A Secure and Privacy Preserving COVID-19 Notification System

by

Yu Sun

A thesis submitted to
the Faculty of Graduate Studies and Research
in partial fulfilment of
the requirements for the degree of
Master of Computer Science

Department of Ottawa-Carleton Institute for Computer Science School of Computer
Science
Carleton University
Ottawa, Ontario, Canada
April 2021

Copyright ©

2021 - Yu Sun

Abstract

The worldwide health crisis caused by the COVID-19 pandemic has been impacting people's life for over one year now. While waiting for an effective treatment, there have been various Contact Tracing Systems (CTS) developed and adopted across the globe to help stop the spread of the virus. Most of these systems are hosted on smartphones and tablets due to their wide adoption rate and their built-in Bluetooth, GPS and WiFi features, which enable easy communication with other devices. While people are concerned about their exposure to the virus, they also worry about the breach of privacy inherent to using CTSs due to their collection of private information. Thus, we need a privacy-preserving and secure architectural design for implementing a Contact Tracing System that should encourage civilians to readily adopt it without any privacy concerns. As of June 2020, we have designed and implemented a fully automated, secure, efficient and privacy-preserving bluetooth-based COVID-19 Notification System (CNS) following the Office of the Privacy Commissioner of Canada (OPC) contact tracing system development guidelines. The design addresses both security and privacy threats during the virus exposure detection and notification processes.

Acknowledgments

First and foremost, with my most sincere gratitude, I would like to acknowledge my main supervisor, Dr. Wei Shi, for her dedication, patience, encouragement, and support. Without her guidance, knowledge and expertise, our COVID-19 Notification System (CNS) and this thesis would not have been possible. I also wish to thank Dr. Jean-Pierre Corriveau, as my co-supervisor from the School of Computer Science.

I would also like to thank all the team members in our CNS work group for their help with polishing the user interface and for offering valuable feedback throughout the process.

I am also thankful to my committee members for providing valuable feedback on my research.

I would also like to thank my family for their encouragement and support for my research interests and educational pursuits.

Table of Contents

Abstract	ii
Acknowledgments	iii
Table of Contents	iv
List of Tables	viii
List of Figures	x
1 Introduction	1
1.1 Background	1
1.2 Motivation	2
1.3 Problem Statement	3
1.4 Major Contribution	5
1.5 Thesis Overview	6
2 Related Work	7
2.1 Centralized Contact Tracing Approaches	7
2.1.1 BlueTrace	8
2.1.2 Pan-European Privacy-Preserving Proximity Tracing (PEPP-PT)	8
2.1.3 Network-based Location Tracking Methodologies	10
2.1.4 Geofencing Methodologies	11

2.2	Decentralized Contact Tracing Approaches	11
2.2.1	Decentralized Privacy-Preserving Proximity Tracing (DP-3T) Protocol	12
2.2.2	Google-Apple Exposure Notifications (GAEN) System	14
2.2.3	Temporary Contact Number (TCN)	14
2.2.4	Private Automated Contact Tracing (PACT)	15
2.2.5	Privacy-Sensitive Protocols And Mechanisms for Mobile Con- tact Tracing	16
2.3	Hybrid Approaches	16
2.3.1	DESIRE	17
3	Use Case Descriptions	18
3.1	Register Application	18
3.2	Turn on/off Contact Tracing	20
3.3	Exchange Encounter Info	20
3.4	Manage Infected User	25
3.5	Upload Encounter Logs	25
3.6	Detect Virus Exposure	25
3.7	Notify Quarantine	29
3.8	Share Family Device	29
4	System Design: Architecture and Overview	32
4.1	Architecture	32
4.1.1	Components	32
4.1.2	Communication Protocol	35
4.1.3	Data Storage	35
4.2	Threat Model and Assumptions	36
4.2.1	Attacker's goal	36

4.2.2	Attack surface	37
4.2.3	Threat	38
4.2.4	Assumption	40
4.3	Solution Overview	41
4.3.1	Notation	41
4.3.2	Registration Process	41
4.3.3	Encounter Exchange Process	44
4.3.4	Notification Process	44
4.4	Detailed Design	45
4.4.1	AppID Generation	45
4.4.2	Encounter Exchange Payload	46
4.4.3	Cryptography Algorithm	47
4.4.4	Signature Algorithm	48
4.4.5	Database Design	49
4.4.6	Detailed Process	49
5	Implementation and Evaluation	53
5.1	Implementation	53
5.1.1	Central Server Configuration	54
5.1.2	Implementation Detail	55
5.2	Evaluation	60
5.2.1	OPC Contact Tracing System Development Guideline	60
5.2.2	Functional Tests	63
5.2.3	Informal Usability Study	75
5.2.4	System Stress Tests	75
5.2.5	Security and Privacy Analysis	76

6	Conclusions and Future Work	78
6.1	Conclusions	78
6.2	Future Work	78
	List of References	80

List of Tables

3.1	Register Application Use Case	21
3.2	Turn off Contact Tracing Use Case	22
3.3	Turn on Contact Tracing Use Case	23
3.4	Exchange Encounter Info Use Case	24
3.5	Manage Infected User Use Case	26
3.6	Upload Encounter Logs Use Case	27
3.7	Detect Virus Exposure Use Case	28
3.8	Notify Quarantine Use Case	29
3.9	Share Family Device Use Case	31
4.1	Notation used in Design and Implementation	43
5.1	A single user registers with a single device	64
5.2	A single user registers with multiple devices	65
5.3	Multiple users register with a single device	65
5.4	Exchange encounter info with different distance between the two devices	67
5.5	Exchange encounter info with obstacles placed between the devices	68
5.6	Turn off contact tracing	69
5.7	Turn on contact tracing	70
5.8	Add infected user to infected user list	71
5.9	Verified infected user uploads encounter logs	72
5.10	Quarantine notification	73

5.11 Share device	74
-----------------------------	----

List of Figures

3.1	Use Case Diagram	19
4.1	The CNS Architecture	34
4.2	Attack Surface for the CNS	37
4.3	The relationship among privacy/security threats and attacker's goals	40
4.4	The CNS Main Processes	42
4.5	Encounter Exchange Info	47
4.6	Encounter Exchange Payload Detail	47
4.7	Database Schema of Client Application	50
4.8	Database Schema of Central Server	50
4.9	Registration Detailed Process	51
4.10	Encounter Exchange Detailed Process	51
4.11	Notification Detailed Process	52
5.1	Central Server Configuration Process	55
5.2	Registration Page	56
5.3	Main Page	57
5.4	Family Device Sharing Page	58
5.5	Notification Page	59
5.6	Usability Study on the Main Page	75

Chapter 1

Introduction

1.1 Background

Since early 2020, a worldwide pandemic outbreak caused by Coronavirus has significantly changed people's life all over the world. To help stop the spread of such an aggressive virus and save more lives while waiting for an effective treatment, governments and authorities have taken various actions, such as forcing lock-downs, closing schools, requiring self-quarantines for travellers, encouraging work-from-home and practicing social distancing.

COVID-19 is highly contagious and can be transmittable through air and surfaces. Even worse, a virus carrier can be contagious without showing any symptoms which entails that by the time this person receives a positive test result, s/he might already have widely spread the virus among those s/he has encountered. To manually interview the infected individuals, gather their encounter histories and then perform risk analysis on each encounter can be expensive and time-consuming. This process is also error-prone: it relies on the correctness of the recollections of individuals. An automated Contact Tracing (CT hereafter) application is thus vitally important to monitor efficiently and at low cost the spread of the disease and to alert people who have been in close proximity with a confirmed COVID-19 case to start a self

quarantine and/or get tested immediately.

The wide adoption of mobile devices, such as smartphones and tablets, makes them an ideal choice to host an automated contact tracing system. Indeed, such a device has the ability to keep track of its locations via GPS or WiFi and is equipped with built-in Bluetooth features to establish communication with other devices. A CT application that runs on a mobile device can track the proximity of other mobile devices and immediately alert their users with notifications when two devices are within a predefined proximity, such as the 2 metres social distancing required by the government of Canada in the context of COVID-19 pandemic.

1.2 Motivation

As the COVID-19 pandemic spreads all over the world, people are concerned about whether they have been exposed to the virus. But they are also concerned about their privacy. The traditional manual CT method will require an infected user to provide detailed tracing information that enables the health authorities to discover the path of spreading of the virus. In some countries, even *direct identifiers* (i.e., Name, National ID) are disclosed to the public to help detect potential exposure to the virus. Other personal information (so-called “quasi-identifiers”), such as phone numbers, Date of Birth (DoB), and address is also collected to be used for tracing purposes.

With the disclosure of personal information, privacy concerns increase. For example, one’s social interactions and whereabouts can be easily inferred from the disclosed information. Furthermore, at the beginning of the pandemic (and arguably even now), a lack of understanding of the virus led infected people to be negatively judged and even sometimes be the victims of hate crimes. All these factors result in a significant proportion of the population hesitating (nay refusing) to cooperate in a

contact tracing process because they feel excessively scrutinized.

A CT application can complement manual CT. However its usefulness is directly dependent on its adoption rate. That is, the higher the number of users of a CT application, the larger the number of its users likely to receive the notifications of virus exposure and thus the better the performance of that application. Thus, we aim to find a privacy-preserving approach to implement a contact tracing application that could be widely accepted and adopted. But it is crucial to acknowledge that the more a CT application is adopted, the more it could be exploited by some adversaries (i.e., be the target of cyber attacks). Therefore, the security aspect of a CT application must be scrutinized.

1.3 Problem Statement

Contact tracing is the primary method to control infectious diseases. Traditional manual CT is time consuming and thus rather cost inefficient, and, more importantly, not sufficient for fast spreading and highly infectious diseases like COVID-19. A fully automated, secure, efficient and privacy preserving solution is needed for supporting automated contact tracing. However, security and privacy vulnerabilities are inherent to any contact tracing application. Consequently, it is important to review and evaluate any solution with respect to a set of accepted critical criteria.

In an article entitled “Race to Trace” [1], Ryerson’s Leadership Lab has given five recommendations to evaluate contact tracing technologies:

- Follow privacy-by-design principles using Bluetooth technology only.
- Use a decentralized approach to reduce the security and privacy risks created by a central database.

- Only collect, store and use data that is necessary, and include built-in functionality to automatically and permanently delete mobile data after no more than 30 days.
- Be voluntary, fully opt-in and require informed consent.
- Be transparent through a transparent procurement process.

Similarly, the National Cybersecurity Consortium, led by researchers from 32 universities across Canada, issued a joint statement [2] establishing the ten principles that should be followed when designing, developing and reviewing any contact tracing application, namely: independent expert review, simple design, minimal functionality, data minimization, trusted data governance, cybersecurity, minimum data retention, protection of derived data and meta-data, proper disclosure and consent, and provision to sunset.

In May 2020, the Office of the Privacy Commissioner of Canada (OPC) has also issued a joint statement [3] with its provincial and territorial counterparts to outline some key privacy principles to consider when developing digital contact tracing applications. This statement decrees that any contact tracing application should respect the following principles: consent and trust, legal authority, necessity and proportionality (necessity, proportionality, effectiveness, minimal intrusiveness), purpose limitation, de-identification, time-limitation, transparency, accountability, and safeguards.

In this thesis, we design and implement a secure and privacy preserving COVID-19 Notification System (CNS) following the OPC contact tracing system development guidelines [3]. A traditional full contact tracing system collects personal data from users, stores and computes encounters on a central server in order to track the spread of a virus based on this information. The resulting massive amount of centralized personal data could be a tempting target for malicious users and could be misused

for civilian tracking purposes. The notification function is the key feature of a contact tracing system as it can effectively slow down the spread of the virus by notifying those who have come in close proximity with an infected person within a certain period of time. The CNS we propose does not collect users' whereabouts. Only minimal anonymized meta data are locally stored in a user's device. According to the definition of a decentralized CT application given by OPC, a decentralized CT application empowers devices to manage and control information independently and lets users pool the information when they want and need [4]. Designing a decentralized CT application reduces several security and privacy risks that are associated with the centralized architectures put forth for CTSs.

1.4 Major Contribution

The COVID-19 pandemic is still impacting people's life. Contact Tracing Systems (CTSs) help slowing down the spread of the virus efficiently. However, it is not trivial to find an optimal solution to the trade-off between privacy and performance in CTSs. In this thesis, we report on our attempt at designing and implementing a secure and privacy preserving COVID-19 Notification System (CNS). Our main contributions are:

- The design of a decentralized Bluetooth-based privacy preserving COVID-19 notification application and its implementation on Android devices.
- An analysis of potential security and privacy threats against our proposed CNS.
- A discussion of the functional testing carried out within our research group as well as via high school students.

