This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2022.3166802, IEEE Transactions on Services Computing

QUEST: Privacy-Preserving Monitoring of Network Data: A System for Organizational Response to Pandemics

Shantanu Sharma, Sharad Mehrotra, Nisha Panwar, Nalini Venkatasubramanian, Peeyush Gupta, Shanshan Han, and Guoxi Wang

Abstract—Most modern organizations today support network infrastructure to provide ubiquitous network coverage at their premises. Such a network infrastructure consisting of a set of access points deployed at different locations in buildings can be used to support coarse-level localization of individuals, who connect to the infrastructure using their mobile devices. This paper describes a system, entitled QUEST that supports a variety of applications (*e.g.*, identifying hotspot regions, finding people who are potentially exposed to a condition such as COVID-19, occupancy count of a region/floor/building) based on network data to empower organizations to maintain safety at their workplace/premises. QUEST builds the above functionalities while fully protecting the privacy of individuals. QUEST incorporates computationally- and information-theoretically-secure protocols that prevent adversaries from gaining knowledge of an individual's location history (based on WiFi data). 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 our organization over a very large dataset of over 50M rows.

Index Terms—WiFi connectivity data, computation and data privacy, exposure tracing, decentralized solution.

1 INTRODUCTION

The ongoing COVID-19 pandemic with rapid and widespread global impact, has caused havoc over the past year — at the time of writing this paper, over 231 million individuals have been infected. The pandemic has caused over 4.7 million 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, and Australia, were among the first to utilize cellular data records or data from Bluetooth-enabled apps to perform contact tracing. In addition, several commercial and academic solutions (*e.g.*, GAEN by Google-Apple collaboration [1], European PEPP-PT [2], and [23], [28], [39], [62]) 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. However, such techniques suffer from limited adoption and significant privacy issues, as we will discuss in detail in §2. For example, Google-Apple's GAEN protocol [1] is still not in use, by all the states in the US, due to privacy concerns [3], [38].

In this paper, instead of developing a tool/system using Bluetooth or cellular data, we take a radically different path by focusing on a specific type of sensor data — the WiFi connectivity data. In an organizational WiFi network, whenever a person's device connects to a WiFi access point, a network event is generated that essentially contains the MAC address of the connecting device, the MAC address of the access point, and the time of connection. Such a connectivity event can provide a ***coarse location*** of an individual, since it can locate a person to the region covered by the WiFi access point. Our focus on WiFi (connectivity) events/datasets is motivated by the following reasons:

1) *Ubiquitous Nature of WiFi.* WiFi connectivity is essentially ubiquitous and available in numerous organizations, such as office buildings and campuses, universities, and shopping complexes.
2) *No Additional Infrastructure Cost.* Developing applications based on WiFi data does not require deploying new sensors, and hence, can be used in the existing places with WiFi connectivity.
3) *Network-Centric and Passive Nature of WiFi Connectivity Data.* The application development using WiFi data does not require users to download and install an application on their mobile devices, as well as, does not require storing any information on mobile devices. WiFi data is collected and processed at the network side, and hence, users do not need to participate in any data storage or processing.
4) *A Turnkey Approach.* Unlike other WiFi-based techniques (*e.g.*, signal strength, time of flight, angle of arrival approaches) that require extensive calibration, training, or fingerprinting to work well across different settings, WiFi connectivity-based approach is robust and can be deployed with little or no training.
5) *Accuracy.* WiFi data – coupled with semantic knowledge of buildings and their users (as is often the case with organizations) – can be used to achieve high accuracy not just at the level of the region covered, but even finer-grained at the room level location, as shown recently in [48].

1.1 Our Contribution: QUEST

Given that WiFi (connectivity) data associates people with spaces dynamically and continuously, WiFi data can provide valuable insight into the organization, its functioning, and its culture. Motivated by the value of WiFi data, we design secure and privacy-preserving cloud-based services (related to COVID-19), where organizational WiFi data can be outsourced and analyzed. We describe our proposed solution, entitled QUEST that exploits existing WiFi infrastructure (prevalent in almost every modern organization) to support different applications that empower organizations to evaluate and tune directives for safe operations, while protecting the privacy of the individuals in their premises.

S. Sharma is with New Jersey Institute of Technology, USA. S. Mehrotra, N. Venkatasubramanian, P. Gupta, S. Han, and G. Wang are with University of California, Irvine, USA. N. Panwar is with Augusta University, Georgia, USA.

Particularly, QUEST leverages WiFi data (the data generated when a device connects to wireless access points, see §4 for details) to support the following applications:

A1: Exposure map application: inputs a period of time (*e.g.*, the past 14 days – the possible incubation time of coronavirus) and an (encrypted) identity of a device (say $\mathcal{E}(d_i)$) of an infected individual (who anonymously volunteers such information) and outputs a list of pairs of the time period and *exposed* regions within organizations where $\mathcal{E}(d_i)$ was present. (Note that the connectivity events contain information that can be used to identify the device (and hence the owner), and thus, can be used to establish *coarse-level localization* based on the access point's location, (as has been studied in [61], [65], [67], [68]).

A2: Exposure tracing application: inputs the output of exposure map application A1 (*i.e.*, a list of pairs of the time period and exposed locations) and outputs a list of the time period and (encrypted) device ids/addresses, where device-ids correspond to device-ids that were present at the exposed locations specified in the input. Note that by this application, QUEST does not support contact tracing, but it provides an approximation and makes contact tracing easier, by identifying people who potentially should be contact traced (by first identifying locations visited by the infected person, using application A1).

A3: Occupancy count application: inputs a period of time and outputs a list of the count of devices connected to access points. Moreover, depending on the information of access points covering a region of a building, it provides the number of devices (*i.e.*, the count of people) in each region. This application helps in finding potential overcrowded regions (both inside/outside buildings).

The key to QUEST is *privacy-preserving mechanisms* that implement the above functionalities at a public cloud, while preventing the cloud from gaining the ability to learn the identity of individuals, either those who may have been infected or those who could have been exposed, by observing the dataset collected by QUEST. QUEST is designed as an end-to-end system that (*i*) collects WiFi (connections) events/data, (*ii*) transforms the device identifier/address (typically MAC address of the device) using a secure hash function such that an adversary (which may the cloud) cannot confirm the transformed representation to a specific identity (*e.g.*, MAC address), (*iii*) encrypts the WiFi connectivity data with the transformed device representation in order to store ciphertext data at the cloud, and (*iv*) generates encrypted queries (called *trapdoors*) for query execution on the encrypted data and answering the above-mentioned applications.

Note that while we list only three applications above, QUEST is designed to be general enough to support other applications over WiFi data, such as tracking a person on a particular day, tracking when two persons met in the last five months, etc.

CQUEST and IQUEST. QUEST supports two different cryptographic alternatives for secure data processing to support different security levels.

The first is a *computationally secure encryption-based mechanism*, entitled CQUEST that encrypts data using a variant of searchable encryption methods. Note that computationally secure techniques can be broken by a computationally efficient adversary. The second approach is an *information-theoretical secure* technique, entitled IQUEST that is based on a string-matching technique [29] over secret-shares generated using Shamir's secret-sharing algorithm [58]. Note that information-theoretical secure techniques are secure regardless of the computational capabilities of an adversary. Note that information-theoretical secure techniques provide a higher level of security than computationally secure techniques. Both CQUEST and IQUEST support the above-mentioned applications. We have deployed CQUEST at University of California Irvine (UCI) [4], as well as, tested the system on large WiFi datasets. Note that in this paper, we will also present experimental results of IQUEST on large WiFi datasets. These datasets were collected as a part of the TIPPERS smartspace testbed at UCI [50] and are also used for scalability studies. (Please see interfaces of the three applications in Appendix B.)

1.2 Advantages of QUEST

QUEST comes with the following advantages over other approaches based on Bluetooth or GPS data:

Privacy-by-design. QUEST is an end-to-end privacy-preserving exposure tracing and occupancy count system based on WiFi technology. Compared to other WiFi-based proposals [61], [65], [67], [68], QUEST only deals with encrypted data and hence prevents leakages of user location to the cloud or to other users. For maintaining data privacy, QUEST exploits both types of cryptographic techniques computationally secure and information-theoretically secure techniques in such a way that an adversary cannot learn past behavior or predict the future behavior of any user. Furthermore, QUEST prevents the privacy of the users who visit multiple organizations, by producing ciphertext such that even from jointly observing data of multiple organizations an adversary does not learn any information of any users.

Passive solution. Since WiFi is ubiquitous in modern organizations, QUEST is passive in terms of not requiring users to download apps/update OS/organizations to deploy sensors, and collects no additional user data (other than what is already being captured to support WiFi access). Thus, QUEST can be deployed and used by simply notifying individuals about the existence of QUEST at the organization's premises, instead of seeking explicit user consent. Note that in contrast, Bluetooth-based solutions (*e.g.*, GAEN by Google/Apple) require OS upgrades and installation of apps, which limit their adoption, while GPS-based solutions only work in outdoors.

Organization-based. QUEST is a decentralized solution, *i.e.*, QUEST allows each organization (*e.g.*, universities and offices) to take steps autonomously and independently to maintain the safety of their premises by warning people about possible exposure on their premises and finding occupancy count at coarse level; (unlike Bluetooth-based solutions [1], [2], [5], [6], [23], [28], [39], [62] requiring *centralization* of the data by a single organization such as Google or Apple about all people who use their app). Though QUEST uses the public cloud to store the data of multiple organizations, each organization can use the same or different cloud vendors. In other words, the public cloud plays the role of data storage and encrypted search and does not perform any contact tracing. Moreover, QUEST encrypts data in a way that the public cloud cannot learn anything from jointly looking at ciphertext data that belongs to multiple organizations.

No calibration. Compared to WiFi connectivity data, approaches based on signal strength, time of flight, and angle of arrival may be more accurate. However, they require extensive calibration/-training/fingerprinting to work well across different settings. WiFi connectivity data, in contrast, provides a turnkey solution (no calibration/training). Equally importantly, it can be implemented in the encrypted domain; the alternate solutions require signal processing on an encrypted domain that adds complexity.

1.3 Discussion

In the present time (at the time of writing of the paper February 2022), organizations such as schools and universities are notifying students and faculties about possible exposure through co-occupancy due to an infected person in classrooms or confined areas. QUEST is designed to empower organizations to determine such co-occupancy in a privacy-preserving manner. Note that QUEST is not a tool to replace contact tracing based on distance, time, and immunity. Furthermore, our intention by occupancy count is to count the total number of connected devices to an access point, and depending on the background information about the location that the access point covers, we find an approximate occupancy count of the location. In situations such as duplicate devices, the presence of spurious devices (such as printers/machines) in buildings that may artificially affect the occupancy counts, missing sensor values (due to disconnections), and location ambiguity due to the coarse nature of the region covered by an access point, QUEST cannot find occupancy count close to the correct value. There are tools such as Locater [48] that exploit semantic information (lifted directly from data) about the affinity of people to each other and to locations to clean WiFi data. Such tools (Locater) reach accuracy as high as 92-93% establishing WiFi signal as a viable technology for indoor localization and for occupancy determination. Locater is not the only tool out there that is using WiFi connectivity for awareness about indoor occupancy. There are at least two recent startup ventures exploring such a technology [7], [8]. The focus of QUEST is not building a new tool for cleaning WiFi data to accurately perform localization using WiFi. To keep the paper focused on privacy techniques, QUEST considers the simplistic assumption that WiFi device connectivity event between WiFi access-points and the device locates an individual to the precision required to determine exposure, as well as, to accurately compute occupancy. QUEST can be used over the cleaned WiFi data using tools (such as [7], [8], [48]) to achieve more accurate occupancy counts.

1.4 Outline of the Paper

§2 provides detailed related work and compares QUEST against other approaches designed specifically for COVID-19. §3 provides the model and security requirments. §4 provides the architecture of QUEST. §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.*, Opaque [66] and multi-party computation (MPC)-based Jana [17]; we discuss tradeoffs between security and performance.

2 Related Work and Comparison

This section discusses new approaches designed for COVID-19 contact tracing, several prior proximity-based solutions to monitor the spread of infections, and compares them against QUEST.

Comparison with COVID-19 proximity finding approaches. Several approaches for preventing the spread of coronavirus are based on Bluetooth data-based secure proximity detection. Among them, the most famous is GAEN by Google/Apple [1]. Also, Switzerland's SwissCovid [9], South Korea's 100m [6], Singapore's TraceTogether application [5], DP-3T (decentralized privacy-preserving proximity tracing) [62], Stanford University app [10], and [23], [28], [39] are based on Bluetooth-based tracking. Enigma MPC, Inc. [11] developed SafeTrace that requires users to send their encrypted Google Map timeline to a server equipped with Intel Software Guard Extensions (SGX) [26] (secure hardware) that executes contact tracing and finds whether the person got in contact with an impacted person or not. A survey of recent contact tracing applications for COVID-19 may be found in [60].

However, all such methods suffer from several limitations: (*i*) *Limited adoption*: GAEN by Google/Apple require OS upgrades as well as installation of the application. Such a thing is also common in other Bluetooth-based applications. This limits adoption due to inertia. Several studies show GAEN adoption needs to be above 60% for effective contact tracing that is difficult to achieve with non-passive technologies. (*ii*) *Significant privacy concerns*: [24], [38] have shown that data collection process in Bluetooth-based applications jeopardizes the user privacy by either broadcasting, sharing, and/or collecting the data using Bluetooth. Moreover, past experiences have indicated that creating pathways for large organizations (such as Google and Apple) to capture personal data can lead to data theft, *e.g.*, Facebook's Cambridge Analytica situation. The privacy concerns further restrict the adoption of such technologies in parts of the world where privacy is considered to be a paramount concern [21], [39] (*iii*) *Data storage and computation at the device*: all Bluetooth-based applications require to store some data [5], [11], [23], [28], [62] and execute computation [11], [23], [28] at the device.

