Received: 25 January 2022 | Revised: 25 April 2022 | Accepted: 22 May 2022

IET Information Security

DOI: 10.1049/ise2.12070

The Institution of Engineering and Technology WILEY

ORIGINAL RESEARCH

Secure contact tracing platform from simplest private set intersection cardinality

Jiahui Gao[1] | Chetan Surana[2] | Ni Trieu[1]

[1]Biodesign Institution, Arizona State University, Tempe, Arizona, USA

[2]Computer Science, Amazon.com Inc, Seattle, Washington, USA

Correspondence

Jiahui Gao, Biodesign Institution, Arizona State University, 727 E TYLER St, Tempe, AZ 85287, USA.
Email: jhgao@asu.edu

Funding information

National Science Foundation, Grant/Award Numbers: 2031799, 2101052, 2115075; Google

Abstract

Contact tracing is an essential tool for controlling the spread of disease through human populations. However, existing contact tracing applications are either vulnerable to privacy and security attacks or heavy bandwidth/computational requirements on the client's devices. In this work, we introduce SecureCT, a Secure Contact Tracing platform with strong privacy protection and lightweight cost. SecureCT prevents linkage attacks, eliminates replay and relay attacks, and allows the phone's holder to delegate their contact tracing computation to untrusted servers while maintaining the user's privacy. The technical core of our scheme is an efficient Private Set Intersection Cardinality protocol which only relies on symmetric-key primitives. We evaluate its performance to show the feasibility of our proposed system in practice.

1 | INTRODUCTION

In the past 2 years, our life has been dramatically changed by the pandemic of coronavirus SARS-CoV-2 which is also known as COVID-19. This virus can be spread via air and droplets. When an infected person has contact with others physically, it is likely that the virus is spread to them. By tracking down the spread route of the virus, medical workers can make the right plan to contain the spread and inform the potential infection to the related people timely. This process is called contact tracing (CT). At the beginning of the pandemic, CT was done in the form of interviewing, to ask the patient to recall the place they have been and the people they have met, which has low efficiency and a high error rate. A more powerful technique is required to implement contact tracing digitally and efficiently.

A large number of CT mobile applications (apps) have been developed and deployed, with most of them based on the exchange of random and anonymous tokens using Bluetooth (BT) [1–2]. These are generally decentralised systems that alert users if they may have come in close proximity with other positively diagnosed users (infected users). A high adoption rate is critical for the success of CT apps in helping curb the spread of severe diseases. However, adoption is low as these systems are prone to a host of attacks, like linkage attack, relay attack, and replay attack [3–5].

Most CT apps have a service provider (server) in the loop which stores tokens of infected users. User devices periodically query the server and download the tokens of infected users. They check whether there is a match between the set of downloaded tokens and the set of tokens received from other users they were in close contact with. This involves a download of a huge set of data from the server, periodically, hence making it computationally inefficient for client mobile devices. Moreover, the coronavirus spread through common surfaces that have been touched by infected users. Informing users to avoid geographical areas where many positively diagnosed users have visited can help lessen the spread of the disease through surface transmission. Decentralised, BT-based systems based on proximity of devices cannot handle this case. GPS-based methods that match location traces of users may not be as accurate as BT-based systems and are vulnerable to dictionary attacks [6]. Hence a secure, efficient, and scalable protocol for CT that considers both contact transmission and surface transmission is needed.

No matter using the BT-based method or the GPS-based method, the privacy of the user is another concern. There is a

need for a framework that is robust against attacks and information leakage, and is computationally light and efficient. In this work, we aim to prevent two important attacks: linkage attack and replay/relay attack. In the linkage attack, we consider valid tokens which are generated from the same device and may be broadcast at several places. The tokens of the same client cannot be linked together by any participant. Most BT-based contact tracing systems are vulnerable to this attack. For example, Seiskari [7] installed BLE-sniffing devices to different known physical locations and collected contact tracing tokens. By keeping track of when and where they received which tokens, authors in [7] can identify the travel route of the individuals. To prevent the linkage attack, prior work [8, 9] relies on private set intersection cardinality (PSI-CA), which is used to check how many tokens held by a user match the tokens in a set stored on a server without the users revealing their tokens. In this work, we propose a more efficient PSI-CA protocol, which can be integrated into a contact tracing system to improve the system's performance.

To prevent the replay and relay attacks, prior work [5, 10] propose delayed authentication so that the CT server uses public verification to authenticate user tokens. However, their authenticated system is not efficient, especially on the user's device. In addition, the identity of COVID-19 positive users might be revealed during the authenticated process. To eliminate the replay and relay attacks, we integrate the random tokens generation of the BT approach with GPS and timestamps [11] such that the transformed token contains the user's secret BT token and location only in an encrypted form.