1.5 Thesis Overview

The remainder of the thesis is organized as follows. In Chapter 2, we summarize related work in centralized, decentralized and hybrid implementations of CTSs. In Chapter 3, we explain the use cases of the proposed CNS. In Chapter 4, we describe the threat model, the architecture and the design of the proposed system. In Chapter 5, we compare our design against other solutions proposed in Chapter 2 and provide a detailed security analysis of the proposed design. In Chapter 6, we conclude and discuss future reach directions.

Chapter 2

Related Work

In this section, we discuss various approaches adopted by academic and industrial research for implementing CTSs.

For CTSs, the simplest way is to compare a user’s moving trails collected by GPS, but this method would leak some sensitive information like user’s geo-location [5]. It can also be implemented by matching the WiFi access points instead of location information, which can only work well with wireless coverage like malls, restaurants, and many indoor places [6]. Further, the address of WiFi access points is still a risk that leaks the location information. Other methods like Bluetooth Low Energy are also adopted to CTSs. As a respiratory infectious disease like COVID-19, people are infected by close contact within about 2 meters, in which Bluetooth can communicate with each other [7]. With almost all smartphones are equipped with Bluetooth, it’s convenient to popularize, and does not need any location information to be involved.

2.1 Centralized Contact Tracing Approaches

For a centralized architect, the core functionalities are implemented in a registered central server, which manages the unique key for each user and performs calculations to determine the potential exposure users [8].

2.1.1 BlueTrace

Singapore is the first country to establish a national deployment of a Bluetooth-based contact tracing system called TraceTogether, which is based on a privacy-preserving protocol named BlueTrace [9].

BlueTrace protocol facilitates contact tracing in a way that logs Bluetooth encounters between registered devices while preserving user's privacy data. After a user installs the app, the user needs to use a phone number to register their device with a central server. The central server then generates a random and unique `userId` which only associates with this user's phone number and sends symmetrically encrypted `userId` back to the device. Registered devices exchange messages with each other via Bluetooth to log the encounters, each message consists of an encrypted `userId`, a created time, and an expiry time, along with a random Initialisation Vector and an Auth tag. When a user is diagnosed with positive, the user will upload the encounter logs stored in their device voluntarily to the central server. And the central server will decrypt these messages to get the encounter `userIds` and validity period based on timestamps. Then filtered valid exposure users are contacted by the central server via phone number to receive notifications.

The government of Alberta Launched a mobile contact tracing app called ABTraceTogether [10] in May 2020. ABTraceTogether has used the Bluetrace protocol for its implementation.

2.1.2 Pan-European Privacy-Preserving Proximity Tracing (PEPP-PT)

A multi-national European team has created a Pan-European Privacy-Preserving Proximity Tracing protocol, which shares a similar concept as BlueTrace but doesn't require a user to provide any personal data [11].

Pan-European Privacy-Preserving Proximity Tracing protocol also facilitates contact tracking with logging Bluetooth encounters nearby. A push notification id is generated from the client app locally and sent to the push notification server after a user installs the app and gives consent. Then the client app sends a register request along with the push notification id to the central server. Then the central server generates a 128-bits unique random id and responses client app with a combination of a proof-of-work (PoW) with a Captcha for authentication. When the client app sends back the solution, the unique random id is marked as valid and stored in the central server database, and an access token is generated and sent back to the client app for future API request authentication. If the central server is not responded with a valid solution, the unique random id will be deleted from the central server.

Once a device is registered successfully, the central server generates secrets periodically used for encrypting the unique random id. Registered devices broadcast the encrypted random id sent from the central server to nearby registered devices and stores the encounters log locally. When a user tests positive, the user needs to authenticate their device with a given verification code to be able to upload the encounter logs to the central server. The uploaded encounter logs trigger the risk assessment process on the central server, and the central server will then send a notification to the push notification service to notify the service that there are new encounter logs uploaded. Then the push notification service does a random push to all devices alerting new encounter logs are uploaded, and each device will send a risk assessment request with the access token and encrypted random id to the central server to get it assessed. Finally, the risk score will be calculated and presented to the user on the app screen.

Inria and Fraunhofer AISEC collaborated and proposed ROBust and privacy-preserving proximity Tracing protocol (ROBERT) for the Pan European Privacy-Preserving Proximity Tracing initiative [12][13]. Different than the PEPP-PT approach where a central server generates the secrets periodically to encrypt the random id on user's devices, ROBERT protocol lets the central server generate a list of pseudonyms and send them to each user's device once the user finished registration. Users use these periodically changed pseudonyms to exchange encounter logs between each other. A central server compares the uploaded pseudonyms from infected users and other users to determine whether a user is at risk of virus exposure. However, mentioned by Serge Vaudenay in [14] that the major privacy issue for ROBERT protocol is a malicious central server could remap the random pseudonyms to a long term id and abuse the id to track users.

2.1.3 Network-based Location Tracking Methodologies

Besides Bluetooth proximity tracing, location tracking is another methodology used in a couple of countries for tracking the spread of the virus. Mobile phone location data are collected by a central server managed by the government to track suspected coronavirus cases.

In China, telecommunication carriers need to provide a location tracking record of a user's cell phone. And these tracking records are uploaded to a nationwide telecommunication data analysis platform for suspected cases analysis. Based on the user's health condition and location record analysis result, a health code is issued to her/his device, presenting green, yellow, or red status to indicate the risk score of the current user.

In South Korea, the government established a system that gathers tracking information from various resources to perform contact tracing, such as credit card transactions, CCTV facial recognition data, and mobile telecommunication data. A

user that is potentially exposed to the virus will be notified with a text message.

In Israel, the Ministry of Health launched a location-based contact tracing app, which gathers users' location data locally and compares them to known movements of the infected people to determine whether a user has been exposed to the virus.

2.1.4 Geofencing Methodologies

Instead of using location data to track potential exposure users, some areas are using location data to support self-quarantine for infected users. The location data are not stored in a central server until the locally-stored location data has significantly changed.

In Russia, an app called Social Monitoring was implemented by the City of Moscow. Infected users are asked to register their devices with a central server. The app monitors infected users' GPS info and sends random notifications to users, which requires users to respond immediately with a selfie to prove they are staying at home, if the response is not received in time, the user will be fined.

In Hong Kong, a waistband is required to be worn by people arriving from overseas. The waistband is linked with an app called StayHomeSafe, which registers a user's waistband with her/his phone number. The user's location data is stored by the app and uploaded to the central server once the user leaves home during 14-days quarantine time.

2.2 Decentralized Contact Tracing Approaches

In terms of a decentralized architect, each device handles the main functionalities locally, including generating random identifiers and matching affected user contacts [8].

2.2.1 Decentralized Privacy-Preserving Proximity Tracing (DP-3T) Protocol

Decentralized Privacy-Preserving Proximity Tracing protocol aims at performing proximity tracing with minimal data at a large scale in a secure and privacy-preserving way [15].

In contrast to the centralized approaches, where central servers are responsible for managing the temporary IDs and exposure user detection, the Decentralized Privacy-Preserving Proximity Tracing protocol moves the logic to each device. Once users installed the app on their devices, the app starts to generate pseudo-random IDs locally representing the device. With the app running in the background, it broadcasts the pseudo-random IDs to other registered devices via Bluetooth Low Energy (BLE) beacons [15]. Other registered device stores the periodically-changed IDs locally as encounter logs. When a user is confirmed infected, the user uploads the broadcast random IDs to the central server. Other devices then fetch the random IDs from the central server and compare them with the encounter logs locally to determine whether the user has been exposed to the virus.

Based on this concept, the DP-3T project research team has defined three different protocols to support exposure detection and tracing with trade-offs between privacy and computation cost [15].

- Low-cost decentralized proximity tracing protocol

Instead of asking infected users to send the whole list of broadcast temporary random IDs to the central server and demanding other users to pull down the list for comparison. This protocol only requires infected users to upload key pairs of the random daily secret seed and the date when the user is considered as being contagious. The random daily secret seed is initialized with a random value when a user installs the app, starting from this initialization day, the

seed updates daily based on previous seed value using a hashed encryption function. A list of temporary IDs generated locally on the device to be used for broadcasting on that day. To calculate the temporary IDs, the app uses a pseudorandom generator to generate a value from the random seed and a given fixed string, and then a pseudo-random function is applied to the value to get the final IDs. The infected users upload the random seed and the date to the central sever instructed by the device, and the seed is deleted after uploading is done to leave the infected users untraceable afterward. Other devices then fetch the key pairs from the server, calculate a list of temporary IDs locally and compare the stored encounter logs to determine whether there is a match. This approach not only saves the storage cost on the central server but also saves transmission data between devices and the central server.

- Unlinkable decentralized proximity tracing protocol

Similar but not limited to the previous low-cost protocol, this protocol provides more privacy with a small cost of increased bandwidth. This approach wrapped hashed temporary IDs in a Cuckoo filter before broadcasting to other users. In this case, the set of temporary IDs uploaded from the infected user are hidden from the general public.

- Hybrid decentralized proximity tracing protocol

As a combination of the low-cost protocol and the unlinkable protocol, this hybrid protocol requires devices to generate random seeds for every given time window, such as every 2 hours, and then using these seeds to generate the temporary IDs as the low-cost protocol does. When a user is confirmed positive, s/he only needs to upload relevant seeds that generated in the contiguous time window to the central server, and then the central server broadcasts the seeds to other devices.

2.2.2 Google-Apple Exposure Notifications (GAEN) System

Google and Apple have worked together to implement a basic exposure notification system in support of the privacy-preserving tracing across Android and IOS mobile platforms [16][17].

The Google-Apple Exposure Notifications System shares the same idea as the DP-3T protocol, which distributed a list of valid temporary keys to other devices when a user tests positive. The temporary keys are refreshed every 10-20 minutes. Other devices need to use the fetched temporary keys to calculate possible temporary identifiers broadcast by the infected user. When a match is found, the temporary key is used to decrypt the metadata to get risk level based on the exposure distance and time.

Kessibi, Guillaume, et al [18] have analyzed four contact tracing applications implemented with the GAEN system to figure out whether the attackers can access the temporary keys of infected users. And they found that the temporary keys uploaded by infected users are stored in KeyFiles on a central server, which doesn't have any restrictions to download. So attackers can easily derive temporary identifiers of those infected users based on these temporary keys and expose infected users to privacy threats.

Canada's free COVID-19 exposure notification application COVID Alert [19] has adopted the open-sourced software code compatible with this Google/Apple API developed by Shopify volunteers. The number of downloads(up to April 9, 2021) is 6,448,615 and there are 9 provinces and territories on board.

2.2.3 Temporary Contact Number (TCN)

Temporary Contact Number protocol is a decentralized exposure alert protocol published by TCN Coalition [20].

TCN protocol follows the same general steps as other decentralized protocols have. It is different from others in the way of anonymous identifiers generation. An Ed25519 keypair is generated on a user’s device. The private key is called Report Authorization Key (RAK) and the public key is called Report Verification Key (RVK) [21]. The initial temporary contact key is a domain-separated hashed 256-bits value of the report authorization key. The initial temporary contact key value concatenates the RVK value and hashed with the same domain-separated hash function outputs the first usable temporary contact key. Temporary contact keys are ratcheted values. Temporary contact numbers are a 128-bits value derived from Temporary contact keys. When a user tests positive, her/his device needs to upload the public RVK, the start time, the end time, the temporary contact keys that are used to re-generate possible temporary contact numbers on other users’ devices, and optional memo fields. Like the other decentralized model, an exposure match is determined when other users’ devices compare the local received and stored temporary contact numbers with the re-generated temporary contact numbers based on the temporary contact keys fetched from the central server.

2.2.4 Private Automated Contact Tracing (PACT)

A research team led by MIT Computer Science and Artificial Intelligence Laboratory has published a decentralized automating exposure detection approach using Bluetooth low Energy called Private Automated Contact Tracing Protocol [22].

The PACT protocol is very similar to the DP-3T protocol in structure and details. The difference between these two protocols is the functions used to generate the pseudo-random IDs. In PACT protocol, a 256-bits seed is generated every hour and stored in the local memory, and then every few seconds, a pseudo-random ID is rotated using a pseudorandom function based on the seed and the current time. The pseudo-random ID generated is 28 bytes long, in order to be compatible with the

Apple Find My system. PACT models the pseudo-random ID based on Apple’s Find My protocol since the functionality is already deployed in many existing smartphones, widely distributed and adopted. As a natural extension, the existing Find My system can be used on each phone to periodically broadcast a 28-byte public key that changed every 15 minutes, and the public keys could be used as the pseudo-random IDs to be compared to detect infection exposures.