In contrast, QUEST does not require any effort by users, since QUEST relies on WiFi data that is generated when a device connects with a WiFi network. QUEST is implemented in a decentralized way with each organization that manages data about exposure at their premise instead of centralizing the data as in GAEN or others. Furthermore, QUEST only deals with encrypted data, preventing leakage of user location to an adversary. The key contribution of QUEST is a new protocol with appropriate security mechanisms to ensure both data security and high performance needed to sustain organizational-level installations.

Comparison with other proximity finding approaches. Epic [14] and Enact [54] are based on WiFi signal strength, where a 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 [14] and Enact [54] consider trust in reporting by the infected users and requires storing some information at the smartphone, like Bluetooth-based solutions [5], [11], [23], [28], [62]. 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 [41]. NearMe [43], ProbeTag [49], [56], [51], and [44] proposed similar approaches for proximity detection. [33] provided a solution for proximity testing among the users while hiding their locations by encryption and considered user-to-user-based 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 users, since QUEST relies on WiFi connectivity data, which is, obviously, generated when a device connects with a WiFi network.

Comparison with approaches based on WiFi data. We also note that recently, several startups have begun exploring the utility of WiFi data. For instance, Blyncsy Inc. [7] and Occuspace.IO [8] provide a cloud-based service to collect WiFi data in order to determine dynamic occupancy counts of different spaces based on which they support applications, such as dashboards of space utilization, the density of people, programmable triggers to alerts

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2022.3166802, IEEE Transactions on Services Computing

(*e.g.*, overcrowding). COVID-19 has spurred WiFi-based monitoring even further with several academic efforts including [61], [65], [67], [68] that have explored several applications related to monitoring/mitigating COVID-19 through people to people contact at their workspaces. To date, efforts so far have focused on algorithms for locating people based on WiFi connectivity [48], [65], [67] or on building applications using WiFi data [8], [65], [68], but have *not considered the security and privacy challenges* that arise when WiFi data is collected and applications are built on WiFi data.

In contrast, QUEST provides an end-to-end WiFi data security by implementing the two types of cryptographic techniques and prevents the misuse of the WiFi data by any user or the cloud.

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) [27], [46], [47], [59], deterministic encryption [20], and order-preserving encryption (OPE) [13], (*ii*) *information-theoretically secure solutions* that include secret-sharing-based techniques [29], [58] and multi-party computation (MPC) techniques [17]. Computationally secure solutions, such as SSE — PB-tree [46] and IB-tree [47], are efficient in terms of computational time. However, they (*i*) reveal access patterns (*i.e.*, the identity of the row 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 rebuilding 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 a 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 [66], HardIDX [31], and EncDBDB [32]. However, such solutions suffer from similar issues as computationally secure solutions and suffer from additional side-channel (cache-line [37] and branching) attacks [64] that reveal access patterns.

In contrast, QUEST comes with both computationally secure and information-theoretically secure mechanisms, thereby depending on the need the organization can select one or both mechanisms.

3 PRELIMINARY

This section explains the entities involved in deploying QUEST, their trust assumptions, and the desired security requirements.

3.1 Entities

We have the following four major entities in QUEST, see Figure 1.

- **Users ($\mathcal{U}$):** are individuals, who carry their mobile devices that connect to the WiFi network on the organization's premise and generate connection events. The device id or the device address (*i.e.*, MAC address) serves as the identity of the user, and the corresponding WiFi access point they connect to identify the location. Users are allowed to execute exposure tracing applications via QUEST over the (encrypted) WiFi connectivity data. Also, users are allowed to know the output of the exposure map application, *i.e.*, a list of pairs of the time periods and exposed regions.

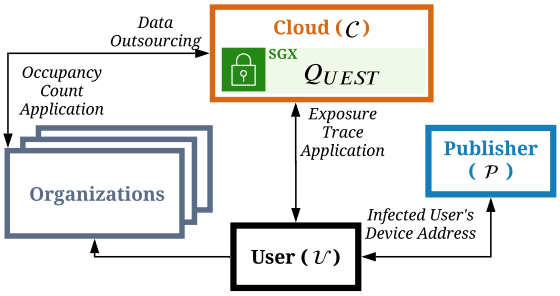


Fig. 1: Entities in QUEST.

A user is not trusted with the cleartext data of other users. In other words, a user may also wish to learn the behavior of other users by executing exposure tracing applications on behalf of other users. Each user $\mathcal{U}_i$ has their own public and private keys, denoted by PuK_{ui} and PrK_{ui}, respectively. The public/private key distribution is done by a trusted authority, and improving/modifying the generation/distribution of the public/private keys is outside the scope of this paper.

Note that we assume that before accessing the WiFi services at the organization, users register their device MAC addresses along with their identifiable information (such as a public key) at the organization. Such information is maintained in a file, called ***registry*** at the organization. Note that such type of registry information is presently maintained by several organizations, such as universities and office campuses.

- **An organization ($\mathcal{O}$):** owns WiFi infrastructure (*e.g.*, WiFi access points/routers). We assume that the organizations are not malicious in terms of data collection and usage. WiFi infrastructure at the organization generates connection events 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. Such connection events are sent to QUEST hosted at the public cloud in a secure manner (by encrypting with the public key of the secure hardware hosted at the cloud. Note that the encrypted data sent by an organization is never stored at the cloud, the secure hardware just reads the encrypted data from the network via accessing sockets, executes the proposed methods via QUEST to appropriately encrypt the data, and this data is stored at the cloud). Also, the organization sends the encrypted registry to the cloud. The organization is allowed to execute the occupancy count application via QUEST over the (encrypted) WiFi data. We assume that $\mathcal{O}$ is trusted, but does not want to participate in executing applications. An $\mathcal{O}_i$ has their own public and private keys, denoted by PuK_{oi} and PrK_{oi}, respectively.
- **The public cloud ($\mathcal{C}$):** hosts QUEST. We assume that a public cloud is not trusted with the cleartext data and code of QUEST. Particularly, we assume that public cloud servers are honest-but-curious (HBC). Such an adversarial model is considered widely in data outsourcing techniques [22], [27], [29]. An HBC adversary may wish to learn information about the data, but never tamper with the data/query/results.

Due to the untrusted environment at the cloud, QUEST is executed inside a secure tamper-proof hardware enclave ($\mathbb{E}$), such as Intel Software Guard eXtensions (SGX) [26].[1] We assume that the secure enclave $\mathbb{E}$ has its own public and private keys, denoted by $PuK_{\mathbb{E}}$ and $PrK_{\mathbb{E}}$, respectively. Also, we assume that SGX is not prone to side-channel (cache-line, branch shadow, page-fault [45], [63], [64]) attacks, as other work [31], [32], [66] on SGX also

1. The assumption of secure hardware at untrusted third-party machines is consistent with emerging system architectures; *e.g.*, Intel machines are equipped with SGX [12].

assumed the same.

As mentioned before, QUEST reads the encrypted data from the network via accessing sockets. Then, QUEST decrypts the received WiFi data and then, appropriately encrypts the data (using the proposed algorithm) on which encrypted queries can be executed based on trapdoors (*i.e.*, encrypted queries) generated by QUEST. QUEST's goal is to ensure that the ciphertext it produces cannot be used to reveal the user behavior. Also, QUEST maintains the encrypted registry based on which before executing exposure map and/or tracing applications, QUEST verifies the identity of the user. Also, QUEST authenticates the organization before executing the occupancy count application.

- **A publisher $\mathcal{P}$:** publishes the secure hash digests (using a hash function $\mathcal{H}$ with key κ) of device-ids of confirmed infected people, iff ***infected people wish to reveal their device-ids to*** $\mathcal{P}$. The publisher is assumed to be trusted, and the role of a publisher can be played by hospitals or CDC. Note that by this way infected individuals empower organizations to identify infected locations by them over time in the organization's premises. The key κ is only known to QUEST and $\mathcal{P}$. (The key κ can, also, be distributed by the trusted authority (which distributes the public/private keys to all entities) to QUEST and $\mathcal{P}$).

3.2 Security Requirements and QUEST

Now, we discuss the security requirements and briefly provide an overview of how does QUEST address them (note that the formal security requirements will be discussed in Section 3.2):

R1: Preventing the cloud to track users. WiFi (connectivity) data contains the MAC address of a device that could be used to track people at the granularity of the neighborhood of access points they connect to. Thus, a system must prevent the cloud to track individuals from using WiFi data without their consent. Moreover, since QUEST might be deployed in multiple organizations simultaneously, the system needs to prevent the cloud to track users across organizations. These requirements need the design of an efficient cryptographic mechanism that will produce *secure ciphertext* (called *ciphertext indistinguishability property*) to prevent the cloud from tracking an individual over the ciphertext data belonging to either one or multiple organizations.

Our approach. To securely encrypt the data, QUEST executes two layers of encryption, the first converts the device address into a secure hash-digest (by using a hash function $\mathcal{H}$ and key κ, which is unknown to all entities except QUEST and the publisher; see §3.1), and the second encrypts the data using the concatenated secure key of QUEST s_q and the public key of the organization for which QUEST is working (see details in §5.2). Thus, to know the device address to ciphertext mapping, an adversary needs to know κ, s_q, and $\mathcal{H}$, which are hidden from the adversary.

R2. Restricting users from accessing other users' data. A user may wish to learn about other users, information such as who is suffering from COVID to harass them. Thus, it is required that the system must not reveal the device address and information of the real COVID-19 patient in cleartext to any other users. Moreover, a user may wish to learn the past behavior of other users based on WiFi data. Hence, it is also required that the system provides information to the user that is based on their device address only.

Our approach. In QUEST, only the publisher publishes a secure list of the real infected people, iff they wish to reveal their device address to QUEST (securely), by using $\mathcal{H}$ and κ. Based on the secure list of the real infected people, QUEST only produces the potentially exposed locations via exposure map application and potentially exposed device addresses via exposure tracing application, after user authentication. Furthermore, to restrict more, QUEST can produce a binary answer when executing the exposure tracing application, *i.e.*, QUEST can maintain the list of potentially exposed devices securely and can return an answer 0 or 1 to the user if their device address intersects with the list (see §5.2 for details).

R3: Light-weight cryptographic solution. While QUEST could be built using existing secure data processing techniques/systems (*e.g.*, searchable-symmetric encryption (SSE) [46], [47], CryptDB [53], and Arx [52]) or secure hardware-based systems (*e.g.*, Opaque [66], EnclaveDB [55], and Cypherbase [16]), such solutions exhibit significant limitations, when serving as a building block for our purpose. First, the underlying encryption technology has to sustain data rates in the order of several thousands of connectivity events per minute.[2] Second, the system must be capable of supporting near real-time answers to queries over millions of records. Such workloads are simply impractical to support using existing cryptographic approaches. Many SSEs solutions [59], not supporting indexes, require linear scans to process queries. While indexable techniques have been explored [46], [47], such techniques do not support efficient frequent data insertion due to building entire indexes at the trusted side for each insert operation. While recent approaches exploiting secure hardware (*e.g.*, [32], [55], [66]) have explored scalable batch-based data insertion, they suffer from significant computational overheads (see the experimental study in §7.2).

Our approach. Given the above limitations of existing cryptographic approaches, we build QUEST using deterministic encryption (DET). There are several advantages of using DET — first, DET-based approaches can support dynamic insertion and index-based retrieval, especially for point queries (and also for range queries with a discretization of the domain). Also, industrial systems, such as Microsoft Always Encrypted [15], support DET. While DET-based solutions scale to the need of QUEST, naïvely using DET will reveal data distribution by observing data-at-rest. Such an approach will offer very little security, especially when connectivity patterns of a device could lead to the disclosure of the user identity. Instead, the encryption mechanism in QUEST exploits the limited nature of queries (that need to be supported to store data using DET) in such a way it does not reveal the distribution and provides strong security guarantees similar to SSEs. This is achieved by devising a special encryption and encrypted query (called *trapdoor*) generation techniques, which we refer to as CQUEST (§5).

3.3 Scope the Problem

There are other aspects in developing a secure system for sensor data outsourcing as listed below. QUEST is not designed to deal with these aspects, and we assume that one can use existing protocols to deal with these aspects. (*i*) *Authentication protocols and a secure network.* We assume the existence of a public/private key-based authentication protocol [42] among different entities of QUEST. Also, we assume the existence of secure communication protocols that can detect/mitigate network-level attacks, *e.g.*, man-in-the-middle attacks. (*ii*) *Trusted sensors.* We assume that sensors

2. A medium-size college campus may have several thousand access points that can produce data at the average rate of ≈100 connectivity event per second [67].

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2022.3166802, IEEE Transactions on Services Computing

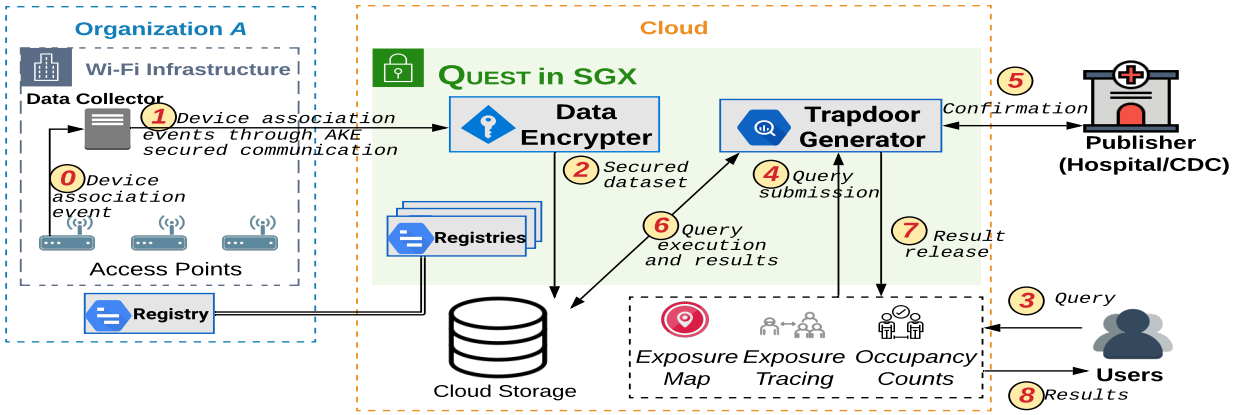


Fig. 2: QUEST system.

