Cognome	Nome	Matricola
Della Monica	Davide	0622701345
di Somma	Vincenzo	0622701283
Falanga	Armando	0622701140
Gravina	Salvatore	0622701063
Guarino	Ferdinando	0622701321

> *“The urge to destroy is also a creative urge” - Pablo Picasso*

Decentralized Privacy-Preserving Contact Tracing

Tema del progetto:

Estensione del sistema DP3T con la seguente finalità: consentire la condivisione di informazioni aggiuntive da parte dell'utente con centri di ricerca epidemiologici per fini statistici garantendo un alto livello di privacy

Entità coinvolte:

- **Cittadino non infetto**: è un cittadino al quale al momento non è arrivata comunicazione di essere infetto.
- **Cittadino infetto**: è un cittadino al quale è già arrivata la comunicazione di essere infetto.
- **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. Coordina anche i laboratori per assistere i pazienti nelle loro operazioni di comunicazione al sistema della loro positività.
- **Centri di ricerca epidemiologici**: ricevono dal sistema informazioni utili ad elaborare modelli di contagio

Obiettivi del progetto:

- Permettere agli epidemiologi di elaborare modelli più raffinati attraverso informazioni più dettagliate sui contatti fornite facoltativamente dagli utenti con il massimo livello di privacy possibile e senza rivelarne l'identità.

Task previsti:

1. Descrizione dettagliata (ma non completamente formale) delle proprietà di completezza, sicurezza e privacy desiderate.
2. Progettazione di un protocolli sicuri basati su DP3T per la comunicazione tra i dispositivi coinvolti.
3. Analisi euristica ma sufficientemente ricca di dettagli che validi la bontà della progettazione rispetto alle proprietà elencate nel punto 1).

4. Decideremo in un secondo momento cosa implementare.

Reference

- DP3T White Paper
- Privacy and Security Attacks on Digital Proximity Tracing Systems
- DP3T - Data Protection and Security

2 Discussione sulle proprietà di confidenzialità ed integrità che si intendono gestire e descrizione degli avversari.

Requisiti

- **Uso dei Dati**: La raccolta dei dati e il relativo uso devono essere limitati allo scopo della raccolta stessa: proximity tracing **e raccolta di dati per gli epidemiologi**. Ciò implica che il design del sistema debba evitare la raccolta e l'utilizzo di qualunque dato che non è direttamente correlato al compito di rilevare un contatto ravvicinato tra due utenti o utile all'elaborazione di modelli epidemiologici.
- **Inferenza Controllata**: L'inferenza di dati riguardo individui o comunità, ad esempio interazioni sociali o diagnosi mediche, deve essere mantenuta sotto controllo in modo da evitare la diffusione di informazioni sugli utenti non previste. Ogni entità autorizzata dovrebbe venire a conoscenza solo delle informazioni strettamente necessarie ad ottemperare il proprio scopo all'interno del sistema.
- **Privacy**:
  1. Gli spostamenti degli utenti non devono poter essere ricostruiti.
  2. Non deve essere possibile **inferire** informazioni riguardanti utenti che non hanno dato il consenso alla condivisione dei dati con gli epidemiologi.
  3. Non deve essere possibile **condividere** informazioni riguardanti altri utenti che non hanno dato il consenso alla condivisione dei dati con gli epidemiologi.
- **Autenticità**: I contatti notificati agli epidemiologi sono autentici, cioè non possono essere falsificati o simulati.
- **Protezione delle identità**: Dai dati condivisi dagli utenti non deve possibile risalire allo loro identità.

Possibili avversari

- **Tech-savvy user**: Può utilizzare antenne per ascoltare canali di vario genere (Bluetooth, WiFi e rete cellulare). Può de-compilare e modificare l'app. Può accedere al codice sorgente del backend.
- **Eavesdropper**: Può ascoltare le comunicazioni che avvengono sul canale (Bluetooth, WiFi e rete cellulare).

- **Epidemiologi**: Hanno accesso al server backend dei laboratori di ricerca, quindi ai dati condivisi dagli utenti.

3 Progettazione del sistema

Introduzione

L'obiettivo del progetto è quello di poter condividere altri dati con gli epidemiologi, oltre quelli già forniti da DP3T (Vedi *Contact tracing and epidemiological research* in DP3T - Data Protection and Security). Abbiamo focalizzato la nostra attenzione su informazioni contestuali ai contatti registrati in accordo al protocollo DP3T, nello specifico la posizione in cui è avvenuto il contatto e gli EphID degli utenti coinvolti con l'obiettivo di elaborare un modello della diffusione del virus. La discussione della progettazione del sistema si è svolta in maniera incrementale, dove in ogni incremento si è considerato un possibile schema di funzionamento con le relative informazioni condivise dagli utenti e i possibili attacchi dagli avversari precedentemente definiti.