2.2.5 Privacy-Sensitive Protocols And Mechanisms for Mobile Contact Tracing

Chan, Justin, et al. [23] presented a decentralized third-party free approach to assist mobile contact tracing called Privacy-Sensitive Protocols And Mechanisms for Mobile Contact Tracing. This approach uses the same mechanism as the PACT protocol and the DP-3T protocol, all based on producing pseudo-random IDs by iteratively applying a pseudo-random function to a seed. More than other approaches mentioned, this approach assumes that the Transport Layer Security protocol is used to protect the communications between the device users and the central server.

2.3 Hybrid Approaches

In centralized or decentralized models proposed above, the main functionalities of temporary ID generation, risk analysis and notification alert are either fully performed in the central server or all delegated to the user devices. The hybrid approach split these functionalities between the server and the devices [8].

2.3.1 DESIRE

Castelluccia, Claude, et al. [24] published the DESIRE protocol which evolves from the ROBERT centralized modal and improved its major privacy issues. In DESIRE protocol, the temporary Ids that exchange between devices are generated locally on the devices, which allows the user to have full control of disclosing them or not. A Private Encounter Token is generated locally from collected temporary IDs based on the Diffie/Hellman key exchange protocol when contact length and received signal strength satisfied the exposure conditions, and this Private Encounter Token is the secret to identify the encounter of any two nodes uniquely. When users are diagnosed positive, they upload the Private Encounter Token list stored locally to a central server, while the central server is used to compare the Private Encounter Tokens uploaded with the Private Encounter Tokens provided from the other users to determine if it should alert the user based on the risk level.

In our design, we proposed a decentralized framework, which generates exchange identifiers locally and performs virus exposure detection process on each user's devices. The identifiers are unique and do not link to any personal identifier, such as phone numbers. For the notification process, users verify whether they have been exposed to the infected user locally to not leak any social information.

Chapter 3

Use Case Descriptions

There are two types of actors in our CNS: end user and government health officer.

- End user, who can download and install the contact notification application to their devices equipped with Bluetooth, using the device for exchanging encounter info and receiving notifications.
- Government health officer, who can verify and release diagnose results to end users with consents.

Based on the requirements from different types of actors, we have summarized the use cases as Register Application, Turn on/off Contact Tracing, Exchange Encounter Info, Manage Infected User, Upload Encounter Logs, Detect Virus Exposure, Notify Quarantine, and Share Family Device. We draw the Use Case Diagram(UCD) in figure 3.1 according to Unified Modeling Language (UML) [25]. In this chapter, we elaborate on these use cases in detail.

3.1 Register Application

Once an end user voluntarily opts in our CNS, s/he can download and install the mobile application to her/his mobile device from a trust website or repository. As

Figure 3.1: Use Case Diagram

the first step to activate the contact tracing notification feature, the end user needs to register her/his mobile device with the application. Table 3.1 describes the use case of this scenario.

3.2 Turn on/off Contact Tracing

An end user can interact with the main functionalities of the application after the device is successfully registered. As the end user voluntarily opts in the system, s/he can decide when and where to turn on/off the contact tracing or how long the contact tracing is on for her/his device. For example, if the end user has already been self-quarantined at home, s/he can turn off the contact tracing as s/he doesn't need to exchange encounter info within anyone at home, and when the end user ends self-quarantined, s/he can turn the contact tracing back on when needed. Table 3.2 describes the turn off contact tracing use case and table 3.3 describes the turn on contact tracing use case of such scenario.

3.3 Exchange Encounter Info

When contact tracing is on for the end user's device, the application constructs the encounter info. When any two end users closer two meters with the contact tracing turning on in their applications, the encounter info is exchanged between their devices via Bluetooth, and the encounter log is stored in the devices locally. Table 3.4 describes the use case of this scenario. If any of the end users doesn't have contact tracing turning on or Bluetooth has been turned off, the encounter info exchange will not be successful.

Table 3.1: Register Application Use Case

Use Case	UC1
Name	Register Application
Description	A user registers her/his mobile device with the CNS mobile application successfully.
Actors	End user
Pre-conditions	And end user has downloaded the mobile application of CNS from a trusted website or repository to their mobile device, followed the instruction to install the application, and granted the application permissions to use the Bluetooth feature.
Post-conditions	After successful registration, an end user is brought to the main page of the application.
Primary paths	<ol style="list-style-type: none"> 1. An end user opens the mobile application on her/his mobile device. 2. The mobile application presents the registration page. 3. An end user fills in health card number and nickname in the input fields. 4. The mobile application validates all the input. 5. The mobile application generates the random unique AppId based on the given health card number and stores it locally. 6. The mobile application displays the main page to the end user.
Alternative paths	<ol style="list-style-type: none"> 4a. Input health card number is invalid, the mobile application displays a warning message and asks the end user to fill in the field again. 5a. The mobile application is unable to generate the random unique AppId, the mobile application displays an error message.

Table 3.2: Turn off Contact Tracing Use Case

Use Case	UC2
Name	Turn off Contact Tracing
Description	An end user turns off the contact tracing on her/his device.
Actors	End user
Pre-conditions	An end user has successfully registered her/his device with the COVID-19 Contact Notification System.
Post-conditions	A red button displays on the application main page, indicating the contact tracing is turned off on the end user's device, the device will not exchange encounter info with other devices but still be able to receive notifications.
Primary paths	<ol style="list-style-type: none"> 1. An end user opens the application on her/his mobile device. 2. The mobile application displays the main page with a green button. 3. The end User taps on the green button. 4. The mobile application turns off the contact tracing on this device. 5. The mobile application displays the main page with a red button.
Alternative paths	2a. The mobile application displays the main page with a red button, indicating the contact tracing is already turned off on the end user's device, no more action is needed for the end user.

Table 3.3: Turn on Contact Tracing Use Case

Use Case	UC3
Name	Turn on Contact Tracing
Description	An end user turns on the contact tracing on her/his device.
Actors	End user
Pre-conditions	An end user has successfully registered her/his device with the mobile application of CNS.
Post-conditions	A green button displays on the mobile application main page, indicating the contact tracing is turned on on the end user's device, the device can exchange encounter info with other devices and receive notifications.
Primary paths	<ol style="list-style-type: none"> 1. An end user opens the application on her/his mobile device. 2. The mobile application displays the main page with a red button. 3. The end User taps on the red button. 4. The mobile application turns on the contact tracing on this device. 5. The mobile application displays the main page with a green button.
Alternative paths	2a. The mobile application displays the main page with a green button, indicating the contact tracing is already turned on on the end user's device, no more action is needed for the end user.

Table 3.4: Exchange Encounter Info Use Case

Use Case	UC4
Name	Exchange Encounter Info
Description	End users exchange encounter info when they come near within two meters.
Actors	End user
Pre-conditions	End users have contact tracing turning on in her/his application.
Post-conditions	The mobile application receives the encounter info and stores it locally, this process should be silent to the end user.
Primary paths	<ol style="list-style-type: none"> 1. An End user A comes near to an end user B within two meters. 2. The end user A's application sends the encrypted encounter info to the end user B's application, vice versa. 3. The end user A's application receives the encounter info and stores it locally, while the end user B's application does the same.

3.4 Manage Infected User

End users can decide whether and when to push their encounter logs to the system when needed. However, an end user can not randomly upload the info. To prevent false positive results, a government health officer needs to verify if the end user is a valid infected victim before allowing the end user to upload their encounter log. With infected users' consents, the government health officer can add infected users' health card numbers to the system once the user is verified tested positive. And then the infected users' health card numbers are encrypted and stored in the system. Table 3.5 describes the use case of this scenario.

3.5 Upload Encounter Logs

If the end user is tested positive and willing to upload her/his encounter logs to the system. The government health officer will update the end user's health card number in the system and enables the upload functionality for the end user. And then the end user can just tap on the upload button to upload her/his encounter logs stored locally in the device. Table 3.6 describes the use case of this scenario.

3.6 Detect Virus Exposure

If the end user has the CNS mobile application running on her/his device, the mobile application pulls down the verified encounter logs from the system every day and does virus exposure detection locally based on the downloaded encounter logs. If the end user has been determined exposing to the virus, the notification process is enabled. Table 3.7 describes the use case of this scenario.

Table 3.5: Manage Infected User Use Case

Use Case	UC5
Name	Manage Infected User
Description	A government health officer can verify and upload infected users' health card numbers to the system with infected users' consents.
Actors	Government Health Officer
Pre-conditions	A government health officer has verified the infected user's test result, acquired consent from the user, and has access to the system administrator page.
Post-conditions	An infected user's health card number is encrypted and stored in the system database.
Primary paths	<ol style="list-style-type: none"> 1. A government health officer visits the admin system via a secure connection. 2. The admin system presents the login page. 3. The government health officer fills in username and password in the input fields. 4. The admin system validates all the input. 5. The admin system displays the admin dashboard page. 6. The government health officer fills in an infected user's health card number in the input field and then clicks submit button. 7. The admin system validates the input. 8. The admin system encrypts the health card number and stores it in a local database. 9. The admin system pops up a success message to the government health officer.
Alternative paths	<ol style="list-style-type: none"> 4a. Username or password is wrong, the admin system displays a warning message and asks the user to fill in the field again. 7a. The Health card number is invalid, the admin system displays a warning message. 8a. The admin system failed to encrypt or save the health card number to the local database, display an error message to the user.

Table 3.6: Upload Encounter Logs Use Case

Use Case	UC6
Name	Upload Encounter Logs
Description	An infected end user can upload the encounter logs stored locally in her/his device to the system when s/he wants.
Actors	End user
Pre-conditions	The end user is verified positive by the government health officer.
Post-conditions	The end user's local encounter logs are securely uploaded to the system.
Primary paths	<ol style="list-style-type: none"> 1. An end user opens the application on her/his mobile device. 2. The mobile application presents the main page. 3. The end user navigates to the settings page. 4. Application displays upload button on the settings page. 5. An end user taps on the upload button to upload locally stored encounter logs to the system. 6. The mobile application displays a successful message when transferring is done.
Alternative paths	<ol style="list-style-type: none"> 4a. Upload button is not present for the end user as the user is not verified as an infected user nor the government health officer hasn't updated the health card number yet.

Table 3.7: Detect Virus Exposure Use Case

Use Case	UC7
Name	Detect Virus Exposure
Description	An end user's application performs virus exposure detection locally to determine if the end user has been exposed to the virus.
Actors	End user
Pre-conditions	The end user has CNS mobile running on her/his device.
Post-conditions	The notification process of the application is enabled.
Primary paths	<ol style="list-style-type: none"> 1. The mobile application pulls down the encounter logs from the system. 2. The mobile application performs virus detection action. 3. The end user is determined to have been exposed to the virus. 4. The mobile application enables the notification process.
Alternative paths	3a. If the user is not determined exposing to the virus yet, the mobile application continues pulling down new encounter logs from the system and repeats virus detection action.

3.7 Notify Quarantine

Once the notification process is enabled in the end user’s application, the application notifies the end user with a quarantine reminder to ask the end user to start a self-quarantine immediately and suggests the end user take a COVID-19 test in the near health provider if possible. Table 3.8 describes the use case of this scenario.

Table 3.8: Notify Quarantine Use Case

Use Case	UC8
Name	Notify Quarantine
Description	An end user receives a quarantine reminder notification from the application if s/he has been exposed to the virus.
Actors	End user
Pre-conditions	The end user has been detected by the application that s/he has been exposed to virus and notification process is enabled in the application .
Post-conditions	The end user receives a quarantine reminder notification.
Primary paths	<ol style="list-style-type: none"> 1. The mobile application displays a quarantine reminder notification to the end user. 2. The end user taps on the notification message to view detailed information.

3.8 Share Family Device

One device can be used to register multiple users when a group of people has limited devices and these devices need to be shared within the group. This scenario usually happens in a family where only one device is used as the family’s primary device. Each user from the group will need to repeat the registration process to register the application, but the application will only have one active account at one time. When

such sharing devices scenario happens, the user needs to go to the account list page and click on the alias that is matching to his/her account to activate the account, while the previous active account will be deactivated at the same time. Table 3.9 describes the use case of this scenario.