(*i.e.*, WiFi access points) are not malicious and cannot be replicated/spoofed. (*iii*) *Inference from aggregate queries.* An adversary may infer sensitive information from answers to occupancy count application; but, QUEST does not deal with this issue. We can minimize such inferences by limiting the preciseness of answers (*e.g.*, binary results).[3] (*iv*) *Inference from the number of sensor readings.* There could be an inference from the number of sensor readings being outsourced by an organization. *E.g.*, on weekdays and weekends, WiFi access points produce a different number of rows. QUEST does not deal with such an issue, but we can handle it by outsourcing fake rows. (*v*) *Inference due to background knowledge about a user/organization.* There could be an inference based on the background knowledge about a user and/or an organization. For example, if it is well-known that person X is the only person who visits an organization O every day in the morning (e.g., Sunday morning), then encrypting the data cannot hide this fact from an adversary, and based on the encrypted data, the adversary will know that all rows correspond to person X at location O. QUEST is not designed to deal with such an issue.

4 QUEST ARCHITECTURE

QUEST contains the following three major components (see Figure 2) and dataflow among them is shown in Figure 2.

Data collector: works at the organization and 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 l_j at time t_k. Particularly, at the organization side, the collector contains a wireless controller that receives WiFi data from several access points (❶), via several methods, *e.g.*, SNMP (Simple Network Management Protocol) traps [57], [67], recent network management protocol NETCONF [30], or Syslog [34]). Such data along with the registry is securely transmitted to QUEST, which is hosted at a public cloud (❶) over the network.

Data encrypter. QUEST at the cloud contains two modules: *data encrypter* and *trapdoor generator*. Both modules execute inside a secure enclave. Data encrypter implements a cryptographic technique (using CQUEST Algorithm 1 or IQUEST Algorithm 3) over the data that is collected for a fixed interval duration, called *epoch* (the reason of creating epochs will be clear soon in §5) and outputs the secured data that is written in the standard database management system (DBMS) at the servers (❷).

Trapdoor generator. An application is submitted to the trapdoor generator (❸, ❹) that generates the secure encrypted query, called *trapdoor* (using Algorithm 2 or 4) for query execution on secured data. For exposure map application, it receives a hash digest of device-id of infected persons from $\mathcal{P}$ (❺). For exposure tracing application, it authenticates users based on public/private keys and their information in the registry. Trapdoors are sent to the DBMS containing the data at the server that executes queries and sends back encrypted results (❻). The results are decrypted inside the enclave before producing the final answer (❼,❽).

3. Such inferences cannot be eliminated by existing cryptographic techniques. One could potentially use differential privacy [19], while receiving only probabilistic answers.

5 CQUEST PROTOCOL

This section presents computationally-secure methods, CQUEST to encrypt WiFi data and to execute queries on encrypted WiFi data. First, we provide a high-level overview of the protocol.

5.1 High-Level Overview of CQUEST

This section presents the high-level overview of CQUEST and details will be presented in the next section §5.2.

Data encryption. QUEST partitions time into subintervals, called *epochs*, and executes data encryption algorithm for each row of each epoch (that has a unique identifier). In QUEST, the cleartext WiFi dataset contains three columns: device-id (`Dev`), location (`Loc`), and time (`Time`), and each row is of the following form: $\langle d_i, l_i, t_i \rangle$. CQUEST encrypts rows such that we satisfy the following two needs:

1) *Secure ciphertext or ciphertext indistinguishability and untrackable encrypted data.* First note that satisfying this need will result in addressing the three security requirements R1, R2, R3 of §3.2. CQUEST produces non-identical ciphertext for more than one appearance of an identical device address or location, by (*i*) implementing the hash function $\mathcal{H}$ under the key κ over each device address, and (*ii*) adding an increasing counter or a random number with the output of the hash function for a single device address or a location along with the epoch identifier, before encrypting them. Note that this prevents an adversary from learning any information by just looking at the ciphertext.

2) *Efficient query processing.* While having only encrypted device address and location columns for each epoch can answer any query, CQUEST includes two additional encrypted columns for efficient query processing: (*i*) A_{CL} for finding all locations visited by a device in an epoch by placing a list of locations visited by the device in a row corresponding to the device's first appearance in the epoch, and (*ii*) A_u for finding unique devices at different locations in an epoch by creating searchable encrypted values for non-duplicated appearances of a device. A_{CL} (and A_u) column helps in the exposure map (and occupancy) application.

Example 5.1.1. Table 1a shows six rows of WiFi data, and Table 1b shows encrypted rows that are partitioned over two

	Dev	Loc	Time
1	d_1	l_1	t_1
2	d_2	l_2	t_2
3	d_1	l_2	t_3
4	d_2	l_1	t_4
5	d_2	l_1	t_5
6	d_3	l_2	t_6

(a) Cleartext original WiFi dataset.

	A_Δ	A_{id}	A_u	A_L	A_{CL}
1	$\mathcal{E}_{k1}(\Delta_x)$	$\mathcal{E}_{k2}(d_1,1,\Delta_x)$	$\mathcal{E}_{k3}(1,1,\Delta_x)$	$\mathcal{E}_{k4}(l_1,1,\Delta_x)$	$\mathcal{E}_{k5}(r,l_1,l_2)$
2	$\mathcal{E}_{k1}(\Delta_x)$	$\mathcal{E}_{k2}(d_2,1,\Delta_x)$	$\mathcal{E}_{k3}(1,2,\Delta_x)$	$\mathcal{E}_{k4}(l_2,1,\Delta_x)$	$\mathcal{E}_{k5}(r,l_1)$
3	$\mathcal{E}_{k1}(\Delta_x)$	$\mathcal{E}_{k2}(d_1,r)$	$\mathcal{E}_{k3}(1,3,\Delta_x)$	$\mathcal{E}_{k4}(l_2,2,\Delta_x)$	$\mathcal{E}_{k5}(\text{Fake},3)$
4	$\mathcal{E}_{k1}(\Delta_y)$	$\mathcal{E}_{k2}(d_2,1,\Delta_y)$	$\mathcal{E}_{k3}(1,1,\Delta_y)$	$\mathcal{E}_{k4}(l_1,1,\Delta_y)$	$\mathcal{E}_{k5}(r,l_1,l_2)$
5	$\mathcal{E}_{k1}(\Delta_y)$	$\mathcal{E}_{k2}(d_2,r)$	$\mathcal{E}_{k3}(0,r)$	$\mathcal{E}_{k4}(l_1,2,\Delta_y)$	$\mathcal{E}_{k5}(\text{Fake},5)$
6	$\mathcal{E}_{k1}(\Delta_y)$	$\mathcal{E}_{k2}(d_3,1,\Delta_y)$	$\mathcal{E}_{k3}(1,3,\Delta_y)$	$\mathcal{E}_{k4}(l_2,1,\Delta_y)$	$\mathcal{E}_{k5}(r,l_1)$

(b) Encrypted WiFi table for an epoch.

TABLE 1: Original WiFi dataset and encrypted WiFi dataset using CQUEST Algorithm 1.

epochs, each containing three rows. The first epoch is denoted by Δ_x containing the first three rows, and the second epoch is denoted by Δ_y containing the last three rows. The encrypted table (denoted by $\mathfrak{R}$) has five columns: A_Δ for encrypted epoch id, A_{id} for encrypted device address, A_L for encrypted location, A_{CL} for an encrypted list of locations visited by a device in an epoch, and A_u for the uniqueness of devices at different locations in an epoch.

Note that first, we implement the hash function $\mathcal{H}$ under the key κ on the device-id $d_i \in \langle d_i, l_i, t_i\rangle$ to produce $\langle \mathcal{H}_\kappa(d_i), l_i, t_i\rangle$ (to satisfy the above-mentioned first need). In rest of the paper, for simplicity, **we use** d_i **instead of** $\mathcal{H}_\kappa(d_i)$. Now, for producing secure ciphertext, CQUEST adds counter and random numbers in the values of A_{id} and A_L columns; see any value in the second and fourth columns of Table 1b (*e.g.*, $\mathcal{E}(l_2,1)$ and $\mathcal{E}(l_2,2)$).

For efficient query processing (the above-mentioned second need), CQUEST adds: (*i*) an encrypted column A_{CL}, for example, see row 1 in A_{CL} column storing a list of locations visited by d_1 in the epoch Δ_x, while other appearances of d_1, *i.e.*, row 3, in the same epoch Δ_x do not contain the same list; and (*ii*) an encrypted column A_u, for example, see row 1 and row 3 of A_u column that marks device d_1 at location l_1 and l_2 as unique (by $\mathcal{E}_{k3}(1,1,\Delta_x)$) in row 1 and (by $\mathcal{E}_{k3}(1,3,\Delta_x)$) in row 3. ■

Query execution. CQUEST supports the three applications (as mentioned in §1.1). Before executing a query, CQUEST verifies the querier's identity. Then, CQUEST generates the encrypted queries (called *trapdoors*) in the enclave according to the application and fetches only the desired encrypted data in the enclave to produce the final answer after filtering redundant encrypted rows.

Example 5.1.2. Suppose, the device d_1 belongs to an infected person. Such information is provided by the publisher to QUEST by sending $\mathcal{H}_\kappa(d_1)$. Now, assume that we want to find locations visited by $\mathcal{H}_\kappa(d_1)$ in epoch Δ_y, *i.e.*, executing exposure map application for $\mathcal{H}_\kappa(d_1)$ over the epoch Δ_y. Then, CQUEST will generate the following trapdoor $\mathcal{E}(\mathcal{H}_\kappa(d_1),1,\Delta_y)$ (or $\mathcal{E}(d_1,1,\Delta_y)$, as mentioned before that for simplicity we omit using $\mathcal{H}_\kappa(d_1)$) and will fetch a corresponding value from A_{CL} column. ■

5.2 Details of CQUEST

Now, we will present details of CQUEST's algorithms.

CQUEST Key Generation

QUEST encrypter generates a symmetric key, as follows: $(s_q \oplus k_{pko})||column_i$, *i.e*, the key is generated for each column of $\mathfrak{R}$ by XORing the secret-key of QUEST (s_q) and public key of organization (k_{pko}), and then concatenating with the column-id.

Algorithm 1: CQUEST Data Encryption Algorithm.

Inputs: Δ: duration. $\langle d_i, l_j, t_k\rangle$: A row. $\mathcal{H}$: 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 table $\mathfrak{R}$.
Variable: c_{l_i}: A counter variable for location l_i.

1 **Function** $encrypt(\Delta_x)$ **begin**
2 $\quad\forall t_y = \langle d_i, l_j, t_k\rangle \in \Delta_x$:
$\quad\quad l_i \leftarrow create\_list\_device\_location(\text{distinct}(d_i))$
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 $\quad\quad$ **//Allocating epoch identifier to rows and creating column** A_Δ
$\quad\quad \mathfrak{R}.A_\Delta[y] \leftarrow \mathcal{E}_{k1}(\Delta_x)$
7 $\quad\quad$ **//Encrypting device-ids and creating columns** A_{id} **and** A_u
$\quad\quad$ **if** $HTab_{id}[\mathcal{H}(d_i)] \neq 1$ **then** $\mathfrak{R}.A_{id}[y] \leftarrow \mathcal{E}_{k2}(d_i,1,x)$,
$\quad\quad\quad \mathfrak{R}.A_u[y] \leftarrow \mathcal{E}_{k3}(1,y,\Delta_x)$, $\alpha_i[] \leftarrow l_j$
8 $\quad\quad$ **else if** $HTab_{id}[\mathcal{H}(d_i)] == 1 \wedge l_j \notin \alpha_i[]$ **then**
$\quad\quad\quad \mathfrak{R}.A_{id}[y] \leftarrow \mathcal{E}_{k2}(d_i,r)$, $\mathfrak{R}.A_u[y] \leftarrow \mathcal{E}_{k3}(1,y,\Delta_x)$,
$\quad\quad\quad \alpha_i[] \leftarrow l_j$
9 $\quad\quad$ **else if** $HTab_{id}[\mathcal{H}(d_i)] == 1 \wedge l_j \in \alpha_i[]$ **then**
$\quad\quad\quad \mathfrak{R}.A_{id}[y] \leftarrow \mathcal{E}_{k2}(d_i,r)$, $\mathfrak{R}.A_u[y] \leftarrow \mathcal{E}_{k3}(0,r)$
10 $\quad\quad$ **//Encrypting locations and creating columns** A_L **and** A_{CL}
$\quad\quad$ **if** $\mathcal{H}(l_j) \notin HTab_L \wedge HTab_{id}[\mathcal{H}(d_i)] \neq 1$ **then**
$\quad\quad\quad HTab_{id}[\mathcal{H}(d_i)] \leftarrow 1, c_{l_j} \leftarrow 1$,
$\quad\quad\quad \mathfrak{R}.A_L[y] \leftarrow \mathcal{E}_{k4}(l_j, c_{l_j}, \Delta_x)$, $\mathfrak{R}.A_{CL}[y] \leftarrow \mathcal{E}_{k5}(r,\ell_i)$
11 $\quad\quad$ **else if** $\mathcal{H}(l_j) \notin HTab_L \wedge HTab_{id}[\mathcal{H}(d_i)] == 1$ **then**
$\quad\quad\quad c_{l_j} \leftarrow 1$, $\mathfrak{R}.A_L[y] \leftarrow \mathcal{E}_{k4}(l_j, c_{l_j}, \Delta_x)$,
$\quad\quad\quad \mathfrak{R}.A_{CL}[y] \leftarrow \mathcal{E}_{k5}(\text{Fake}, r)$
12 $\quad\quad$ **else if** $\mathcal{H}(l_j) \in HTab_L \wedge HTab_{id}[\mathcal{H}(d_i)] \neq 1$ **then**
$\quad\quad\quad HTab_{id}[\mathcal{H}(d_i)] \leftarrow 1$, $\mathfrak{R}.A_L[y] \leftarrow \mathcal{E}_{k4}(l_j, c_{l_j}+1, \Delta_x)$,
$\quad\quad\quad \mathfrak{R}.A_{CL}[y] \leftarrow \mathcal{E}_{k5}(r,\ell_i)$
13 $\quad\quad$ **else if** $\mathcal{H}(l_j) \in HTab_L \wedge HTab_{id}[\mathcal{H}(d_i)] == 1$ **then**
$\quad\quad\quad \mathfrak{R}.A_L[y] \leftarrow \mathcal{E}_{k4}(l_j, c_{l_j}+1, \Delta_x)$,
$\quad\quad\quad \mathfrak{R}.A_{CL}[y] \leftarrow \mathcal{E}_{k5}(\text{Fake}, r)$
14 $\quad\quad c_{max} \leftarrow max(c_{max}, c_{l_j}), \forall l_j$
15 $\quad$ Delete all hash tables for Δ_x

We denote the key for a column i by k_i in Algorithm 1, and unless not clear, we drop the notation k_i from the description.