Assunzioni

1. Il server di backend degli epidemiologi non mantiene dei log sui dati che riceve dall'esterno;
2. La comunicazione dei dati ai server degli epidemiologi avviene attraverso l'uso di **proxy** come consigliato per il server di backend principale in NR 1 nel documento Privacy and Security Attacks on Digital Proximity Tracing Systems
3. Le comunicazioni con il server degli epidemiologi avvengono in modo programmato anche se l'utente non ha nulla da inviare oppure non ha dato l'autorizzazione alla condivisione dei dati agli epidemiologi (simile all'assunzione fatta in *Contact tracing and epidemiological research* nel documento DP3T - Data Protection and Security)

Scenario generale

- **Eavesdropper**:
  - **Cosa può fare?** Può ascoltare le comunicazioni degli utenti sui canali (Bluetooth, WiFi e rete cellulare)
  - **Problematiche** Dipende dalle informazioni che vengono scambiate, ma in generare può cercare di scoprire informazioni riguardanti gli utenti.
- **Tech-savvy**:
  - **Cosa può fare?** Inviare dati falsi agli epidemiologi
  - **Problematiche**: Può compromettere l'autenticità dei dati forniti dagli utenti agli epidemiologi (es. false posizioni, falsi contatti, ecc...)
  - **Possibili soluzioni**: Si potrebbe richiedere una qualche forma di autenticazione agli utenti, ma ciò potrebbe risultare in una problematica in termini di anonimizzazione degli utenti, andando a violare la proprietà di **Protezione delle identità** (Vedi IR1 in Privacy and Security Attacks on Digital Proximity Tracing Systems)
- **Epidemiologi**:
  - **Cosa possono fare?** Possono inferire altre informazioni sugli utenti attraverso i dati che gli vengono inviati;
  - **Problematiche**: Dipende dai dati che gli utenti condividono;

Scenario n°1: Dati sulla posizione e coppie di EphID di tutti i contatti

- **Epidemiologi**:
  - **Cosa possono fare?** Possono tracciare i movimenti degli utenti, costruire i relativi grafi delle interazioni sociali, associare un'identità agli EphID sulla base degli spostamenti e delle abitudini;
  - **Problematiche**: Il tracciamento della posizione con lo studio delle abitudini portano alla violazione delle proprietà di **Protezione dell'Identità**, **Inferenza Controllata** e ovviamente di **Privacy**.
- I restanti avversari non hanno accesso a tali informazioni, di conseguenza i loro comportamento non cambia

Scenario n°2: Solo coppie EphID inviate ai server degli epidemiologi

- **Epidemiologi**:
  - **Cosa possono fare?** Possono ricostruire il grafo delle interazioni sociali **solo tra utenti che sono risultati infetti**
  - **Problematiche** Le informazioni sui contatti vengono inviate senza discriminare tra coppie nelle quali entrambe le parti hanno dato il consenso, e coppie in cui solo una delle parti ha dato il consenso, di conseguenza vanno a violare il requisito di **Privacy ai punti 2 e 3** e di **Inferenza Controllata**
  - **Possibili soluzioni** Le informazioni sui contatti possono essere condivise con gli epidemiologi solo previo consenso di entrambe le parti.
- I restanti avversari non hanno accesso a tali informazioni, di conseguenza i loro comportamento non cambia

Scenario n°3: Coppie di EphID con accordo sul caricamento dei dati aggiuntivi

Descrizione

In questa variante agli epidemiologi vengono comunque inviate le coppie di EphID relative ai contatti, **ma le due parti del contatto si mettono d'accordo tramite bluetooth per verificare se entrambi hanno dato l'autorizzazione alla condivisione dei dati con gli epidemiologi** e stabiliscono chi dovrà caricarli.

Studio degli avversari

- **Epidemiologi**:
  - **Cosa possono fare?** Possono ricostruire il grafo delle interazioni sociali **solo tra utenti che sono risultati infetti**;
  - **Problematiche** Possono ricostruire soltanto dati di persone che hanno dato il consenso, le proprietà di privacy sono soddisfatte, **dato che gli utenti hanno dato il consenso**;
- **Eavesdropper**:
  - **Cosa possono fare?** Possono ascoltare la comunicazione su canale Bluetooth tra i telefoni, e capire che si stanno accordando sul

caricamento dei dati agli epidemiologi;
    - **Problematiche** Può capire chi ha deciso di dare il consenso agli epidemiologi, non costituisce un problema nel caso di un semplice eavesdropper;
- **Tech-savvy**:
    - **Cosa può fare?** Può capire chi ha dato autorizzazione ai dati aggiuntivi, **come l'eavesdropper**, può modificare il comportamento dell'app in modo tale da poter inviare informazioni agli epidemiologi anche senza il consenso della controparte;
    - **Problematiche** La proprietà di **Privacy** ai punti 2 e 3 non è rispettata;
    - **Possibili soluzioni** Verifica del consenso da parte del server di backend degli epidemiologi.