Table 3.9: Share Family Device Use Case

Use Case	UC9
Name	Share Family Device
Description	A group of end users can share one device when they have limited devices.
Actors	End user
Pre-conditions	Each end user from the group has successfully registered their health card number with the device.
Post-conditions	One end user is active in the application at one time.
Primary paths	<ol style="list-style-type: none"> 1. An end user A from the group obtains the device and opens the application on the device. 2. The end user A navigates to the account list page. 3. The end user A taps on the account s/he previously registered. 4. The mobile application deactivates the previous active account and activates the selected account. 5. The mobile application displays the main page with the selected account name and a green button to indicate the contact tracing is turning on.
Alternative paths	<ol style="list-style-type: none"> 3a. An end user A can't find the account s/he previously registered, s/he needs to navigate to the registration page to register the application with her/his health card. 5a. An end user A's account is not successfully activated, the application displays an error message with a red button on the main page.

Chapter 4

System Design: Architecture and Overview

A primary goal of our design is to facilitate contact tracing with a secure and privacy-preserving approach. In this chapter, we first briefly introduce our CNS architecture. We then discuss the privacy and security threats related to contact notification systems. Lastly, We explain our proposed secure privacy-preserving CNS architecture in detail as well as a protocol that is used to protect against identified threats.

4.1 Architecture

In this section, we explain the general architecture for our proposed CNS regarding components, communication protocol, and data storage.

4.1.1 Components

To reduce security and privacy risks associated with the centralized architecture designs, we proposed a decentralized approach, which contains three main components including a mobile application, a central server, and a admin dashboard. The central

server only stores minimal metadata while the information management and processing happen in the local mobile application. There are three main reasons that a contact tracing system must use a central server:

- The central server is managed by the government or health provider which can be used to verify if the uploaded data is valid from a user based on the user's testing result. If the user tests negative but still uploads her/his contact tracing data, without being verified by the central server, false positive results can be easily reproduced and might cause widespread panic.
- The central server is used as the media to broadcast notifications. Once infected people upload their contact tracing data to the central server, the central server can easily broadcast to the users when needed.
- The central server can protect the privacy of infected people to some degree. Without using a central server, when a person is identified as infected, s/he needs to send messages to all of the contacts in her/his encounter list directly, by doing this, her/his health condition goes public. With the use of a central server, it can eliminate the identity data of the infected people before broadcast the notifications.

Figure 4.1 illustrates the CNS architecture we proposed and the three components we used in our system:

- Mobile application, which is installed on users' mobile devices. When any two devices with mobile application running come close within 2 meters, the application exchanges encounter data with each other via Bluetooth. The encounter data is stored and managed locally, only being uploaded when a user wants to. The client application also notifies users with quarantine reminders when it processes the downloaded encounter logs from the remote central server and finds

Figure 4.1: The CNS Architecture

a match that indicating a user has been exposed to the virus. It implements use cases UC1, UC2, UC3, UC4, UC7, UC8, and UC9 defined in chapter 3.

- Central Server, which is used for sending notifications to the contacts of a confirmed Covid-19 case. It doesn't involve any data processing or managing. It implements use cases UC6 defined in chapter 3.
- Admin Dashboard, which is only accessible by the government health officials and used to enter the card number of a confirmed COVID-19 case, consequently triggers the notification process. It implements use cases UC5 defined in chapter 3.

Different from the GAEN [17] solution, where the payload uploaded from the mobile application to the central server when a user test positive is usually the list of random identifiers created by the infected people's devices, we transfer the encounter logs that infected user's application collected. Each entry of the encounter logs is encrypted by the sender's symmetric key before broadcasting and is encrypted again with the central server's public key when uploading to the central server. The encryption details are explained in section 4.4. This architecture helps prevent an

infected user's identity from disclosed. This is because none of the infected user's info is transferred nor the generated AppIDs be can be decrypted.

4.1.2 Communication Protocol

In this section, we explain the general communication protocols for our notification system.

- For the communication channel between client applications, we use Bluetooth Low Energy (BLE) protocol which is widely used for short-range communication with low consumption and cost [26][27].
- For the communication channel between the mobile application and central server, we use Hypertext Transfer Protocol Secure (HTTPS) protocol, which is used for maintaining a secure communication over a computer network by Transport Layer Security (TLS) protocol [28][29].
- For the communication channel between the central server and admin dashboard, we also use HTTPS protocol for secure communication.

4.1.3 Data Storage

CNS uses different methods to store information in different components.

- Mobile application: we use the SQLite [30] to store the application configuration info and encounter logs locally. Because it is light weighted and widely used in Android, IOS and other embedded system [31][32].
- Central Server: we use the MongoDB [33] to the store server configuration, user public keys, and exposed encounter logs(temporarily). Because the MongoDB is efficient than a relational database like MySQL if we do not consider the ACID transaction [34][35].

The main data structures for application configuration info, encounter logs, system configuration, and user public keys can be described as:

- $AppConfig_D$, which contains user alias, $AppId$, $isStarted$, language setting.
- $EncounterLog_D$, which contains user alias, encounter ID and encounter time.
- $ServerConfig_D$, which contains language setting.
- $UPublicKey_D$, which contains user $AppId$ and public keys.

4.2 Threat Model and Assumptions

In this section, we discuss the possible threats that are specific to the design of CNS that we propose. We first identify the attackers' goal and attack surface, then analyze the threats from these aspects: security and privacy.

4.2.1 Attacker's goal

We identify the 4 possible goals of attackers that may threaten our system below:

- G1: To obtain user's private data by accessing health status, encounter log, and location information from the central server.
- G2: To destroy the system reliability by performing a Denial-of-Service attack to a user's devices or central server.
- G3: To widely spread panic by flooding false notifications.
- G4: To reduce the effectiveness of the system by blocking notifications.

4.2.2 Attack surface

As Figure 4.2 illustrated, there are several attack surfaces in our proposed system:

S1: End user device

S2: Central server

S3: Communication channel between devices

S4: Communication channel between device and central server

Figure 4.2: Attack Surface for the CNS

4.2.3 Threat

Based on the attacker's goal discussed earlier, we identify the threats against our CNS in two aspects: security and privacy.

We discussed the Security Threats (ST) from the perspectives of data integrity, data confidentiality, and general Bluetooth technology vulnerabilities.

Data Integrity risks: malicious users can produce false positive and false negatives exposure results.

Data Confidentiality risks: malicious users can observe the communications between devices and the central server or the communications between devices to obtain data without authorization.

Bluetooth Vulnerabilities risks: malicious users can leverage the existing vulnerabilities of Bluetooth technology to obtain data, harm a user's device and the central server.

ST1: A malicious infected user may add fake encounters to the encounter list before sharing the data with the central server, which results in widespread panic due to false positive results.

ST2: A malicious server may modify the exposure database. Adding fake entries increasing the risk of developing panic while removing designated entries can reduce the efficiency of the system.

ST3: A malicious attacker can impersonate the server and deliver false positive and false negative notifications to users.

ST4: A malicious attacker can impersonate another user to broadcast fake identifiers to the other users.

ST5: A malicious infected attacker can upload a fake encounter list to the server

as s/he has been verified infected by the central server and enabled to upload encounter logs.

ST6: A malicious attacker can upload unverified exposure data to the central server database.

ST7: A malicious attacker can perform an Eavesdrop attack on the network channel between a device and the server to obtain data.

ST8: The distance between two encountering devices detected by Bluetooth may not be accurate, therefore, leads to a false positive result.

ST9: A malicious attacker can perform DoS attacks on a user's device by flooding a large volume of packets to a user's device to block the device from performing any function.

ST10: A malicious attacker can perform DoS attacks on the central server.

ST11: A malicious attacker can make use of Bluetooth's vulnerabilities, such as blue bugging, bluejacking, and bluesnarfing[36], to harm a user's device as Bluetooth is required to be on all the time for a contact tracing app.

Privacy Threats (PT) are the risks that participant users' private information is disclosed.

PT1: A malicious server can deanonymize a user by mapping the metadata such as IP address with a random identifier. Once a user's identity reveals, the malicious server can use this data to track a specific user.

PT2: A malicious server can discover a social graph based on the exposure data received.

PT3: A malicious server can derive private data such as location data from the received exposure data.

PT4: A malicious attacker can attack the server database to steal private data.

PT5: A malicious attacker can attack a user's device to steal the user's private data.

PT6: A malicious attacker can identify an infected person by comparing identifiers with the data stored in the central server database.

Figure 4.3: The relationship among privacy/security threats and attacker's goals

4.2.4 Assumption

For our proposed solution, we make the following assumptions:

A1: The database of the central server is protected by the firewall. Only an authenticated user can access the database.

A2: The general limitations of Bluetooth technology are not considered, like the proximity exchange can still successfully performed when there is a wall between two devices within a valid distance.

A3: All authenticated users are trusted. They would not share the password or other authenticated info with others.

Considering the above-mentioned STs and PTs we add specific designs to remove each identified threat. In section 4.4, we discuss design details that are specifically addressing these threats.

4.3 Solution Overview

Considering the relations among the use cases in Chapter 3, we combine the use cases into three main processes, including a registration process, encounter info exchange process, and notification process.

4.3.1 Notation

The notation used for system design is displayed in Table 4.1.

4.3.2 Registration Process

As Figure (<https://covidcts.carleton.ca/pic/register.png>) illustrates:

- 1: User downloads and installs the Application. Then User input the health card number and alias, start the registration process. Multiple users can register on the same device with various health card numbers for sharing the same device

Figure 4.4: The CNS Main Processes

Table 4.1: Notation used in Design and Implementation

Notation	Meaning
<i>AppId</i>	Mobile application identity
<i>Key_Public_Server</i>	ECC encryption public key of central server
<i>Key_Private_Server</i>	ECC Decryption private key of central server
<i>Key_Aes_User_i</i>	AES encryption key of user i
<i>Key_Public_User_i</i>	ECC encryption public key of user i
<i>Key_Private_User_i</i>	ECC Decryption private key of user i
<i>Exchange_Payload_i</i>	The exchange information of user i when they meet with others
<i>Encounter_Log_i</i>	The contact list of user i
<i>Con_Payload_i</i>	The uploaded contact list of infected user i
<i>Bro_Payload</i>	The broadcast potential infected list of central server
Sign(data, key)	signature algorithm function
Encrypt(data, key)	encryption algorithm function
Decrypt(data, key)	decryption algorithm function
Mask(data)	mask algorithm function
Key_Gen()	symmetric key generation algorithm function
Key_Pair_Gen()	asymmetric key pair generation algorithm function

scenario, while one user can also register on multiple devices with the same health card number as the user might have multiple mobile devices.

- 2: The system uses a mask method to mask the health card number and generate the App ID.

$$AppId \leftarrow Mask(Health_Card_Number)$$

- 3: Then the application automatically generates two sets of keys. One is a pair of a public key and a private key, which is used for digital signature. Another one is a symmetric key, which is used for encrypting the App ID.

$$((Key_Public_User, Key_Private_User), Key_Aes_User) \leftarrow Key_Gen()$$

- 4: Then the application uploads the public key to the central server and receives the central server public key.

4.3.3 Encounter Exchange Process

As Figure (<https://covidcts.carleton.ca/pic/exchange.png>) illustrates:

- 5: When a user turns on the application, the mobile application generates the exchange payload by the symmetric key, which is not shared with anyone else making the payload only can be decrypted by himself/herself.

$$Exchange_Payload \leftarrow Encrypt(AppId, Time : Key_Sym)$$

- 6: When two users meet with each other within two meters, they exchange the payload.
- 7: The mobile application encrypts the payload with the receiver's symmetric key and stores it in the receiver's encounter log.

$$INSERT(Encrypt(Payload : Key_Sym), Encounter_Log)$$

4.3.4 Notification Process

As Figure (<https://covidcts.carleton.ca/pic/notification.png>) illustrates:

- 8: When Person A is tested positive, the health center requests his/her health card number and computes the masked App Id, which can grant permission to Person A for uploading his/her contact list to the central server.
- 9: Person A encrypts the local contact Id list with the health center's public key and his own digital signature.

$$Con_Payload \leftarrow Sign(Encrypt(Contact_List : Key_Pub_{server}), : Key_Pri_a)$$

- 10: The central server verifies the list and adds its own digital signature to the payload, then broadcasting the info to all users.

$$Bro_Payload \leftarrow Sign(Verify(Con_Payload : Key_Pub_a), : Key_Pri_server)$$

- 11: When Person B and Person C get the list, they can verify it with the health center and trust the info. They will try to decrypt each item in the list with their own private key, the item can be decrypted successfully if they have encountered the infected user previously.

$$Is_Exposure \leftarrow Decrypt(Verify(Bro_Payload : Key_Pub_server), : Key_Pri_{b|c})$$

4.4 Detailed Design

4.4.1 AppID Generation

During the registration process, the application would generate a unique AppID, which is an identifier in the whole system. For GAEN [17] solution, it uses some random numbers as AppIDs. Although the probability of generating redundant AppIDs is low, it still doesn't guarantee the uniqueness of identifiers. If any two users have the same AppID, a false notification scenario would happen. We use the health card number with a data masking method to generate the AppID and store the AppID locally. The data masking is used to hide sensitive data by modifying the original content [37]. In our system, the mask algorithm is described below, and the unique AppID is 20 bytes long.