CQUEST Data Encryption Method

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

Selecting epochs and creating column A_Δ (Lines 6). We use bulk encryption. 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 make it hard to track a person based on time.[4] 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. Thus, we allocate an identical epoch identifier to all rows belonging to epoch Δ_x and encrypt the identifier. Epoch identifiers allow searching based on time.[5] 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 and creating column A_{id} (Lines 7-9). On each device id d_i of an epoch, CQUEST first implements the hash function $\mathcal{H}$ under key κ, that results in $\mathcal{H}_\kappa(d_i)$. For simplicity, we

4. 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 access points.

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

Algorithm 2: CQUEST query execution algorithm.

Inputs: $\mathcal{H}$: Hash function. $\mathcal{E}$: encryption function. Registry[]: As defined in §3.1. $\mathcal{T}$: A fixed interval (*e.g.*, 14 days).
Output: Answers to queries.
1 **Function** $Exposure\_Map(q(d_i, \text{Time}))$ **begin**
2 **Q**: Generate trapdoors $\mathcal{E}(d_i, 1, \Delta_t)$: Δ_t is the epoch-id covers the requested Time
3 **S** $\rightarrow$ **Q**: loc, $epoch[*, *] \leftarrow$ Location and epoch-ids from A_{CL} and A_Δ corresponding to $\mathcal{E}(d_i, 1, \Delta_t)$ from A_{id}
4 **Q**: Decrypt loc, $epoch[*, *]$ and produce answers
5 **Function** $Exposure\_Trace(q(d_i, \text{Time}))$ **begin**
6 **Q**: loc, $epoch[*, *] \leftarrow Location\_Trace(q(d_i, \text{Time}))$
7 **Q**: Generate trapdoors: $\forall l_i \in loc$: $\mathcal{E}(l_i, m)$, $m \in \{1, \text{max counter for any location}\}$
8 **S** $\rightarrow$ **Q**: $id[] \leftarrow$ Values from A_{id} corresponding to $\mathcal{E}(l_i, m)$ and $\mathcal{T}$ covers $epoch[]$
9 **Q**: Decrypt $id[]$
10 **User** $\rightarrow$ **Q**: know about their exposure
11 **Q**: Authenticate the user against the registry information and if successful, perform intersection of $id[]$ and the user device address and return the appropriate answer
12 **Function** $Occupancy\_count(q(\text{Time}))$ **begin**
13 **Q**: Generate trapdoors: $\mathcal{E}(1, y, \Delta_t)$, $y =$ max rows in any epoch, $\Delta_t =$ epoch-id covers the requested Time
14 **S** $\rightarrow$ **Q**: $loc[] \leftarrow$ Location values from A_L corresponding to $\mathcal{E}(1, y, \Delta_t)$ from A_u
15 **Q**: $\forall l_i \in$ Decrypt($loc[]$), $count_{l_i} \leftarrow count_{l_i} + 1$

use d_i to denote $\mathcal{H}_\kappa(d_i)$ in the following. As mentioned before in §3.2, CQUEST uses deterministic encryption, which reveals frequency of a value via ciphertext, also known as *frequency-count attack*. To prevent such information leakage and efficiently finding the first appearance of d_i in any epoch (when executing exposure trace application), CQUEST encrypts the first appearance of d_i in an epoch Δ_x as: $\mathcal{E}(d_i, 1, \Delta_x)$, where 1 shows the first appearance of d_i in the epoch Δ_x. In addition, CQUEST maintains a hash table ($HTab_{id}$) with value one for d_i in the epoch Δ_x. All the other appearances of the device d_i in the epoch Δ_x are encrypts as $\mathcal{E}(d_i, r)$, where r is a random number used to produce secure ciphertext, *i.e.*, preventing the frequency count of a value via ciphertext, in the epoch Δ_x.

Uniqueness of the device and creating column A_u (Lines 7-9). To execute the occupancy count application, we need to know unique devices at each location in the epoch Δ_x. Thus, when a device d_i appears for the first time at a location in y^{th} row, we add its uniqueness by $\mathcal{E}(1, y, \Delta_x)$. (As will become clear soon, it will avoid CQUEST to decrypt all encrypted device-ids for knowing distinct devices in Δ_x.)

Encrypting locations and creating columns A_L and A_{CL} (Lines 10-13). First, we need to produce different ciphertexts for multiple appearances of a location to prevent frequency-count attack, while data is at-rest. To do so, CQUEST uses a counter variable for each location and increments by 1, when the same location appears again in a row of the same epoch (and could, also, add epoch identifier, like d_i's encryption). Second, we need to deal with d_i that moves to different locations in an epoch Δ_x. Note that based on $\mathcal{E}(d_i, 1, \Delta_x)$, we can search only the first appeared location of d_i in Δ_x. Thus, CQUEST collects all locations visited by d_i in Δ_x and adds to the combined-locations column A_{CL} in a row having $\mathcal{E}(d_i, 1, \Delta_x)$. CQUEST pads the remaining values of A_{CL} by encrypted fake values.

CQUEST Query Execution

Algorithm 2 explains encrypted query (called *trapdoor*) generation process and their execution at CQUEST. We denote the process in the enclave at CQUEST by **Q** and the process outside of the enclave by **S**. Below, we explain the execution of our three applications. Table 9 in Appendix A shows SQL for the three applications.

Exposure map application (lines 1-4). This application takes as input a period of time and a secure device address (belonging to a real infected person) provided by the publisher. Before obtaining the a secure device address (denoted by $\mathcal{H}(d_i)$), CQUEST authenticate the publisher. Based on $\mathcal{H}(d_i)$, CQUEST produces a list of pairs of time period and exposed regions within organizations where $\mathcal{H}(d_i)$ was present. To do so, **Q** creates and sends trapdoors for d_i as: $\mathcal{E}(d_i, 1, \Delta_t)$,[6] where t is the epoch-identifiers that can cover the desired queried time (line 2). **S** executes a selection query for fetching the values of A_{CL} column corresponding to all encrypted query trapdoors (line 3). The answers to the selection query are given to **Q** that decrypts them to know the exposed or impacted locations in a given epoch (line 4).

Example 5.2.1. Suppose d_1 belongs to an infected person in Table 1b, and we wish to know the location visited by d_1 in epoch Δ_x. To execute exposure map application, **Q** creates trapdoor for d_1, as: $\mathcal{E}(d_1, 1, \Delta_x)$. **S** checks the trapdoor in A_{id} column and sends the corresponding value of A_{CL} column, *i.e.*, $\mathcal{E}_{k5}(r, l_1, l_2)$ to **Q**. On decrypting the received answer, **Q** knows the impacted locations are l_1 and l_2. ■

Exposure tracing application (lines 5-11). This application takes the output of exposed location tracing application A1, *i.e.*, loc, $epoch[*, *]$ (line 6). To produce a list of device ids that may have been exposed (*i.e.*, were at the infected locations), **Q** executes creates trapdoors for all such infected locations (line 7), 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 8). All such values are sent to **Q** that decrypts them to know the final list of potentially exposed device addresses in the hash digest form (line 9).

If users request to know their exposure (*i.e.*, the presence at the infected location), **Q** first verifies the user, performs the hash function $\mathcal{H}$ under the key κ on the user device address, and then executes an intersection between the hash digests of user device address and the list of potentially exposed device addresses in hash digest form (line 11). Depending on the answer to the intersection, Q informs the user.

Example 5.2.2. Suppose, in a time duration covered by epoch Δ_x, we wish to know the impacted people that may in contact with the infected person whose device-id is d_1. From Example 5.2.1, 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. ■

Occupancy count application (lines 12-15). In order to find occupancy of locations in a given time, **Q** generates trapdoors as: $\mathcal{E}(1, y, \Delta_t)$, where y is the maximum number of rows in any epoch, and send such trapdoors to **S**. **S** executes a selection query for fetching the values of A_L column corresponding to all

6. For simplicity, we denote a queried device-id by d_i.

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

encrypted query trapdoors over A_u column (line 3). The answers to the selection query are given to **Q** that decrypts and counts the appearances of each location (line 4).

Example 5.2.3. Suppose that we want to find occupancy in a time duration covered by epoch Δ_y. **Q** generates the following three trapdoors to be searched by **S** in A_u column: $\mathcal{E}(1,1,\Delta_y)$, $\mathcal{E}(1,2,\Delta_y)$, and $\mathcal{E}(1,3,\Delta_y)$. Based on these trapdoors, **S** sends $\mathcal{E}(l_1,1)$ and $\mathcal{E}(l_2,1)$ from A_L columns. On receiving the encrypted location values, **Q** decrypts them and counts the number of rows for each location. ■

Advantages of cQUEST. cQUEST's approach is simple, but maintains hash tables during encryption of rows 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 [46], [47] that require rebuilding the entire index due to data insertion at the trusted size). cQUEST's query execution algorithm avoids reading, decrypting the entire data of an epoch to execute a query, (unlike SGX-based systems [66]); thus, saves computational overheads. Also, the key generation by XORing s_q and k_{pko} prevents the adversary to learn any information by observing the encrypted data belonging to two (or multiple) different organizations, since one of the keys will be surely different at different organizations.

Optimizations for the exposure tracing application. We provide two optimizations for trapdoor generation in $Exposure\_Trace()$. §7 will show the impact of such optimizations. Note that in the exposure trace application (Line 7 of Algorithm 2), cQUEST generates 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 the overhead in generating multiple trapdoors. 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.

Optimizations for the occupancy count application. The occupancy count application depends on counting the unique devices at each location. In cQUEST query execution algorithm (Line 13 Algorithm 2), we generate the number of trapdoors equals to the maximum number of rows in any epoch. We can avoid generating so many trapdoors, by encrypting and outsourcing counter per epoch and per location, as mentioned above. Note that based on the counter per epoch and per location, we can find the unique device at any location by just decrypting the counter value for the desired location in the desired epoch.

Information Leakage Discussion

cQUEST address all three security requirements, mentioned in §3.2, by (*i*) producing secure ciphertext or ciphertext indistinguishable data, (*ii*) authenticating the user and producing a binary answer for the exposure trace application, and hence restricting a user to ask a query about other users, and (*iii*) carefully using deterministic encryption techniques. Experiment 3 in §7.1 will also show the efficiency of cQUEST. The query execution reveals access-patterns (like SSEs or SGX-based systems [32], [37], [64], [66]). Thus, an adversary, by just observing the query execution, may learn additional information, *e.g.*, which of the rows correspond to an infected device id (by observing $Exposure\_Map$), how many people may get infected by an infected person (by observing $Exposure\_Map$). Also, since cQUEST is based on encryption, a computationally efficient adversary can break the underlying encryption technique.

6 iQUEST PROTOCOL

We provide one more version of QUEST, called iQUEST that provides information-theoretic security, *i.e.*, security regardless of the computational capabilities of an adversary. iQUEST is based on Shamir's secret-shares [58] and string-matching operation [29] on secret-shares. Sharing-sharing techniques are also quantum secure and use multiple non-colluding cloud servers to store data in the secret-share form. These servers could be any cloud instantiations from one or more vendors, such as Microsoft Azure, AWS, or Google Cloud.

6.1 Background on Secret-Sharing

We provide an overview of Shamir's secret-shares [58] and the string matching algorithm [29] that are building blocks of iQUEST.

Shamir's secret-sharing (SSS) [58]. Informally, in secret-sharing techniques, a user can take a value, convert it into multiple pieces (called *shares*), and store different shares at different non-colluding servers. A server can compute an operation over individual shares, and then, the results computed over the shares are sent to the user that computes the final answer after interpolation. Formally, in using SSS [58], the database (DB) owner divides a *secret value*, say S, into c different *shares*, and sends each share to a set of c non-colluding participants/servers. These servers cannot know the secret S until they collect $c' < c$ shares. In particular, the DB owner randomly selects a polynomial of degree c' with c' random coefficients, *i.e.*, $f(x) = a_0 + a_1x + a_2x^2 + \cdots + a_{c'}x^{c'}$, where $f(x) \in \mathbb{F}_p[x]$, p is a prime number, $\mathbb{F}_p$ is a finite field of order p, $a_0 = S$, and $a_i \in \mathbb{N}(1 \leq i \leq c')$. The DB owner distributes the secret S into c shares by placing $x = 1, 2, \ldots, c$ into $f(x)$. The secret can be reconstructed based on any $c'+1$ shares using Lagrange interpolation [25]. Note that $c' \leq c$, where c is often taken to be larger than c' to tolerate malicious adversaries that may modify the value of their shares.