Scenario n°4: Coppie di EphID con accordo sulla condivisione dei dati e verifica dell'autorizzazione

Descrizione

Modifica dello **Scenario n°3** con l'aggiunta che **appena l'utente decide di dare o meno il consenso invia un commitment agli epidemiologi con la sua scelta**.

Schema di commitment

1. L'utente sceglie se dare o meno il consenso alla condivisione dei dati e apre una connessione sicura con gli epidemiologi (e.g. HTTPS)
2. L'utente genera un **nonce** casuale ed invia **commit := H(msg| nonce)**, dove *msg* sarà *"yes"* oppure *"no"* a seconda della sua scelta
3. Il server di backend degli epidemiologi risponde con l'ID associato al *commit*, l'utente conserverà l'*ID*, il *nonce* e ovviamente *msg*;
4. Più tardi, se l'utente risulterà positivo, oltre a caricare il suo SKt sul backend del sistema DP3T, invierà anche la terna **(ID,msg,nonce)**;
5. Verificando il contenuto del *commit* gli epidemiologi potranno determinare se l'utente infetto ha fornito o meno il consenso alla condivisione dei dati, in quest'ultimo caso potranno scartare i contatti che lo riguardano.

Studio degli avversari

- **Epidemiologi**: stesse cose del protocollo precedente
- **Eavesdropper**: stesse cose del protocollo precedente
- **Tech-savvy**:
    - **Cosa può fare?** Le stesse cose dello schema generale e possono capire chi ha dato il consenso con una versione modificata dell'app;
    - **Problematiche** Le stesse cose del schema generale;
    - **Possibili soluzioni** Bisogna trovare un modo per evitare l'accordo tramite bluetooth.

Proposta di soluzione: Coppie di EphID con secret sharing per la ricostruzione del contatto

Descrizione

Modifica dello **Scenario n°3** con l'utilizzo di **Shamir secret sharing**: i contatti vengono trattati come un segreto di cui entrambi gli utenti conservano uno share, che

condividono solo se hanno dato il consenso alla condivisione dei dati agli epidemiologi, i quali possono ricostruire il contatto **solo** se hanno entrambi gli share.

Schema di secret sharing

Il segreto consiste nella coppia **s:=(EphID1,EphID2)**, in sui supponiamo che **EphID1 > EphID2**. Lo schema utilizzato è Shamir Secret Sharing (2,2):

1. Alice e Bob si incontrano e si scambiano i relativi EphID;
2. Dato che il polinomio ha la forma **p(x) = s + a*x % P** (dove *P* è un numero primo t.c. P>s) e occorre generare un solo coefficiente, si è scelto di utilizzare **a = H(EphID1|EphID2)**;
3. I punti verranno scelti calcolando l'hash dell'SKt concatenato ad un *nonce* generato casualmente quando l'utente da l'autorizzazione alla condivisione dei dati con gli epidemiologi, supponendo che Alice abbia dato il consenso per i dati agli epidemiologi ella invierà il punto **(x = H(SKt|nonce), y = p(H(SKt|nonce)))** al server backend degli epidemiologi;
4. Al server bisognerà inviare un ID per poter individuare il contatto, calcolato come **ID = H(EphID2|EphID1)** (L'ordine è l'opposto di quello utilizzato per il coefficiente).
5. Quando uno dei due utenti diventerà infetto e dovrà caricare l'SKt sul backend di DP3T allora invierà anche il *nonce* che ha utilizzato per la scelta dei punti

Le considerazioni effettuate su questo schema valgono in ambito **Random Oracle Model**, cioè si suppone che l'output della funzione di hash sia completamente casuale. Nella realtà si sceglierà di utilizzare una funzione di hash collision resistant (e.g. SHA256). Non indichiamo esplicitamente il numero primo *P*, osserviamo però che quanto più è grande *P* tanto più è robusto lo schema di secret sharing.

Studio degli avversari

Lo studio degli avversari è spostato nella sezione successiva, dato che questo è lo schema definitivo.

4 Analisi della confidenzialità e della integrità del sistema proposto

Analizzeremo ora le proprietà rispettate dallo schema descritto nella proposta finale.