Besides, We have implemented this masking algorithm by Android Native Development Kit [38][39], which allows using C/C++ code within the Android, and use the code obfuscation technology [40][41], which can provide anti-compile protection to prevent malicious user can derive personal data from the metadata.

Algorithm 1: Health Card Number Masking Algorithm

Input : Health_Card_Number

Output: AppID

```

for  $i \leftarrow 0$  to  $\text{len}(\text{Health\_Card\_Number})$  do
     $T1[i] \leftarrow (\text{Health\_Card\_Number}[i] + i) * \text{Coef}[i] \bmod (10 \text{ or } 26);$ 
     $T2[\text{Position}[i]] \leftarrow T1[i];$ 
     $\text{Padding}[i:i+\text{Len}[i]] \leftarrow \text{Random}(\text{Health\_Card\_Number})[i:i+\text{Len}[i]];$ 
     $\text{AppID}[k:k+\text{Len}[i]+1] \leftarrow T[i] + \text{Padding}[i:i+\text{Len}[i]];$ 

```

4.4.2 Encounter Exchange Payload

When the contact tracing feature is switched on for both users' devices and they stay close within two meters, the encounter payload is exchanged between these two users. When devices communicate via Bluetooth, transferring plain text is not secure. In order to make sure the communication channel is secure, the application constructs the encounter payload by encrypting the unique AppID with a random value generated based on the time of that occurrence.

Besides, the payload also contains the version, which is used for the protocol upgrade in the future. Moreover, the TxPowerLevel is used to calculating the distance, and Service UUID is used to identify our tracing service.

In Bluetooth 4.2, the advertising data consists of up to 31 bytes of user-configurable data, and an additional 31 bytes can be sent as a scan response to a scan request [42][43]. So the total length that can be sent is 62 bytes at most. In our system, the length of the payload is 58 bytes, which contains 48 bytes encrypt AppID, 1 byte version, 1 byte TxPowerLevel, and 8 bytes service UUID.

Based on the encounter exchange payload, we can calculate the distance between two users using the formula [44][45][46] below:

Figure 4.5: Encounter Exchange Info

Service UUID (4*2 bytes)	Encrypt AppId (48 bytes)	TxPowerLevel (1 bytes)	Version (1 bytes)
-----------------------------	-----------------------------	---------------------------	----------------------

Figure 4.6: Encounter Exchange Payload Detail

$$distance = \begin{cases} \left(\frac{RSSI}{RSSI_{TX}} \right)^{10}, & \text{if } \frac{RSSI}{RSSI_{TX}} < 1.0 \\ \left(\frac{RSSI}{RSSI_{TX}} \right)^{7.7095} \times 0.89976 + 0.111, & \text{else} \end{cases} \quad (4.1)$$

The RSSI is the received signal strength indicator, and the $RSSI_{TX}$ is the average RSSI when the devices are in one meter distance with a different TxPowerLevel.

4.4.3 Cryptography Algorithm

In our system, we have two types of cryptography algorithms, including symmetric and asymmetric algorithms. For the symmetric algorithms, the key used for the encryption and decryption process is the same. For the asymmetric algorithms, which are also known as public-key cryptography, the key used for encryption is different from the key used for decryption but both are generated together. The symmetric algorithm is fast and requires fewer computing resources, but the main disadvantage is the distribution of the key. Because it used the same key for encryption and decryption information, it must be distributed to those who need access to the data, which poses

a security risk. In contrast, an asymmetric algorithm uses a public key for encryption and a private key for decryption to solve the problem of key distribution. However, the trade-off is that asymmetric encryption systems run very slowly compared to symmetric encryption and require more computing resources due to the very long key length [47][48]. So we choose the appropriate cryptography algorithm considering the performance and security level of different algorithms.

So, we use the Advanced Encryption Standard (AES) algorithm, which is a widely used symmetric algorithm [49][50], to encrypt and decrypt the AppID in the encounter exchange payload. Because we do not need to share the key, the encounter info would be only encrypted and decrypted locally. Considering the length limitation when using the Bluetooth Low Energy protocol, we set the block size equal to 128 bits and use Cipher Block Chaining (CBC) mode [51][52] to padding the plain text.

Besides, we use the asymmetric algorithms, specifically the Elliptic Curve Cryptography (ECC) algorithm [53][54], to encrypt and decrypt the messages in the channel between the client application and the central server, as the keys need to be shared between the client application and the central server in order to do the encryption and decryption process for uploading and downloading encounter logs. Compared with Rivest-Shamir-Adleman (RSA), another commonly used asymmetric algorithms, ECC can provide the same or higher security level with a shorter key length and better performance [55]. A 256-bit public key of the ECC provides the same security as 3072 bits of RSA public key [56][57].

Figure (<https://covidcts.carleton.ca/pic/encryption.png>) illustrates the overall encryption and decryption process.

4.4.4 Signature Algorithm

The digital signature is an important part of our system, which allows us to verify the data integrity and data origin, in case of the man-in-the-middle attack. When

the client application communicates with the central server, they need to sign and verify the data is sending from each other directly. We use the elliptic curve signature scheme Edwards-curve Digital Signature Algorithm (EdDSA), which combines ECC and Digital Signature Algorithm (DSA) [58]. Specifically, we use the Ed25519, which combines the SHA-512 [59] and Curve25519 for the signing and verifying the message [60].

Figure (<https://covidcts.carleton.ca/pic/signature.png>) illustrates the digital signature process.

4.4.5 Database Design

For the data storage in the mobile application and central server, we use two different types of databases, including SQLite and MongoDB. The SQLite is used to store the information locally, and do not share this information to the central server only if he/she is confirmed and agrees to upload the encounter logs. The median incubation period for the COVID-19 virus is around 5 days, and people who have been exposed to the virus develop symptoms within 11 days according to medical research [61]. World Health Organization (WHO) [62] suggests people who have been infected or exposed to the virus should do self-quarantine for at least 14 days. Hence, the encounter logs' minimal valid period is 14 days, based on this rule, the central server only stores the encounter logs temporarily, which would be deleted after 14 days. Besides, all the encounter logs are encrypted, the central server cannot decrypt them and know who is in the encounter logs.

4.4.6 Detailed Process

For the registration process, after a user downloads and install the application, upon entering one's health card number in the input field, the application would generate

Encounter Log	Account Info
log : text	Alias : varchar
Insert_Date : timestamp	Is_Enable : boolean
	Language : varchar
	Is_BLE_Started : boolean

Figure 4.7: Database Schema of Client Application

UPublicKey	Encounter Log	Infected Person
AppId : varchar	log : text	AppId : varchar
Public_Key : text	Insert_Date : timestamp	Insert_Date : timestamp
Insert_Date : timestamp		

Figure 4.8: Database Schema of Central Server

the unique AppID and immediately discard the health card number after. It is important to know that a user's health card number is only used to generate a unique ID and will be saved neither locally nor elsewhere. Then the application generates two sets of keys and stores them locally, including a public-private key pair and a symmetric key. Then the application pulls the central server's public key, encrypt its own public key, and uploads it to the central server along with the AppID. The central server would store the user public key in the database locally.

When two users carrying their devices with the contact tracing function switched on, come to close proximity, the application would share the payload via Bluetooth. Once the application receives the payload via Bluetooth, it encrypts the payload with the symmetric key again and stores the encrypted payload along with a timestamp into the encounter log locally.

When a user is confirmed to carry Coronavirus, a health officer will enter this person's health card number. The server then masks the AppID on the central server.

Figure 4.9: Registration Detailed Process

Figure 4.10: Encounter Exchange Detailed Process

The encounter log list stored in this infected user's device and has been encrypted with the central server's public key and his/her digital signature is sent to the central server. Communication between the application and the central server utilizes HTTPS, which provides the authenticated encryption. And the digital signature enables one to verify the integrity and data origin authentication of the broadcasted encounter log. Once the central server receives such a list, it verifies the digital signature and the AppID first, if the payload is confirmed to be valid, the central server adds its own digital signature to the encounter log and broadcasts the list.

Figure 4.11: Notification Detailed Process

When a user receives an encounter log from the central server, it verifies the signature first and then the application tries to decrypt each item from the list using its own private key. As the payload exchanges between the two devices are encrypted with each one's symmetric key, the application should only be able to decrypt an item that is encrypted previously using its own symmetric key. Once decryption is done successfully, the application displays a notification indicating that the owner of this account has been exposed to a virus in the last 14 days.

Chapter 5

Implementation and Evaluation

In this chapter, we give implementation details for our solution and discuss how we make decisions for each step.

5.1 Implementation

In the section, we explain our system implementation in detail and discuss the choices we made.

Hardware and Software Components: we summarize the capabilities of each component in this section and explain the implementation of these components.

- Mobile Device: An Android device with operating system versioning from 7.0 to 10, with inbuilt Bluetooth feature version 4.0 or up and WIFI module. The mobile device is the primary device for a user to download and install the mobile application of CNS with, it should be able to connect to the network for receiving notifications and have enough memory for basic computation. In our prototype, we have used three different devices for testing:
 - 1) Xiaomi 8 with Android 8, 6GB RAM and 64GB memory
 - 2) Sony with Android 7, 2GB RAM and 32GB memory

3) Huawei with Android 7, 2GB RAM and 32GB memory

- **Mobile Application:** An Android application has functionalities of registration, turning on/off contact tracing feature, generating key pairs, exchanging encounter info, pulling notifications, and verifying virus exposure. In our prototype, we have implemented the application with Java and C++.
- **Central Server:** A Central Server has functionalities of receiving and verifying uploaded encountered logs, storing valid logs in a database, and broadcasting the exposure info to all users. In our prototype, we implemented these features in Java and hosted the service on a Linux machine with Ubuntu 20.04 LTS, 16GB RAM, and 512GB disk memory.
- **Database:** The database is used for temporarily storing uploaded encounter logs, it has been configured with a trigger that automatically deleting records older than 14 days. The database is only accessible to authenticated users. In our prototype, we used MongoDB as the database on the central server.

5.1.1 Central Server Configuration

The first step for contact tracing is system administrator needs to configure a valid central server. As our assumption mentioned earlier, the system administrator is a trusted authenticated user and will not do any harm to the server. There are two steps for the central server configuration:

The system administrator configures a database instance on the central server. Two tables are created in the database, one for storing the received AppID and public key, another one for storing the uploaded encounter logs. The system administrator will configure a trigger for the database to automatically delete older records.

The system administrator deploys the web service jar file with the Apache Tomcat

Figure 5.1: Central Server Configuration Process

web server and starts up the services. We have implemented the web services with the Java SpringBoot framework in our prototype. Once the services are up, it will automatically generate a public-private keypair, connect to the database and publish web services for receiving and broadcasting messages.

5.1.2 Implementation Detail

After a user installs the mobile application on her/his device, s/he clicks on the app icon to launch the application. The application displays the registration page to the user, as illustrated in Figure 5.2. There are two input fields for a user to enter her/his health card number and nickname. Once a user clicks on the create account button, the application validates the user's input and triggers the registration process.

As Figure 5.3 illustrates, the application brings a user to the main page when the registration process finishes. There is a green button indicating the contact tracing

Why do we need your health insurance number?

Pourquoi avons-nous besoin de votre numéro d'assurance maladie?

English Version

French Version

Figure 5.2: Registration Page

feature is on. A user can tap on the button to switch off the contact tracing feature when needed, and then the button turns red.

English Version

French Version

Figure 5.3: Main Page

When a group of people needs to share one device, the device can be registered with multiple users. Each user from the group needs to repeat the registration steps as described above to register the device, but the device will only have one active account at each point in time. As Figure 5.4 illustrates, a user can navigate to the account list page and clicks on her/his alias to activate the account, the other accounts

are deactivated in the meantime.

Figure 5.4: Family Device Sharing Page

When a user is confirmed to have been exposed to the virus, the notification page displays as Figure 5.5 illustrates.

Figure 5.5: Notification Page

5.2 Evaluation

In this section, we evaluate our solution based on three aspects: compliance to the OPC Contact Tracing System Development Guideline [3], the fulfillment of functional requirements, and security analysis.