String matching over secret-sharing. Now, we explain the string matching algorithm of [29].

DB 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 DB owner wishes to outsource Y; thus, 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 2) and sends shares to servers.

Values	Polynomials	Ist shares	IInd shares	IIIrd shares
0	$0+2x$	2	4	6
1	$1+8x$	9	17	25

TABLE 2: Secret-shares of $\langle 1,0,0,1\rangle$, created by the DB 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 DB owner (see Table 3). Note since a user can use any polynomial, it prevents an adversary to learn an equality condition by observing query predicates and databases.

Values	Polynomials	Ist shares	IInd shares	IIIrd shares
0	$0+3x$	3	6	9
1	$1+7x$	8	15	22

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

Servers: String-matching operation. 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 4).

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 4: Servers' computation.

User: result reconstruction. The 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.

Note. String matching operation over secret-shared dataset executes multiplication operation, which increases the degree of the polynomial. If the shares are created using polynomials of degree one and the length of the string is ℓ, then we need at least $2\ell + 1$ shares to execute string matching operation.

6.2 High-Level Overview of IQUEST

This section presents the high-level overview of IQUSET and details will be presented in the next section. Data outsourcing and query execution in IQUEST setting is shown in Figure 3.

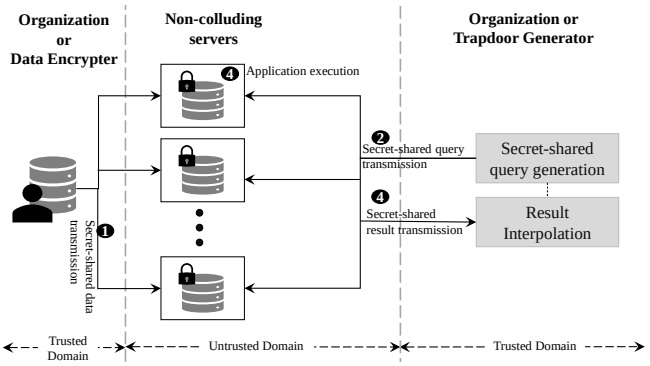


Fig. 3: Data outsourcing and query execution in secret-shared settings.

Secret-share creation of WiFi data. Likewise CQUEST, IQUEST partitions time into subintervals, called epochs, and executes secret-sharing creation algorithm for each row of each epoch. Likewise CQUEST, IQUEST satisfy the three security requirements by ensuring ensures ciphertext indistinguishability and untrackability of a user from secret-shared dataset, and by appending additional columns for efficient query execution.

Before creating secret-shares of a device address, IQUEST performs the hash function $\mathcal{H}$ under the key κ, and then implements secret-share creation algorithm over the entire row. Note that IQUEST creates two types of shares for each value of device address and location: one is for performing string matching operation using [29] and another is for retrieving the value. Further note that having two different types of shares also helps in efficient query execution, as will become clear soon. We will denote columns for string matching operation by A_{sm*} and columns for value retrieval by A_{s*}, where $*$ can be device id or location. For efficient execution of occupancy count, IQUEST adds one more column (denoted by A_{su}) to capture the unique devices in an epoch.

	Dev	Loc	Time
1	d_1	l_1	t_1
2	d_2	l_2	t_2
3	d_1	l_2	t_3
4	d_2	l_1	t_4
5	d_2	l_1	t_5
6	d_3	l_2	t_6

(a) Cleartext original WiFi dataset.

	A_Δ	A_{smid}	A_{sid}	A_{su}	A_{smL}	A_{sL}
1	Δ_x	$\mathrm{SSS}(d_1)$	$\mathrm{S}(d_1)$	$\mathrm{S}(1)$	$\mathrm{SSS}(l_1)$	$\mathrm{S}(l_1)$
2	Δ_x	$\mathrm{SSS}(d_2)$	$\mathrm{S}(d_2)$	$\mathrm{S}(1)$	$\mathrm{SSS}(l_2)$	$\mathrm{S}(l_2)$
3	Δ_x	$\mathrm{SSS}(d_1)$	$\mathrm{S}(d_1)$	$\mathrm{S}(1)$	$\mathrm{SSS}(l_2)$	$\mathrm{S}(l_2)$
4	Δ_y	$\mathrm{SSS}(d_2)$	$\mathrm{S}(d_2)$	$\mathrm{S}(1)$	$\mathrm{SSS}(l_1)$	$\mathrm{S}(l_1)$
5	Δ_y	$\mathrm{SSS}(d_2)$	$\mathrm{S}(d_2)$	$\mathrm{S}(0)$	$\mathrm{SSS}(l_1)$	$\mathrm{S}(l_1)$
6	Δ_y	$\mathrm{SSS}(d_3)$	$\mathrm{S}(d_3)$	$\mathrm{S}(1)$	$\mathrm{SSS}(l_2)$	$\mathrm{S}(l_2)$

(b) Secret-shared WiFi table for an epoch.

TABLE 5: Original WiFi dataset and secret-shared WiFi dataset using IQUEST Algorithm 3.

Example 6.2.1. Table 5a shows six rows of WiFi data, and Table 5b shows secret-shared rows that are partitioned over two epochs, each containing three rows. The first epoch is denoted by Δ_x containing the first three rows, and the second epoch is denoted by Δ_y containing the last three rows. The secret-shared table (denoted by $S(\mathfrak{R})_i$) has six columns: A_Δ for epoch identifier in cleartext, A_{smid} for device ids on which we can execute string matching, A_{sid} for device id for retrieval operation, A_{su} for storing the uniqueness of devices in an epoch and locations, A_{smL} for locations on which we can execute string matching, and A_{sL} for location for retrieval operation. ■

Query execution over secret-shared datasets. IQUEST supports the three applications (as mentioned in §1.1). Before executing a query, IQUEST verifies the querier's identity. Then, IQUEST generates trapdoors in the enclave according to application and fetches only the desired secret-shared data in the enclave to produce the final answer after interpolating the retrieved rows and filtering redundant rows; see details below.

6.3 Details of IQUEST

Now, we will present details of IQUEST's algorithms.

IQUEST Data Outsourcing Method

IQUEST uses Algorithm 3 for creating secret-shares of input WiFi table R. As clear by the description of SSS in §6.1, a secret-sharing algorithm creates multiple shares of a value. In the enclave, IQUEST will produce multiple shares of a value (device id, location, or time) and place each share into a set of non-colluding servers. The share transmission from the enclave to the servers can be done using anonymous routing protocol [35] to hide the information about the receiver servers from the adversary, and the transmission can happen without storing shares on the disk at the server where the enclave is hosted.

Note that IQUEST *Algorithm 3 creates shares for string matching denoted by* $\mathrm{SSS}(v)$ *and creates share of a complete value denoted by* $\mathrm{S}(v)$. $\mathrm{SSS}(v)$ of the value v is created by following the strategy of string matching algorithm as mentioned in §6.1. $\mathrm{S}(v)$ of the value v is created by implementing Shamir'secret-sharing algorithm over v. To create shares, IQUEST randomly selects a polynomial of an identical degree. Note that *if the adversary cannot collude with two servers, then we can use polynomials of degree one*, (since based only on one share the adversary cannot learn anything about the data). Table 5b 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 table $S(\mathfrak{R})_i$ with six columns, denoted by A_Δ, A_{smid}, A_{sid}, A_{su}, A_{smL}, and A_{sL}. Algorithm 3 works as follows:

Algorithm 3: Secret-share creation algorithm.

Inputs: Δ: duration. $\langle d_i, l_j, t_k \rangle$: A row. $\mathcal{H}$: A hash function known to only IQUEST.
Output: $S(\mathfrak{R})_i(A_{smid}, A_{sid}, A_{su}, A_{smL}, A_{sL}, A_{\Delta})$: An i^{th} encrypted table R with six columns.
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 $HTab_{id} \leftarrow init\_hash\_table\_device()$
3 **for** $t_y = \langle d_i, l_j, t_k \rangle \in \Delta_x$ **do**
4 **Allocating epoch identifier to rows and creating column** A_{Δ}
$\mathfrak{R}.A_{\Delta}[y] \leftarrow identifier(\Delta_x)$,
5 **Creating shares of device-ids and creating columns** A_{smid}**,** A_{sid}
$val \leftarrow last\_v\_bits(\mathcal{H}(d_i))$
6 $\mathfrak{R}.A_{smid}[y] \leftarrow \mathrm{SSS}(val), \mathfrak{R}.A_{sid}[y] \leftarrow \mathrm{S}(val)$
7 **Creating shares of the uniqueness of device-ids and creating columns** A_u
if $HTab_{id}[\mathcal{H}(d_i)] \neq 1$ **then** $\mathfrak{R}.A_{su}[y] \leftarrow \mathrm{S}(1), \alpha_i[] \leftarrow l_j$
8 **else if** $HTab_{id}[\mathcal{H}(d_i)] == 1 \wedge l_j \notin \alpha_i[]$ **then**
$\mathfrak{R}.A_{su}[y] \leftarrow \mathrm{S}(1), \alpha_i[] \leftarrow l_j$
9 **else if** $HTab_{id}[\mathcal{H}(d_i)] == 1 \wedge l_j \in \alpha_i[]$ **then**
$\mathfrak{R}.A_{su}[y] \leftarrow \mathrm{S}(0)$
10 **Creating shares of locations and creating columns** A_{smL}**,** A_{sL}
$\mathfrak{R}.A_{smL}[y] \leftarrow \mathrm{SSS}(l_j), \mathfrak{R}.A_{sL}[y] \leftarrow \mathrm{S}(l_j)$
$HTab_{id}[\mathcal{H}(d_i)] \leftarrow 1$

Epoch-ids and creating column A_{Δ} (Line 4). Likewise CQUEST, IQUEST partition the input WiFi data into epochs and allocate an identical epoch identifier to all rows of an epoch.

Secret-shares of devices and creating columns A_{smid}, A_{sid} (Lines 5-6). Likewise CQUEST, on each device id d_i of an epoch, IQUEST implements the hash function $\mathcal{H}$ under key κ, that results in $\mathcal{H}_{\kappa}(d_i)$, and for simplicity, we use d_i to denote $\mathcal{H}_{\kappa}(d_i)$ in the following. We create two types of shares of each device id d_i, one is denoted by $\mathrm{SSS}(d_i)$ that is used for string matching operation and stored in A_{smid}, and another is just a Shamir's secret-share of the entire device-id, denoted by $\mathrm{S}(d_i)$ and stored in A_{sid}. The values in A_{smid} help in searching for a device-id when executing the exposure map application, while the values in A_{sid} helps in fetching the device-id when executing the exposure tracing application.

Aside. Recall that creating secret-shares for string matching requires converting the hash digest of device-ids into a *unary vector*; as shown in Table 2. However, it increases the length of device-ids significantly (*i.e.*, $16 \times 128 = 2,048$, often a hash digest contains 12 hexadecimal digits (a combination of numbers $0, 1, \ldots, 9$ and alphabets A, B, ... F), and thus, every single hash digest digit will use a unary vector of size 16). Instead, we use the last $v > 1$ digits of the digest. With a very low probability, the last v digits of two hash digests will be identical.

Uniqueness of devices and creating column A_{su} (Lines 7-9). To find unique devices at each location in an epoch, IQUEST assigns value $v = 1$ when a device d_i appears for the first time at a location in an epoch; otherwise, $v = 0$, and then, creates $\mathrm{S}(v)$.

Secret-shares of location and creating columns A_{smid}, A_{sid} (Line 10). Likewise two types of secret-shares for device-ids, IQUEST creates two types of shares of each location l_i, one is $\mathrm{SSS}(l_i)$ – stored in A_{smL}, and another is $\mathrm{S}(l_i)$ stored in A_{sL}.

Differences between data outsourcing methods of CQUEST and IQUEST. 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 rows 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 column; hence, IQUEST does not keep column 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: Secret-shared relation, *i.e.*, the output of Algorithm 3.
Output: Answers to queries.
Notation: $\otimes$: string matching operation
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** $Exposure\_Map(q(\mathcal{H}_{\kappa}(d_i), \mathrm{Time}))$ **begin**
2 **Q** $\rightarrow$ **S**: $\gamma \leftarrow \mathrm{SSS}(\mathcal{H}_{\kappa}(d_i))$, Δ_t, where Δ_t is the epoch-id covers the requested Time
3 **S**: $sLoc[]$, $epoch[] \leftarrow (A_{smid}[j] \otimes \gamma) \times A_{sL}, \Delta_t\ j \in \{1, y\}$, $y =$ #rows in Δ_t
4 **Q**: $location[]$, $epoch[] \leftarrow \mathrm{interpolate}(sLoc[]), epoch[]$
5 **Function** $Exposure\_Trace(q(d_i, \mathrm{Time}))$ **begin**
6 **Q**: $location[]$, $epoch[] \leftarrow Exposure\_Map(q(d_i, \mathrm{Time}))$
7 **Q** $\rightarrow$ **S**: $sssLoc[] \leftarrow \mathrm{SSS}(location[])$, Δ_t: t covers the requested Time
8 **S**: $\forall i \in \{1, |sssLoc[]|\}, \forall j \in \{1, y\}$, $y =$ #rows in Δ_t, $sID[i, j] \leftarrow (sssloc[i] \otimes A_{smL}[j]) \times A_{sid}[j]$
9 **Q**: $id[] \leftarrow \mathrm{interpolate}(sID[*, *]), \forall i \in \{1, |sID[*, *]|\}$
10 **User** $\rightarrow$ **Q**: know about their exposure
11 **Q**: Authenticate the user against the registry information and if successful, perform intersection of $id[]$ and the user device address and return the appropriate answer
12 **Function** $Occupancy\_count(q(\mathrm{Time}))$ **begin**
13 **Q** $\rightarrow$ **S**: Δ_t: t covers the requested Time
14 **S** $\rightarrow$ **Q**: $sLoc[j] \leftarrow A_{su}[j] \times A_{sL}[j], \forall j \in \Delta_t$
15 **Q**: $location[] \leftarrow \mathrm{interpolate}(sLoc[])$
16 **Q**: $\forall l_i \in location[]$, $count_{l_i} \leftarrow count_{l_i} + 1$

IQUEST 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.