- **Epidemiologi**:
  - **Cosa possono fare?** Possono costruire modelli sulla diffusione del virus a partire dai contatti che coinvolgono infetti ed elaborare considerazioni sulle tempistiche dei meccanismi di contagio;
- **Eavesdropper**:
  - **Cosa possono fare?** Può ascoltare le comunicazioni sui canali coinvolti nel sistema;
- **Tech-savvy**:
  - **Cosa possono fare?** Generare contatti falsi da inviare ai server degli epidemiologi, anche utilizzando EphID raccolti nella realtà;
  - **Problematiche**: Se gli utenti ai quali gli EphID sono associati non diventano infetti allora non c'è modo per distinguere i contatti falsi

da quelli veri.

Gli epidemiologi non rappresentano più una problematica perché non possono venire a conoscenza di contatti per i quali almeno delle due parti non abbia dato il consenso alla condivisione dei dati. Anche l'eavesdropper, che potrebbe cercare di scoprire informazioni sugli utenti, non rappresenta una minaccia poiché non è in grado di capire chi tra gli utenti condivide i dati con gli epidemiologi, tale risultato è ottenuto grazie all'utilizzo del **secret sharing**, poiché gli utenti non si accordano pubblicamente sulla condivisione, e con l'utilizzo di dummy package e proxy (così come scritto nelle assunzioni della fase progettuale) che prevengono l'analisi del traffico. Invece il **Tech-Savvy** è ancora capace di generare contatti falsi sia con EphID di altri utenti captati sul canale Bluetooth, sia EphID generati a caso. Tuttavia il meccanismo di generazione degli share basato sui valori hash delle SKt che vengono inviati agli epidemiologi permette a questi ultimi di verificare l'autenticità del contatto quando uno dei due utenti coinvolti nel contatto segnalato diventa infetto. Ricordiamo che da una SKt è possibile calcolare sia gli SKt successivi che i relativi EphID. Per la verifica è sufficiente calcolare gli hash delle SKt presenti sul backend di DP3T e di quelle dei giorni successivi concatenate al relativo *nonce* sul backend di DP3T e trovare una corrispondenza tra le x degli share nel server degli epidemiologi e gli hash calcolati, dopodiché verificare che uno degli EphID coinvolti nel contatto venga effettivamente generato da quella SKt. Questa operazione di verifica non è possibile nel caso in cui nessuno dei due EphID risulterà appartenere ad un utente che si è dichiarato infetto. Nonostante ciò, per quanto utili le informazioni relative a contatti tra utenti che non diventano infetti sono meno cruciali rispetto a quelle riguardanti gli infetti e dunque questa situazione rappresenta un compromesso accettabile.

- **Uso dei Dati**: La proprietà è verificata poiché non vengono raccolti dati aggiuntivi rispetto a quelli previsti da DP3T, con la sola differenza che agli epidemiologi vengono inviati gli EphID di tutti i contatti, con e senza utenti infetti, ma ciò rientra nello scopo del sistema da noi proposto.
- **Inferenza Controllata**: Anche se è possibile ricostruire le interazioni sociali, ciò non costituisce una violazione della proprietà perché non è possibile associare un identità agli SKt.
- **Privacy**: La proprietà è rispettata dato che non è possibile tracciare gli utenti, ne inferire o condividere informazioni su utenti che non hanno dato l'autorizzazione alla condivisione dei dati con gli epidemiologi (in tal caso si avrà un solo share, che non dà nessuna informazione sul segreto, cioè il contatto)
- **Autenticità**: La proprietà non è completamente soddisfatta dato che la verifica dell'autenticità dei contatti è possibile solo se uno degli utenti risulterà infetto.
- **Protezione delle identità**: La proprietà è verificata poiché dai dati condivisi dagli utenti non è possibile risalire all'identità degli stessi.

5 Parte pratica

Scopo dell'implementazione

Lo scopo dell'implementazione è stato quello di simulare parte del comportamento del sistema, sono stati simulati i seguenti comportamenti:

1. Creazione di uno share da parte di un utente a seguito di un contatto
2. Ricostruzione del segreto da parte degli epidemiologi
3. (**Parzialmente**) Notifica dell'infezione al main server
4. Eliminazione dei contatti ritenuti falsi da parte degli epidemiologi

Il punto *3* è stato implementato parzialmente perché non si effettua un controllo da parte dell'autorità sanitaria sul caricamento come richiesto in DP3T White Paper ma incluso perché propedeutico alla simulazione degli scenari descritti nel nostro progetto.

La simulazione riguarda 3 entità:

1. Il server degli epidemiologi, rappresentato dal modulo `lab_svr`
2. Il server dell'autorità sanitaria, rappresentato dal modulo `main_svr`
3. Gli utenti (avversari inclusi), rappresentati dalla classe `User` e dagli script di test

Strumenti utilizzati

- Sanic per la creazione dei web server
- OpenSSL per la creazione dei certificati delle CA (root CA e intermediate CA) e dei server
- PostgreSQL per i database
- Cryptography per le primitive crittografiche adoperate

main_svr

Questo modulo rappresenta il server dell'autorità sanitaria sul quale vengono caricati gli SKt degli utenti che risultano infetti. Come descritto in DP3T White Paper, gli utenti scaricano periodicamente gli SKt dei nuovi infetti dal server principale, ad esempio fornendo l'id dell'ultimo SKt letto dal database.