5.2.1 OPC Contact Tracing System Development Guideline

As mentioned in section 1.3, a critical guideline should be used to evaluate any contact tracing technologies to ensure the design is secure and privacy preserving. In this section, we discuss how our proposed solution follows the OPC principles [3]here:

- Consent and trust: The use of our notification system is purely voluntary and we grant users full control of their data. Users need to give consents in the steps of application registration and encounter logs uploading as these steps involving releasing their locally stored encounter data to the central server for notification purposes. Users can turn off contact tracing or opt-out from the system at any time with metadata removed from the device as well.
- Legal authority: A legal basis must be clear and meaningful for a CTS, in our proposed solution, the central server must be hosted by the government or health authorities as to the encounter logs that infected people uploaded with consents should be verified and only used for quarantine notifications purpose.
- Necessity and Proportionality: Two-meter physical distancing has been confirmed as one of the most effective measures to reduce the spread of illness during an outbreak. In our proposed system, each encounter log is generated when any two users come close within two meters, it has been tailored in a way that it is only used as a piece of evidence for proving any two users have

encountered each other without disclosing any personal information or location information. Such encounter logs meet our specific purpose for notifying users when they are proved to be exposed to the virus. The data collection is minimized in our CNS.

- **Effectiveness:** As our contact notification application installed on a user's device pulls encounter logs uploaded by infected users from the central server at least once a day for processing virus exposure detection locally. If any encounters of the user have been confirmed infected, s/he should get the notification within 24 hours.
- **Minimal intrusiveness:** We have only leveraged Bluetooth technology for data transferring, and when a user installs the application to their devices, the application only requires permission for Bluetooth feature, user can disable any other permissions and the application still works. The only data we have used in our design is the health card number but it is not persistent anywhere as it is only used for generating the unique masking AppID, which is part of the constructed encounter data payload. Most importantly, this health card number is immediately discarded as soon as an AppID is generated. It is neither saved on the running device itself nor transferred elsewhere. Other than this data, we don't require any other personal data in our system.
- **Purpose Limitation:** The purpose of our system is to notify users when they have been exposed to the COVID-19 virus and advise them to start a self-quarantine in order to slow down the spread of the virus. Based on this purpose, we don't require users to upload any personal information other than encounter logs stored in their device, as encounter logs are the only measurement for virus exposure detection.

- **De-identification:** We have adopted a decentralized approach for our design, in which the central server is only used for encounter log uploading and broadcasting, as all the data is encrypted with a public key, the central server is not able to derive any personal information from the uploaded encounter logs or target specific user based on the communications. The virus exposure detection and notification steps happen in each device locally. Therefore, the encounter logs sent by the central server cannot disclose which user has triggered such notification. In other words, neither a user can know who he/she has been in contact with contracted Coronavirus, nor can any health officials working on the central server can know who would receive a notification triggered by a confirmed Covid-19 case.
- **Time-Limitation:** As our proposed solution only aims at assisting people to slow down the spread of the illness during this difficult time. Furthermore, a contact that had happened more than 14 days is no longer relevant in this specific Covid-19 pandemic. Therefore, the central server automatically deletes any AppID of a confirmed Covid-19 case that has been entered for more than 14 days. 14 days after the last confirmed case is entered into the server, no data remains on the central server. And after the pandemic, users can easily remove/delete this notification app from their mobile devices.
- **Transparency:** All the processes in our design are transparent to users and this design should only be used under the governance of the government or health authorities.
- **Accountability:** If our proposed system goes production, the development should be reviewed and audited by oversight authorities to ensure accountability.

- **Safeguards:** Any data transferred between devices or between a device and the central server is encrypted. For the communication between the device and the central server, we used a digital signature to verify the sender is trusted. The database of the central server is protected by the firewall and only accessible for authenticated users while all authenticated users are trusted. The security analysis on the chosen encryption algorithms falls outside the scope of this thesis.

5.2.2 Functional Tests

In this section, we describe the functional tests we have performed based on the use cases of our proposed CNS in chapter 3.

Before functional testing, We have published the implemented application to our testing server and used three testing devices with different configurations to download and install the application for testing compatibility, the application can be installed and works fine in all three devices.

We start with testing Registration functionality. The basic Registration functionality for our client application is to allow user to create an account with her/his health card number for notification purpose. We have considered multiple registration scenarios, such as a user only has one primary device, a user has more than one primary device and a group of users share one primary device. These testing scenarios are listed in Table 5.1, Table 5.2, and Table 5.3. The testing results indicate that one user can register multiple mobile devices as s/he needed with the same health card number and one device can also be shared within a trusted group with multiple health card numbers. Each account manages its data and not interfere with others.

In the field testing for Exchange Encounter Info functionality, as Table 5.4 lists, we have switched on the contact tracing feature for all devices and placed them in different distances to test whether encounter info are successfully exchanged. The

Table 5.1: A single user registers with a single device

Function Name: Registration	
Test Scenario: A single user registers with a single device	
Test Case	RAT1
Description	Check a new account can be successfully created in the client application when user register with a valid health card number.
Test Steps	<ol style="list-style-type: none"> 1. Open the application on mobile device. 2. Input health card number and nickname. 3. Tap on the register button.
Test Data	Health card number = 123-456-7890-AB Nickname = Kevin
Expected Result	Registration should be successful and user is brought to main page with new account being active.
Actual Result	As expected.
Test Case	RAT2
Description	Check a new account can't be created in the client application when user register with a invalid health card number.
Test Steps	Repeat steps 1-3 in test case PAT1.
Test Data	Health card number = 123-456-7890-A! Nickname = Michelle
Expected Result	Registration should be failed and user is asked to input the correct health card number.
Actual Result	As expected.
Test Case	RAT3
Description	Check a new account can't be created in the client application when user tries to register with the same health card number for the second time on the same device.
Test Steps	<ol style="list-style-type: none"> 1. Repeat steps 1-3 in test case RAT1 to have the health card number registered once. 2. Repeat steps 1-3 in test case PAT1 again.
Test Data	Health card number = 123-456-7890-AB Nickname = Kevin
Expected Result	Registration should be failed and user is told the health card number has already been registered.
Actual Result	As expected.

Table 5.2: A single user registers with multiple devices

Function Name: Register Application	
Test Scenario: A single user registers with multiple devices	
Test Case	RAT4
Description	Check new accounts can be successfully created in the client application when user register with the same health card number on different devices.
Test Steps	<ol style="list-style-type: none"> 1. Repeat steps 1-3 in test case RAT1 to have the health card number registered once on device A. 2. Repeat steps 1-3 in test case PAT1 again to register with the same health card number on device B.
Test Data	Health card number = 123-456-7890-AB Nickname = Kevin
Expected Result	Registration should be successful for user on both devices.
Actual Result	As expected.

Table 5.3: Multiple users register with a single device

Function Name: Register Application	
Test Scenario: Multiple users register with a single device	
Test Case	RAT5
Description	Check more than one account can be successfully created in the client application when different users register with valid health card numbers on the same devices.
Test Steps	<ol style="list-style-type: none"> 1. Repeat steps 1-3 in test case RAT1 to register an account for user Kevin. 2. Repeat steps 1-3 in test case PAT1 again to register an account for user Michelle.
Test Data	Health card number = 123-456-7890-AB, 123-456-7890-CD Nickname = Kevin, Michelle
Expected Result	Registration should be successful for both users, and the new accounts are listed in the account list page.
Actual Result	As expected.

testing result indicates that any two devices only exchange the encounter info when the distance between them is shorter than two meters.

As it occurs naturally that obstacles, such as walls and windows might deteriorate the Bluetooth signal, we also test the scenario that whether the encounter info is exchange successfully when there is an obstacle placed between two devices. We describe this scenario in Table 5.4 Based on the testing result, we find the signal can still go through an obstacle and the encounter info is successfully exchanged. However, this is also an limitation for Bluetooth-based contact tracing system, as this might generate potential false positive result when detecting virus exposure. For example, in a townhouse where neighbours sharing the same wall, they can still exchange encounter info while they are both staying at home. If one of them is confirmed to have been exposed to the virus, the other one will also receive notification even though they haven't meet each other. For such scenario, we encourage users to turn off contact tracing feature while they are at home to eliminate such impact, but in real life, such scenario is not avoidable. For future work we can do is to determine if two users encounter based on Bluetooth signal strength, but this is not in the scope of this thesis.

Then we turn off the contact tracing feature in one of the device and repeat the above testing steps, we noticed that no encounter info exchange happens when the feature is turned off. If user's mobile device restarts anytime after the user has turned off the contact tracing feature, the contact tracing feature is still disabled when user opens the application again until user gives consent to turn the feature back on. These scenarios are listed in Table 5.6 and Table 5.7.

Once a user is confirmed infected, a health officer enters the health card number of this user in the admin dashboard which triggers the encounter log uploading process. Only infected users are granted with uploading permission to upload encounters logs to the central server. In our testing, we have repeated these steps for testing the

Table 5.4: Exchange encounter info with different distance between the two devices

Function Name: Exchange Encounter Info	
Test Scenario: Exchange encounter info with different distance between the two devices.	
Test Case	EEIT1
Description	Check encounter info is exchanged between two devices when the distance is smaller than two meters.
Test Steps	<ol style="list-style-type: none"> 1. Download and install client applications in device A and B. 2. Register new accounts with different health card numbers for device A and B individually. 3. In a playground, one person holds device A with contact tracing turning on in the client application and stand still. 4. Starting with having more than two meters distance from where device A is placed, another person holds device B and walking towards the device A.
Test Data	Health card number = 123-456-7890-AB, 123-456-7890-CD Nickname = Kevin, Michelle
Expected Result	Alarm alerts for both devices when the distance is smaller than two meters, indicating a new entry is added to encounter logs for both devices.
Actual Result	As expected.
Test Case	EEIT2
Description	Check encounter info is not exchanged between two devices when the distance is greater than two meters.
Test Steps	<ol style="list-style-type: none"> 1. Repeat steps 1-3 in test case EEIT1. 2. Starting with where device A is placed, another person holds device B and walk away from device A.
Test Data	Health card number = 123-456-7890-AB, 123-456-7890-CD Nickname = Kevin, Michelle
Expected Result	Alarm alerts go off when the distance is greater than two meters, indicating no new entry is added to encounter logs for any of the devices.
Actual Result	As expected.

Table 5.5: Exchange encounter info with obstacles placed between the devices

Function Name: Exchange Encounter Info	
Test Scenario: Exchange encounter info with obstacles placed between the devices.	
Test Case	EEIT3
Description	Check encounter info is exchanged between two devices when the distance is smaller than two meters with obstacles placed in between.
Test Steps	<ol style="list-style-type: none"> 1. Repeat step 1-2 in test case EEIT1. 2. Using a wall as the mocking obstacle, place device A on one side of the wall with contact tracing turning on in the client application. 3. Hold device B on the other side of the wall with more than two meters distance and walk towards the wall.
Test Data	Health card number = 123-456-7890-AB, 123-456-7890-CD Nickname = Kevin, Michelle
Expected Result	Alarm alerts for both devices when the distance is smaller than two meters, indicating a new entry is added to encounter logs for both devices.
Actual Result	As expected.

Table 5.6: Turn off contact tracing

Function Name: Turn on/off Contact Tracing	
Test Scenario: Turn off contact tracing.	
Test Case	TOCT1
Description	Check encounter info is not exchanged between two devices when one of the device has contact tracing feature turned off in the application.
Test Steps	<ol style="list-style-type: none"> 1. Repeat step 1-2 in test case EEIT1. 2. Place device A and device B within two meters with contact tracing feature on for both devices. 3. Turn off the contact tracing feature on device A.
Test Data	Health card number = 123-456-7890-AB, 123-456-7890-CD Nickname = Kevin, Michelle
Expected Result	Alarm alerts go off, indicating no encounter token exchange happens anymore.
Actual Result	As expected.
Test Case	TOCT2
Description	Check contact tracing feature is still disabled after user restart her/his device when contact tracing feature is turned off in the application.
Test Steps	<ol style="list-style-type: none"> 1. Repeat step 1-3 in test case TOCT1. 2. Restart device A. 3. Open application in device A.
Test Data	Health card number = 123-456-7890-AB, 123-456-7890-CD Nickname = Kevin, Michelle
Expected Result	No alarm alert, indicating no encounter token exchange happens after the device is restarted.
Actual Result	As expected.