1.1 | Our contribution

In this work, we make the following contributions:

- We propose a novel and deployment-friendly PSI-CA protocol which relies only on symmetric-key primitives (e.g. AES).
- We design and implement a contact tracing system, SecureCT, that can provide strong privacy guarantees. It is able to eliminate replay and relay attack using GPS.
- We implement SecureCT and evaluate it on the client's phone using Google Pixel 3. For the client set size $n = 2^{11}$, without including the time spent waiting on the server's response, the client requires a running time of 208 milliseconds and only 32 KBs of communication. The server requires 35 s to perform CT for the server set size $N = 10^9$.

1.2 | Organisation

In Section 2, we begin with the related work of Contact Tracing systems as well as the protocols of PSI-CA. Then we give the details of primitives for our design, security model, and potential attacks in Section 3. The PSI-CA is presented in Section 4. The SecureCT contact tracing system is demonstrated in Section 5. Finally, we present the details and the result of our implementation in Section 6.

2 | RELATED WORK

In this section, we overview the state of the art in contact tracing and PSI-CA.

2.1 | Decentralised contact tracing

There are two main categories of CT approach: centralised and decentralised. In a centralised approach, a trusted third party is required. The TraceTogether app is a typical example that is launched by the Singapore government. In TraceTogether, the central authority (the government server) registers and stores user details and unique identifiers, and assigns a set of contact tokens to be broadcast at specific times. An infected user shares all the received broadcast tokens with the central authority, who then uses the tokens to identify and follow-up with users who have come in contact with him. This system could be misused as a surveillance system, where the central authority can learn graphs of user interaction.

In this work, we focus on decentralised contact tracing and review two popular types of contact tracing systems.

2.1.1 | GPS-based construction

It is very important for the patient to recall the place they visited and the people they met before they tested positive. Based on this information, the analyst can rebuild the trajectory of the patient and make the corresponding plan to track and contain the spread of the virus. A natural way to implement contact tracing digitally is to record the physical location of the people and find the potential contact upon that.

In the GPS-based construction, the location information of the user is collected for contact tracing analysis. In reference [12], the authors proposed a network-centric WiFi sensing approach for digital contact tracing. By collecting the Wifi logs of device associations to access points within the network, a graph structure capturing the user device trajectory can be generated. The intersection of the trajectories can be gained by using efficient time-evolving graphs and algorithms.

Safe Paths [13], extended to Path Check, is one contact tracing approach that is based on GPS location traces of users. The app logs the user's GPS location periodically. The location is quantised to a geographical area using Geohash [11]. The app then uses a one-way hash function to mask the Geohash and timestamp. An infected user's hashes are shared to a central server maintaining a public list. Other devices can download this list and detect an exposure using set intersection. This approach may not be as effective as BT-based techniques and involves a large number of hashes to be stored locally and downloaded from a server. It is susceptible

to dictionary attacks [6], where a one-way deterministic hash used to mask private information can be potentially reversed.

2.1.2 | BLE-based construction

Most of the current decentralised CT systems are based on Bluetooth Low Energy (BLE). BLE is a radio specification for short-range communication and is well suited for proximity detection due to its accuracy and feasibility. The BLE-based CT protocols are designed in a very similar way as follows:

1. Alice and Bob are two users of the contact tracing protocol. They download and install the app on their smartphone.
2. When Alice and Bob meet each other, their phone will generate and exchange the token.
3. Suppose Alice is tested positive for the disease. She will upload all the tokens she generates to a third-party server.
4. At the server, a list of tokens from the user who is tested positive can be maintained and published or a query mechanism can be provided to the user for checking their contacts.

Google/Apple Exposure Notification (GAEN) solution and Decentralised Privacy-Preserving Proximity Tracing (DP3T) [1] are built based on this idea of sharing tokens via Bluetooth devices. As discussed in Section 1, the typical BLE-based CT approach remains susceptible to various attacks. For example, in GAEN, when Alice is diagnosed with the disease, her daily diagnosis keys (used to generate the tokens) are uploaded to the server. Thus, Alice's anonymous identifier tokens, as they are broadcast each day, can be linked to each other. The tokens can also be linked across days if Alice frequently appears at the same place. According to the calculation in [9], DP3T with Cuckoo filters requires the users to download 110 MB each day for 40,000 new daily infections. It costs each user $1/day using Google Fi network $10/GB. The GAEN solution would cost $0.10/day although their design is more vulnerable to linkage attacks than the DP3T. PSI-CA was introduced to prevent the linkage attack in the CT [8, 9, 14]. We review their PSI-CA protocols in Section 2.2.