Riportiamo di seguito il file `main_svr/views.py` in cui è contenuto il codice opportunamente documentato, su come vengono gestiti i caricamenti di nuovi SKt da parte di infetti e di come vengono richiesti i nuovi SKt da parte degli utenti.

```
class MainView(HTTPMethodView):
    async def get(self, request):
        """
        Questo metodo si occupa di servire le richieste di nuovi SKt.
        :param request: Client request.
        :return: Response object.
        """
        # Nel caso non venga fornito il parametro last_id utilizziamo
        # questa request per controllare che il server sia in funzione.
        try:
            last_id = int(request.args['last_id'][0])
        except KeyError:
            return response.text("OK")
        logger.info("Requested SKts from : " + str(last_id))

        # Costruiamo la query per richiedere tutti gli SKt
        # con id maggiore dell'ultimo id letto dall'utente.
        query = infects.select().where(infects.c.id > last_id)
        rows = await request.app.db.fetch_all(query)
```

```
        # Restituiamo all'utente una response in formato json e
        # contenente tutti gli SKt successivi a last_id trovati nel database.
        return response.json({
            'skts': [{'id': row['id'], 'skt': row['skt'], 'nonce': row['nonce'],
                      'start_date': row['start_date'].isoformat(),
                      'created_date': row['created_date'].isoformat()} for row in
rows]
        })

    async def post(self, request):
        """
        Questo metodo si occupa di servire i caricamenti di nuovi SKt sul database.
        :param request: Client reqeust.
        :return: Response object.
        """
        # Si accettano solamente request di tipo json
        if request.content_type != 'application/json':
            return response.text('BAD REQUEST', status=400)

        # Nel json vi saranno due campi obbligatori
        prs_js = json.loads(request.json)
        try:
            # L'SKt dell'utente segnalato infetto
            skt = prs_js['skt']
            # La data d'inizio dell'infezione #
            # (Solitamente 14 giorni prima dei sintomi)
            str_date = prs_js['start_date']
            start_date = datetime.strptime(str_date, request.app.config.DATE_FORMAT)
        except KeyError:
            return response.text('BAD REQUEST', status=400)

        # Vi sarà anche un campo opzionale: il nonce
        # utilizzato per la notifica dei contatti agli
        # epidemiologi.
        try:
            nonce = prs_js['nonce']
        except KeyError:
            # Nel caso il nonce non sia presente
            # significa che l'utente non ha dato
            # l'autorizzazione alla condivisione
            # dei dati con gli epidemiologi
            nonce = None

        # Controlliamo se si tratta di una notifica duplicata,
        # in tal caso la scartiamo
        query = infects.select().where(infects.c.skt == skt)
        rows = await request.app.db.fetch_all(query)

        if len(rows) > 0:
            return response.text('FORBIDDEN', status=403)
```

```
        # Se la notifica risulta valida, allora l'aggiungiamo al database
        if nonce is None:
            query = infects.insert().values(skt=skt, start_date=start_date)
        else:
            query = infects.insert().values(skt=skt, nonce=nonce,
start_date=start_date)

        await request.app.db.execute(query)
        return response.text("OK")
```

lab_svr

Questo modulo rappresenta il server degli epidemiologi, si occupa di ricevere i contatti dagli utenti che hanno dato l'autorizzazione alla condivisione dei dati e di controllare periodicamente la presenza di nuovi SKt sul server principale (main_svr), in modo da scartare i contatti ritenuti non autentici.

Le procedure principali sono contenute in `lab_svr/views.py` e `lab_svr/tasks.py`. Nel primo file viene gestito il caricamento di nuovi share, nel secondo il controllo di nuovi SKt sul server principale dell'autorità sanitaria.

