UC Irvine

UC Irvine Previously Published Works

Title

Quest: Practical and Oblivious Mitigation Strategies for COVID-19 using WiFi Datasets

Permalink

https://escholarship.org/uc/item/0411t3xd

Authors

Gupta, Peeyush

Mehrotra, Sharad

Panwar, Nisha

et al.

Publication Date

2020-05-05

Copyright Information

This work is made available under the terms of a Creative Commons Attribution License, available at https://creativecommons.org/licenses/by/4.0/

Peer reviewed

eScholarship.org Powered by the California Digital Library
University of California

QUEST: Practical and Oblivious Mitigation Strategies for COVID-19 using WiFi Datasets*

Peeyush Gupta[1], Sharad Mehrotra[1], Nisha Panwar[2], Shantanu Sharma[1],
Nalini Venkatasubramanian[1], and Guoxi Wang[1]
[1]University of California, Irvine, USA. [2]Augusta University, USA.
Email: sharad@ics.uci.edu, shantanu.sharma@uci.edu

arXiv:2005.02510v1 [cs.DB] 5 May 2020

ABSTRACT

Contact tracing has emerged as one of the main mitigation strategies to prevent the spread of pandemics such as COVID-19. Recently, several efforts have been initiated to track individuals, their movements, and interactions using technologies, *e.g.*, Bluetooth beacons, cellular data records, and smartphone applications. Such solutions are often intrusive, potentially violating individual privacy rights and are often subject to regulations (*e.g.*, GDPR and CCPR) that mandate the need for opt-in policies to gather and use personal information. In this paper, we introduce QUEST, a system that empowers organizations to observe individuals and spaces to implement policies for social distancing and contact tracing using WiFi connectivity data in a passive and ***privacy-preserving manner***. The goal is to ensure the safety of employees and occupants at an organization, while protecting the privacy of all parties. QUEST incorporates computationally- and information-theoretically-secure protocols that prevent adversaries from gaining knowledge of an individual's location history (based on WiFi data); it includes support for accurately identifying users who were in the vicinity of a confirmed patient, and then informing them via opt-in mechanisms. QUEST supports a range of privacy-enabled applications to ensure adherence to social distancing, monitor the flow of people through spaces, identify potentially impacted regions, and raise exposure alerts. We describe the architecture, design choices, and implementation of the proposed security/privacy techniques in QUEST. We, also, validate the practicality of QUEST and evaluate it thoroughly via an actual campus-scale deployment at UC Irvine over a very large dataset of over 50M tuples.

Keywords

COVID-19, contact tracing, location tracing, social distancing, crowd-flow, WiFi.

1. INTRODUCTION

The ongoing COVID-19 pandemic with rapid and widespread global impact, has caused havoc over the past few months — at the time of writing of this paper, over 3 million individuals have been infected. The epidemic has caused over 200,000 global casualties, and the world economy to come to a screeching halt. Several (non-pharmacologic) steps are being taken by governments and organizations to restrict the spread of the virus, including social distancing measures, quarantining of those with confirmed cases, lock-down of non-essential businesses, and contact-tracing methods to identify and warn potentially exposed individuals. These tracking and tracing measures utilize a range of technological solutions. Countries, *e.g.*, Israel, Singapore, China, Taiwan, and Australia, utilize cellular data records or data from Bluetooth-enabled apps to perform contact tracing. Other countries, *e.g.*, India, have begun manual contact tracing by interviewing patients.

Recently, commercial and academic solutions (*e.g.*, Apple-Google collaboration [1], European PEPP-PT [2], Israel's The Shield [3], Singapore's TraceTogether [4], South Korea's 100m [5], and [40, 14, 28, 23, 57]) aim to provide secure contact tracing using Bluetooth-based proximity-detection. Using this approach, users can check if they have been exposed to a potential carrier of the virus by performing a private set intersection of their data with the secured public registry of infected people. While this approach is a step towards protecting the privacy of individuals, there are several limitations: First, the collection and sharing of such personal information can compromise the privacy of individuals — there are growing fears that this could also lead to misuse of data (now or in the future), *e.g.*, mass surveillance of communities and targeting of specific populations [24, 57]. Second, such methods require users to opt-in to broadcast, share, and collect the data using Bluetooth — past work has highlighted limited adoption of such technologies, especially, in parts of the world where privacy is considered to be a paramount concern [40, 19]. Third, contact tracing using Bluetooth or GPS-based proximity sensing has been shown to have false positives/negatives, leading to limited accuracy [23, 40]. Finally, past experiences have indicated that creating pathways for large organizations to capture personal data can lead to data theft, *e.g.*, Facebook's Cambridge Analytica situation and [51].

Contact tracing approaches are reactive in nature and aim to detect exposure after it occurs. We argue that proactive and preventive approaches are critical to contain and mitigate the spread. For instance, the ability to monitor public spaces (*e.g.*, classrooms, restaurants, malls), which are expected to have significant density and population flow, can be used by organizations (*e.g.*, campuses) to observe the extent to which employees (and employers) are adhering to social distancing directives. In fact, based on recent media articles [6, 7] and conversations with our university leadership,[1] the importance of such applications will increase further as organizations consider ways forward to reopen and resume operations. Today, organizations are working to help strike the right balance between onsite/online operations that afford both business continuity and public safety.

*We are thankful to Dhrubajyoti Ghosh for helping us in the system installation.
This material is based on research sponsored by DARPA under agreement number FA8750-16-2-0021. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of DARPA or the U.S. Government. This work is partially supported by NSF grants 1527536 and 1545071.

[1]Including epidemiologists in the public health school.

This paper describes our proposed solution, entitled QUEST that exploits existing WiFi infrastructure (prevalent in almost every modern organization) to support a sleuth of applications that empower organizations to evaluate and tune directives for safe operation, while protecting the privacy of the individuals in their premises. Particularly, QUEST leverages WiFi connectivity data (the data generated when a device connects to wireless access-points, see §4 for details) to support applications for social distancing adherence, crowd-flow, contact tracing, and exposure notifications within premises (both inside/outside buildings). The WiFi data collected is appropriately secured to prevent leakage of personally identifiable information (*e.g.*, MAC address of the mobile device) and outsourced to a public (cloud) server. On the outsourced data, QUEST allows application execution to occur in a privacy-preserving manner.

QUEST supports two different cryptographic alternatives for secure data processing; the choice of the approach is based on underlying security requirements of the organization. The first is a computationally secure encryption-based mechanism, entitled CQUEST that encrypts data using a variant of searchable encryption methods. The second approach called IQUEST, based on a string-matching technique [29] over secret-shares generated using Shamir's secret-sharing algorithm [54]. Both methods support the above-mentioned applications. IQUEST offers a higher level of security, when using untrusted servers, since it is information-theoretically secure, and moreover, does not reveal access-patterns (*i.e.*, the identity of tuples satisfying the query). We have deployed QUEST at UC Irvine [8], as well as, tested the system on large WiFi datasets. These datasets were collected as a part of the TIPPERS smartspace testbed at UCI [48] and will also be used for scalability studies.

QUEST offers several distinct advantages compared to other ongoing contact tracing efforts that have focused on using GPS, cellular infrastructure, and proximity sensors (*e.g.*, Bluetooth) [9, 4, 1, 23, 28, 40]. These include:

- *A Decentralized organizational solution.* QUEST is designed as a tool to be used independently and autonomously by organizations (*e.g.*, universities, individual shops/shopping complexes, and airports) to monitor adherence of their policies for social distancing, crowd-flow, and, to warn people about possible exposure on their premise. The organizational aspect of QUEST brings several advantages. First, the solution is amenable for organizational-level control to ensure that warning and alerts are not misused to spread false information, unlike some of the recent tools which are being targeted by malicious adversaries to spread propaganda and misinformation [10, 11]. Second, unlike solutions such as the one being designed by mobile OS platform vendors (viz. Apple and Google), in QUEST, both data collection and usages remain decentralized to the level of an organization and, thus, end-users do not need to trust any single organization/authority with their data.

- *A robust solution that works both inside buildings and outdoors.* Since QUEST is based on WiFi technology, it has a distinct advantage of being able to monitor both inside buildings (organizational premises) and in outside spaces, due to the ubiquitous nature of WiFi coverage in both indoors and outdoors of campuses. The use of WiFi data brings in several additional advantages: First, QUEST does not require any additional hardware expenses or deployment of any new technology that might be prohibitive in terms of cost and limited in terms of deployment. Second, since WiFi connectivity events are generated automatically by current WiFi protocols, QUEST is entirely passive, *i.e.*, it does not require users to deploy any new applications or make changes to their mobile devices. Third, the technology is platform independent, since data collection is implemented entirely on the infrastructure side.

- *Privacy-by-design.* QUEST supports the above-mentioned applications in a privacy-preserving manner by exploiting computationally secure and information-theoretically secure techniques. Thus, QUEST does not provide additional information about people, their locations, or their health status to any organization that they do not already have. Also, an adversary cannot learn past behavior or predict the future behavior of any user. Since the ciphertext representations of a person across organizations are different, even from jointly observing data of multiple organizations to know any specific person has been to the premises of one or more organizations.[2]

Outline. §3 provides the model and security properties. §4 provides an overview of QUEST and its applications. §5 provides CQUEST protocol. §6 provides IQUEST protocol. We evaluate QUEST in §7 and compare it with other state-of-the-art approaches, *e.g.*, Intel Software Guard Extensions (SGX) [26] based Opaque [62] and multi-party computation (MPC)-based Jana [17]; we discuss tradeoffs between security and performance.

2. RELATED WORK AND COMPARISON

In this section, we discuss the new approaches designed for COVID-19 contact tracing, several prior research approaches have explored proximity-based solutions to monitor the spread of infections, and compare against QUEST.

Comparison with COVID-19 proximity finding approaches. Several recent approaches for preventing the spread of coronavirus are based on Bluetooth data-based secure proximity detection. For example, Canetti et al. [23] present a person proximity detection method based on Bluetooth-enabled devices. However, this method requires to store parts of the data at the user device. [23], also, argued that GPS-based proximity detection inside a building can give false results. Stanford University [9] is also developing applications based on Bluetooth data. Singapore's TraceTogether application [4], also, works based on Bluetooth-based tracking. However, [24] showed that TraceTogether jeopardizes the user privacy. DP-3T (decentralized privacy-preserving proximity tracing) [57] proposed a proximity tracing system based on Bluetooth data. Google and Apple [1] are developing Bluetooth beacon-based contact tracing, while preserving the user privacy and location privacy. Similar work is also proposed in [28, 40] for Bluetooth data-based secure proximity detection, based on the private set intersection. Enigma MPC, Inc. [12] develops SafeTrace that requires users to send their encrypted Google Map timeline to a server equipped with Intel SGX [26] that executes contact tracing and finds whether the person got in contact with an impacted person or not. A survey of recent contact tracing application for COVID-19 may be found in [56]. However, all such methods require either to install an application [57, 4] at the device, to store some data [23, 28] at the device, to execute computation [23, 12, 28] at the device, to explicitly opt-in to enable Bluetooth-based beacon exchange [1, 28, 40], or jeopardize the user privacy [24].

In contrast, QUEST does not require any effort by users, since we rely on WiFi data that is generated when a device connects with a WiFi network. QUEST discovers the most accurate proximity of

[2] Organizations today, if they so desire, can capture and trace individuals based on their WiFi connectivity data. QUEST, obviously, cannot prevent such a use of WiFi data. The key-point is that while QUEST stores secured WiFi data at the cloud, the data-at-rest or query execution will not reveal any additional information to the organizations.

a person inside a building, unlike GPS-based approaches. Also, while using the servers, IQUEST provides complete security, due to using secret-sharing based technique. QUEST not only provides contact tracing, but also provides other applications (Table 1).

Comparison with other proximity finding approaches. Epic [15] and Enact [50] are based on WiFi signal strength, where a dynamic user scans the surrounding's wireless signals, access-points, and records in their phones. The infected user sends this information to a server that notifies other users and requests them to find their chances of contact. However, Epic [15] and Enact [50] consider trust in reporting by the infected users and requires storing some information at the smartphone, like Bluetooth-based solutions [57, 4, 23, 12, 28]. Another problem with such signal strength-based methods is in developing models to compare WiFi signals and have issues related to spatial, temporal, and infrastructural sensing [42]. NearMe [43], ProbeTag [47], [52], [49], and [44] proposed similar approaches for proximity detection. The seminal work [21] proposed distance-bounding protocol to estimate an upper-bound on the physical proximity of the device through the round-trip time measurements, by exchanging unique challenge-response pairs between a sender and a receiver. [34] provided a solution for proximity testing among the users while hiding their locations by encryption and considered user-to-user- and server-based proximity testing. Note that all such methods require *active participation* from the users.

