo script bash, dopo aver pulito la directory corrente ed aver creato la struttura delle cartelle necessaria al funzionamento del sistema, crea la chiave privata per la `rootCA` e autofirma il suo certificato. Viene quindi creata la struttura delle cartelle per la `intermediateCA`, la chiave privata di `intermediateCA` e la richiesta di un certificato alla `rootCA`.

Dopo la creazione dei due certificati, viene creata la catena di certificati contenente entrambi i certificati della `rootCA` e della `intermediateCA`.

Al termine della creazione, viene eseguita la verifica della suddetta catena.

Per l'esecuzione automatica dello script, i comandi di `openssl` hanno come parametro `-batch` il quale permette di utilizzare i valori di default presenti nella configurazione senza l'intervento dell'utente.

`server.py`

Il server è implementato nello script python `server.py`. Dopo l'apertura del socket `ssl`, resta in ascolto per qualche secondo (nello scenario reale, la gestione dei socket è parallelizzata) e, una volta ricevuto il messaggio, procede alla verifica. Il messaggio ricevuto è strutturato nel seguente modo:

- 32 byte per il punto x della curva ellittica;
- 32 byte per il punto y della curva ellittica;
- 16 byte per l' $EphID_i$;
- 64 byte per la firma dell' $EphID_i$ da parte di un cittadino infetto.

Il messaggio viene quindi diviso e sfruttando i metodi messi a disposizione dalla libreria `Crypto`, viene generata la chiave pubblica dell'infetto (a partire dalle coordinate (x, y) ricevute), che verrà infine utilizzata per la verifica della firma. Al termine della verifica, viene comunicato l'esito positivo o negativo.

Implementazione client

Per l'implementazione del lato client, sono state utilizzate le seguenti librerie di python:

- **socket**: per la creazione del socket ed accettare le connessioni;
- **ssl**: wrapper per rendere sicura la connessione del socket;
- **Crypto**: necessario per l'implementazione di diverse funzioni, utilizzando gli strumenti crittografici messi a disposizione dalla libreria:
 - Generazione di chiavi pubbliche e private;

Implementazione

- Generazione di *EphID* attraverso strumenti di codifica;
- Produzione e verifica di firme;
- Documentazione: [PyCryptodome>>Docs>>API documentation](#)

`script_sender.py`

Attraverso tale script è possibile simulare il comportamento di un dispositivo client che invia pacchetti Bluetooth, sia esso appartenente a un cittadino infetto o non. È possibile simulare l'invio di un *EphID* generato da una chiave di un cittadino infetto passando come parametro dello script il valore 1; è invece possibile simulare l'invio di un *EphID* generato da una chiave di un cittadino non infetto passando il valore 0 come parametro di tale script.

Dopo aver generato (o, se già presente, aggiornato) la propria *SK*, il sender produrrà *N EphID* da inviare durante il corso della giornata corrente, selezionerà il particolare *EphID_i* da inviare in broadcast, lo firmerà, e finalmente invierà un pacchetto BLE composto dall'IV (necessario al receiver per ricostruire gli *EphID*), dall'*EphID* stesso e dalla firma. Quest'ultima, come specificato in [fase di progettazione](#), sarà generata tramite algoritmo ECDSA se l'utente è infetto, mentre sarà una stringa di 0 se l'utente non è infetto.

Poiché si tratta di una simulazione, è questo stesso script a salvare il pacchetto e l'eventuale chiave pubblica in un file utile al receiver. In un contesto reale, i pacchetti vengono invece scambiati con il receiver via BLE, mentre le chiavi pubbliche vengono richieste dal receiver al server.

`script_receiver.py`

Attraverso tale script è possibile simulare il comportamento di un dispositivo client che vuole verificare se sono stati ricevuti pacchetti Bluetooth provenienti da pazienti infetti. In caso di riscontro positivo, inizia una comunicazione con il server (specificata nel [prossimo paragrafo](#)). È possibile simulare la ricezione di un *EphID* generato da un cittadino infetto onesto passando come parametro di esecuzione dello script il valore 0; è invece possibile simulare la ricezione di un *EphID* generato da un avversario che si finge un utente infetto passando il valore 1 come parametro di tale script.

Dopo aver letto (ed eventualmente aggiornato) le chiavi pubbliche e i pacchetti ricevuti dai file appropriati, verrà eseguito un ciclo in cui, per ogni chiave pubblica *SK_i* e per ogni pacchetto *EphID_i*, verrà eseguita una cifratura con chiave *SK_i* (ottenendo così tutti gli *N EphID* generabili da *SK_i*), e si verificherà se un *EphID_i* sia effettivamente generabile da *SK_i*; in caso positivo, verrà verificata la relativa firma (per evitare di segnalare pacchetti prodotti da avversari) e saranno comunicati al server *SK_i*, *EphID_i* e la firma.

`client.py`

Attraverso tale script è possibile simulare la connessione al server da parte dell'applicazione, inviando un messaggio contenente le informazioni necessarie alla verifica della violazione della quarantena. Dopo la creazione di una socket e la connessione al server, l'applicazione client procede a verificare l'identità del server tramite la verifica del certificato ricevuto. Nel caso in cui tale verifica fallisca, viene mostrato un messaggio di errore e l'esecuzione dell'applicazione viene interrotta.

Viene quindi letto e mostrato a video il messaggio di benvenuto ricevuto dal server, dopodiché un pacchetto (formato come descritto [nel paragrafo precedente](#)) viene inviato al server.

Il server, una volta effettuata la verifica, invierà un messaggio di conferma (dall'esito positivo o negativo), che verrà poi stampato a video.