lab_svr/views.py

```
class ShareView(HTTPMethodView):
    async def get(self, request):
        """
        Restituisce una response di carattere generico, usata per testare lo stato
        di funzionamento del server.
        :param request: Client request.
        :return: Response object.
        """
        return response.text("OK")

    async def post(self, request):
        """
        Gestisce le richieste di caricamento di nuovi share da parte degli utenti.
        :param request: Client request contenete lo share.
        :return: Response object.
        """
        # Vengono servite solo request contenenti json
        if request.content_type != 'application/json':
            response.text('FORBIDDEN', status=403)
        prs_js = json.loads(request.json)

        # Controllo che nel json siano presenti i campi attesi
        try:
            # ID associato all'evento di contatto
            # tra due utenti.
            cont_id = prs_js['contact_id']
            # Coordinate del punto nel polinomio,
            # cioè dello share.
            x = prs_js['share']['x']
```

```
            y = prs_js['share']['y']
        except KeyError:
            return response.text('ERROR', status=400)

        # Controllo se è presente uno share con lo stesso contact_id,
        # in modo da poter ricostruire il segreto, cioè la coppia
        # di EphID coinvolti nell contatto.
        get_shares_q = shares.select().where(shares.c.contact_id == cont_id)
        rows = await request.app.db.fetch_all(get_shares_q)

        query = None
        if len(rows) == 0:
            # Se non è stato trovato nessuno share, allora presumo che
            # questo sia uno share associato ad un nuovo contatto e quindi
            # lo aggiungo al database.
            query = shares.insert().values(contact_id=cont_id, x=x, y=y)
        elif len(rows) == 1:
            row = rows[0]
            # Nel caso sia stato già trovato uno share, controlliamo se è
            # diverso da quello che vogliamo caricare e in tal caso lo inseriamo.
            if row['x'] != x:
                query = shares.insert().values(contact_id=cont_id, x=x, y=y)
            else:
                response.text('FORBIDDEN', status=403)
        else:
            return response.text('FORBIDDEN', status=403)

        if query is not None: await request.app.db.execute(query)

        # Controllo se ci sono due share realtivi ad un contatto.
        rows = await request.app.db.fetch_all(get_shares_q)
        if len(rows) == 2:
            # Se ci sono due share allora ricostruisco il contatto/segreto.
            x1 = int.from_bytes(bytes.fromhex(rows[0]['x']), 'big')
            y1 = int.from_bytes(bytes.fromhex(rows[0]['y']), 'big')
            x2 = int.from_bytes(bytes.fromhex(rows[1]['x']), 'big')
            y2 = int.from_bytes(bytes.fromhex(rows[1]['y']), 'big')

            sh1 = [x1, y1]
            sh2 = [x2, y2]

            s = ss.recover_secret([sh1, sh2])
            s_b = s.to_bytes(32, 'big')

            # Il segreto contiene gli EphID concatenati.
            ephid1 = s_b[:16].hex()
            ephid2 = s_b[16:].hex()

            # Salvo il nuovo contatto nel database.
            query = contacts.insert().values(id=cont_id, ephid1=ephid1, ephid2=ephid2)
            await request.app.db.execute(query)
        return response.text('ACCEPTED')
```

lab_svr/tasks.py

```
def setup_tasks():
    """
    Configura i task da eseguire periodicamente.
    :return: None
    """
    # La funzione di hash utilizzata.
    hash_f = hashes.Hash(hashes.SHA256(), backend=default_backend())

    @task(start=timedelta(seconds=5), period=timedelta(days=1))
    async def daily_task(app):
        # Carico l'ultimo ID letto.
        try:
            f = open(app.config.ID_FILE, 'r')
            last_id = int(f.readline())
        except (IOError, ValueError):
            # Se non ho nessun nuovo ID allora li scarico tutti.
            last_id = -1

        # Invio la request al server principale per farmi restituire la lista degli
SKt caricati
        # dall'ultimo controllo.
        params = {'last_id': last_id}
        resp = requests.get('https://' + app.config.MAIN_SVR + '/', params=params,
verify=app.config.CA_DIR)

        rows_list = resp.json()['skts']
        # Itero sugli SKt ricevuti
        for rec in rows_list:
            # Per ogni SKt ricevuto calcolo quanti giorni è stato infetto, in
            # modo da generare i reletivi EphID.
            start_date = datetime.strptime(rec['start_date'], app.config.DATE_FORMAT)
            created_date = datetime.strptime(rec['created_date'],
app.config.DATE_FORMAT)
            days = (created_date - start_date).days
            # Inizio l'SKt attuale da utilizzare nei controlli
            act_skt = bytes.fromhex(rec['skt'])
            # Vedo se è disponibile un nonce utilizzato per caricare
            # i dati al server degli epidemiologi.
            try:
                act_nonce = bytes.fromhex(rec['nonce'])
            except (TypeError, KeyError):
                act_nonce = None

            # Nel caso sia presente il nonce allora calcolo le x
            # utilizzate per generare gli share.
            hash_out_str = None
            if act_nonce is not None:
                h = hash_f.copy()
                h.update(act_skt + act_nonce)
                hash_out = h.finalize()
```

```
                hash_out_str = hash_out.hex()

            # Itero sui giorni in cui l'utente risultava infetto.
            for i in range(0, days):
                # Per ogni giorno predispongo il generatore per gli EphID
                prf = hmac.HMAC(act_skt, hashes.SHA256(), backend=default_backend())
                prf.update(b"broadcast key")
                prf_out = prf.finalize()
                bit_gen = AESCounter(key=int.from_bytes(prf_out[16:], "big"))
                gen = Generator(bit_gen)

                # Itero sulle n parti della giornata a cui corrisponde
                # un EphID
                for j in range(0, app.config.DAY_PARTS):
                    # Genero l'EphID prendendo i successimi 16 byte
                    ephid = gen.bytes(16)
                    # Predispongo la chiamata della funzione
                    # definita sul database e che si occupa di eliminare
                    # i contatti e gli share considerati non autentici.
                    query_txt = text('SELECT "DELETE_FAKE"(:ephid,:hash)')
                    query_txt = query_txt.bindparams(ephid=ephid.hex(),
hash=hash_out_str)
                    result = await app.db.execute(query_txt)
                    if app.config.DEBUG:
                        logger.info('EphID : ' + ephid.hex() + ', SKt : ' +
act_skt.hex())
                        logger.info("Fake contacts found : " + str(result) +
                                    ', Nonce : ' + str(act_nonce or 'None'))
                # Calcolo l'SKt del giorno successivo.
                h = hash_f.copy()
                h.update(act_skt)
                act_skt = h.finalize()

        # Nel caso abbia ricevuto nuovi SKt allora salvo
        # l'ID dell'ultimo SKt restituito dal server principale.
        if len(rows_list) > 0:
            f = open(app.config.ID_FILE, 'w')
            f.write(str(rows_list[-1]['id']))
```