In contrast, QUEST does not require active participation from the user, since it relies on WiFi connectivity data, which is, obviously, generated when a device gets connected with a WiFi network.

Background on cryptographic techniques. We may broadly classify existing cryptographic techniques into two categories: (*i*) *Computationally secure solutions* that includes encryption-based techniques such as symmetric-searchable encryption (SSE) [55, 27, 45, 46], deterministic encryption [18, 20], and order-preserving encryption (OPE) [13], (*ii*) *information-theoretically secure solutions* that include secret-sharing-based techniques [54, 29] and multi-party computation (MPC) techniques [17]. Computationally secure solutions, such as SSE — PB-tree [45] and IB-tree [46], are efficient in terms of computational time. However, they (*i*) reveal access-patterns (*i.e.*, the identity of the tuple satisfying the query), (*ii*) do not scale to a large-dataset due to dependence of a specific index structure, (*iii*) are not efficient for *frequent data insertion*, since it requires to rebuild the entire index at the trusted side, and (*iv*) cannot protect data from a computationally-efficient adversary or the government legislation/subpoena that may force to give them the data in cleartext. In contrast, information-theoretically secure solutions hide access-patterns, as well as, secure against a computationally-efficient adversary or the government legislation/-subpoena, (if the shares of the data are placed at the public servers under the different jurisdiction). Instead of using any cryptographic solution, one may also use *secure hardware-based solutions* that include Intel Software Guard eXtension (SGX) [26] based systems, *e.g.*, Opaque [62], Bunker and Fort [16], HardIDX [32], and EncDBDB [33]. However, such solutions suffer from similar issues as computationally secure solutions and suffer from additional side-channel attacks, such as cache-line [38] and branching attacks [59] that reveals access-patterns.

3. PRELIMINARY

This section explains the entities involved in deploying QUEST, the adversarial model, and the desired security properties.

3.1 Entities

We have the following two major entities in QUEST.

1. An *organization* o_i, who owns and deploys WiFi infrastructure (*e.g.*, WiFi access-points), and hosts QUEST that receives WiFi (connectivity) data (from the infrastructure) of the form: $\langle d_i, l_i, t_i \rangle$, where d_i is the i^{th} device-id and t_i is the time when d_i connects with a WiFi access-point l_i. Prior to outputting the data, QUEST appropriately implements a cryptographic technique to prevent misuse of the data from an adversary.

2. The *untrusted public (cloud) servers* that host the secured data, outsourced by QUEST, on which they execute applications. We assume that the servers support any database system, *e.g.*, MySQL.

Also, we assume two additional entities: a *querier* and a *publisher* ($\mathcal{P}$). A *querier* (which may be the organization or any (authenticated) person) is allowed to execute QUEST applications on the secured data (via QUEST). Further, only for *contact tracing application*, we assume a *trusted publisher* $\mathcal{P}$ (*i.e.*, hospitals), who publishes a secured list $\mathcal{L}$ of device-ids of confirmed infected person. We assume that QUEST executes a secure authentication protocol with $\mathcal{P}$ to confirm the queried device-id as the device-id of an infected person, before executing contact tracing application. Thus, it prevents the querier to execute a query for any device-id.

3.2 Adversarial Model

As we have two entities in QUEST. Below, we discuss their adversarial behavior and our assumptions.

Organizations. We assume that organizations have their own security keys (*e.g.*, public and private keys). A *system that hosts* QUEST*, its security keys, and programs must be secured and untampered by anyone* including the organizations, and this is an inevitable assumption, as well as, similar to the database-as-a-service (DaS) model [39], (where it is assumed that an adversary cannot filtrate the information stored at the database owners). Otherwise, any cryptographic mechanism cannot be implemented on WiFi data. Also, it is important to mention that o_i can capture their infrastructure data and can do any computation on the data as per their desire, without informing anyone. Controlling such organizations and preventing the misuse of the data is not in the scope of QUEST. Recall that QUEST's goal is to prevent o_i to track individuals or to run any applications other than those supported by QUEST on the data collected by QUEST.

Cloud servers. We assume that the public servers are honest-but-curious (HBC) and/or malicious adversaries. Such adversarial models are considered widely in data outsourcing techniques [39, 27, 22, 41, 29, 58, 61]. An HBC adversary may wish to learn the user information by observing query execution, while a malicious adversary deviates from the algorithm. Since dealing with a malicious adversary, we use an information-theoretically secure solution that uses Shamir's secret-sharing [54]. Thus, we follow the restriction of Shamir's secret-sharing that the adversary cannot collude with all (or possibly the majority of) the servers. Prior to sending the data to the server, we assume that QUEST authenticates the servers. Also, we assume that the secret-shared data transmission to the servers is done using an anonymous routing protocol [36], and it prevents an adversary to eavesdrop on a majority of communication channels between QUEST and the servers, and thus, preventing the adversary to know the servers that store the secret-shares.

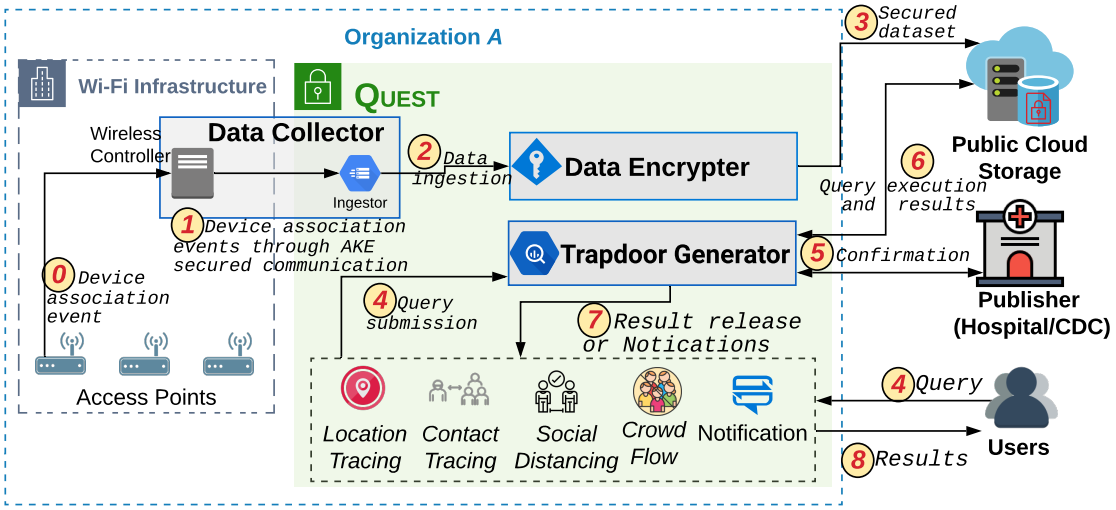


Figure 1: QUEST system.

3.3 Security Properties

In the above-mentioned adversarial model, an adversary wishes to reveal user privacy by learning from data-at-rest and query execution. Thus, a secure algorithm must prevent an adversary to learn the data by just observing (*ii*) cryptographically-secure data and (*ii*) query execution and deduce which tuples satisfy the query (*i.e.*, access-patterns). Also, we need to ensure that a querier cannot execute a query for device-ids not published by $\mathcal{P}$. Thus, we need to maintain the following properties:

Ciphertext indistinguishability. In the proposed scheme, the data contains user device-id. Thus, the *indistinguishability* of the user device-ids and locations is a vital requirement. Thus, the adversary, just by observing the secured dataset, cannot deduce that any two tuples belong to the same user/location or not. Note that satisfying indistinguishability property also prevents the adversary from learning any information from jointly observing two datasets belonging to two different organizations.

Secure query execution. It requires to maintain: (*i*) *Query privacy* that prevents the adversary from distinguishing between two query predicates (for the same or different device-ids and locations) by observing the query predicates or by observing the two queries' execution, *i.e.*, access-patterns. (*ii*) *Execution privacy* that enforces the adversary to behave identically and to provide an identical answer to the same query. (Since an adversary cannot distinguish between two query predicates, it should follow the same protocol for each query execution to prove its non-adversarial behavior.)

Satisfying these two properties achieve indistinguishability property during data-at-rest/query execution and do not reveal any information about the device-ids/locations. We can, formally, define it using the algorithm's real execution at the servers against the algorithm's ideal execution at a trusted party having the same data and the same query predicate. An algorithm reveals nothing if the real and ideal executions of the algorithm return the same answer.

Definition: Query privacy. *For any probabilistic polynomial time (PPT) adversary having a secured relation and any two input query predicates, say p_1 and p_2, the adversary cannot distinguish p_1 or p_2, either by observing the query predicates or by query output.*

Definition: Execution privacy. *For any given secured relation, any query predicate p issued by any real user U, there exists a PPT user U' in the ideal execution, such that the outputs to U and U' for the query predicate p on the secured data are identical.*

Note that satisfying the above two requirements (which are widely considered in many cryptographic approaches [22, 41, 29]) will hide access-patterns, thus, the adversary cannot distinguish two different queries and the satisfying output tuples. However, such a secure algorithm (as given in §6) incurs the overhead. Thus, we relax the access-pattern-hiding property (similar to existing searchable encryption or secure-hardware-based algorithms) and, also, present efficient access-pattern revealing algorithm, CQUEST (§5). In Appendix A, we provide security property of CQUEST.

4. QUEST ARCHITECTURE

QUEST contains the following three major components (see Figure 1):

Data collector. It collects WiFi connectivity (or association event) data of form $\langle d_i, l_j, t_k \rangle$, when a device d_i connects to a WiFi access-point (AP) l_j at time t_k. Particularly, at the infrastructure side, the collector contains a wireless controller that receives WiFi data from several APs (❶), via several methods, *e.g.*, SNMP (Simple Network Management Protocol) traps [53, 63], recent network management protocol NETCONF [30], or Syslog [35]) and forwards WiFi data to QUEST (❶) over the network using the secure networking protocol (*e.g.*, SSH [60]). QUEST receives and handles a large amount of streaming WiFi data at a very high rate (❶). However, the encrypter may not be able to handle a sudden burst of data, due to the overhead of security techniques and may drop some data. Thus, QUEST data collector includes an ingester (*e.g.*, Apache Kafka, Storm, and Flume) that acts as a buffer between the wireless controller and the encrypter (❷).

Data encrypter. The encrypter collects data for a fixed interval duration, called *epoch* (the reason of creating epochs will be clear soon in §5) and then implements a cryptographic technique (based on the desired security level, using CQUEST Algorithm 1 or IQUEST 3) and outputs the secured data that is outsourced to the servers (❸).

Trapdoor generator. A query/application is submitted to the trapdoor generator (❹) that generates the secure trapdoor (using Algorithm 2 or 4) for query execution on secured data. For (user) contact tracing, it confirms the submitted device-id as the real device-id of an infected person or not, from the publisher (❺). The trapdoors are sent to the servers that execute queries and send back encrypted results (❻). The results are decrypted before producing the final

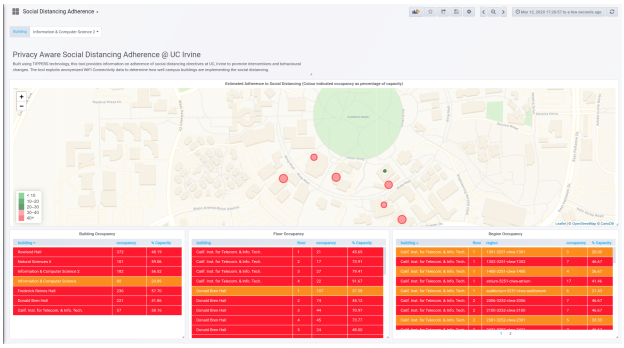


Figure 2: Social distancing application interface before lockdown.