Table 5.7: Turn on contact tracing

Function Name: Turn on/off Contact Tracing	
Test Scenario: Turn on contact tracing.	
Test Case	TOCT3
Description	Check encounter info is exchanged between two devices when both devices have contact tracing feature turned on in the application.
Test Steps	<ol style="list-style-type: none"> 1. Repeat steps in TOCT2. 2. Turn on contact tracing in application for device A again.
Test Data	Health card number = 123-456-7890-AB, 123-456-7890-CD Nickname = Kevin, Michelle
Expected Result	Alarm alerts for both devices, indicating encounter token exchange happens.
Actual Result	As expected.
Test Case	TOCT4
Description	Check contact tracing feature is still enabled after user restart her/his device when contact tracing feature is turned on in the application.
Test Steps	<ol style="list-style-type: none"> 1. Repeat steps in test case TOCT3. 2. Restart device A. 3. Open application in device A.
Test Data	Health card number = 123-456-7890-AB, 123-456-7890-CD Nickname = Kevin, Michelle
Expected Result	Alarm alerts for both devices, indicating encounter token exchange happens.
Actual Result	As expected.

encounter log uploading feature, as listed in Table 5.8 and Table 5.9.

Table 5.8: Add infected user to infected user list

Function Name: Manage Infected User	
Test Scenario: Add infected user to infected user list.	
Test Case	MIU1
Description	Check a new entry has been added to the infected user list.
Test Steps	1. Login to admin system 2. Input infected user's health card number and click submit.
Test Data	Health card number = 123-456-7890-AB
Expected Result	A new entry has been added to the infected user list in database, the new entry is not input health card number but a masked AppId derived from it.
Actual Result	As expected.
Test Case	MIU2
Description	Check a duplicated entry can't be added to the infected user list.
Test Steps	1. Repeat steps in test case MIU1. 2. Input infected user's health card number again and click submit.
Test Data	Health card number = 123-456-7890-AB
Expected Result	Only one entry has been added to the infected user list in database, no duplicated entry exists in the infected user list.
Actual Result	As expected.

Once the encounter logs are uploaded to the testing server, only the users who have encountered the infected user should receive quarantine notifications. We also test the scenario that two users met after one of them is already confirmed as infected, in this case, the other user will receive a notification right away. These scenarios are listed in Table 5.10.

When we perform the test for registration functionality, we mentioned one device can be shared with a group of trusted users and each account should manage its data.

Table 5.9: Verified infected user uploads encounter logs

Function Name: Upload Encounter Logs	
Test Scenario: Verified infected user uploads encounter logs.	
Test Case	UEL1
Description	Check encounter logs have been uploaded by the verified infected user.
Test Steps	<ol style="list-style-type: none"> 1. Repeat steps in EEIT1 to have device A and device B registered for user Kevin and Michelle separately, place two devices together to exchange encounter info and generate encounter logs. 2. Login to admin system, use user Kevin as mocked infected user. 3. Input infected user's health card number and click submit. 4. Open application on device A and click upload button.
Test Data	Health card number = 123-456-7890-AB, 123-456-7890-CD Nickname = Kevin, Michelle
Expected Result	New encounter logs are uploaded to testing server and updated in database.
Actual Result	As expected.
Test Case	UEL2
Description	Check encounter logs can't be uploaded by the users that are not verified as infected.
Test Steps	<ol style="list-style-type: none"> 1. Repeat step 1-3 in UEL1. 2. Open application on device B.
Test Data	Health card number = 123-456-7890-AB, 123-456-7890-CD Nickname = Kevin, Michelle
Expected Result	No upload button is available to user Michelle on device B as she is not verified as infected.
Actual Result	As expected.

Table 5.10: Quarantine notification

Function Name: Quarantine Notification	
Test Scenario: User receives notification if s/he has encountered a confirmed COVID-19 case.	
Test Case	QN1
Description	Check notification have been received by user who encounters a person before he/she confirmed as a COVID-19 case.
Test Steps	<ol style="list-style-type: none"> 1. Register device A, B, C for user Kevin, Michelle and Judy respectively. 2. Put device A and B together to make encounter exchange happen for Kevin and Michelle. 3. Put device B and C together to make encounter exchange happen for Judy and Michelle. 4. Use user Kevin as mocked infected user and submit his health card number with admin system.
Test Data	<p>Health card number = 123-456-7890-AB, 123-456-7890-CD, 123-456-7890-EF</p> <p>Nickname = Kevin, Michelle, Judy</p>
Expected Result	Notification is received by device B for Michelle, while no notification is received by device C for Judy as Judy hasn't met with Kevin.
Test Case	QN2
Description	Check notification have been received by user who encounters a confirmed COVID-19 case.
Test Steps	<ol style="list-style-type: none"> 1. Register device A, B, C for user Kevin, Michelle and Judy respectively. 2. Put device A and B together to make encounter exchange happen for Kevin and Michelle. 3. Use user Kevin as mocked infected user and submit his health card number with admin system. 4. Put device A and C together to make encounter exchange happen for Judy and Kevin after Kevin is confirmed as infected.
Test Data	<p>Health card number = 123-456-7890-AB, 123-456-7890-CD, 123-456-7890-EF</p> <p>Nickname = Kevin, Michelle, Judy</p>
Expected Result	Notifications are received by both device B for Michelle and device C for Judy.

This means if user A and B are both registered on the same device when A is the only person that has been exposed to the virus and receives quarantine notifications, the application shouldn't display a notification when B's account is active. We illustrate this scenario in table 5.11

Table 5.11: Share device

Function Name: Share Family Device	
Test Scenario: Each account on the same device should manage its own notifications.	
Test Case	SFD1
Description	Check notification have been received by the user who encounters a COVID-19 case in a family while others don't.
Test Steps	<ol style="list-style-type: none"> 1. Register device A for user Kevin, B for Michelle and Judy respectively. 2. Activate Michelle's account on device B, put device A and B together to make encounter exchange happen for Kevin and Michelle. 3. Use user Kevin as mocked infected user and submit his health card number with admin system. 4. Notice device B receives notifications when Michelle is active. 5. Activate Judy's account on device B.
Test Data	<p>Health card number = 123-456-7890-AB, 123-456-7890-CD, 123-456-7890-EF</p> <p>Nickname = Kevin, Michelle, Judy</p>
Expected Result	Notification is not sent to device B when Judy's account is active.

Based on our testing, our implemented prototype can fulfill the functionalities required by a Contact Tracing Notification System: registration, turning on/off contact tracing, switching account, contact exchange, exposure detection, and notification.

We have tested our implemented mobile application with families, friends, and people within the research group. Approximately over 40 people aged from 9 to 80 have participated in the testing to ensure the functionalities are easy to use by users of all age groups.

5.2.3 Informal Usability Study

After functional tests, we also performed an informal usability study with all participants. We have collected user input on the choice of color, text style, size and wording, button actions and page layout design. More precisely, we posted our draft design in a google document shared with all participants. We asked the participants to provide feedback on each of the above-mentioned user interface items using the comments function. Figure 5.6 illustrates an example of our usability study on the main page.

Fix 4: Main Page

We really want to minimize the text and maximize ease of use.

The button should be placed much higher up (After all, it provides the main functionality) so it is seen first.

I'm not too fond of this slide toggle (Adds extra complication in my opinion), can it just be a toggle button like you see with wifi or a play/pause?

Figure 5.6: Usability Study on the Main Page

5.2.4 System Stress Tests

We further performed a stress test on both central server and individual mobile devices to analyze the robustness and reliability of our system. Considering the database in

the central server, each entry in the encounter log table is 136 bytes (128 bytes of encrypted AppId and 8 bytes of timestamp). Assume the confirmed Covid-19 cases have increased by 50,000 every day, and each confirmed individual has 1000 encounters. Then the encounter logs consume 7GB of memories in the central server database per day. Each Encounter log is deleted after 14 days. Therefore, the maximum load in the database is 98GBs. MongoDB can easily handle such data usage in a distributed environment. Besides the database, all the other services are implemented as stateless and can be easily scaled up. For the data storage of a mobile device, each device only needs 2M space if the owner meets 1000 encounters per day.

5.2.5 Security and Privacy Analysis

In this section, we explained how we address the threats outlined in Section 3.1.3 in our design.

[Threat ST1, ST4, ST5]: The encounter tokens exchanged between devices are encrypted by each device's symmetric keys and stored in the local device with a timestamp. These tokens can only be decrypted by the symmetric keys used to encrypt them. Malicious infected user is not able to add any valid encounter log to the list without having the making AppID and the symmetric key, as it can't be decrypted and verified later in the notification process.

[Threat ST2, PT1, PT2, PT4]: The exposure database in our design is protected by a firewall and only accessible by the authenticated users. On the central server, as soon as a patient's health number is entered, it generates the same AppID automatically. This health number is not saved on the server, nor sent anywhere else. Data related to each patient is saved on the server for 14 days only. It is permanently deleted afterward. Based on our assumption, all the authenticated users are trusted, thus the exposure database is trusted. Also, the encounter log uploaded to the central server is encrypted with symmetric keys, the central server is not able to derive info

from it and discover the social graph.

[Threat ST3]: Messages exchanged between device and server are sent along with digital signature according to our design, which is used to verify the identity of the sender. Impersonate false notification will not be accepted by the device.

[Threat ST6]: The health care officer verifies a user's health card number before allowing them to upload the encounter log to the central server. Unverified exposure data will not enter into the system.

[Threat ST7]: All the data transferred between the device and the server is encrypted with a public key of the device or the central server. Even if the malicious attacker is observing the channel, the attacker is not able to derive useful info from encrypted messages.

[Threat ST8, ST9, ST11, PT5]: False positive results might happen when the distance between two devices detected by Bluetooth is not accurate or there are obstacles such as walls between devices that can't be detected by Bluetooth (however, the latter also blocks the virus spread). And there are other common issues caused by Bluetooth usage, such as flooding Bluetooth messages, Bluetooth technology common vulnerabilities analysis is out of the scope of this thesis.

[Threat ST10, PT6]: In our design, the central server is protected by a firewall. When the malicious attacker attempts DoS attacks on the server, filtering the harmful IPs to block the traffic can resolve this issue. Furthermore, only generated AppIDs are temporarily stored on the central server for 14 days. Even if this information is obtained by an adversary, no meaningful information relates to any real person can be deducted.

[Threat PT3]: There is no location data transferred or used as metadata in our design.

Chapter 6

Conclusions and Future Work

6.1 Conclusions

The COVID-19 pandemic has been impacting everyone’s life for over a year. A CTS enables the government to efficiently slow down the spread of the virus. However, we believe that an optimal solution to the trade-off between privacy and performance for CTSs is not trivial to obtain. In this thesis, we propose an implementation of a secure and privacy-preserving COVID-19 Notification System. We discuss in detail an actual implementation of our CNS for Android devices, based on our proposed design. The latter follows the CTS development guidelines [3] of the Office of the Privacy Commissioner of Canada (OPC) and addresses potential privacy and security threats against Bluetooth-based CTSs. With its simple user interface and minimum data usage, our design supports registration, turning on/off of the contact-tracing feature, switching accounts, encounter information exchange, exposure detection and notification.

6.2 Future Work

Some limitations remain in our implemented CNS solution:

L1: In our solution, the exposure detection and notification process runs locally in each user’s device. In order to strictly preserve privacy, our current design requires the device try to decrypt all entries received from the central server. Should our system be widely adopted, the list of encounters received from the central server could be long. Therefore, resulting computations may lead to draining the device’s battery faster than normal. This does constitute a scalability limitation of the current implementation.

L2: Given the Bluetooth feature of the device using our CNS is required to be on all the time in order to perform the contact exchange process, its inherent security vulnerabilities can be potentially dangerous for a user’s private data and device [63][64]. A sensor-based CNS could be a more desirable solution.

Based on these limitations, we believe future work is required to enhance our current solution. In particular, the data structure used for the encounter token can be improved to shorten the payload exchanged between devices and the central server. Also, applying filters to the encounter list that is used for exposure detection may help with the scalability limitation.

Finally, should our CNS be used in the context of a specific organization, proper roles and permissions management can restrict the power each authenticated user has and thus prevent ‘power’ users from abusing the system.