Exposure Map (lines 1-4). After verifying the publisher and on receiving the hash digest $\mathcal{H}_{\kappa}(d_i)$ from the publisher for a device id belonging to an infected person, **Q** creates SSS of $\mathcal{H}_{\kappa}(d_i)$ (denoted by γ) and sends it to each non-colluding servers along with the desired epoch identifier (line 2). 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 3). Finally, **Q** receives shares from all servers, interpolates them, and it results in all locations visited by the infected person (line 4).

Example 6.3.1. Suppose d_1 belongs to an infected person in Table 5b. To execute exposure map application for an epoch Δ_x, **Q** generates SSS of d_1, say γ. **S** checks γ against the first three shares (via string-matching operation) in A_{smid} and results in $\langle 1, 0, 1, \rangle$ (of secret-shared form) that is position-wise multiplied by $\langle \mathrm{S}(l_1), \mathrm{S}(l_2), \mathrm{S}(l_2) \rangle$. 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 $\langle l_1, l_2 \rangle$, *i.e.*, l_1, l_2 are potentially exposed locations. ■

Exposure tracing (lines 5-11). First, **Q** executes $Exposure\_Map()$ for knowing the exposed or impacted locations by an infected person (line 6). Then, **Q** creates SSS of all impacted locations (denoted by $sssLoc[]$) and sends them

to the servers along with the desired epoch-identifier (which is the same as in $Exposure\_Map()$; line 6). **S** executes string-matching operation over each value of A_{smL} against each value of $sssLoc[]$ in the desired epoch and results in either 0 or 1 of secret-shared form. Then, the i^{th} result of string-matching operation is multiplied by the i^{th} value of A_{sid}, resulting in the secret-shared device-ids, if (potentially) exposed to the infected person; otherwise, the secret-shared device-id value will become 0 of secret-shared form (line 8). Finally, **Q** receives shares from all servers, interpolates them, and results in the hash digest of device addresses of the impacted people (line 9).

Now, likewise CQUEST, if users request to know their exposure (*i.e.*, the presence at the infected location), **Q** verifies the user, performs the hash function $\mathcal{H}$ under the key κ on the user device address, and then executes an intersection between the hash digests of user device address and the list of potentially exposed device addresses in hash digest form. Depending on the answer of the intersection, **Q** informs the user.

Example 6.3.2. We continue from Example 6.3.1, where d_1 was the device of an infected person in Table 5b and impacted locations were $\langle l_1, l_2 \rangle$ that were known to **Q** after executing $Exposure\_Map(*)$ (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 three shares (via string-matching operation) in A_{smL}. It will result in two vectors: $\langle 1, 0, 0 \rangle$ of secret-shared form corresponding to γ_1 and $\langle 0, 1, 1 \rangle$ 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 $\langle d_1, 0, 0 \rangle$ and $\langle 0, d_2, d_1 \rangle$ of secret-shared form to **Q**. **Q** interpolates the vectors and knows that the device d_2 belongs to an impacted person. ■

Occupancy count (lines 12-16). **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).

Example 6.2.3. Suppose, we want to find occupancy of all location in epoch Δ_x. **Q** sends the desired epoch identifier Δ_x to **S** that executes position-wise multiplication and sends the output of the following to **Q**: $\langle S(1) \times S(l_1), S(1) \times S(l_2), S(1) \times S(l_2)$. **Q** interpolates the received answers, counts the number of each location as: $l_1 = 1$ and $l_2 = 2$. ■

Advantages of IQUEST over CQUEST. Note that IQUEST executes an identical operation on each row for executing an application, and it hides access-patterns, *i.e.*, the identity of rows that satisfy a query. Furthermore, due to using SSS, IQUEST provides the highest level of security, as well as addresses all the security requirements, mentioned in §3.2. In addition, IQUEST is fault-tolerant, due to using multiple servers.

Information Leakage Discussion

Since Algorithm 3 uses different polynomials of the same degree for creating shares of a value, an adversary cannot learn anything by observing the shares. Thus, IQUEST produces ciphertext indistinguishable or secure ciphertext dataset. Also, the query execution 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 rows satisfy the query. In addition, since query execution Algorithm 4 performs an identical operation on each share (*e.g.*, lines 3,8,14), IQUEST hides access patterns; thus, the adversary cannot learn anything from the query execution, also. Hence, in IQUEST provides stronger security than CQUEST.

IQUEST also authenticates the user and produces a binary answer for the exposure trace application, and hence restricting a user to ask a query about other users. However, as we will see in Experiment 3 in §7.1, while providing strong security guarantees, IQUEST is a little bit slower as compared to CQUEST.

7 Experimental Evaluation

QUEST has been deployed at University of California Irvine (UCI), to support occupancy count on a daily basis. (Please see interfaces of the three applications in Appendix B.) Since the exposure map and tracing applications are based on the device address of an infected person, we simulate such a scenario to evaluate the performance of QUEST. 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 the secured datasets. A 16GB RAM machine at the local-side hosts worked as the data collector that is hosted at the university IT department, which manages the WiFi infrastructure at the university.

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 $2(\ell + y) + 1$ shares are required (as mentioned in §6.1), where $\ell = 3$ (the length of device-ids, line 5 Algorithm 3) and $y = 1$ (a single secret value in column A_{sL}, line 10 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 our three applications: exposure map, exposure tracing, and occupancy count over the 10M and 50M datasets, as mentioned above.

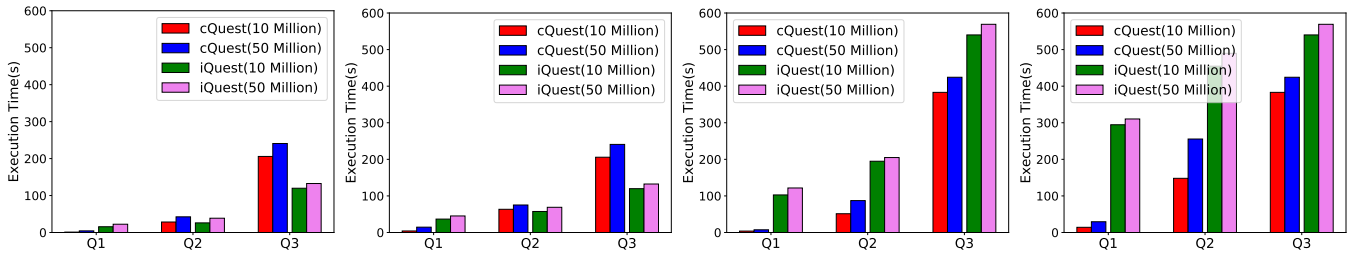
7.1 Performance Evaluation of QUEST

This section presents how does QUEST behave on different parameters and evaluates the scalability of QUEST.

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 rows/min, and IQUEST Algorithm 3 can create secret-shares of ≈38,935 rows/min. Though the throughput of Algorithm 3 is significantly less than Algorithm 1 due to creating 9 (different) shares, Algorithm 3 sustains UCI level workload.

Exp 2: Metadata size. Recall that Algorithm 1 (Algorithm 3) for CQUEST (IQUEST) 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 more than the metadata size for IQUEST, since CQUEST uses two hash tables (see line 3 Algorithm 1) and one list of visited places by each device, while IQUEST uses only one hash table (line 2 Algorithm 3). Nevertheless, metadata overheads remain small for both techniques.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2022.3166802, IEEE Transactions on Services Computing



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

Fig. 4: Exp 3: Scalability test of 10M and 50rows with varying other parameters.

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: (*i*) varying the number of infected people from 1 to 100, (*i*) varying the days for tracing from 1 to 14 days, and (*iii*) varying dataset size from 10M to 50M. Figure 4 shows the result of this experiment. In Figure 4, ***Q1 denotes exposure map, Q2 denotes exposure tracing, and Q3 denotes occupancy count applications.*** Note that *in Figure 4, we have combined all three applications to compare all of them. However, only exposure map and exposure tracing applications exploit the number of infected persons*, which we vary from 1 to 100. We execute all three applications for 1 and 14 days on 10M and 50M rows.

In exposure map application (Q1), a device has visited between 1 to 55 locations in 1 epoch. Note that Q1 using CQUEST took less time in all three cases (*i.e.*, varying the number of the infected person, number of days, and the dataset size), since it uses an index on A_{id} column (line 3 Algorithm 2); while IQUEST took more time, since it scans all data depending on the queried interval (line 3 Algorithm 4). As the number of infected people increases, the query time increases too. The cost analysis follows the same argument for the exposure tracing application (Q2) that is an extension of the exposure map application (Q1). Since in the exposure tracing application (Q2), we also find potentially impacted people after executing the exposure map application (Q1), exposure tracing application (Q2) takes more time than Q1.

For the occupancy count application (Q3) in Figures 4a and 4b, 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 rows 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 14 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. Observe that for occupancy count application in Figures 4c and 4d, 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: Impact of optimization. We have implemented the optimization method to minimize the value of max location counter (§5) for CQUEST and measured the performance improvement over 10M rows, while fixing the number of infected people to 100 and interval duration to 1-day. The *counter per epoch* for the exposure tracing application reduced the computation time from 63s (Figure 4b) to ≈35s and used 128KB more space to maintain the counter; while the *counter per epoch and per location* for the exposure tracing application took only ≈2sec with 55MB space to store the counters.

We have also implemented the optimization method for the occupancy count application that finds unique devices in an epoch. The proposed optimization (*i.e.*, outsourcing encrypted counter per epoch and per location) reduces the time of the occupancy count application from 179.4s (Figure 4b) to 1s.

Exp 5: Memory access-patterns. Recall that access patterns refer to the identity of rows that satisfy a query. Figure 6 shows a sequence of memory accesses by CQUEST and IQUEST. For this, we run the exposure tracing application multiple times, by 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. It also experimentally validate that IQUEST hides the access patterns, while CQUEST does not hide access patterns.

Exp 6: Impact of communication. Recall that IQUEST creates multiple shares of a value, places them on multiple non-colluding servers, and fetches the shared data that is propositional to the epoch size when executing an application. Thus, we need to measure the impact of communication on IQUEST. Table 8 shows the amount of data transfer using IQUEST and the data transfer time using different transfer speeds.

From Table 8, it is clear that IQUEST incurs communication overhead, while IQUEST provides a high-level of security. In particular, the exposure map application requires us to fetch ≈32MB data from each server when the tracing period was 14-days for an infected person. As the exposure tracing application requires two communication rounds (the first for knowing the impacted locations and another for knowing the impacted device ids), the exposure tracing application incurs significant 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. In the occupancy count application, we also need to fetch data corresponding to all locations in epochs that cover 14-day time duration. Thus, the occupancy count application, also, requires fetching ≈3.5GB data from each server.

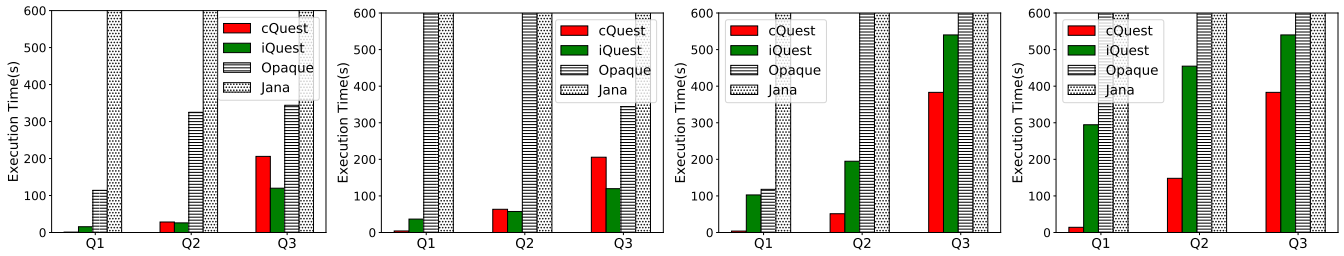
To calculate the data transfer time, we calculated the size of the data to be transferred and divided by different data transfer speeds to find the time required to move the data.

7.2 Comparing QUEST against Other Systems

Now, we compare QUEST against two existing systems (since these systems were available to us and work on any dataset).

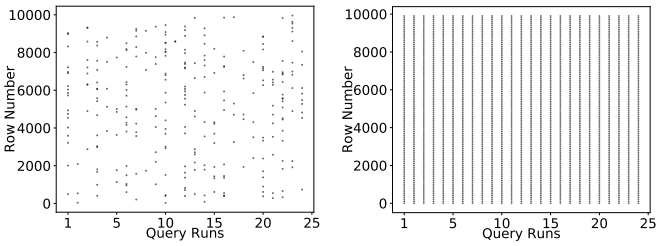
Exp 7: Using other existing systems to support QUEST applications. Since CQUEST uses SGX-based processing, we

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2022.3166802, IEEE Transactions on Services Computing



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

Fig. 5: Exp 7: Using other systems (secure hardware based Opaque and secret-sharing-based Jana) vs CQUEST and IQUEST on 10M.



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

Fig. 6: Exp 5: Access-patterns created by QUEST.