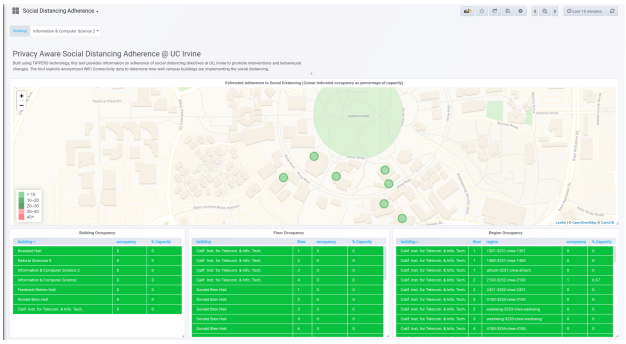


Figure 3: Social distancing application interface after lockdown.

answer (❼). Further, the organization may alert the users appropriately (via emails or phones), if devices have allowed the organization to inform about it, during their registration (❽).

QUEST Applications. On the secured data, QUEST supports the following diverse applications, which monitor/mitigate the spread of COVID-19 (Table 1 lists the application in SQL):

1. ***Location tracing***: traces all locations that were visited by an infected person in the past 14 days (the possible incubation time of coronavirus). Once the information of an infected person is provided to trapdoor generator, it, first, confirms from the publisher about the infected person, and then, generates trapdoors to find the locations visited by the person during the desired time interval.

2. ***User tracing*:** traces all users that were in the vicinity of an infected person in the past 14 days. Note that this is a natural extension of the previous application, by tracing all people who were at the infected locations at the (*bounded*) interval time (*e.g.*, +/-15minutes), when an infected person was there.

3. ***Social distancing*:** finds the locations and/or users in the campus that are not following social distancing rule. The idea is to use WiFi dataset to create a predefined occupancy knowledge at the granularity of buildings, floors, and regions within buildings. Now, the dynamic occupancy levels of such buildings (along with the knowledge of the capacity of rooms/floors/buildings) help in establishing to what degree different parts of the buildings are (or have been) occupied. Such a measure can help develop a quantitative metric, a social distance adherence index (*e.g.*, at UCI, 6 feet distancing requirement was translated roughly into 12.5% occupancy).

   Figure 2 shows the interface of social distancing application at UCI before the lockdown was announced. Figure 2 shows social distancing at different granularity, such as regions, floors, and buildings, where red-colored dots show that the buildings are not following social distancing rule. Figure 3 shows the interface of social distancing application at UCI after the lockdown was announced,

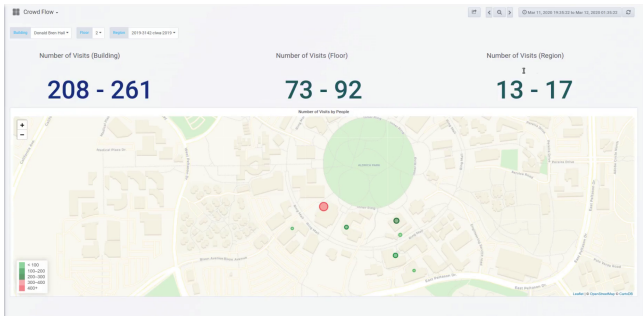


Figure 4: Crowd-flow application interface after lockdown.

where green-colored dots show that the buildings are following social distancing rule.

4. ***Crowd-flow*:** finds locations that were visited by many people in a day, and hence, need frequent cleaning. Note that this is a natural extension of social distancing application. This application provides information to individuals about the number of people visiting a given region over a given time period. Such information can be vital for people wishing to avoid crowded areas and also for the cleaning staff to determine places where disinfecting might be necessary. Figure 4 shows the interface of crowd-flow application after the lockdown.

5. ***Notification*:** enables all (desired) users to receive notifications, if they are tentative suspects. Note that often when connecting to a WiFi network, it may ask email address or phone number; QUEST exploits such information for notifications, (if allowed by the user).

5. CQUEST PROTOCOL

This section presents computationally-secure methods, CQUEST to encrypt WiFi data and to execute queries on encrypted WiFi data.

Key generation. QUEST encrypter generates a symmetric key: $(s_q \oplus k_{pko})||attribute_i$, *i.e*, the key is generated for each attribute of $\mathfrak{R}$ by XORing the secret-key of QUEST (s_q) and public key of organization (k_{pko}), and then concatenating with the attribute-id. We denote the key for an attribute i by k_i in Algorithm 1, and unless not clear, we drop the notation k_i from the description.

Data Encryption Method

Algorithm 1 provides pseudocode of proposed data encryption method that is executed at QUEST encrypter. It takes tuples of an epoch, produces an encrypted relation $\mathfrak{R}$ with five attributes. Table 2b shows an example of the produced outputs by Algorithm 1, which works as follows:

Selecting epoch. We use the bulk encryption method. Note that WiFi access-points capture time in milliseconds and ping the same device after a certain interval, during which the device can move. These two characteristics of WiFi data capture makes it hard to track a person based on time.[3] Thus, we discretize time into equal-length intervals, called *epoch*, and store a special identifier for each interval (that maps to the wall-clock time). An epoch x is denoted by Δ_x and is identified as a range of begin and end time. All sensor readings during that time period are said to belong to that epoch. There are no gaps between epochs, *i.e.*, the end time of the previous epoch is the same as the begin time of the next epoch. For simplicity, we identify each epoch by its beginning.

Encrypting device-ids: Attribute A_{id} (Lines 6-8). Since a device d_i can appear multiple times in an epoch, we need to prevent the

[3]For example, a query to find a device's location at 11:00am, cannot be executed in a secure domain, due to unawareness of millisecond-level time generated by APs.

Applications	SQL syntax
Location tracing	SELECT DISTINCT locationId FROM WiFiData INNER JOIN InfectedUsers ON WiFiData.macId = InfectedUsers.macId WHERE timestamp > t_1 AND timestamp < t_2
Contact tracing	SELECT DISTINCT WifiData.macId FROM WiFiData LEFT OUTER JOIN InfectedUsers ON WiFiData.macID = InfectedUsers.macId (SELECT locationId, timestamp FROM WiFiData INNER JOIN InfectedUsers ON WiFiData.macId = InfectedUsers.macId WHERE timestamp > t_1 AND timestamp < t_2) AS InfectedLocations WHERE WiFiData.locationId = InfectedLocations.locationId AND EXTRACT(WiFiData.timestamp, Δ) = EXTRACT(InfectedLocations.timestamp, Δ) AND InfectedUsers.macId IS NULL
Social distancing	SELECT DISTINCT COUNT(MacId) FROM WiFiData, (SELECT WiFiData.locationId, timestamp, COUNT(DISTINCT MacId)/capacity AS socialDistancing FROM WiFiData INNER JOIN Location ON WiFiData.locationId = Location.locationId WHERE timestamp > t_1 AND timestamp < t_2 GROUP BY WiFiData.locationId, timestamp/300 HAVING socialDistancing > maxAllowed}) AS Violations WHERE WiFiData.locationId = Violations.LocationId AND EXTRACT(WifData.timestamp, delta) = EXTRACT(Violations.timestamp, delta)
Crowd-flow	SELECT locationId, COUNT(DISTINCT macId) AS usersVisited FROM WiFiData WHERE timestamp > t_1 AND timestamp < t_2 GROUP BY locationId ORDER BY usersVisited DESC LIMIT K

Table 1: A sample of supported applications by QUEST in SQL.

frequency-count attack, while data-at-rest. Also, during query execution, we want to know whether d_i is present in the desired epoch at least once or not. To do so, we encrypt the first appearance of d_i in the epoch as $\mathcal{E}(d_i, 1, x)$ and maintain a hash table with value one for d_i in the epoch. Otherwise, we use any random number r and encrypts as $\mathcal{E}(d_i, r)$. We add the epoch-id with $\mathcal{E}(d_i, 1, x)$ to make d_i's first appearance indistinguishable in other epochs.

Uniqueness of the device: Attribute A_u (Lines 6-8). To execute applications such as social distancing and crowd-flow, we need to know unique devices at each location in Δ_x. Thus, when a device d_i appears for the first time at a location in y^{th} tuple, we add its uniqueness by $\mathcal{E}(1, y, \Delta_x)$. (It will avoid QUEST to decrypt all encrypted device-ids for knowing distinct devices in Δ_x.)

Encrypting locations: Attributes A_L and A_{CL} (Lines 9-12). First, we need to produce different ciphertexts for multiple appearances of a location to prevent frequency-count attack, while data-at-rest. To do so, we use a counter variable for each location and increment by 1, when the same location appears again in a tuple of Δ_x (and could, also, add x, like d_i's encryption). Second, we need to deal with d_i that moves to different locations in Δ_x. Note that based on $E(d_i, 1, x)$, we can search only the first appeared location of d_i in Δ_x. Thus, we collect all locations visited by d_i in Δ_x and add to the combined-locations attribute A_{CL} in a tuple having $E(d_i, 1, x)$. We pad the remaining values of A_{CL} by encrypted fake values.

Epoch-ids: Attribute A_Δ (Lines 13). Finally, we allocate an identical epoch identifier[4] to all tuples belonging to Δ_x and encrypts it. It allows search based on time, *e.g.*, based on epoch-id.[5]

Query Execution

Table 1 shows applications supported by QUEST in SQL, and Algorithm 2 explains trapdoor generation process at QUEST (denoted by **Q**) for those applications and query execution at the public server (denoted by **S**).[6]

Algorithm 1: Data Encryption Algorithm.

Inputs: Δ: duration. $\langle d_i, l_j, t_k \rangle$: A tuple. $\mathcal{H}$: Hash function. $\mathcal{E}$: encryption function. PRF: a pseudo-random generator.
Output: $\mathfrak{R}(A_{id}, A_u, A_L, A_{CL}, A_\Delta)$: An encrypted relation $\mathfrak{R}$ with five attributes.
Variable: c_{l_i}: A counter variable for location l_i.

1 **Function** $encrypt(\Delta_x)$ **begin**
2 $\quad \forall t_y = \langle d_i, l_j, t_k \rangle \in \Delta_x$:
$\quad\quad \ell_i \leftarrow create\_list\_device\_location(\text{distinct}(d_i))$
3 $\quad HTab_{id} \leftarrow init\_hash\_table\_device()$,
$\quad\quad HTab_L \leftarrow init\_hash\_table\_location()$
4 $\quad$ **for** $t_y = \langle d_i, l_j, t_k \rangle \in \Delta_x$ **do**
5 $\quad\quad r \leftarrow \text{PRF}()$
6 $\quad\quad$ **if** $HTab_{id}[\mathcal{H}(d_i)] \neq 1$ **then** $\mathfrak{R}.A_{id}[y] \leftarrow \mathcal{E}_{k1}(d_i, 1, x)$,
$\quad\quad\quad \mathfrak{R}.A_u[y] \leftarrow \mathcal{E}_{k2}(1, y, \Delta_x), \alpha_i[] \leftarrow l_j$
7 $\quad\quad$ **else if** $HTab_{id}[\mathcal{H}(d_i)] == 1 \wedge l_j \notin \alpha_i[]$ **then**
$\quad\quad\quad \mathfrak{R}.A_{id}[y] \leftarrow \mathcal{E}_{k1}(d_i, r), \mathfrak{R}.A_u[y] \leftarrow \mathcal{E}_{k2}(1, y, \Delta_x)$,
$\quad\quad\quad \alpha_i[] \leftarrow l_j$
8 $\quad\quad$ **else if** $HTab_{id}[\mathcal{H}(d_i)] == 1 \wedge l_j \in \alpha_i[]$ **then**
$\quad\quad\quad \mathfrak{R}.A_{id}[y] \leftarrow \mathcal{E}_{k1}(d_i, r), \mathfrak{R}.A_u[y] \leftarrow \mathcal{E}_{k2}(0, r)$
9 $\quad\quad$ **if** $\mathcal{H}(l_j) \notin HTab_L \wedge HTab_{id}[\mathcal{H}(d_i)] \neq 1$ **then**
$\quad\quad\quad HTab_{id}[\mathcal{H}(d_i)] \leftarrow 1, c_{l_j} \leftarrow 1, \mathfrak{R}.A_L[y] \leftarrow \mathcal{E}_{k3}(l_j, c_{l_j})$,
$\quad\quad\quad \mathfrak{R}.A_{CL}[y] \leftarrow \mathcal{E}_{k4}(r, \ell_i)$
10 $\quad\quad$ **else if** $\mathcal{H}(l_j) \notin HTab_L \wedge HTab_{id}[\mathcal{H}(d_i)] == 1$ **then**
$\quad\quad\quad c_{l_j} \leftarrow 1, \mathfrak{R}.A_L[y] \leftarrow \mathcal{E}_{k3}(l_j, c_{l_j})$,
$\quad\quad\quad \mathfrak{R}.A_{CL}[y] \leftarrow \mathcal{E}_{k4}(\text{Fake}, r)$
11 $\quad\quad$ **else if** $\mathcal{H}(l_j) \in HTab_L \wedge HTab_{id}[\mathcal{H}(d_i)] \neq 1$ **then**
$\quad\quad\quad HTab_{id}[\mathcal{H}(d_i)] \leftarrow 1, \mathfrak{R}.A_L[y] \leftarrow \mathcal{E}_{k3}(l_j, c_{l_j} + 1)$,
$\quad\quad\quad \mathfrak{R}.A_{CL}[y] \leftarrow \mathcal{E}_{k4}(r, \ell_i)$
12 $\quad\quad$ **else if** $\mathcal{H}(l_j) \in HTab_L \wedge HTab_{id}[\mathcal{H}(d_i)] == 1$ **then**
$\quad\quad\quad \mathfrak{R}.A_L[y] \leftarrow \mathcal{E}_{k3}(l_j, c_{l_j} + 1), \mathfrak{R}.A_{CL}[y] \leftarrow \mathcal{E}_{k4}(\text{Fake}, r)$
13 $\quad\quad \mathfrak{R}.A_\Delta[y] \leftarrow \mathcal{E}_{k5}(\Delta_x)$
14 $\quad\quad c_{max} \leftarrow max(c_{max}, c_{l_j}), \forall l_j$
15 $\quad$ Delete all hash tables for Δ_x