List of References

- [1] A. S. B. K. . C. Z. Masoodi, M.J., “Race to trace: Security and privacy of covid-19 contact tracing apps..” <https://www.cybersecurepolicy.ca/racetotrace>, 2020.
- [2] N. C. C. (NCC), “Statement on privacy-respecting and trust-worthy covid-19 tracing apps..” <https://uwaterloo.ca/cybersecurity-privacy-institute/news/coronavirus-statement>, 2020.
- [3] O. of the Privacy Commissioner of Canada, “Supporting public health, building public trust: Privacy principles for contact tracing and similar apps..” https://www.priv.gc.ca/en/opc-news/speeches/2020/s-d_20200507/, 2020.
- [4] O. of the Privacy Commissioner of Canada, “Contact tracing technologies and protocol..” https://priv.gc.ca/en/privacy-and-transparency-at-the-opc/opc-parl-bp/indu_20200529/ctt_20200529/, 2020.
- [5] J. A. Sacks, E. Zehe, C. Redick, A. Bah, K. Cowger, M. Camara, A. Diallo, A. N. I. Gigo, R. S. Dhillon, and A. Liu, “Introduction of mobile health tools to support ebola surveillance and contact tracing in guinea,” *Global Health: Science and Practice*, vol. 3, no. 4, pp. 646–659, 2015.
- [6] A. Trivedi, C. Zakaria, R. Balan, A. Becker, G. Corey, and P. Shenoy, “Wifitrace: Network-based contact tracing for infectious diseases using passive wifi sensing,” *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, vol. 5, no. 1, pp. 1–26, 2021.
- [7] C. Gomez, J. Oller, and J. Paradells, “Overview and evaluation of bluetooth low energy: An emerging low-power wireless technology,” *Sensors*, vol. 12, no. 9, pp. 11734–11753, 2012.

- [8] N. Ahmed, R. A. Michelin, W. Xue, S. Ruj, R. Malaney, S. S. Kanhere, A. Seneviratne, W. Hu, H. Janicke, and S. K. Jha, “A survey of covid-19 contact tracing apps,” *IEEE Access*, vol. 8, pp. 134577–134601, 2020.
- [9] J. Bay, J. Kek, A. Tan, C. S. Hau, L. Yongquan, J. Tan, and T. A. Quy, “Blue-trace: A privacy-preserving protocol for community-driven contact tracing across borders,” *Government Technology Agency-Singapore, Tech. Rep*, 2020.
- [10] G. of Alberta, “Help prevent the spread of covid-19 with the abtracetoegether mobile app.” <https://www.alberta.ca/ab-trace-together.aspx>, 2020.
- [11] P.-P. e.V. i.Gr., “Pan-european privacy-preserving proximity tracing..” <http://github.com/pepp-pt>, 2020.
- [12] C. Castelluccia, N. Bielova, A. Boutet, M. Cunche, C. Lauradoux, D. Le Métayer, and V. Roca, “Robert: Robust and privacy-preserving proximity tracing,” 2020.
- [13] A. S. Krishnan, Y. Yang, and P. Schaumont, “Risk and architecture factors in digital exposure notification.,” *IACR Cryptol. ePrint Arch.*, vol. 2020, p. 582, 2020.
- [14] S. Vaudenay, “Centralized or decentralized? the contact tracing dilemma.” *Cryptography ePrint Archive*, Report 2020/531, 2020. <https://eprint.iacr.org/2020/531>.
- [15] C. Troncoso, M. Payer, J.-P. Hubaux, M. Salathé, J. Larus, E. Bugnion, W. Lueks, T. Stadler, A. Pyrgelis, D. Antonioli, *et al.*, “Decentralized privacy-preserving proximity tracing,” *arXiv preprint arXiv:2005.12273*, 2020.
- [16] Apple, “Privacy-preserving contact tracing..” <https://covid19.apple.com/contacttracing>, 2020.
- [17] Google, “Exposure notifications: Using technology to help public health authorities fight covid-19..” https://www.google.com/intl/en_us/covid19/exposurenotifications/, 2020.
- [18] G. Kessibi, M. Cunche, A. Boutet, C. Castelluccia, C. Lauradoux, D. Le Métayer, and V. Roca, “Analysis of diagnosis key distribution mechanism in contact tracing applications based on google-apple exposure notification (gaen) framework,” 2020.

- [19] G. of Canada, “Covid alert: Covid-19 exposure notification application privacy assessment.” <https://www.canada.ca/en/public-health/services/diseases/coronavirus-disease-covid-19/covid-alert/privacy-policy/assessment.html>, 2021.
- [20] T. Coalition, “Temporary contact number protocol.” <https://github.com/TCNCoalition/TCN>, 2020.
- [21] E. Daw, T. Coalition, and C. Vista, “Component-based comparison of privacy-first exposure notification protocols,” *IACR Cryptol. ePrint Arch.*, vol. 2020, p. 586, 2020.
- [22] MIT, “Pact: Private automated contact tracing..” <https://pact.mit.edu/>, 2020.
- [23] J. Chan, S. Gollakota, E. Horvitz, J. Jaeger, S. Kakade, T. Kohno, J. Langford, J. Larson, S. Singanamalla, J. Sunshine, *et al.*, “Pact: Privacy sensitive protocols and mechanisms for mobile contact tracing,” *arXiv preprint arXiv:2004.03544*, 2020.
- [24] C. Castelluccia, N. Bielova, A. Boutet, M. Cunche, C. Lauradoux, D. Le Métayer, and V. Roca, “Desire: A third way for a european exposure notification system leveraging the best of centralized and decentralized systems,” 2020.
- [25] M. Fowler, *UML distilled: a brief guide to the standard object modeling language*. Addison-Wesley Professional, 2004.
- [26] R.-S. Cheng, W.-J. Hong, J.-S. Wang, and K. W. Lin, “Seamless guidance system combining gps, ble beacon, and nfc technologies,” *Mobile Information Systems*, vol. 2016, 2016.
- [27] E. Mackensen, M. Lai, and T. M. Wendt, “Bluetooth low energy (ble) based wireless sensors,” in *SENSORS, 2012 IEEE*, pp. 1–4, IEEE, 2012.
- [28] T. Dierks and E. Rescorla, “The transport layer security (tls) protocol version 1.2,” 2008.
- [29] E. Rescorla and T. Dierks, “The transport layer security (tls) protocol version 1.3,” 2018.
- [30] S. Lee, “Creating and using databases for android applications,” *International Journal of Database Theory and Application*, vol. 5, no. 2, 2012.

- [31] L. Junyan, X. Shiguo, and L. Yijie, "Application research of embedded database sqlite," in *2009 International Forum on Information Technology and Applications*, vol. 2, pp. 539–543, IEEE, 2009.
- [32] C. Bi, "Research and application of sqlite embedded database technology," *wseas transactions on computers*, no. 1, 2009.
- [33] Z. Wei-Ping, L. Ming-Xin, and C. Huan, "Using mongodb to implement textbook management system instead of mysql," in *2011 IEEE 3rd International Conference on Communication Software and Networks*, pp. 303–305, IEEE, 2011.
- [34] C. Györödi, R. Györödi, G. Pecherle, and A. Olah, "A comparative study: MongoDB vs. mysql," in *2015 13th International Conference on Engineering of Modern Electric Systems (EMES)*, pp. 1–6, IEEE, 2015.
- [35] B. Dipina Damodaran, S. Salim, and S. M. Vargese, "Performance evaluation of mysql and mongodb databases," *Int. J. Cybern. Inform.(IJCI)*, vol. 5, 2016.
- [36] G. Legg, "The bluejacking, bluesnarfing, bluebugging blues: Bluetooth faces perception of vulnerability," *TechOnline* <http://www.wirelessnetdesignline.com/showArticle.jhtml>, 2005.
- [37] G. Sarada, N. Abitha, G. Manikandan, and N. Sairam, "A few new approaches for data masking," in *2015 International Conference on Circuits, Power and Computing Technologies [ICCPCT-2015]*, pp. 1–4, IEEE, 2015.
- [38] F. Liu, *Android native development kit cookbook*. Packt Publishing Ltd, 2013.
- [39] F. Wei, X. Lin, X. Ou, T. Chen, and X. Zhang, "Jn-saf: Precise and efficient ndk/jni-aware inter-language static analysis framework for security vetting of android applications with native code," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pp. 1137–1150, 2018.
- [40] P. Faruki, H. Fereidooni, V. Laxmi, M. Conti, and M. Gaur, "Android code protection via obfuscation techniques: past, present and future directions," *arXiv preprint arXiv:1611.10231*, 2016.
- [41] M. Chua and V. Balachandran, "Effectiveness of android obfuscation on evading anti-malware," in *Proceedings of the Eighth ACM Conference on Data and Application Security and Privacy*, pp. 143–145, 2018.

- [42] S. Academy, “Bluetooth low energy scanning and advertising.” https://dev.ti.com/tirex/explore/node?node=AD4sGbaamTCyn0DvZgBAsg__krol.2c__LATEST/.
- [43] ReelyActive, “Best practices for ble identifiers.” <https://reelyactive.github.io/diy/best-practices-ble-identifiers/>.
- [44] A. A. Kalbandhe and S. C. Patil, “Indoor positioning system using bluetooth low energy,” in *2016 International Conference on Computing, Analytics and Security Trends (CAST)*, pp. 451–455, IEEE, 2016.
- [45] R.-S. Cheng, J.-L. Li, W.-J. Hong, and K. W. Lin, “Development of an indoor positioning platform for irobot create 2,” in *2018 IEEE International Conference on Applied System Invention (ICASI)*, pp. 669–672, IEEE, 2018.
- [46] M. Al Qathrady and A. Helmy, “Improving ble distance estimation and classification using tx power and machine learning: A comparative analysis,” in *Proceedings of the 20th ACM International Conference on Modelling, Analysis and Simulation of Wireless and Mobile Systems*, pp. 79–83, 2017.
- [47] M. Agrawal and P. Mishra, “A comparative survey on symmetric key encryption techniques,” *International Journal on Computer Science and Engineering*, vol. 4, no. 5, p. 877, 2012.
- [48] M. S. Henriques and N. K. Vernekar, “Using symmetric and asymmetric cryptography to secure communication between devices in iot,” in *2017 International Conference on IoT and Application (ICIOT)*, pp. 1–4, IEEE, 2017.
- [49] D. Selent, “Advanced encryption standard,” *Rivier Academic Journal*, vol. 6, no. 2, pp. 1–14, 2010.
- [50] V. Rijmen and J. Daemen, “Advanced encryption standard,” *Proceedings of Federal Information Processing Standards Publications, National Institute of Standards and Technology*, pp. 19–22, 2001.
- [51] D. Bujari and E. Aribas, “Comparative analysis of block cipher modes of operation,” in *International Advanced Researches & Engineering Congress*, pp. 1–4, 2017.
- [52] M. Vaidehi and B. J. Rabi, “Design and analysis of aes-cbc mode for high security applications,” in *Second International Conference on Current Trends In Engineering and Technology-ICCTET 2014*, pp. 499–502, IEEE, 2014.

- [53] J. Lopez and R. Dahab, “An overview of elliptic curve cryptography,” 2000.
- [54] M. Amara and A. Siad, “Elliptic curve cryptography and its applications,” in *International workshop on systems, signal processing and their applications, WOSSPA*, pp. 247–250, IEEE, 2011.
- [55] N. Thiranant, Y. S. Lee, and H. Lee, “Performance comparison between rsa and elliptic curve cryptography-based qr code authentication,” in *2015 IEEE 29th International Conference on Advanced Information Networking and Applications Workshops*, pp. 278–282, IEEE, 2015.
- [56] A. J. Menezes, *Elliptic curve public key cryptosystems*, vol. 234. Springer Science & Business Media, 1993.
- [57] D. Hankerson, A. J. Menezes, and S. Vanstone, *Guide to elliptic curve cryptography*. Springer Science & Business Media, 2006.
- [58] O. Sury and R. Edmonds, “Edwards-curve digital security algorithm (eddsa) for dnssec,” tech. rep., RFC 8080 (Proposed Standard)., 7 pages. <https://doi.org/10.17487/RFC8080>, 2017.
- [59] D. Eastlake and T. Hansen, “Us secure hash algorithms (sha and hmac-sha),” 2006.
- [60] J. Brendel, C. Cremers, D. Jackson, and M. Zhao, “The provable security of ed25519: theory and practice,” *IEEE Security & Privacy*, 2021.
- [61] S. A. Lauer, K. H. Grantz, Q. Bi, F. K. Jones, Q. Zheng, H. R. Meredith, A. S. Azman, N. G. Reich, and J. Lessler, “The incubation period of coronavirus disease 2019 (covid-19) from publicly reported confirmed cases: estimation and application,” *Annals of internal medicine*, vol. 172, no. 9, pp. 577–582, 2020.
- [62] W. H. Organization, “Criteria for releasing covid-19 patients from isolation.” <https://www.who.int/news-room/commentaries/detail/criteria-for-releasing-covid-19-patients-from-isolation>, 2020.
- [63] T. Panse and P. Panse, “A survey on security threats and vulnerability attacks on bluetooth communication,” *International Journal of Computer Science and Information Technologies*, vol. 4, no. 5, pp. 741–746, 2013.
- [64] H. O’Sullivan, “Security vulnerabilities of bluetooth low energy technology (ble),” *Tufts University*, 2015.