Funzione per eliminazione contatti non autentici

```
--
-- Name: DELETE_FAKE(character, character); Type: FUNCTION; Schema: public; Owner:
postgres
--

CREATE FUNCTION public."DELETE_FAKE"(ephid character, hash character) RETURNS integer
    LANGUAGE plpgsql
    AS $$DECLARE
out integer;
BEGIN
create temp table to_delete(contact_id CHAR(64));
```

```
if(hash is not null) then
insert into "to_delete"
SELECT contact_id FROM "Share" WHERE contact_id IN (SELECT id FROM "ContactEvent"
WHERE ephid1 = ephid OR ephid2 = ephid) AND x != hash GROUP BY contact_id;
else
insert into "to_delete"
SELECT contact_id FROM "Share" WHERE contact_id IN (SELECT id FROM "ContactEvent"
WHERE ephid1 = ephid OR ephid2 = ephid) group by contact_id;
end if;
out := (select count(*) from "to_delete");
DELETE FROM "ContactEvent" WHERE id IN (select * from to_delete);
DELETE FROM "Share" WHERE contact_id IN (select * from to_delete);
DROP TABLE "to_delete";
RETURN out;
END$$;


ALTER FUNCTION public."DELETE_FAKE"(ephid character, hash character) OWNER TO
postgres;

SET default_tablespace = '';

SET default_table_access_method = heap;
```

Libreria secret sharing

Questa libreria implementa lo schema di secret sharing come descritto nella nostra [proposta di soluzione](#Proposta di soluzione: Coppie di EphID con secret sharing per la ricostruzione del contatto).