***Location tracing* (lines 1-5).** First, **Q** verifies the identity of the infected user device d_i from the publisher $\mathcal{P}$ (line 2). Then, **Q**

[4] One may assign the begin time of each epoch as an identifier, *e.g.* 4:00, 4:15, and 4:30, while the epoch duration is 15 minutes, or an increasing counter.

[5] Based on epoch-ids, we can execute query to find device's location at any desired time, *e.g.*, 11:00am.

[6] For simplicity, we denote a queried device-id by d_i. In practice, depending upon the publisher $\mathcal{P}$, such a device-id might be encrypted. in which case **Q** may need to securely obtain the real device-id from $\mathcal{P}$ during verification (line 2 Algorithm 2)

Algorithm 2: CQUEST query execution algorithm.

Inputs: $\mathcal{H}$: Hash function. $\mathcal{E}$: encryption function. capacity_{l_i}: The capacity of location l_i. distanceIndex: Maximum % of allowed people. Registry[]: The list of users allowed sending them notifications.
Output: Answers to queries.

1 **Function** $Location\_Trace(q(d_i, \text{Time}))$ **begin**
2 **if** Q $\leftrightarrow \boldsymbol{\mathcal{P}}$: Verify d_i Successful **then**
3 **Q**: Generate trapdoors $\mathcal{E}(d_i, 1, \Delta_t)$: t covers the requested Time
4 **S** $\rightarrow$ **Q**: $loc[] \leftarrow$ Location values from A_{CL} corresponding to $\mathcal{E}(d_i, 1, \Delta_t)$
5 **Q**: Decrypt $loc[]$ and produce answers
6 **Function** $User\_Trace(q(d_i, \text{Time}))$ **begin**
7 **Q**: $loc[] \leftarrow Location\_Trace(q(d_i, \text{Time}))$
8 **Q**: Generate trapdoors: $\forall l_i \in loc$: $\mathcal{E}(l_i, m)$, $m \in \{1, \text{max counter for any location}\}$
9 **S** $\rightarrow$ **Q**: $id[] \leftarrow$ Values from A_{id} corresponding to $\mathcal{E}(l_i, m)$
10 **Q**: Decrypt $id[]$ and $Notification(id[])$
11 **Function** $Social\_Distance(q(\text{Time}))$ **begin**
12 **Q**: Generate trapdoors: $\mathcal{E}(1, y, \Delta_t)$, y is max rows in any epoch, t covers the requested Time
13 **S** $\rightarrow$ **Q**: $loc[] \leftarrow$ Location values from A_L corresponding to $\mathcal{E}(1, y, \Delta_t)$
14 **Q**: $\forall l_i \in$ Decrypt($loc[]$), $count_{l_i} \leftarrow count_{l_i} + 1$
15 **Q**: **if** $count_{l_i} > \text{capacity}_{l_i} \times \text{distanceIndex}$ **then** Issue alarm
16 **Function** $Crowd\_Flow(q(\text{Time}))$ **begin**
17 **Q**: $Social\_Distance(q(\text{Time}))$
18 **Function** $Notification(id[])$ **begin**
19 **Q**: **if** $\forall i, id[i] \in$ Registry[] **them** Send notification to $id[i]$

creates and sends trapdoors for d_i as: $\mathcal{E}(d_i, 1, \Delta_t)$, where t is the epoch-identifiers that can cover the desired queried time (line 3). **S** executes a selection query for fetching the values of A_{CL} column corresponding to all encrypted query trapdoors (line 4). The answers to the selection query are given to **Q** that decrypts them to know the impacted locations (line 5).

Example 5.1. Suppose d_1 belongs to an infected person in Table 2b, and all four tuples belong to an identical epoch x. To execute location tracing, **Q** creates trapdoor for d_1, as: $\mathcal{E}(d_1, 1, x)$. **S** checks the trapdoor in A_{id} column and sends the corresponding value of A_{CL} column, *i.e.*, $\mathcal{E}_{k4}(r, l_1, l_2)$ to **Q**. On decrypting the received answer, **Q** knows the impacted locations as l_1 and l_2.

***User tracing* (lines 6-10).** First, **Q** executes $Location\_Trace()$ to know the impacted locations by the infected person (line 7). Then, **Q** creates trapdoors for all such locations (line 8), as: $\mathcal{E}(l_i, m)$, where l_i is the i^{th} impacted location and m is the maximum counter value for any location in any epoch, as obtained in Algorithm 1's line 14.[7] **S** executes a selection query for the trapdoor (or a join query between a table having all trapdoors and another table having the encrypted WiFi data) to know the corresponding values of A_{id} column (line 9). All such values are transmitted to **Q** that decrypts them to know the final answer (line 10). If any of the impacted users have subscribed to notification service, then they are informed.

Example 5.2. Suppose, we wish to know the impacted people that may in contact with the infected person whose device-id is d_1. From Example 5.1, we know that $\langle l_1, l_2 \rangle$ are the impacted locations. Suppose the maximum counter value for any location (c_{max}) is two. Thus, **Q** generates trapdoors as follows: $\mathcal{E}(l_1, 1)$, $\mathcal{E}(l_1, 2)$, $\mathcal{E}(l_2, 1)$, $\mathcal{E}(l_2, 2)$, and sends them to **S**. **S** executes a selection query over A_L column for such trapdoors and sends device-ids from A_{id} column, corresponding to the trapdoors. After the decryption, **Q** knows that d_2 is the device of a person that was in contact with the infected person whose device-id is d_1.

***Social distancing* (lines 11-15).** **Q** generates and sends trapdoors $\mathcal{E}(1, y, \Delta_t)$ to **S** to find the unique devices in the desired epochs (line 12).[8] **S** executes a selection query for the trapdoors (or a join query between a table having all trapdoors and another table having the encrypted WiFi data) to know the unique devices in the desired epochs and sends the qualified values from A_L column (line 13) to **Q**. **Q** decrypts the received locations and counts the appearance of each location (line 14). Then, **Q** issues an alarm, if the counter value for a location exceeds the predefined rule for social distancing, denoted by distanceIndex (line 15).

Aside. Note that we can also know the devices that do not follow the predefined rule for social distancing, by fetching the qualified values from A_{id} along with values of A_L.

Example 5.3. Assume that if more than one person appear at a location during a given epoch, then it shows that people at the location are not following the predefined social distancing rules, *i.e.*, in this example, $\text{distanceIndex} = 1$. **Q** generates the following four trapdoors: $\mathcal{E}(1, 1, x)$, $\mathcal{E}(1, 2, x)$, $\mathcal{E}(1, 3, x)$, and $\mathcal{E}(1, 4, x)$. Based on these trapdoors, **S** sends $\mathcal{E}(l_1, 1)$, $\mathcal{E}(l_2, 1)$, and $\mathcal{E}(l_2, 2)$. On receiving the encrypted location values, **Q** decrypts them, counts the number of each location, and finds that the location l_2 is not following the social distancing rule.

Information leakage discussion. Although the data-at-rest does not reveal any information, the query execution reveals access-patterns (like SSEs or SGX-based systems [38, 59, 62, 31, 33]). Thus, an adversary, by just observing the query execution, may learn additional information, *e.g.*, which of the tuples correspond to an infected person (by observing $Location\_Trace$), how many people may get infected by an infected person (by observing $User\_Trace$), which tuples correspond to unique devices by observing queries on A_u or A_{CL}, and which locations are frequently visited by users (by observing $Social\_Distance$). Also, since CQUEST is based on encryption, a computationally-efficient adversary can break the underlying encryption technique.

Pros. Though the approach is simple, CQUEST maintains hash tables during encryption of tuples belonging to an epoch. Nevertheless, the size of hash tables is small for an epoch, (see §7). CQUEST efficiently deals with dynamic data, due to independence from an explicit indexable data structure, (unlike indexable SSE techniques [45, 46] that require to rebuild the entire index due to data insertion at the trusted size). Algorithm 2 avoids reading, decrypting the entire data of an epoch to execute a query, (unlike SGX-based systems [62]); thus, saves computational overheads. Also, the key generation by XORing s_q and k_{pko} prevents the adversary to learn any information by observing at the encrypted data belonging to two different organizations, since one of the keys will be surely different at different organizations.

Cons. Algorithm 1 increases the dataset size by adding two additional columns. Algorithm 2 reveals access-patterns; hence, the adversary may deduce information based on access-patterns. Similar to DaS model [39], the trapdoor generator has to decrypt the retrieved tuples, possibly to filter them, and to execute a small computation (*e.g.*, group by operation line 14 of Algorithm 2). Also, as we mentioned earlier that QUEST has a limitation that encrypter or trapdoor modules should not be tampered, by anyone, likewise DaS model [39].

Optimizations. We provide four optimizations: two for trapdoor generation in $User\_Trace()$, and the other two for trapdoor generation for $Social\_Distance()$. §7 will show the impact of such optimizations.

[7] Generating and sending trapdoors for impacted locations equals to the maximum counter value may incur computation and communication overheads. Thus, we will suggest an optimization for preventing this.

[8] Sending trapdoors that are equal to the number of tuples in the desired epoch may incur communication overheads. Soon, we will provide optimizations for avoiding such trapdoor generation and transmission.

	Dev	Loc	Time
1	d_1	l_1	t_1
2	d_2	l_2	t_2
3	d_1	l_2	t_1
4	d_1	l_1	t_3

(a) WiFi dataset.

	A_{id}	A_u	A_L	A_{CL}	A_Δ
1	$\mathcal{E}_{k1}(d_1, 1, x)$	$\mathcal{E}_{k2}(1, 1, x)$	$\mathcal{E}_{k3}(l_1, 1)$	$\mathcal{E}_{k4}(r, l_1, l_2)$	$\mathcal{E}_{k5}(x)$
2	$\mathcal{E}_{k1}(d_2, 1, x)$	$\mathcal{E}_{k2}(1, 2, x)$	$\mathcal{E}_{k3}(l_2, 1)$	$\mathcal{E}_{k4}(r, l_1)$	$\mathcal{E}_{k5}(x)$
3	$\mathcal{E}_{k1}(d_1, r, x)$	$\mathcal{E}_{k2}(1, 3, x)$	$\mathcal{E}_{k3}(l_2, 2)$	$\mathcal{E}_{k4}(\text{Fake}, 3)$	$\mathcal{E}_{k5}(x)$
4	$\mathcal{E}_{k1}(d_1, r, x)$	$\mathcal{E}_{k2}(0, r)$	$\mathcal{E}_{k3}(l_1, 2)$	$\mathcal{E}_{k4}(\text{Fake}, 4)$	$\mathcal{E}_{k5}(x)$

(b) Encrypted WiFi relation for an epoch.