2.2 | Server-aided PSI-cardinality

Private set intersection (PSI) allows two parties to compute the intersection of their datasets without revealing any additional information. The description of functionality is given in section 3.5 Over the last several years PSI has become truly practical with extremely fast cryptographically secure implementations [15, 16]. We refer the reader to [17] for additional discussion and motivation of PSI. Recently, private contact tracing applications related to COVID-19 [6, 8, 9, 14] found PSI-CA as the ultimate cryptographic tool, allowing multiple participants (users and healthcare providers) to privately match contact information and notify users who may have been infected. In this work, we mainly focus on a variant of PSI problems, PSI-CA. The functionality of PSI-CA is to allow parties to learn the size of the intersection and nothing else. Particularly, this functionality can be achieved in a "server-aided" way in which there is a helping cloud server to do some of the computation for the participants. Below we consider the works most relevant to ours.

- **DH-based PSI-CA** [14]: Epione [14] is one of the first works that applies PSI-CA into CT to prevent the linkage attack. Instead of using CT tokens (referring the token in some of the earlier contact tracing schemes that do not hide the identity of the corresponding user) for matching, their protocol uses PRF values of these tokens. The PRF computation is implemented via DH-based OPRF [18]. To make PSI-CA efficient for a large server-side database and a small client-side database, Epione relies on keyword-PIR [19, 20] which allows a client to check whether their PRF is in the server's data, without revealing the PRF itself to the server. As a result, their PSI-CA protocol has communication complexity $O(n \log N)$ which is linear in the size of the smaller set (n), and logarithmic in the larger set size N. However, it requires each user to perform $O(n)$ exponentiations (public-key operations) for DH-based OPRF and $O(N)$ symmetric-key operations for keyword PIR computation.
- **Delegated PSI-CA** [9]: Catalic, a delegated contact tracing system proposed in [9], allows multiple untrusted cloud servers to do the most of contact tracing computation so that the efficiency of the PSI-CA protocol on the client's device can be improved. A set of non-colluding cloud servers take the secret shares of the token from the client and jointly perform oblivious distributed key PRF (Odk-PRF) [9] with a backend server holding a set of tokens from infected patients. In the end, only one cloud server learns the PRF value of the client's token and nothing else. By having these values, the cloud server can compute PSI-CA with less computation for the client. The client's computation and communication complexity of the PSI-CA protocol in Catalic is linear in the size of the smaller set $O(n)$, and is independent of the larger set's size N. However, Catalic system requires at least two non-colluding cloud servers with a heavy computation/communication cost of Odk-PRF. In addition, the underlying OPRF of Odk-PRF is based on Oblivious Transfer [21], which is not deployment-friendly.
- **Function Secret Sharing (FSS) based PSI-CA** [8]: Dittmer et al. [8] introduces a variant of PSI-CA (so-called weighted PSI-CA) in which each token of the client has an associated secret weight. The weight indicates a proximity estimate (e.g. "is there a wall between us?") that enables a more fine-grained tracing response. The weighted PSI-CA is based on cheap FFS constructions [22, 23], thus it is efficient on both the client's and server's sides. Concretely, in the FSS-based PSI-CA, the computation complexity of the client and server is $O(n)$ and $O(N)$, respectively. The communication complexity is $O(n)$. However, their construction assumes that there exist two non-colluding

servers, each holding an identical set of infected tokens. This assumption is not realistic in the context of contact tracing.

3 | PRELIMINARIES

In this section, we introduce the notation and the primitives for our contact tracing system and the PSI-CA protocol which will be discussed in the later sections.

3.1 | Notations

In this work, the computational and statistical security parameters are denoted by κ, λ respectively. We use [.] notation to refer to a set. For example, $[m]$ implies the set $\{1, 2, \ldots, m\}$. Additionally, we use $[i, j]$ to denote the set $\{i, i + 1, \ldots, j\}$. Other special notations for the data structure will be introduced before the usage.

3.2 | Geohash

Geohashing [11] is a convenient geocoding system that can encode a location latitude and longitude into a string of letters and digits, with the length of encoding defining the precision. It is a hierarchical spatial data structure that divides geographical areas into the grid like buckets. A useful property of a geohash is arbitrary precision, allowing one to gradually remove characters from the end, and reducing the length while losing precision. The longer the prefix of geohashes of the two locations, the closer they are spatially.

A geohash from GPS coordinates is computed by interleaving two binary strings, one each for the latitude and longitude, with bits recursively splitting the grid into intervals. The calculation of a geohash can be elucidated with an example. The interval is between −90 and 90° for latitude and between −180 and 180° for longitude. For example, the first four bits of a GPS coordinate with latitude 19.5 is 1001. The first bit is 1 for it lies in the second half of the first interval. Then, 0 is noted for it lies in the first half of the interval 0–90, followed by 0 for the interval 0–45, 1 for interval 0–22.5, and so on recursively, until the desired accuracy is reached. The interleaved binary strings for longitude and latitude are represented as letters and digits using the base-32 encoding. In the implementation of the BT plus GPS protocol proposed in this work, geohash of length 8 is chosen, to accommodate for reasonable accuracy of proximity detection.