Criteria	Exposure Map	Trace	Occ. count
Dataset size	32MB	3.6GB	3.6GB
Trans. speed 25MB/s	Neg.	≈2.5m	≈2.5m
Trans. speed 100MB/s	Neg.	≈1m	≈1m
Trans. speed 500MB/s	Neg.	≈11s	≈11s

TABLE 8: Exp 6: IQUEST: the amount of data to be transferred, and the required time to transfer data on different data transfer speeds. Neg. refers to negligible.

compare CQUEST against the state-of-the-art SGX-based system Opaque [66]. Also, we compare IQUEST against the state-of-the-art secret-sharing-based system Jana [17], since IQUEST is built over secret-sharing techniques. We tried one more secret-sharing-based system, namely SMCQL [18]; however, it does not support any arbitrary as well as a large amount of dataset. For this experiment, we took 10M rows of WiFi dataset and vary: (*i*) the number of infected people from 1 to 100 and (*ii*) the days for tracing from 1 to 14.

We inserted data using non-deterministic encryption [36] in Opaque and using the underlying secret-sharing mechanism in Jana. Then, we used their query execution mechanisms for our three applications. Figure 5 shows the impact of using different systems for supporting our three applications, denoted by Q1, Q2, and Q3. Note that we drop any query that took more than 1000s.

In *terms of performance*, 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. In terms of security, 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 [37] and branch-shadow [64]) attacks. Furthermore, if data belonging to multiple organizations is non-deterministically encrypted and hosted at the same cloud, then while Opaque does not need to develop any encryption or query execution algorithm, by just observing the ciphertext dataset, an adversary may deduce the information about those users who work in multiple organization.

In *terms of performance*, IQUEST is efficient compared to Jana that takes more than 1000s in each application. 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. In *terms of security*, IQUEST and Jana provide *identical security* by hiding access-patterns, due to executing identical operations on each row.

8 CONCLUSION

In this paper, we designed, developed, and validated a system, called QUEST for privacy-preserving presence/exposure tracing and occupancy count 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. Particularly, QUEST comes with both flavors of the security, namely computational security via CQUEST and information-theoretic security via IQUEST. The capabilities provided by QUEST are vital for organizations to resume operations after a community-scale lockdown. Additionally, QUEST shows an interesting and practical use-case of cryptographic techniques, explored for data outsourcing.

Future Directions. Below, we discuss a few directions in which QUEST can be extended.

1) **Malicious entities and use of blockchains.** This paper focused on protocols to collect, store, and search data in encrypted form to support the three applications. In doing so, our solutions have assumed both the organization and the cloud to be non-malicious — they could be honest-but-curious and may wish to learn the behavior of individuals either from the ciphertext or query execution. We have not considered approaches to protect against attacks, such as an organization or the cloud deliberately deleting the data about an individual (log truncation attack), or maliciously modifying users' data. If such attacks were to be considered, solutions such as blockchains would be required to address such concerns and to write authenticated tamper-proof logs, which could be written into the blockchain.
2) **QUEST with cleaned WiFi data.** As mentioned earlier, there are some issues with WiFi also, as: duplicate devices, the presence of spurious devices (such as printers/machines) in buildings that may artificially affect the occupancy counts, missing sensor values (due to disconnections), and location ambiguity dye to coarse nature of region covered by an access point, etc. As mentioned before, there are tools (such as [7], [8], [48]) that exploit semantic information about locations and people to clean WiFi data and reach accuracy as high as 92-93%. One can think about using tools to clean WiFi data and then use QUEST and compare the accuracy among two different scenarios. Another interesting direction would be to use data cleaning tools at the cloud equipped with secure hardware (such as Intel Software Guard eXtension – SGX).

REFERENCES

[1] Apple's and Google's COVID-19 contact tracing technology, available at: https://tinyurl.com/wfw9ojr.
[2] PEPP-PT, available at: https://github.com/pepp-pt.
[3] States are finally starting to use the Covid-tracking tech Apple and Google built — here's why, available at: tinyurl.com/y6yun3y8.
[4] QUEST Applications: available at: https://tippersweb.ics.uci.edu/covid19/d/IwAc1O9Wk/covid-19-effort-at-uc-irvine?orgId=1.
[5] TraceTogether, available at: https://www.tracetogether.gov.sg/.
[6] South Korea's 100m: available at: https://tinyurl.com/yb5mj9o6.
[7] Blyncsy Inc. https://blyncsy.com/mercury-contact-tracing.
[8] Occuspace IO Inc. https://occuspace.io/.
[9] Switzerland's SwissCovid: available at: https://tinyurl.com/z3vvpz4.
[10] Stanford University's COVID-Watch: https://covid-watch.org/.
[11] SafeTrace, available at: https://github.com/enigmampc/safetrace.
[12] https://tinyurl.com/yzcvexcm.
[13] R. Agrawal et al. Order-preserving encryption for numeric data. In *SIGMOD*, pages 563–574, 2004.
[14] T. Altuwaiyan et al. EPIC: efficient privacy-preserving contact tracing for infection detection. In *ICC*, pages 1–6, 2018.
[15] P. Antonopoulos et al. Azure SQL database always encrypted. In *SIGMOD*, pages 1511–1525, 2020.
[16] A. Arasu et al. Orthogonal security with cipherbase. In *CIDR*, 2013.
[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] J. Bater et al. SMCQL: secure query processing for private data networks. *Proc. VLDB Endow.*, 10(6):673–684, 2017.
[19] J. Bater et al. Shrinkwrap: Efficient SQL query processing in differentially private data federations. *PVLDB*, 12(3):307–320, 2018.
[20] M. Bellare et al. Deterministic and efficiently searchable encryption. In *CRYPTO*, pages 535–552, 2007.
[21] C. Bi et al. Familylog: A mobile system for monitoring family mealtime activities. In *PerCom*, pages 21–30, 2017.
[22] R. Canetti et al. Adaptively secure multi-party computation. In *STOC*, pages 639–648, 1996.
[23] R. Canetti et al. Anonymous collocation discovery: Taming the coronavirus while preserving privacy. *CoRR*, abs/2003.13670, 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*, 2016:86, 2016.
[27] R. Curtmola et al. Searchable symmetric encryption: Improved definitions and efficient constructions. *JCS*, 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] B. Fuhry et al. Hardidx: Practical and secure index with SGX in a malicious environment. *JCS*, 26(5):677–706, 2018.
[32] B. Fuhry et al. Encdbdb: Searchable encrypted, fast, compressed, in-memory database using enclaves. *CoRR*, abs/2002.05097, 2020.
[33] R. Gelles et al. Multiparty proximity testing with dishonest majority from equality testing. In *ICALP*, pages 537–548, 2012.
[34] R. Gerhards et al. Rfc 5424: The syslog protocol. *IETF*, 2009.
[35] D. M. Goldschlag et al. Onion routing. *Commun. ACM*, 42(2), 1999.
[36] S. Goldwasser and S. Micali. Probabilistic encryption. *J. Comput. Syst. Sci.*, 28(2):270–299, 1984.
[37] J. Götzfried et al. Cache attacks on Intel SGX. In *EUROSEC*, 2017.
[38] Y. Gvili. Security analysis of the covid-19 contact tracing specifications by apple inc. and google inc. *IACR*, page 428, 2020.
[39] A. Hekmati et al. CONTAIN: privacy-oriented contact tracing protocols for epidemics. *CoRR*, abs/2004.05251, 2020.
[40] Y. Ishai et al. Private large-scale databases with distributed searchable symmetric encryption. In *RSA*, pages 90–107, 2016.
[41] M. B. Kjærgaard et al. Challenges for social sensing using wifi signals. In *WMSCSS*, pages 17–21, 2012.
[42] H. Krawczyk. SIGMA: the 'SIGn-and-MAc' approach to authenticated Diffie-Hellman and its use in the IKE-protocols. In *CRYPTO*, 2003.
[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] S. Lee et al. Inferring fine-grained control flow inside SGX enclaves with branch shadowing. In *USENIX*, pages 557–574, 2017.
[46] R. Li et al. Fast range query processing with strong privacy protection for cloud computing. *PVLDB*, 7(14):1953–1964, 2014.
[47] R. Li et al. Adaptively secure conjunctive query processing over encrypted data for cloud computing. In *ICDE*, pages 697–708, 2017.
[48] Y. Lin et al. LOCATER: cleaning wifi connectivity datasets for semantic localization. *Proc. VLDB Endow.*, 14(3):329–341, 2020.
[49] M. Maier et al. Probetags: Privacy-preserving proximity detection using wi-fi management frames. In *WiMob*, pages 756–763, 2015.
[50] S. Mehrotra et al. TIPPERS: A privacy cognizant iot environment. In *PerCom W*, pages 1–6, 2016.
[51] J.-L. Meunier. Peer-to-peer determination of proximity using wireless network data. In *PerComW*, pages 70–74, 2004.
[52] R. Poddar et al. Arx: A strongly encrypted database system. *IACR Cryptology ePrint Archive*, 2016.
[53] R. A. Popa et al. CryptDB: processing queries on an encrypted database. *Commun. ACM*, 55(9):103–111, 2012.
[54] A. Prasad and D. Kotz. ENACT: encounter-based architecture for contact tracing. In *WPA@MobiSys*, pages 37–42, 2017.
[55] C. Priebe et al. EnclaveDB: A secure database using SGX. In *SP*, pages 264–278, 2018.
[56] P. Sapiezynski et al. Inferring person-to-person proximity using wifi signals. *IMWUT*, 1(2):24:1–24:20, 2017.
[57] C. Schlener and S. Vasudev. Flexible snmp trap mechanism, Jan. 30 2001. US Patent 6,182,157.
[58] A. Shamir. How to share a secret. *Commun. ACM*, 22(11), 1979.
[59] D. X. Song et al. Practical techniques for searches on encrypted data. In *SP*, pages 44–55, 2000.
[60] Q. Tang. Privacy-preserving contact tracing: current solutions and open questions. *CoRR*, abs/2004.06818, 2020.
[61] A. Trivedi et al. Wifitrace: Network-based contact tracing for infectious diseases using passive wifi sensing. *IMWUT*, 5(1):37:1–37:26, 2021.
[62] C. Troncoso et al. Decentralized privacy-preserving proximity tracing overview of data protection and security. 2020. Available at: https://github.com/DP-3T/documents.
[63] J. Wang et al. Interface-based side channel attack against intel SGX. *CoRR*, abs/1811.05378, 2018.
[64] W. Wang et al. Leaky cauldron on the dark land: Understanding memory side-channel hazards in SGX. In *CCS*, pages 2421–2434, 2017.
[65] K. Yoshida et al. Contact analysis on COVID-19 using a campus network: poster abstract. In *SenSys*, pages 752–753, 2020.
[66] W. Zheng et al. Opaque: An oblivious and encrypted distributed analytics platform. In *NSDI*, pages 283–298, 2017.
[67] M. Zhou et al. EDUM: classroom education measurements via large-scale wifi networks. In *UbiComp*, pages 316–327, 2016.
[68] M. Zhou et al. Mobicamp: a campus-wide testbed for studying mobile physical activities. In *WPA@MobiSys*, pages 1–6, 2016.

Shantanu Sharma is an assistant professor Department of Computer Science, New Jersey Institute of Technology, USA. He received his Ph.D. in Computer Science in 2016 from Ben-Gurion University, Israel. During his Ph.D., he worked with Prof. Shlomi Dolev and Prof. Jeffrey Ullman. He obtained his Master of Technology (M.Tech.) degree in Computer Science from National Institute of Technology, Kurukshetra, India, in 2011. He was awarded a gold medal for the first position in his M.Tech. degree. His research interests include secure and privacy-preserving database systems, trustworthy smart spaces, and distributed algorithms.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2022.3166802, IEEE Transactions on Services Computing

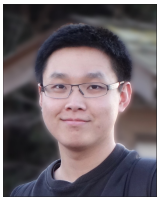
Sharad Mehrotra received the PhD degree in computer science from the University of Texas, Austin, in 1993. He is currently a professor in Department of Computer Science, University of California, Irvine. Previously, he was a professor with the University of Illinois at Urbana Champaign. He has received numerous awards and honors, including the 2011 SIGMOD Best Paper Award, 2007 DASFAA Best Paper Award, SIGMOD test of time award, 2012, DASFAA ten year best paper awards for 2013 and 2014, 1998 CAREER Award from the US National Science Foundation (NSF), and ACM ICMR best paper award for 2013. His primary research interests include the area of database management, distributed systems, secure databases, and Internet of Things.

Nisha Panwar is an assistant professor at Augusta University, Georgia. She obtained her Ph.D. in Computer Science from Ben-Gurion University, Israel, in 2016, where he worked with Prof. Shlomi Dolev and Prof. Michael Segal. She received her Master of Technology (M.Tech.) degree in Computer Engineering from National Institute of Technology, Kurukshetra, India in 2011. She was a Post Doc at University of California, Irvine, USA. Her research interests include security and privacy issues in IoT systems, as well as, in vehicular networks, computer network and communication, and distributed algorithms.

Nalini Venkatasubramanian received the PhD degree in computer science from the University of Illinois, Urbana-Champaign in 1998. She is a professor of computer science at the university of California, Irvine. From 1991 to 1998, she was a member of technical staff and software designer engineer for Hewlett-Packard. Her research interests are networked and distributed systems, internet technologies and applications, ubiquitous computing and urban crisis responses. She received US National Science Foundation (NSF) Career Award in 1990, best paper award from IEEE Consumer Communications and Networking (CCN) Conference in 2006, and best student poster award from IEEE/ACM 11th International Symposium on Cluster, Cloud and Grid Computing (CCGRID2011).

Peeyush Gupta is a Ph.D. student, advised by Prof. Sharad Mehrotra, at University of California, Irvine, USA. He obtained his Master of Technology degree in Computer Science from Indian Institute of Technology, Bombay, India, in 2013. His research interests include IoT data management, time series database systems, and data security and privacy.