	A_{smid}	A_{sid}	A_{su}	A_{smL}	A_{sL}	A_Δ
1	$\text{SSS}(d_1)$	$\text{S}(d_1)$	$\text{S}(1)$	$\text{SSS}(l_1)$	$\text{S}(l_1)$	x
2	$\text{SSS}(d_2)$	$\text{S}(d_2)$	$\text{S}(1)$	$\text{SSS}(l_2)$	$\text{S}(l_2)$	x
3	$\text{SSS}(d_1)$	$\text{S}(d_1)$	$\text{S}(1)$	$\text{SSS}(l_2)$	$\text{S}(l_2)$	x
4	$\text{SSS}(d_1)$	$\text{S}(d_1)$	$\text{S}(0)$	$\text{SSS}(l_1)$	$\text{S}(l_1)$	x

(c) Secret-shared WiFi relation for an epoch.

Table 2: Original WiFi dataset, encrypted WiFi dataset using Algorithm 1, and secret-shared WiFi dataset using Algorithm 3.

Location counters. Note that Line 8 of Algorithm 2 requires us to generate the number of trapdoors equals to the maximum counter values (*i.e.*, maximum connection events at a location in any epoch (Line 14 of Algorithm 1)). It may incur overhead in generating trapdoors and sending them to the server. Thus, we can reduce the number of trapdoors by keeping two types of counters: (*i*) *counter per epoch* to contain the maximum connection events at a location in each epoch, and (*ii*) *counter per epoch and per location* to contain the maximum connection events at each location in each epoch.

Trapdoor generation for uniqueness finding. Line 12 Algorithm 2 requires QUEST's encrypter to generate and send the number of trapdoors equals to the maximum number of tuples in any epoch. We can avoid sending many trapdoors by encrypting uniqueness of the device, as follows: $\mathcal{E}_k(\mathcal{E}_{k'}(1, \Delta_x), y)$ (at Line 6-8 of Algorithm 1), where k is known to **S** and $k' = (s_q \oplus k_{pko})||attribute$ is unknown to **S**. Thus, for social distancing query execution, **Q** needs to send to **S** only $\gamma = \mathcal{E}_{k'}(1, \Delta_x)$, and then, **S** can generate all the desired trapdoors as $\mathcal{E}_k(\gamma, y)$, where y is the number of rows in the desired epoch.

In the above-mentioned optimization, QUEST's encrypter does not need to generate all trapdoors and sends them the server, and the server will generate the desired trapdoors. While it will reduce the communication cost, the computation cost at the server will remain identical to the method given in Algorithm 2 Lines 9. Thus, in order to reduce the computation cost at the server, we can also outsource the hash table created for locations ($HTab_L$, Line 3 Algorithm 1), after each epoch. Now, to execute the social distancing application, QUEST needs to ask the server to send the encrypted hash tables for all the desired epochs. Since the hash table contains the number of unique devices at each location, it will provide the correct answer to the social distancing application after decryption at QUEST.

6. IQUEST PROTOCOL

To overcome the information leakages due to CQUEST, we provide a completely secure solution, IQUEST that is based on string-matching operation [29] on secret-shares [54].

Background: String-matching over secret-shares. As a building block, first, we explain the string matching of Dolev et al. [29] using the following example.

Data Owner: outsourcing searchable-secret-share (SSS). Assume there are only two symbols: X and Y; thus, X and Y can be written as $\langle 1, 0\rangle$ and $\langle 0, 1\rangle$. Suppose, the owner wishes to outsource Y; thus, she creates *unary vector* $\langle 0, 1\rangle$. But, to hide exact numbers in $\langle 0, 1\rangle$, she creates secret-shares of each number using polynomials of an identical degree (see Table 3) and sends shares to servers.

Values	Polynomials	I$^{\text{st}}$ shares	II$^{\text{nd}}$ shares	III$^{\text{rd}}$ shares
0	$0 + 2x$	2	4	6
1	$1 + 8x$	9	17	25

Table 3: Secret-shares of $\langle 1, 0, 0, 1\rangle$, created by the owner.

User: SSS query generation. Suppose a user wishes to search for Y. She creates unary vectors of Y as $\langle 0, 1\rangle$, and then, creates secret-shares of each number of $\langle 0, 1\rangle$ using any polynomial of the same degree as used by the owner (see Table 4). Note that since a user can use any polynomial, it prevents an adversary to learn an equality condition by observing query predicates and databases.

Values	Polynomials	I$^{\text{st}}$ shares	II$^{\text{nd}}$ shares	III$^{\text{rd}}$ shares
0	$0 + 3x$	3	6	9
1	$1 + 7x$	8	15	22

Table 4: Secret-shares of $\langle 1, 0, 0, 1\rangle$, created by the user.

Servers: String-matching operation. Now, each server has a secret-shared database and a secret-shared query predicate. For executing the string-matching operation, the server performs bit-wise multiplication and then adds all outputs of multiplication (see Table 5).

Server 1	Server 2	Server 3
$2 \times 3 = 6$	$4 \times 6 = 24$	$6 \times 9 = 54$
$9 \times 8 = 72$	$17 \times 15 = 255$	$25 \times 22 = 550$
78	279	604

Table 5: Servers' computation.

User: result reconstruction. User receives results from all servers and performs Lagrange interpolation [25] to obtain final answers:
$\frac{(x-2)(x-3)}{(1-2)(1-3)} \times 72 + \frac{(x-1)(x-3)}{(2-1)(2-3)} \times 255 + \frac{(x-1)(x-2)}{(3-1)(3-2)} \times 550 = 1$
Now, if the final answer is 1, it shows that the secret-shared database at the server matches the user query.

Data Outsourcing Method

IQUEST uses Algorithm 3 for creating secret-shares of input WiFi relation R. *Note that Algorithm 3 when creating SSS or Shamir's secret-shares of a value (denoted by* $\text{SSS}(v)$ *and* $\text{S}(v)$*, respectively), randomly selects a polynomial of an identical degree.* Table 2c shows an example of the output of Algorithm 3. Algorithm 3 selects an epoch duration (like CQUEST (§5)) and produces an i^{th} secret-shared relation $S(\Re)_i$ with six attributes, denoted by A_{smid}, A_{sid}, A_{su}, A_{smL}, A_{sL}, and A_Δ. Note that *if the adversary cannot collude any two non-communicating servers, then we can use polynomials of degree one*, and in this case, there is no need to create more than $2l + 2$ shares, where l is the maximum length of a secret, to obtain an answer to a query in one communication round between the user and servers. Algorithm 3 works as follows:

Secret-shares of devices: Attributes A_{smid}, A_{sid} (Lines 4-5). We create two types of shares of each device id, one is SSS that is used

Algorithm 3: Secret-share creation algorithm.

Inputs: Δ: duration. $\langle d_i, l_j, t_k \rangle$: A tuple. $\mathcal{H}$: A hash function known to only IQUEST. z: a secret of proxy, unknown to organization.
Output: $S(\mathfrak{R})_i(A_{smid}, A_{sid}, A_{su}, A_{smL}, A_{sL}, A_{\Delta})$: An i^{th} encrypted relation R with six attributes.
Functions: $\mathrm{SSS}(v)$: A function for creating searchable secret-shares of v. $\mathrm{S}(v)$: A function for creating Shamir's secret-shares of v.

1 **Function** $create\_shares(\Delta_x)$ **begin**
2 $\quad HTab_{id} \leftarrow init\_hash\_table\_device()$
3 $\quad$ **for** $t_y = \langle d_i, l_j, t_k \rangle \in \Delta_x$ **do**
4 $\quad\quad val \leftarrow last\_v\_bits(\mathcal{H}(d_i))$
5 $\quad\quad \mathfrak{R}.A_{smid}[y] \leftarrow \mathrm{SSS}(val), \mathfrak{R}.A_{sid}[y] \leftarrow \mathrm{S}(val)$
6 $\quad\quad$ **if** $HTab_{id}[\mathcal{H}(d_i)] \neq 1$ **then** $\mathfrak{R}.A_{su}[y] \leftarrow \mathrm{S}(1), \alpha_i[] \leftarrow l_j$
7 $\quad\quad$ **else if** $HTab_{id}[\mathcal{H}(d_i)] == 1 \wedge l_j \notin \alpha_i[]$ **then** $\mathfrak{R}.A_{su}[y] \leftarrow \mathrm{S}(1), \alpha_i[] \leftarrow l_j$
8 $\quad\quad$ **else if** $HTab_{id}[\mathcal{H}(d_i)] == 1 \wedge l_j \in \alpha_i[]$ **then** $\mathfrak{R}.A_{su}[y] \leftarrow \mathrm{S}(0)$
9 $\quad\quad \mathfrak{R}.A_{smL}[y] \leftarrow \mathrm{SSS}(l_j), \mathfrak{R}.A_{sL}[y] \leftarrow \mathrm{S}(l_j)$
10 $\quad\quad \mathfrak{R}.A_{\Delta}[y] \leftarrow identifier(\Delta_x), HTab_{id}[\mathcal{H}(d_i)] \leftarrow 1$

for string matching operation and stored in A_{smid}, and another is just a Shamir's secret-share of the entire device-id stored in A_{sid}. The purpose of storing the same device-id in two different formats is to speed-up the computation. Particularly, values in A_{smid} help in string-matching operation, when we want to search for a device-id (*e.g.*, location tracing application), and values in A_{sid} helps in fetching the device-id (*e.g.*, user tracking application for retrieving device-ids based on infected locations).

Aside. Recall that creating secret-shares for string matching requires to convert the device-id into a *unary vector*; as shown in Table 3. However, it increases the length of device-ids significantly (*i.e.*, $12 \times 16 = 192$, often a device-id (MAC-ID) contains 12 hexadecimal digits (a combination of numbers $0, 1, \ldots, 9$ and alphabets A, B, ... F), and thus, every single MAC-ID digit will use a unary vector of size 16). Thus, we, first, execute a hash function (only known to IQUEST) on each device-id to map to a smaller length string, by taking the last $v < 12$ digits of the digest. Hashing may result in a collision, by mapping two different device-ids to the same digest, with a very low probability. For example, for a 256-bit hash function, the probability of collision in mapping all possible 32-bit integers is $2^{64}/2^{256+1} = 1/2^{193}$, which is negligible.

Uniqueness of devices: Attribute A_{su} (Lines 6-8). Similar to CQUEST's Algorithm 1, we assign value one when d_i appears for the first time at a location in an epoch; otherwise, zero. After that we create secret-shares of the value.

Secret-shares of location: Attributes A_{smid}, A_{sid} (Line 9). Likewise two types of secret-shares for device-ids, we create two types of shares of each location, one is SSS – stored in A_{smL}, and another is a Shamir's secret-share of the location stored in A_{sL}.

Outsourcing epoch-ids: Attributes A_{Δ} (Line 10). Finally, for all tuples of Δ_x, we outsource an epoch identifier in cleartext.

Differences between data outsourcing methods of CQUEST and IQUEST. Though CQUEST is an encryption-based method and IQUEST is a secret-sharing-based method, they, also, differ the way of keeping metadata (in Algorithms 1 and 3). First, IQUEST does not keep a hash table for locations to maintain their occurrences in tuples of an epoch. Second, IQUEST does not need to first find all locations visited by a device during an epoch and adds them in a special attribute; hence, IQUEST does not keep attribute A_{CL}. Note that these differences occur, due to exploiting the capabilities of SSS and selecting different polynomials for creating shares of any value, thereby, different occurrences of an identical value appear different in secret-shared form.

Algorithm 4: IQUEST query execution algorithm.

Inputs: $\mathcal{H}$: Hash function. $\mathrm{capacity}_{l_i}$: The capacity of location l_i. $\mathrm{distanceIndex}$: Maximum % of allowed people.
Output: Answers to queries.
Functions: $\mathrm{SSS}(v)$ and $\mathrm{S}(v)$: From Algorithm 3. $\mathrm{interpolate(shares)}$: An interpolation function that takes shares as inputs and produces the secret value.