3.3 | Oblivious Key-Value Store (OKVS)

An OKVS [24] is a data structure in which a sender, holding a set of key-value mapping $P = \{(x_i, y_i), i \in [n]\}$ with pseudo-random values y_i, wishes to hand that mapping over to a receiver who is able to evaluate the mapping on any input but without revealing the keys x_i. Formally, an Oblivious Key-Value Store consists of two algorithms:

- Encode$(P) \rightarrow T$: a randomised algorithm that takes as input a set of n key-value pairs $P = \left\{(k_i, v_i)_{i\in[n]}\right\}$ from the key-value domain $\mathcal{K} \times \mathcal{V}$, and outputs an OKVS table T.
- Decode$(x, T) \rightarrow y$: a deterministic algorithm that takes as input a table T, a key x and outputs a value y.

The correctness of the OKVS is that if for all key-value pairs $A \subseteq \mathcal{K} \times \mathcal{V}$ with distinct keys and pseudo-random values, Encode$(A) = T$ and $(k, v) \in A$ then *Decode* $(T, k) = v$.

An OKVS is secure if the values v_i are chosen uniformly then the output of Encode hides the choice of the keys k_i.

3.4 | Hash table data structures

3.4.1 | Cuckoo hashing

In the scheme of Cuckoo hashing, there is a hash table of β bins denoted $B[1\ldots\beta]$. k random hash functions $h_1, \ldots, h_k : \{0,1\}^\star \rightarrow [\beta]$ are chosen to generate the position index for the input element. There is an additional storage called stash in case some elements failed to find an empty bin. The client uses a variant of Cuckoo hashing such that each item $x \in X$ is placed in exactly one of the β bins. Using the Cuckoo analysis [20] based on the set size $|X|$, the parameters β, k are chosen so that with high probability $(1 - 2^{-\lambda})$ every bin contains at most one item, and no item has to be placed in the stash during the Cuckoo eviction (i.e. no stash is required) It is a scheme with worst case constant lookup and deletion time, and amortised constant insertion time. On inserting an item, it uses the first hash function. If an item already exists there, the current item replaces it, and the evicted item is re-inserted using the subsequent hash function. Repeat till the process settles. If there is a cycle, a *rehash* is performed by choosing new hash functions $h_1, \ldots, h_k : \{0,1\}^\star \rightarrow [\beta]$.

3.4.2 | Simple hashing

With simple hashing, items in the input set Y are inserted into β bins using the same set of k Cuckoo hash functions (i.e. each item $y \in Y$ appears k times in the hash table). Using a standard ball-and-bin analysis based on k, β, and the input size of the client $|X|$, one can deduce an upper bound η such that no bin contains more than η items with high probability.

3.5 | PSI-CA

Private set intersection cardinality is a security protocol which allows parties to learn the size of the intersection of their input sets and nothing else. We consider two parties setting with a

helping server. The description of its functionality is given by Figure 1.

3.6 | Security models

The protocols in this work are scrutinised under specific security and adversarial models. Consider that multiple parties agree to cooperatively compute a function f, and also agree to share the evaluation result to a particular party. Two classical security models are the colluding and non-colluding models [14]. In a colluding model, a subset of parties may be dishonest and collude during the execution of the protocol. In a non-colluding model, the parties are independent and do not collude.

There are two adversarial model definitions. In the honest but curious model (semi-honest model), the parties strictly follow the protocol without deviation but may attempt to learn extra information from the execution script apart from that intended by the protocol. In the malicious model, the adversary or dishonest party may attempt any polynomial time strategy such as supplying invalid inputs, deviating and executing different computation, so as to disrupt the protocol to leak information.

In this work, the non-colluding and semi-honest setting is considered, where the parties are assumed not to collude and follow the protocol's description.

3.7 | Attacks

The aforementioned approaches introduced in Section 2 are all vulnerable to some of the attacks including relay attacks, linkage attacks by users or servers, and also false reporting by users. We list and illustrate these possible attacks below.

- Linkage Attack: Linkage attack allows attacker to refer the identity of anonymous data by linking it to some non-anonymous dataset. In the case of contact tracing, linkage attack can be applied by both the server and the client. The server can do it by observing the contact token it received. In our proposed SecureCT, this is prevented by having all the tokens randomly generated. For clients, all they will receive from the protocol is the number of infected people they have been in contact, they cannot apply linkage attack to any arbitrary client.
- Social graph reconstruction: A determined malicious adversary can learn a part of the social graph in a centralised system. The server can learn the social subgraphs with contacts between the diagnosed users and the people they have come in contact with. A determined user can obtain proof of encounter with a diagnosed person in a decentralised system [10].
- Replay and relay attack—Identification of diagnosed users: An adversary, whether an individual, group, or organisation, can collect contact tokens, from the app or using strong Bluetooth receivers, along with the time and place of collection. In a decentralised system, the tokens of the diagnosed users are public. The adversary can use this to a posteriori identify the user that was diagnosed [10].
- False encounter and false reporting: An adversary may install artificial broadcasters, and/or falsely report as positively diagnosed, to increase false positive exposure alerts.