secretsharing.py

```
#####
# Questo modulo contiene le funzioni che implementano la variante di Shamir's Secret
Sharing proposta nel nostro progetto.
#
# Il codice che segue è una modifica dell'implementazione pubblicata sulla pagina
# https://en.wikipedia.org/wiki/Shamir%27s_Secret_Sharing
# sotto i termini di CC0 e OWFa
# https://creativecommons.org/publicdomain/zero/1.0/
# http://www.openwebfoundation.org/legal/the-owf-1-0-agreements/owfa-1-0
#####


from cryptography.hazmat.backends import default_backend
from cryptography.hazmat.primitives import hashes

# 13-esimo numero primo di Mersenne
_PRIME = 2 ** 521 - 1


def _eval_at(poly: int, x: int, prime: int) -> int:
```

```
    """
    Valuta il polinomio (lista di coefficienti [a0, a1]) nel punto x.
    Usata per generare uno share in make_share.
    : param poly : lista dei coefficienti del polynomio
    : param x : numero intero per cui valutare il polinomio
    : param p : il numero primo che definisce il campo finito
    : return : il valore P(x)
    """

    accum = 0
    for coeff in reversed(poly):
        accum *= x
        accum += coeff
        accum %= prime

    return accum


def make_share(poly, x, prime=_PRIME):
    """
    Genera uno share [x, P(x)]
    : param poly : lista dei coefficienti del polinomio come restituiti da make_poly
    : param x : il numero intero in cui valutare il polinomio
    : param p : il numero primo che definisce il campo finito
    : return : la lista [x, P(x)]
    """

    point = [x, _eval_at(poly, x, prime)]

    return point


def make_polynomial(secret: bytes) -> bytes:
    """
    Genera i coefficienti di un polinomio di grado 1.
    : param secret : il segreto da condividere, come sequenza di byte. Usato come
termine noto del polinomio in formato intero
    : return : i coefficienti del polinomio come lista di interi [a0,ai]
    """

    digest = hashes.Hash(hashes.SHA256(), backend=default_backend())
    digest.update(secret)
    a1 = digest.finalize()
    poly = [int.from_bytes(secret, "big"), int.from_bytes(a1, "big")]

    return poly


def _extended_gcd(a, b):
    """
    Extended Euclidean Algorithm utilizzato per la divisione in aritmetoca modulare.
    Chiamata da _divmod.
```

```
    : param a,b : interi
    """

    x = 0
    last_x = 1
    y = 1
    last_y = 0
    while b != 0:
        quot = a // b
        a, b = b, a % b
        x, last_x = last_x - quot * x, x
        y, last_y = last_y - quot * y, y

    return last_x, last_y


def _divmod(num, den, p):
    """
    Calcola (num/den) mod p.
    Il valore di ritorno sarà tale che: den * _divmod(num, den, p) mod p == num
    : param num,den : interi
    : param p : il numero primo che definisce il campo finito
    : return : (num/den) mod p
    """

    inv, _ = _extended_gcd(den, p)

    return num * inv


def _lagrange_interpolate(x, x_s, y_s, p):
    """
    Trova il valore P(x) dati n punti (xi, yi). Utilizzata per ricavare il segreto
(P(0)).
    : param x : il punto in cui calcolare il valore del polinomio, intero
    : param x_s : tupla contenenti le x dei punti
    : param y_s : tupla contenente le y dei punti
    : param p : il numeor primo che definisce il campo finito
    : return : P(x)
    """

    k = len(x_s)
    assert k == len(set(x_s)), "points must be distinct"

    def PI(vals):
        accum = 1
        for v in vals:
            accum *= v
        return accum

    nums = []
    dens = []
```

```
    for i in range(k):
        others = list(x_s)
        cur = others.pop(i)
        nums.append(PI(x - o for o in others))
        dens.append(PI(cur - o for o in others))
    den = PI(dens)
    num = sum([_divmod(nums[i] * den * y_s[i] % p, dens[i], p)
               for i in range(k)])

    return (_divmod(num, den, p) + p) % p


def recover_secret(shares, prime=_PRIME) -> int:
    """
    Ricostruisce il segreto partendo dai due share.
    : param shares: lista di due shares come restituiti da make_share
    : param prime: Il numero primo che definisce il campo finito
    : return: il segreto come numero intero
    """

    if len(shares) < 2:
        raise ValueError("need at least two shares")
    x_s, y_s = zip(*shares)

    return _lagrange_interpolate(0, x_s, y_s, prime)


def make_secret(EphID1: bytes, EphID2: bytes) -> bytes:
    """
    Costruisce il segreto come concatenazione di due EphID
    : param EphID1, EphID2: i due EphIDs come sequenze di byte
    : return : il segreto come sequenza di byte
    """

    if EphID1 > EphID2:
        secret = EphID1 + EphID2
    else:
        secret = EphID2 + EphID1

    return secret
```

Certificati per i server

I server dispongono di certificati e di relative chiavi per la connessione sicura con gli utenti (In modo da non permettere al techsavvy di intercettare il valore della *x* negli share) e tra i server stessi. I certificati sono stati creati con l'utilizzo di OpenSSL, creando due Certification Authorities, la prima è una root CA e la seconda è una intermediate CA che si occupa di firmare i certificati dei server (Come nomi alternativi si è utilizzato `127.0.0.1` e `localhost` dato che non si ha un DNS a disposizione). La struttura della directory è la seguente.

```
cyb-certs
├──CA
│   ├──certs
│   ├──newcerts
│   └──private
├──intermediate-CA
│   ├──certs
│   ├──newcerts
│   └──private
├──lab
└──main
```

I comandi utilizzati sono i seguenti:

```
CA/ >openssl req -new -x509 -config ca-config.txt -nodes -newkey rsa:2048 -extensions
x3_ca
intermediate-CA/ >openssl req -new -nodes -out req.pem -newkey rsa:2048 -keyout
key.pem
CA/ >openssl ca -config ca-config.txt -policy policy_strict -in req.pem -out cert.pem
-extensions v3_intermediate_ca
# Nel caso del certificato di un server
lab/ >openssl ca -config int-config.txt -policy policy_loose -in req.pem -out cert.pem
-extensions server_cert
```