1 **Function** $Location\_Trace(q(d_i, \mathrm{Time}))$ **begin**
2 $\quad$ **Q** $\leftrightarrow \boldsymbol{\mathcal{P}}$: Verify d_i
3 $\quad$ **Q** $\rightarrow$ **S**: $\gamma \leftarrow \mathrm{SSS}(d_i)$, Δ_t: t covers the requested Time
4 $\quad$ **S**: $sLoc[] \leftarrow (A_{smid}[j] \otimes \gamma) \times A_{sL}, j \in \{1, y\}, y =$ #tuples in Δ_t
5 $\quad$ **Q**: $location[] \leftarrow \mathrm{interpolate}(sLoc[])$
6 **Function** $User\_Trace(q(d_i, \mathrm{Time}))$ **begin**
7 $\quad$ **Q**: $location[] \leftarrow Location\_Trace(q(d_i, \mathrm{Time}))$
8 $\quad$ **Q** $\rightarrow$ **S**: $sssLoc[] \leftarrow \mathrm{SSS}(location[])$, Δ_t: t covers the requested Time
9 $\quad$ **S**: $\forall i \in \{1, |sssLoc[]|\}, \forall j \in \{1, y\}, y =$ #tuples in Δ_t, $sID[i, j] \leftarrow (ssloc[i] \otimes A_{smL}[j]) \times A_{sid}[j]$
10 $\quad$ **Q**: $id[] \leftarrow \mathrm{interpolate}(sID[*, *]), \forall i \in \{1, |sID[*, *]|\}$
11 $\quad$ **Q**: $Notification(id[])$ of **Algorithm 2**
12 **Function** $Social\_Distance(q(\mathrm{Time}))$ **begin**
13 $\quad$ **Q** $\rightarrow$ **S**: Δ_t: t covers the requested Time
14 $\quad$ **S** $\rightarrow$ **Q**: $sLoc[j] \leftarrow A_{su}[j] \times A_{sL}[j], \forall j \in \Delta_t$
15 $\quad$ **Q**: $location[] \leftarrow \mathrm{interpolate}(sLoc[])$
16 $\quad$ **Q**: $\forall l_i \in location[], count_{l_i} \leftarrow count_{l_i} + 1$
17 $\quad$ **Q**: **if** $count_{l_i} > \mathrm{capacity}_{l_i} \times \mathrm{distanceIndex}$ **then** Issue alarm
18 **Function** $Crowd\_Flow(q(\mathrm{Time}))$ **begin**
19 $\quad$ **Q**: $Social\_Distance(q(\mathrm{Time}))$

Query Execution

Algorithm 4 explains secret-shared query generation at IQUEST (denoted by **Q**), query execution at the server (denoted by **S**), and final processing before producing the answer at **Q**. Note that in Algorithm 4, $\otimes$ *denotes string-matching operation and* $\times$ *denotes normal arithmetic multiplication*. Below, we explain query execution for different applications over secret-shares.

***Location tracing* (lines 1-5).** First, **Q** verifies the device id d_i (as the real device of an infected person) from the publisher $\mathcal{P}$ (line 2). Then, **Q** creates SSS of d_i (denoted by γ) and sends it to each non-communicating server along with the desired epoch identifier (line 3). Each server executes string-matching operation over each value of A_{smid} against γ in the desired epoch, and it will result in either 0 or 1 (recall that string-matching operation results in only 0 or 1 of *secret-shared form*). Then, the i^{th} result of string-matching operation is multiplied by i^{th} value of A_{sL}, resulting in the secret-shared location, if impacted by the user; otherwise, the secret-shared location value will become 0 of secret-shared form (line 4). Finally, **Q** receives shares from all servers, interpolates them, and it results in all locations visited by the infected person (line 5).

Example 6.1. Suppose d_1 belongs to an infected person in Table 2c. To execute location tracing, **Q** generates SSS of d_1, say γ. **S** checks γ against the four shares (via string-matching operation) in A_{smid} and results in $\langle 1, 0, 1, 1 \rangle$ (of secret-shared form) that is position-wise multiplied by $\langle \mathrm{S}(l_1), \mathrm{S}(l_2), \mathrm{S}(l_2), \mathrm{S}(l_1) \rangle$. Thus, **S** sends $\langle l_1, 0, l_2, l_1 \rangle$ of secret-shared form to **Q** that interpolates them to obtain the final answer as $\langle l_1, l_2 \rangle$.

***User tracing* (lines 6-11).** First, **Q** executes $Location\_Trace()$ to know the impacted locations by the infected person (line 7). Then, **Q** creates SSS of all impacted locations (denoted by $sssLoc[]$) and sends them to the servers along with the desired epoch-identifer (which is the same as used when knowing infected locations in line 7). **S** executes string-matching operation over each value of A_{smL} against each value of $sssLoc[]$ in the desired epoch, and it will result in either 0 or 1 of secret-shared form). Then, the i^{th} result of string-matching operation is multiplied by i^{th} value of A_{sid}, resulting in the secret-shared device-ids, if impacted by the infected person; otherwise, the secret-shared location value will become 0

of secret-shared form (line 9). Finally, **Q** receives shares of all impacted people from all servers, interpolates them, and it results in all impacted people (line 10). All such impacted users are notified using $Notification(*)$ function of Algorithm 2.

Example 6.2. We continue from Example 6.1, where d_1 was the device of an infected person in Table 2c and impacted locations were $\langle l_1, l_2 \rangle$ that were known to **Q** after executing $Location\_Trace(*)$ (line 1). Now, to find impacted people, **Q** generates SSS of l_1 and l_2, say γ_1 and γ_2, respectively. **S** checks γ_1 and γ_2 against the four shares (via string-matching operation) in A_{smL}. It will result in two vectors: $\langle 1, 0, 1, 1 \rangle$ of secret-shared form corresponding to γ_1 and $\langle 0, 1, 0, 0 \rangle$ of secret-shared form corresponding to γ_2. Then, the vectors are position-wise multiplied by $\langle \mathrm{S}(d_1), \mathrm{S}(d_2), \mathrm{S}(d_1), \mathrm{S}(d_1) \rangle$. Thus, **S** sends $\langle d_1, 0, d_1, d_1 \rangle$ and $\langle 0, d_2, 0, 0 \rangle$ of secret-shared form to **Q**. **Q** interpolates the vectors and knows that the device d_2 belongs to an impacted person.

***Social distancing* (lines 12-17).** **Q** sends the desired epoch identifier to the servers (line 13). Based on the desired identifier, each server multiplies the i^{th} value of A_{su} with the i^{th} value of A_{sL}, and it results in all locations having the unique devices. The server sends all such locations to **Q** (line 14). First, **Q** interpolates the received locations (line 15) and then, counts the appearance of each location (line 16). Finally, **Q** issues an alarm, if the counter value for a location exceeds the predefined rule for social distancing, denoted by $\mathrm{distanceIndex}$ (line 17).

Aside. Note that we can also know the devices that do not follow the predefined rule for social distancing, by multiplying the i^{th} value of A_{su} with the i^{th} value of A_{sid} in the desired epoch, and in experiment section §7, we will find all such device-ids in our experiments.

Example 6.3. Suppose that $\mathrm{distanceIndex} = 1$, *i.e.*, if there are more than one person at a location during a given epoch, then it shows that people at the location are not following the predefined social distancing rules. Suppose all four tuples of Table 2c belongs to an epoch. **S** executes position-wise multiplication and send the output of the following to **Q**: $\langle S(1) \times S(l_1), S(1) \times S(l_2), S(1) \times S(l_2), S(0) \times S(l_1) \rangle$. **Q** interpolates the received answers, counts the number of each location (as $l_1 = 1$ and $l_2 = 2$), and finds that the location l_2 is not following the social distancing rule.

Information leakage discussion. Since Algorithm 3 uses different polynomials of the same degree for creating shares of a secret, an adversary cannot learn anything by observing the shares. Algorithm 4 creates secret-shares of a query predicate that appears different from the secret-shared data. Thus, the adversary by observing the query predicate cannot learn which tuples satisfy the query. Furthermore, since Algorithm 4 performs an identical operation on each share (*e.g.*, lines 4,9,14), it hides access-patterns; thus, the adversary cannot learn anything from the query execution, also. Hence, in IQUEST provides stronger security than CQUEST.

Pros. Due to hiding access-patterns, IQUEST provides stronger security and satisfies the security properties given in §3.3, *i.e.*, query and execution privacy. Also, it prevents a computationally efficient adversary to know anything from the ciphertext. Also, it is fault-tolerant, due to using multiple servers.

Cons. As known, a fully secure system incurs performance overheads. Due to executing an identical operation on each share, and hence, not using any index structure, IQUEST incurs the computational cost at the server. Also, since the servers send a secret-shared vector (having 0 or the desired value) of size equals to the numbers of tuples in the desired epoch, it incurs the communication cost. Nevertheless, §7) will show that such overheads are not very high.

Optimization. In social distancing application using IQUEST, it may turn out that we need to send a significant amount of data from the servers to QUEST. To avoid such communication, **Q** can send SSS of all the locations (denoted by $smLoc[]$) to servers. The servers can do the following:

$count_i \leftarrow \Sigma_{1 \le j \le y}(A_{smL}[j] \otimes sssLoc[i]) \times A_{su}[j], \forall i \in \{1, |smLoc[]|\}$

The servers execute string-matching operation for each location of $smLoc[]$ against each value of A_{smL} in each desired epoch and adds the output of string matching operation. Thus, for each desired epoch, the servers send $|sssLoc[]|$ numbers to **Q** that interpolates them to know the number of unique devices at each location in the desired epochs. Note that this method will outperform the method given in Algorithm 4 Line 13, if the number of tuples in each epoch are more than the number of locations in $smLoc[]$.

7. EXPERIMENTAL EVALUATION

QUEST is deployed at UCI, where it is being used to support social distancing and crowd flow applications [8]. This section evaluates the scalability of QUEST to evaluate its practicality for larger deployments and for all supported applications. We used AWS servers with 192GB RAM, 3.5GHz Intel Xeon CPU with 96 cores and installed MySQL to store secured dataset. A 16GB RAM machine at the local-side hosts QUEST that communicates with AWS servers.

Dataset. We used WiFi association data generated using SNMP traps at the campus-level WiFi infrastructure at UCI that consists of 2000 access-points with four controllers. Experiments used real-time data received at one of the four controllers (that collects real-time WiFi data from 490 access-points spread over 40+ buildings). Using this WiFi data, we created two types of datasets, refer to Table 6. For evaluating IQQUEST, we created nine shares, since at most $2(x + y) + 1$ shares are required, where $x = 3$ (the length of device-ids, line 4 Algorithm 3) and $y = 1$ (a single secret value in column A_{sL}, line 9 Algorithm 3).

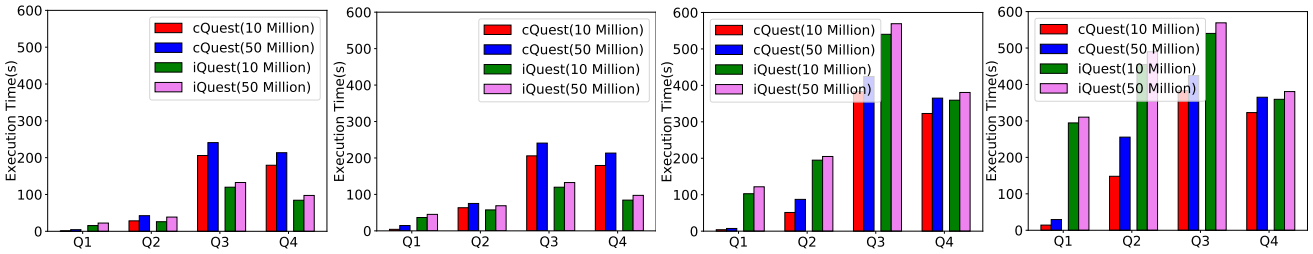
#rows	Cleartext size	Days covered	Encrypted size	Secret-Share size
10M	1.4GB	14	5GB	25GB
50M	7.0GB	65	13GB	65GB

Table 6: Characteristics of the datasets used in experiments.

Queries. We executed all queries (Q1: social distancing, Q2: Contact tracing, Q4: Crowd-flow), see Table 1. We modified 'Q3: social distancing query' by also fetching device-ids that do not follow distancing rules, in addition to fetching locations information.