In this work, our proposed SecureCT framework is robust against all the attacks listed above.

PARAMETERS:
- A Sender $\mathcal{S}$ with input $Y = \{y_1, \ldots, y_N\}$.
- A Receiver $\mathcal{R}$ with input $X = \{x_1, \ldots, x_n\}$.
- A cloud server $\mathcal{C}$.

FUNCTIONALITY:
- Wait for input set X and Y from the $\mathcal{S}$ and $\mathcal{R}$.
- Give the server $\mathcal{C}$ nothing.
- Give the $\mathcal{R}$ an intersection set size $|X \bigcap Y|$

FIGURE 1 Functionality of private set intersection cardinality

4 | SIMPLEST SERVER-AIDED PSI-CA

In this section, we present the simplest server-aided PSI-CA in which the computation utilises a third-party non-colluding cloud server. Our proposed protocol does not require OT-based OPRF [25] or Odk-PRF [9] (e.g. public-key base-OT), thus, it relies only on symmetric-key primitives. To the best of our knowledge, this is the only construction with such a property.

4.1 | Technical overview

Consider an untrusted cloud server $\mathcal{C}$ who helps to perform PSI-CA on behalf of the receiver $\mathcal{R}$. Our protocol consists of two main phases. In the first phase, the receiver $\mathcal{R}$ chooses a random key k which is sent to the sender $\mathcal{S}$. On the other hand, $\mathcal{R}$ computes the PRF values $x'_i \leftarrow F(k, x_i), \forall i \in [n]$, and sends them to the cloud server $\mathcal{C}$. One can consider this phase as executing oblivious PRF where the sender $\mathcal{S}$ knows the PRF key k and the cloud server $\mathcal{C}$ learns the PRF values $F(k, x_i)$ without knowing the key k. However, different from traditional OPRF, $\mathcal{C}$ learns nothing about the underlying values x_i. Having the PRF key k, the sender computes the PRF values $y'_i \leftarrow F(k, y_i), \forall i \in [N]$.

Our second phase replies on OKVS and takes the PRF values x'_i, y'_i as inputs. More precisely, the sender $\mathcal{S}$ encodes the points $P = \{(y'_1, v_1), \ldots, (y'_N, v_N)\}$ into an OKVS table $T \leftarrow \text{Encode}(P)$ which is sent to the cloud server $\mathcal{C}$. Here, the set $V = \{v_1, \ldots, v_N\}$ is randomly chosen by the sender $\mathcal{S}$. The cloud server $\mathcal{C}$ knows a table T, so she decodes it on every x'_i and obtains a set $W = \{w_1, \ldots, w_n\}$. According to the OKVS's

functionality, we have $w_i \in V$ if x'_i was encoded in T, otherwise, w_i is random. In addition, the cloud server $\mathcal{C}$ cannot infer any information from W due to the randomness property of the OKVS. To allow the receiver $\mathcal{R}$ to learn only the intersection size, the sender $\mathcal{S}$ and the cloud server $\mathcal{C}$ respectively send a set V and W to the receiver $\mathcal{R}$ in a randomly shuffled order. At this point, the receiver can count how many items are in the intersection by computing $W \cap V$ as $|W \cap V| = |\{x'_i\}_{i\in[n]} \cap \{y'_i\}_{i\in[N]}| = |X \cap Y|$. In addition, the shuffling makes the receiver learn nothing about which specific item was in common (i.e. which w_i corresponds to the item $x_j \in X$). Thus, the intersection set is not revealed which can prevent the linkage attack in the contact tracing scenarios.

Garimella et al. [24] lists several OKVS constructions with different encoding/decoding costs. The most efficient OKVS scheme is based on a 3-Hash Garbled Cuckoo Table (3H-GCT) in which: the encoding time for encoding N items in the OKVS is $O(N\lambda)$; the decoding time for decoding n elements is $O(n\lambda)$; and the length of the OKVS table T is $1.27N + \log(N) + \lambda$.

However, 3H-GCT is not deployment-friendly as it involves complicated peeling/unpeeling processes. Thus, in the implementation of SecureCT, we use a deployment-friendly OKVS variant, a polynomial-based OKVS scheme, in which the encoding and decoding algorithms are exactly polynomial interpolation and evaluation. In polynomial-based OKVS, the encoding/decoding time takes $O(N\log(N)^2)$ and the table's length is N.

4.2 | Construction

Our server-aided PSI-CA protocol is presented in Figure 2. It closely follows the technical overview described in Section 4.1. Recall that the set V is pseudo-random and known by both parties, $\mathcal{S}$ and $\mathcal{R}$. Thus, the set V can be generated from a PRG seed known by these parties. In our construction, we reuse the PRF key k as the PRG seed. Clearly, the outputs of PRG and PRF are independent, and their distributions are uniform.