Shanshan Han is a PhD student in computer science at UC Irvine. Her research interests are secure database systems and data privacy.



Guoxi Wang is a Networked Systems Ph.D. candidate at UC Irvine. His research interest includes data privacy in the Internet of Things and edge computing. His current research focus is on the systems that support privacy-preserving sensing data collection in smart communities.

APPENDIX A
SQL QUERIES OF THE THREE APPLICATIONS

Table 9 provides SQL queries for our three applications: exposure map, exposure trace, and occupancy count.

APPENDIX B
INTERFACES OF THE THREE APPLICATIONS

Figure 7 shows the interface of the exposure map and exposure tracing applications. Since we do not have the device address of a real infected person, as mentioned in our experiments also, we simulated the execution of the exposure map and exposure tracing applications. The left-hand side of Figure 7 shows that there is a user 49352 (user name is anonymized here) who got COVID-19 and visited which building in the past 14 days. The right-hand side of Figure 7 shows other users who got potentially exposed to the infected person 49352.

Figure 8 shows the interface of the occupancy count application at University of California Irvine, before the lockdown was announced. Figure 8 shows the occupancy count only at floors level. On clicking, one can also see the occupancy count at different granularity, such as regions buildings. Note that instead of showing the number of unique devices, we represent occupancy as low, medium, and high, depending on a pre-specified threshold. Figure 9 shows the interface of the occupancy count application at University of California Irvine after the lockdown was announced.

C FORMAL SECURITY PROPERTY

We have discussed our security requirements and how does Quest handle them in §3.2. We have also discussed information leakage discussion from CQUEST and IQUEST. Below, we defined our desired security properties formally.

Recall that in the cloud-based setting, an adversary, which may be the cloud or a user, 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. Also, we need to ensure that a querier cannot learn the information about infected people or potentially exposed people. 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. It requires that the adversary, just by observing the secured dataset, cannot deduce that any two rows 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 maintaining: (*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. (*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

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2022.3166802, IEEE Transactions on Services Computing

Applications	SQL syntax
Exposure Map	SELECT DISTINCT locationId FROM WiFiData INNER JOIN InfectedUsers ON WiFiData.macId = InfectedUsers.macId WHERE timestamp $> t_1$ AND timestamp $< t_2$
Exposure Trace	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
Occupancy count	SELECT locationId, DISTINCT COUNT(MacId) FROM WiFiData WHERE timestamp $> t_1$ AND timestamp $< t_2$ GROUP BY locationId, EXTRACT(WifData.timestamp, delta)

TABLE 9: Three supported applications by QUEST in SQL.

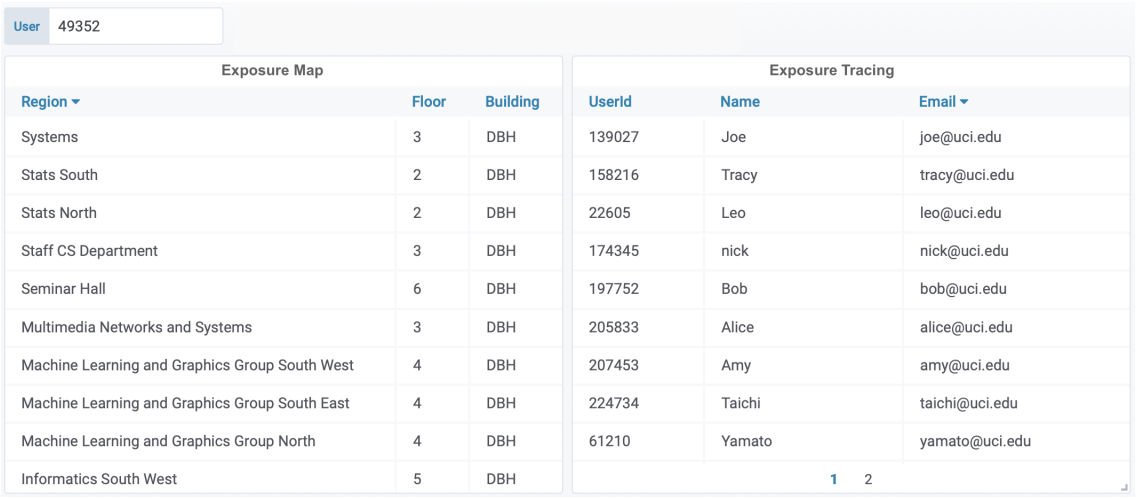


Fig. 7: Exposure map and trace applications interface.

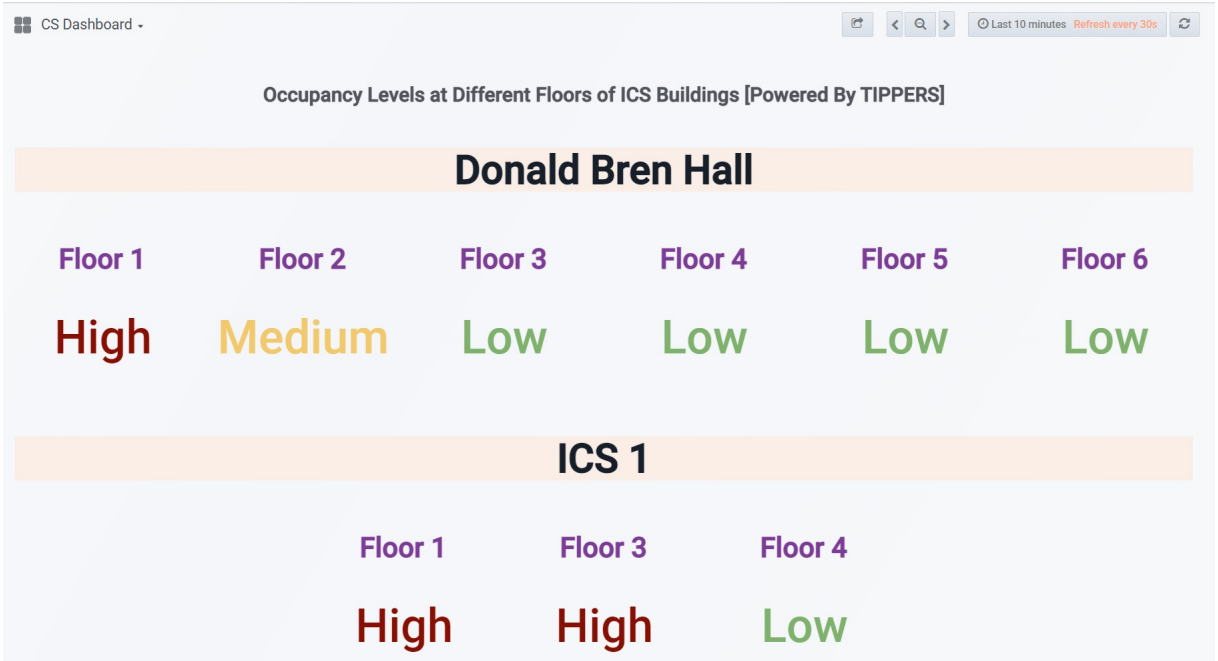


Fig. 8: Occupancy count application interface before lockdown.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2022.3166802, IEEE Transactions on Services Computing

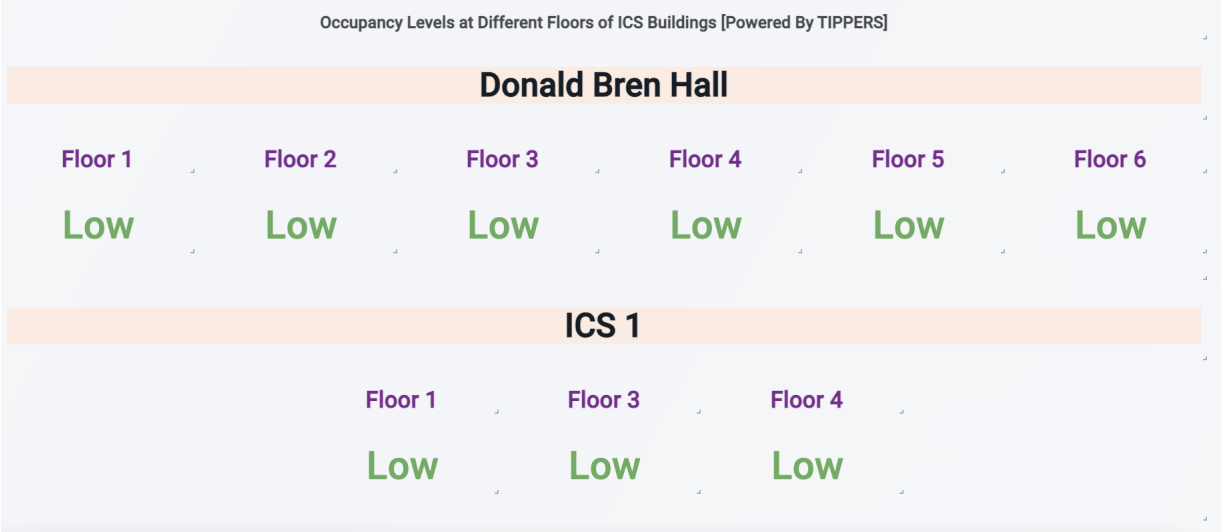


Fig. 9: Occupancy count application interface after lockdown.

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 table 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 table, 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 security properties (which are widely considered in many cryptographic approaches [29], [40]) will hide access patterns, thus, the adversary cannot distinguish between two different queries and the satisfying output rows. However, such a secure algorithm (*i.e.*, IQUEST as given in §6) incurs the computation overhead, as clear from Experiment 3 in §7.1. Thus, we relax the access pattern hiding property (similar to existing searchable encryption or secure-hardware-based algorithms) and, also, presented an efficient access-pattern revealing algorithm, CQUEST in §5. Observe that IQUEST satisfies all the above-mentioned properties, by producing ciphertext secure dataset and by executing identical operations regardless of the data as well as queries.

In order to define the security property for techniques, we follow the standard security definitions of symmetric searchable encryption techniques [27] that define the security in terms of leakages: (*i*) *setup leakage* $\mathcal{L}_s$ that includes the leakages from the encrypted database size and leakages from metadata size, and (*ii*) *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 rows are retrieved to answer a query)). Observe that in access pattern hiding techniques such as IQUEST, there will be *no* $\mathcal{L}_q$ leakage, while the adversary will only know the size of the encrypted dataset via $\mathcal{L}_s$.

Based on these leakages $\mathcal{L}_s$ and $\mathcal{L}_q$, the security notion provides guarantees that an encrypted database reveals no other information (*i.e.*, occupancy count of a location or tracking an individual by the cloud) 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 (a similar setting will work for IQUEST):

1) $(K, \mathfrak{R})) \leftarrow Setup(1^k, R)$**:** is a probabilistic algorithm that takes as input a security parameter 1^k and a table R. It outputs a secret key K and an encrypted table $\mathfrak{R}$. This algorithm (as given in Algorithm 1) is executed at QUEST's encrypter, before outputting the encrypted table at the server.
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 DBMSs hosted at the server to retrieve the desired rows (in the enclave).
3) $results \leftarrow Query\_Exe(trapdoor\{1, \ldots, q\}, \mathfrak{R})$**:** is a deterministic algorithm. It takes the encrypted table $\mathfrak{R}$ 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 the chosen-keyword attack* (IND-CKA) model [27]. IND-CKA prevents an adversary from deducing the cleartext values of data from the encrypted table or from the query execution, except for what is already known.

Security Definition. Now, based on the formal definition of CQUEST, we define the security of CQUEST.

Let $\Psi = (Setup, Trapdoor\_Gen, Query\_Exe)$ be a row 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 table R and sends it to a simulator. The simulator runs $Setup$ algorithm and produces an encrypted table $\mathfrak{R}$ that is sent to $\mathcal{A}$. The adversary $\mathcal{A}$ executes a polynomial number of queries on the encrypted tables $\mathfrak{R}$ 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 table R'. Note that this table may or may not be identical to the table 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 table $\mathfrak{R}'$ 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 tables, R_1 and R_2, having an identical number of attributes and an identical number of rows. Relations R_1 and R_2 may or may not overlap. The simulator simulates the role of QUEST encrypter to produce an encrypted table 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 table encrypted by the simulator, based on the query execution. The adversary cannot differentiate between the two encrypted tables, since if the adversary cannot find which encrypted table is produced by the simulator with probability non-negligibly different from 1/2, then the query execution reveals nothing about the table.

Proof outline. We first construct a simulator that can build the entire encrypted dataset based only on $\mathcal{L}_s$ and $\mathcal{L}_q$. In order to do that, the simulator executes cQUEST data encryption Algorithm 1 and regards the leakage $\mathcal{L}_s$. The simulator picks n number of any random rows and encrypts them using Algorithm 1 such that the size of the simulated encrypted data matches with the size of the real encrypted data. Note that this is possible for the simulator to produce such a dataset, since the simulator knows $\mathcal{L}_s$.

Now, we need to show that the simulator can also simulate queries and can produce trapdoors for the queries. Note that according to the leakage $\mathcal{L}_q$, the simulator knows which rows were accessed for all queries that have been executed, due to the revealed access patterns via $\mathcal{L}_q$. Thus, the simulator can simulate the tokens such that the simulated token regards $\mathcal{L}_q$. Now if a probabilistic polynomial time adversary issues a query (*i.e.*, a keyword), the simulator can generate a trapdoor for this query as above. The trapdoor given by the simulator and the query result produced by the simulated encrypted dataset are indistinguishable to the adversary, because of identical leakages $\mathcal{L}_s$ and $\mathcal{L}_q$ from the real encrypted dataset and the simulated encrypted dataset. Thus, cQUEST is IND-CKA secure.