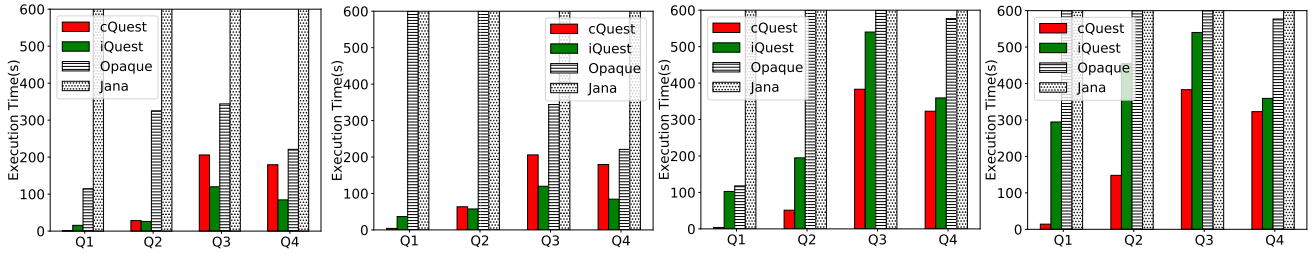
Exp 1: Throughput. In order to evaluate the overhead of CQUEST and IQUEST at the ingestion time, we measured the throughput (rows/minute) that QUEST can sustain. CQUEST Algorithm 1 can encrypt ≈494,226 tuples/min, and IQUEST Algorithm 3 can create secret-shares of ≈38,935 tuples/min. While the throughput of Algorithm 3's is significantly less than Algorithm 1, as it needs to create 9 (different) shares, it can sustain UCI level workload on the relatively weaker machine used for hosting QUEST.

Exp 2: Metadata size. Recall that Algorithm 4 (Algorithm 2) for IQUEST (CQUEST) maintains hash-tables for a certain duration. Table 7 shows the size of hash tables created for epochs of different sizes: 15min, 30min, and 60min. Note that the metadata size for CQUEST is larger than IQUEST, since CQUEST uses two hash tables (line 3 Algorithm 2) and one list of visited places by each device, while IQUEST uses only one hash table (line 2 Algorithm 4) and the list. Metadata overheads remain small for both techniques.



(a) 1-infected, 1-day. (b) 100-infected, 1-day. (c) 1-infected, 14-days. (d) 100-infected, 14-days.

Figure 5: Exp 3: Scalability test of 10M and 50rows with varying other parameters.



(a) 1-infected, 1-day. (b) 100-infected, 1-day. (c) 1-infected, 14-days. (d) 100-infected, 14-days.

Figure 6: Exp 4: Using other systems (secure hardware based Opaque and MPC-based Jana) vs CQUEST and IQUEST on 10M.

Epoch duration	CQUEST	IQUEST
15min	1.96MB	0.93MB
30min	3.40MB	1.37MB
60min	5.84MB	2.10MB

Table 7: Exp 2: Size of hash tables, for different epoch sizes.

Exp 3: Scalability. We measured the scalability of QUEST in three scenarios, by varying the number of infected people, days for tracing, and dataset size. Figure 5 shows results for 1-100 infected users for Q1, Q2 and execution of Q1-Q4 over 1-14 days duration on 10M, 50M rows. In Q1, a device has visited between 1 to 55 locations in 1 epoch. Note that Q1 using CQUEST took less time in all four cases, since it uses an index on A_{id} column (line 4 Algorithm 2); while IQUEST took more time, since it scans all data depending on the queried interval (line 4 Algorithm 4). As the number of infected people increases, the query time increases too. Cost analysis follows the same argument as Q2 that is an extension of Q1.

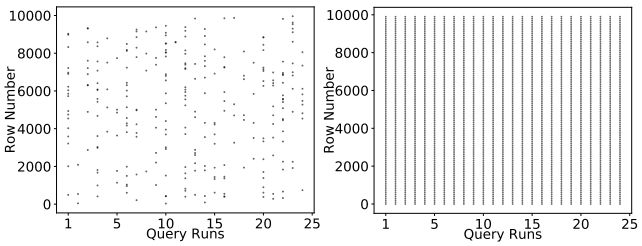
For Q3 and Q4 in Figures 5a and 5b, IQUEST took less time than CQUEST. The reason is: IQUEST performs multiplication on i^{th} values of A_{sL} and A_{su} (line 14 Algorithm 4), and the cost depends on the number of tuples in the desired epochs. However, CQUEST joins a table of size $y \times \Delta_t \times x$ with the encrypted WiFi data table on A_L column to obtain the number of locations having unique devices (line 13 Algorithm 2), where y is the maximum appearance of a location in any epoch (can be of the order of 10,000, causing a larger join table size), Δ_t is the number of desired epochs, and x is the number of locations. Also, note that for Q3 and Q4 in Figures 5c and 5d, IQUEST took more time than CQUEST, since the increase in the cost of multiplication operations (due to larger dataset of 14-days tracing period) in IQUEST overtook the increase in the cost of join in CQUEST. It shows CQUEST is more scalable than IQUEST.

Exp 4: Using other existing systems to support QUEST applications. We note an **alternative solution**, where one may output non-deterministically encrypted [37] or secret-shared WiFi data (via QUEST's encrypter), on which the queries can be executed using existing SSEs [45, 46], secure hardware-based systems, *e.g.*, Opaque [62], or MPC-systems. Note that this solution does not need to develop any encryption or query execution algorithm. However, it may allow an adversary to deduce the user locations by observing datasets belonging to different organizations and *may incur the high computational cost*, as will be clear below.

Thus, to see the impact of using existing systems to support QUEST applications, we used SGX-based Opaque [62] and MPC-based Jana [17] on 10M rows only (since these systems were available to us and work on any dataset). Now, we can compare CQUEST against computationally-secure Opaque and IQUEST against information-theoretically-secure Jana. We inserted data, using non-deterministic encryption in Opaque and using the underlying secret-sharing mechanism in Jana. Then, we used their query execution mechanisms for queries Q1-Q4. Figure 6 shows the impact of using different systems for supporting our four queries on 10M rows. We drop any query that took more than 1000s.

Observe that CQUEST works well compared to Opaque, since CQUEST uses index-based retrieval, while Opaque reads entire data in secure memory and decrypts it. CQUEST and Opaque provides *the same security*, *i.e.*, ciphertext indistinguishability, and reveals access-patterns. Note that CQUEST reveals access-patterns via index-scan, while Opaque reveals access-patterns due to side-channel (cache-line [38] and branch-shadow [59]) attacks. Also, IQUEST is efficient compared to Jana that takes more than 1000s in each query. The reason is: IQUEST does not require communication among servers due to using string-matching over secret-shares [29], while Jana requires communication among servers, since Jana is based on MPC techniques that require communication among server during a computation to compute the answer. But, IQUEST and Jana provide *identical security* by hiding access-patterns, due to executing identical operations on each tuple.

Exp 5: Impact of optimization. We implemented improved methods to minimize the value of max location counter (§5) and mea-



(a) Access-patterns of CQUEST. (b) Access-patterns of IQUEST.

Figure 7: Exp 6: Access-patterns created by QUEST.

sured the performance improvement over 10M rows, while fixing the number of infected people to 100 and interval duration to 1-day. When we used *counter per epoch* for Q2, it reduced the computation time from 63 (Figure 5b) to ≈35s and used 128KB more space to maintain the counter; while using *counter per epoch and per location*, Q2 took only ≈2sec with 55MB space to store the counters.

We also implemented the improved method for uniqueness finding by outsourcing encrypted hash tables for each epoch. It reduced the time of Q4 (that finds unique devices in each epoch) from 179.4s to 1s. Further, we incorporated this improved method in Q3 (that also finds the devices that does not follow social distancing rule) with *counter per epoch and per location* optimization, and it reduced the time of Q3 from 206s to 2s.

Exp 6: Access-patterns. Figure 7 shows a sequence of memory accesses by CQUEST and IQUEST. For this, we run Q2 multiple times, selecting different device-ids each time over a fixed set of epochs. It is clear that IQUEST accesses the same memory locations (accesses all the rows of the given set of epochs) and produces an output for each accessed row for different queries, while CQUEST accesses different memory locations (different rows for different device-ids) for answering different queries.

Exp 7: Impact of communication. Table 8 shows the amount of data transfer using CQUEST and the data transfer time using different transfer speeds. From Table 8, it is clear that CQUEST is communication efficient, while CQUEST reveals information from access-patterns. In particular, without using optimization methods (as described in §5), Q3 and Q4 incur significant communication overheads, *i.e.*, fetch ≈95MB data from the server. However, the optimization methods reduce such data size to ≈57KB.

Criteria	Q1	Q2	Q3	Q4
Without optimization	1.4KB	42.2KB	95MB	95MB
With optimization	N/A	N/A	56.6KB	14.4KB
Trans. speed 25MB/s	Neg.	Neg.	≈2.5m	Neg.
Trans. speed 100MB/s	Neg.	Neg.	≈1m	Neg.
Trans. speed 500MB/s	Neg.	Neg.	≈11s	Neg.

Table 8: Exp 7: CQUEST: amount of data transfer and required time (Neg. refers to negligible).

Table 9 shows the amount of data transfer using IQUEST and the data transfer time using different transfer speeds. From Table 9, it is clear that IQUEST incurs communication overhead, while IQUEST provides a high-level of security. In particular, Q1 requires us to fetch ≈32MB data from each server when tracing period was 14-days for an infected person. As Q2 requires two communication rounds (the first for knowing the impacted location and another for knowing the impacted device ids), it incurs signification communication cost by fetching ≈3.5GB data from each server. The reason is: we need to fetch data corresponding to 55 locations that a user can visit during an epoch. Q3, also, incurs the same communication overhead. Q4 requires to download ≈32MB data from each server for executing social distancing over 14-days. However, when we use the improved method (as described in §6) for Q4, we need to fetch only 2.1MB data.

Criteria	Q1	Q2	Q3	Q4
Without optimization	32MB	3.6GB	3.6GB	32MB
With optimization	N/A	N/A	N/A	2.1MB
Trans. speed 25MB/s	Neg.	≈2.5m	≈2.5m	Neg.
Trans. speed 100MB/s	Neg.	≈1m	≈1m	Neg.
Trans. speed 500MB/s	Neg.	≈11s	≈11s	Neg.

Table 9: Exp 7: IQUEST: amount of data transfer and required time (Neg. refers to negligible).

8. LESSONS LEARNT

In this paper, we designed, developed, and validated a system, called QUEST for privacy-preserving presence and contact tracing at the organizational level using WiFi connectivity data to enable community safety in a pandemic. QUEST incorporates a flexible set of methods that can be customized depending on the desired privacy needs of the smartspace and its associated data. We anticipate that capabilities provided by QUEST are vital for organizations to resume operations after a community-scale lockdown — the passive approach to information gathering in QUEST can enable continuous information awareness to encourage social distancing measures and identify settings and scenarios, where additional caution should be exercised. Ongoing discussions with campus administration at UC Irvine to utilize QUEST's capabilities for a staged and guided reopening of campus have highlighted the value of the privacy and security features embedded in QUEST. The living lab experience at UC Irvine will enable us to tune the underlying cryptographic protocols for other useful applications including dynamic occupancy counts and context-aware messaging to encourage safe operations.

9. REFERENCES

[1] Apple's and Google's COVID-19 contact tracing technology, available at: https://tinyurl.com/wfw9ojr.

[2] Pan-European Privacy-Preserving Proximity Tracing: available at: https://www.pepp-pt.org/.

[3] Israel's The Shield: available at: https://tinyurl.com/y75bqjj9.

[4] TraceTogether, available at: https://www.tracetogether.gov.sg/.

[5] South Korea's 100m: available at: https://tinyurl.com/yb5mj9o6.

[6] Georgia's daily coronavirus deaths will nearly double by August with relaxed social distancing, model suggests, available at: https://tinyurl.com/ydy53cfc.

[7] Polls: Americans dont want to end social distancing policies despite financial devastation, available at: https://tinyurl.com/ybvtfn9a.

[8] QUEST Applications: available at: https://tippersweb.ics.uci.edu/covid19/d/IwAc1O9Wk/covid-19-effort-at-uc-irvine?orgId=1.

[9] Stanford University's COVID-Watch, available at: https://covid-watch.org/.

[10] Fake news about the coronavirus is hazardous to your health. Don't fall for it: Doctor, available at: https://tinyurl.com/ybk4b5lo.