4.2.1 | Correctness

To show correctness of our construction, we consider two following cases based on whether $x_i \in X$ is in the intersection of X and Y:

- Case 1: Suppose x_i is an element in the set of Y, $\exists y_j \in Y$, such that $y_j = x_i$. Then we have $x'_i = F(k, x_i)$ which equals to $y'_j = F(k, y_j)$. When decoding the OKVS table T using x'_i, the receiver obtains w_i. Based on the correctness of OKVS,

PARAMETERS:
- Set size n and N.
- A sender $\mathcal{S}$, a receiver $\mathcal{R}$, and a cloud server $\mathcal{C}$
- A pseudo-random function $F : (\{0,1\}^\star, \{0,1\}^\kappa) \to \{0,1\}^\kappa$
- A pseudo-random generator $PRG : \{0,1\}^\kappa \to \{0,1\}^\star$
- An OKVS primitive with Encode and Decode algorithms described in Section 3.3.

INPUTS:
- Sender $\mathcal{S}$ has input $Y = \{y_1, \ldots, y_N\}$, where $y_i \in \{0,1\}^\kappa$ for $i \in [N]$.
- Receiver $\mathcal{R}$ has input $X = \{x_1, \ldots, x_n\}$, where $x_i \in \{0,1\}^\kappa$ for $i \in [n]$.
- Cloud server $\mathcal{C}$ has no input.

PROTOCOL:
1. The receiver $\mathcal{R}$ chooses a random PRF key k and sends it to the sender $\mathcal{S}$
2. Upon receiving the key k from $\mathcal{R}$, the sender $\mathcal{S}$ computes:
   - A pseudo-random set $V = \{v_1, \ldots, v_N\}$ generated by PRG as $v_1||\ldots||v_N \leftarrow PRG(k)$, where each v_i has κ-bit length.
   - PRF values $y'_i = PRF(k, y_i), \forall y_i \in Y$
   - An OKVS table $T \leftarrow \mathsf{Encode}(\{(y'_i, v_i)\}_{i\in[N]})$
3. The sender $\mathcal{S}$ sends T to the cloud server $\mathcal{C}$.
4. The receiver $\mathcal{R}$ computes $x'_i \leftarrow F(k, x_i), \forall x_i \in X$, and sends a set $X' = \{x'_1, \ldots, x'_n\}$ to the cloud server $\mathcal{C}$.
5. Upon receiving T from the sender $\mathcal{S}$ and X' from the receiver $\mathcal{R}$, the cloud server $\mathcal{C}$ computes $w_i = \mathsf{Decode}(T, x'_i), \forall x'_i \in X'$, and sends $W = \{w_{\pi(1)}, \ldots, w_{\pi(n)}\}$ to the receiver $\mathcal{R}$, where $\pi : ([n]) \to ([n])$ is a random shuffled function chosen by $\mathcal{C}$.
6. $\mathcal{R}$ generates a pseudo-random set $V = \{v_1, \ldots, v_N\}$ from PRG as $v_1||\ldots||v_N \leftarrow PRG(k)$, and outputs $|V \cap W|$.

FIGURE 2 Our server-aided PSI-CA construction

$w_i = v_j$ where v_j is the corresponding value of y'_j from the encode process of OKVS. In other words, there is one-to-one mapping from $x_i = y_j$ to $w_i = v_j$. Thus, this gives a contribution to $|W \cap V|$ so that the receiver $\mathcal{R}$ can learn.

- Case 2: Suppose x_i is not an element in the set of Y. The decode result of $x'_i = F(k, x_i)$ is a random value since x'_i was never used in the encode process of OKVS. There is no contribution to $|W \cap V|$ from x_i.

4.2.2 | Security

We turn to show the security of our PSI-CA construction by the following theorem.

Theorem 1 *Given the OKVS functionality described in Section 3.3, the PSI-CA construction of Figure 2 securely implements the PSI-CA functionality with the presence of an untrusted semi-honest cloud server $\mathcal{C}$, malicious sender $\mathcal{S}$ and malicious receiver $\mathcal{R}$.*

Proof We exhibit simulators for simulating these three following cases: corrupt sender $\mathcal{S}$, corrupt receiver $\mathcal{R}$, and corrupt cloud server $\mathcal{C}$. For the first two cases, we describe simulation in both the semi-honest and malicious settings. We argue the indistinguishability of the produced transcript from the real execution.

Simulating sender

The simulator is given the sender's input Y and obtains the PRF key k from the honest receiver. Since the key k is randomly chosen by the receiver, we can replace k with random.

In the semi-honest setting, the sender gives the set Y and k to the ideal world and receives nothing. In the real world, he receives an empty output. Therefore, the simulation is perfect.

In the malicious setting, the simulator runs the sender internally and might encode a malicious pair into the OKVS. One can simulate this action as changing the sender's input, thus, which trivially concludes the simulation.