[11] During this coronavirus pandemic, 'fake news' is putting lives at risk: UNESCO, available at: https://tinyurl.com/y78jhdbl.

[12] SafeTrace, available at: https://github.com/enigmampc/safetrace.

[13] R. Agrawal et al. Order-preserving encryption for numeric data. In *SIGMOD*, pages 563–574, 2004.

[14] A. Aktay et al. Google COVID-19 community mobility reports: Anonymization process description (version 1.0). *CoRR*, abs/2004.04145, 2020.

[15] T. Altuwaiyan et al. EPIC: efficient privacy-preserving contact tracing for infection detection. In *ICC*, pages 1–6, 2018.

[16] G. Amjad et al. Forward and backward private searchable encryption with SGX. In *Proceedings of the 12th European Workshop on Systems Security, EuroSec@EuroSys 2019, Dresden, Germany, March 25, 2019*, pages 4:1–4:6, 2019.

[17] D. W. Archer et al. From keys to databases - real-world applications of secure multi-party computation. *Comput. J.*, 61(12):1749–1771, 2018.

[18] M. Bellare et al. Deterministic and efficiently searchable encryption. In *CRYPTO*, pages 535–552, 2007.

[19] C. Bi et al. Familylog: A mobile system for monitoring family mealtime activities. In *PerCom*, pages 21–30, 2017.

[20] A. Boldyreva et al. On notions of security for deterministic encryption, and efficient constructions without random oracles. In *CRYPTO*, pages 335–359, 2008.

[21] S. Brands et al. Distance-bounding protocols (extended abstract). In *EUROCRYPT*, pages 344–359, 1993.

[22] R. Canetti et al. Adaptively secure multi-party computation. In *STOC*, pages 639–648, 1996.

[23] R. Canetti, A. Trachtenberg, and M. Varia. Anonymous collocation discovery:taming the coronavirus while preserving privacy, 2020.

[24] H. Cho, D. Ippolito, and Y. W. Yu. Contact tracing mobile apps for covid-19: Privacy considerations and related trade-offs, 2020.

[25] R. M. Corless and N. Fillion. A graduate introduction to numerical methods. *AMC*, 10:12, 2013.

[26] V. Costan and S. Devadas. Intel SGX explained. *IACR Cryptology ePrint Archive*, 2016:86, 2016.

[27] R. Curtmola et al. Searchable symmetric encryption: Improved definitions and efficient constructions. *Journal of Computer Security*, 19(5):895–934, 2011.

[28] D. Demirag et al. Tracking and controlling the spread of a virus in a privacy-preserving way. *CoRR*, abs/2003.13073, 2020.

[29] S. Dolev et al. Accumulating automata and cascaded equations automata for communicationless information theoretically secure multi-party computation. *Theor. Comput. Sci.*, 795:81–99, 2019.

[30] R. Enns et al. Netconf configuration protocol. Technical report, RFC 4741, December, 2006.

[31] S. Eskandarian et al. Oblidb: Oblivious query processing for secure databases. *Proc. VLDB Endow.*, 13(2):169–183, 2019.

[32] B. Fuhry et al. Hardidx: Practical and secure index with SGX in a malicious environment. *Journal of Computer Security*, 26(5):677–706, 2018.

[33] B. Fuhry et al. Encdbdb: Searchable encrypted, fast, compressed, in-memory database using enclaves. *CoRR*, abs/2002.05097, 2020.

[34] R. Gelles et al. Multiparty proximity testing with dishonest majority from equality testing. In *ICALP*, pages 537–548, 2012.

[35] R. Gerhards et al. Rfc 5424: The syslog protocol. *Request for Comments, IETF*, 2009.

[36] D. M. Goldschlag et al. Onion routing. *Commun. ACM*, 42(2):39–41, 1999.

[37] S. Goldwasser and S. Micali. Probabilistic encryption. *J. Comput. Syst. Sci.*, 28(2):270–299, 1984.

[38] J. Götzfried et al. Cache attacks on Intel SGX. In *EUROSEC*, pages 2:1–2:6, 2017.

[39] H. Hacigümüs et al. Providing database as a service. In *ICDE*, pages 29–38, 2002.

[40] A. Hekmati et al. CONTAIN: privacy-oriented contact tracing protocols for epidemics. *CoRR*, abs/2004.05251, 2020.

[41] Y. Ishai et al. Private large-scale databases with distributed searchable symmetric encryption. In *RSA*, pages 90–107, 2016.

[42] M. B. Kjærgaard et al. Challenges for social sensing using wifi signals. In *Workshop on Mobile systems for computational social science*, pages 17–21, 2012.

[43] J. Krumm et al. The nearme wireless proximity server. In *UbiComp*, pages 283–300, 2004.

[44] J. C. Krumm et al. Proximity detection using wireless signal strengths, Mar. 24 2009. US Patent 7,509,131.

[45] R. Li et al. Fast range query processing with strong privacy protection for cloud computing. *PVLDB*, 7(14):1953–1964, 2014.

[46] R. Li et al. Adaptively secure conjunctive query processing over encrypted data for cloud computing. In *ICDE*, pages 697–708, 2017.

[47] M. Maier et al. Probetags: Privacy-preserving proximity detection using wi-fi management frames. In *WiMob*, pages 756–763, 2015.

[48] S. Mehrotra et al. TIPPERS: A privacy cognizant iot environment. In *PerCom W*, pages 1–6, 2016.

[49] J.-L. Meunier. Peer-to-peer determination of proximity using wireless network data. In *PerComW*, pages 70–74, 2004.

[50] A. Prasad and D. Kotz. ENACT: encounter-based architecture for contact tracing. In *WPA@MobiSys*, pages 37–42, 2017.

[51] L. Radaelli et al. Quantifying surveillance in the networked age: Node-based intrusions and group privacy. *CoRR*, abs/1803.09007, 2018.

[52] P. Sapiezynski et al. Inferring person-to-person proximity using wifi signals. *IMWUT*, 1(2):24:1–24:20, 2017.

[53] C. Schlener and S. Vasudev. Flexible snmp trap mechanism, Jan. 30 2001. US Patent 6,182,157.

[54] A. Shamir. How to share a secret. *Commun. ACM*, 22(11):612–613, 1979.

[55] D. X. Song et al. Practical techniques for searches on encrypted data. In *SP*, pages 44–55, 2000.

[56] Q. Tang. Privacy-preserving contact tracing: current solutions and open questions. *CoRR*, abs/2004.06818, 2020.

[57] C. Troncoso et al. Decentralized privacy-preserving proximity tracing overview of data protection and security. 2020. Available at: https://github.com/DP-3T/documents.

[58] C. Wang et al. Secure ranked keyword search over encrypted cloud data. In *ICDCS*, pages 253–262, 2010.

[59] W. Wang et al. Leaky cauldron on the dark land: Understanding memory side-channel hazards in SGX. In *CCS*, pages 2421–2434, 2017.

[60] T. Ylonen et al. The secure shell (ssh) protocol architecture, 2006.

[61] S. Yu et al. Attribute based data sharing with attribute revocation. In *ASIACCS*, pages 261–270, 2010.

[62] W. Zheng et al. Opaque: An oblivious and encrypted distributed analytics platform. In *NSDI*, pages 283–298, 2017.

[63] M. Zhou et al. EDUM: classroom education measurements via large-scale wifi networks. In *UbiComp*, pages 316–327, 2016.

APPENDIX

A. SECURITY PROPERTY FOR ACCESS-PATTERN-REVEALING SOLUTIONS

In order to define security property of CQUEST, we follow the standard security definitions of symmetric searchable encryption techniques [27] that define the security in terms of leakages: *setup leakage* $\mathcal{L}_s$ (that includes the leakages from the encrypted database size and leakages from metadata size) and *query leakage* $\mathcal{L}_q$ (that includes search-patterns (*i.e.*, revealing if and when a query is executed) and access-patterns (*i.e.*, revealing which tuples are retrieved to answer a query)). Based on these leakages, the security notion provides a guarantees that an encrypted database reveals no other information about the data beyond leakages $\mathcal{L}_s$ and $\mathcal{L}_q$.

Now, before defining security property, we need to formally define CQUEST's query execution method that contains the following three algorithms:

1. $(K, \mathfrak{R})) \leftarrow Setup(1^k, R)$**:** is a probabilistic algorithm that takes as input a security parameter 1^k and a relation R. It outputs a secret key K and an encrypted relation $\mathfrak{R}$. This algorithm (as given in Algorithm 1) is executed at QUEST's encrypter, before outsourcing a relation to the cloud.
2. $trapdoor\{1, \ldots, q\} \leftarrow Trapdoor\_Gen(K, query)$**:** is a deterministic algorithm that takes as input the secret key K and a query predicate $query$, and outputs a set of query trapdoors, denoted by $trapdoor\{1, \ldots, q\}$. This algorithm (as given in Algorithm 2) is executed at QUEST's trapdoor generator and $trapdoor\{1, \ldots, q\}$ are sent to the server to retrieve the desired tuples.
3. $results \leftarrow Query\_Exe(trapdoor\{1, \ldots, q\}, \mathfrak{R})$**:** is a deterministic algorithm and executed at the server. It takes the encrypted relation $\mathfrak{R}$ and the encrypted query trapdoors $trapdoor\{1, \ldots, q\}$ as the inputs. Based on the inputs, it produces the results.

In order to define the security notion, we adopt the real and ideal game model security definition [27]. Based on this game, what the security property is provided is known as indistinguishability under chosen-keyword attack (IND-CKA) model [27]. IND-CKA prevents an adversary from deducing the cleartext values of data from the encrypted relation or from the query execution, except what is already known.

Security Definition.

Let $\Psi = (Setup, Trapdoor\_Gen, Query\_Exe)$ be a tuple of algorithms. Let $\mathcal{A}$ be an adversary. Let $\mathcal{L}_s$ be the setup leakage, and let $\mathcal{L}_q$ be the query leakage.

- $Real_{\Psi,\mathcal{A}}(k)$**:** The adversary produces a relation R and sends it to a simulator. The simulator runs $Setup$ algorithm and produces an encrypted relations $\mathfrak{R}$ that is sent to $\mathcal{A}$. The adversary $\mathcal{A}$ executes a polynomial number of queries on the encrypted relations $\mathfrak{R}$ by asking trapdoors for each of the queries from the simulator. Then, the adversary $\mathcal{A}$ executes queries using $Query\_Exe()$ algorithm and produces a bit b.
- $Ideal_{\Psi,\mathcal{A}}(k)$**:** The adversary $\mathcal{A}$ produces a relation R'. Note that this relation may or may not be identical to the relation R, produced in $Real_{\Psi,\mathcal{A}}(k)$. However, $\mathcal{L}_s$ in the ideal world should be identical to the real world. The simulator has neither access to the real dataset R, nor access to the real queries. Instead, the simulator has, only, access to $\mathcal{L}_s$ and $\mathcal{L}_q$. The simulator simulates $Setup$ and $Trapdoor\_Gen$ algorithms. Given $\mathcal{L}_s$ and $\mathcal{L}_q$, the simulator produces an encrypted relation $\mathfrak{R}'$ and the trapdoors for all queries that were previously executed. The adversary executes the queries and produces a bit b.

We say Ψ is $(\mathcal{L}_s, \mathcal{L}_q)$-secure against non-adaptive adversary, iff for any probabilistic polynomial time (PPT) adversary $\mathcal{A}$, there exists a PPT simulator such that: $|Pr[Real_{\Psi,\mathcal{A}(k)} = 1] - [Pr[Ideal_{\Psi,\mathcal{A}(k)} = 1]| \leq negl(k)$, where $negl()$ is a negligible function.

The above real-ideal game provides the following intuition: an adversary selects two different relations, R_1 and R_2, having an identical number of attributes and an identical number of tuples. Relations R_1 and R_2 may or may not overlap. The simulator simulates the role of QUEST encrypter to produce an encrypted relation and provides it to the adversary. On the encrypted data, the adversary executes a polynomial number of queries. The adversarial task is to find the relation encrypted by the simulator, based on the query execution. The adversary cannot differentiate between the two encrypted relations, since if the adversary cannot find which encrypted relation is produced by the simulator with probability non-negligibly different from 1/2, then the query execution reveals nothing about the relation.