Simulating receiver

The simulator is given the receiver's input X, the set $W = \{w_{\pi(1)}, \ldots, w_{\pi(n)}\}$ in a randomly permuted order π: $([n]) \rightarrow ([n])$ chosen by the cloud server combiner $\mathcal{C}$, a set of V, and the intersection size $|X \cap Y|$.

In the semi-honest setting, we consider two cases. For each $x_i \notin X \cap Y$, we can replace the term w_i with an independently random element due to the obliviousness property of the OKVS table T. For each common item $x_i \in X \cap Y$, the value $w_i \leftarrow \mathsf{Decode}(T, x'_i)$ is equal to a value in the set V. We assume that the receiver and $\mathcal{C}$ do not collude, thus the shuffle function π is hidden from the simulator's view. Therefore, we can replace $w_{\pi^{-1}(i)}$ with a random element in V (i.e, the permutation hides the common items). In other words, the simulator only learns $|X \cap Y|$ and Y. The simulation is perfect.

In the malicious setting, the simulation is elementary as it is similar to simulating the malicious sender. More precisely, any malicious action can be considered as the receiver changes his input.

Simulating cloud server

The simulator simulates the view of adversary $\mathcal{A}$, which consists of the PRF values $x'_i = F(k, x_i)$ from the receiver, and an OKVS table $T \leftarrow \mathsf{Encode}\left(\{(y'_i, v_i)\}_{i \in [N]}\right)$ from the sender. We consider two following cases:

- Security for the receiver $\mathcal{R}$: In Step 1 of our protocol, the receiver $\mathcal{R}$ randomly chooses the PRF key k and sends it to the sender $\mathcal{S}$. We assume that $\mathcal{A}$ does not collude with the sender, thus the key k is unknown to $\mathcal{A}$. Thanks to the cryptographic guarantees of the underlying PRF protocol, the PRF outputs can be replaced with randoms. In Step 5, $\mathcal{A}$ evaluates Decode which also produces an output indistinguishable from the real world.
- Security for the sender $\mathcal{S}$: In Step 2 of our protocol, $\mathcal{S}$ encodes a set of key-value pairs $\{(y'_i, v_i)\}_{i \in [N]}$ via Encode algorithm, where $y'_i = F(k, y_i)$ is a PRF value on the item $y_i \in Y$ with the key k unknown by $\mathcal{A}$, and v_i is generated from the secret PRG seed. Because of the PRF property, we replace y'_i with random. In our protocol, the cloud server does not know the PRG seed, we can also replace v_i with random. The Encode functionality takes a set of random pairs, thus its distribution is uniform.

In summary, the output of $\mathcal{A}$ is indistinguishable from the real execution.

4.2.3 | Complexity

We begin by the analysis of the computational complexity. The sender requires to perform $2N$ AES calls to generate the set V and compute N PRF values y'_i. The sender also encodes N items into an OKVS. Denote the computational cost of encoding/decoding OKVS as $|\mathtt{OKVS}|$ which is $O(N\lambda)$ or $O(N\log(N)^2)$ depending on which OKVS variant is used. The sender computational complexity is $2N + |\mathtt{OKVS}|$. The receiver requires to compute $n + N$ AES calls. The cloud server needs to decode n items, which costs $|\mathtt{OKVS}|$.

For the communication complexity, the sender sends an OKVS table encoded with $O(N)$ values to the cloud server. The receiver sends an κ-bit PRF key from the sender, sends n PRF values to the cloud server, and receives n OKVS decoding values from him. In summary, the communication complexity of the sender, the receiver, and the cloud server is $\kappa + |T|$-bit, $\kappa + n(\kappa + \lambda + \log(N))$-bit, and $|T| + n(\lambda + \log(N))$-bit, respectively. Here, T is the size of the OKVS table with $O(N)$ values.

Finally, we consider the round complexity. It is easy to see that our protocol is 1-round.

4.3 | Optimisation: Unbalanced PSI-CA

We present a server-aided PSI-CA protocol in the unbalanced setting where the receiver's set size n is much smaller than the sender's set size N. The unbalanced PSI-CA is a good fit for our running application, contact tracing, where the sender has million diagnosis tokens (e.g. $N = 10^9$) while the receiver has a few thousand tokens (e.g. $n = 10^3$). Recall that our primary goal aims to minimise the communication and computation cost on the receiver's side. However, the construction in Figure 2 requires the receiver compute $N + n$ AES executions. When N is larger, the computation might be a bottleneck, especially on the resource-constrained devices, for example, the end-user's phone or edge device. In this section, we describe an optimisation based on hashing to bins that enables large cost savings on the receiver's side. In particular, the receiver's computation complexity of our optimised construction is linear in the size of the smaller set $O(n)$ and independent of the larger set's size N.

Our main idea is that the receiver and sender use hashing to partition their items into $m = O(n)$ buckets. Each bucket contains a smaller fraction of inputs, which allows all participants to perform computation bin-by-bin. Concretely, we use the Cuckoo-and-Simple hashing scheme [26] such that each bin of the $\mathcal{R}$ consists of at most one item. Thus, the sender is allowed to use only one value v_i for all items in the ith bin. The amount of data the sender has to touch per query is now only the items that were mapped to the same bin as the receiver query. Thus, it is much more efficient computationally on the sender's side. In addition, the receiver only needs to generate $O(n)$ values $V = \{v_1, \ldots, v_m\}$ before computing $V \cap W$. Thus, the receiver's computation complexity reduces to $O(n)$ from $O(N + n)$. Note that variants of this idea have appeared in previous work [26].

We now discuss concrete hashing schemes and present a formal description of our unbalanced PSI-CA in Figure 3. In this construction, the receiver uses Cuckoo hashing with h hash functions and inserts her set of size n into m buckets. The sender maps his set of size N into m buckets using the same set of h hash functions, so-called simple hashing. With the high probability, each of the sender's items appears h times across all over bins. Using a standard ball-and-bin analysis [26] based on h, m, and n, one can deduce an upper bound β such that no sender bin contains more than β items with high probability p. Let $B_{S[i]}$ and $B_{R[i]}$ denote the items in the sender's and receiver's ith bucket, respectively.

The receiver $\mathcal{R}$ computes a PRF $x' \leftarrow F(k, x)$ for an item $x \in B_{R[i]}$ bucket, or chooses a dummy value for the empty bin. He then sends all the PRF values to the cloud server $\mathcal{C}$ in order. Since each $\mathcal{C}$'s bucket contains exactly one item, it allows $\mathcal{C}$ and $\mathcal{S}$ to execute OKVS bin-by-bin with a particular default value v. That is, the v_i values must be assigned bin-wise, instead of item-wise as before in Figure 2. By doing so, the receiver only needs to generate m values v_i from the PRG seeds, which speeds up the receiver's computation cost.

However, all values in the OKVS data structure should be pseudorandom. In the unbalanced PSI-CA, the sender computes encodes a set of points $\left(y'_i, H\left(y'_i\right) \oplus v_b\right)$ into OKVS. Here, $y'_i = F(k, y_i)$ for each item y_i in the bth bucket, v_b is assigned for that bin, and H is an one-way hash function. Upon receiving an OKVS table, the cloud server $\mathcal{C}$ decodes it using the PRF value x' corresponding to that bin, and then removes the mask $H(x')$. We observe that this modification does not impact any of our applications, since the cloud server $\mathcal{C}$ can learn either v_b or random, and all v_b values are different across over bins.

4.3.1 | Correctness and security proofs

Our unbalanced PSI-CA construction is correct by observation, except with the negligible probability of Cuckoo hashing failure. In particular, our constructions fail to be correct if the receiver is unable to hash its items into m bucket. However, we note that we can set parameters so that the probability of such failures is negligible.

The security of our unbalanced PSI-CA construction follows straightforwardly from the security of PSI-CA construction described in Figure 2. Thus, we omit the proof of the following theorem.

Theorem 2 *Given the OKVS functionality described in Section 3.3 and Cuckoo hashing scheme described in Section 3.4, the unbalanced PSI-CA construction of Figure 3 securely implements the PSI-CA functionality with the presence of an untrusted semi-honest cloud server $\mathcal{C}$, semi-honest sender $\mathcal{S}$, and malicious receiver $\mathcal{R}$.*

Note that our unbalanced PSI-CA approach is not secure against a malicious sender. The sender may map y_i only to a subset of the required bins instead of all of them. For example, if the sender puts the point $\left(y'_i, H\left(y'_i\right) \oplus v_b\right)$ only in one bin $B_{S[b]}$ and the receiver indeed counted y into the intersection size $X \cap Y$. It means that the cloud service (so is the receiver) puts its query x_i in bin $B_{S[b]}$. This leaks the information related to other queries that could have been put in that bin.

4.3.2 | Complexity

We first discuss the computation complexity of our server-aided unbalanced PSI-CA construction.

- The receiver first hashed its n elements into $m = O(n)$ bins via the Cuckoo hashing scheme with complexity $O(nh)$. The receiver also needs to run $m = O(n)$ AES to generate the set of V and compute n PRF values of x'. The receiver computational complexity is $O((h + 1)n)$

PARAMETERS:

- Set size n and N.
- A sender $\mathcal{S}$, a receiver $\mathcal{R}$, and a cloud server $\mathcal{C}$
- An one-way hash function $H : \{0,1\}^\kappa \to \{0,1\}^\kappa$
- A pseudo-random function $F : (\{0,1\}^\star, \{0,1\}^\kappa) \to \{0,1\}^\kappa$
- A pseudo-random generator $PRG : \{0,1\}^\kappa \to \{0,1\}^\star$
- An OKVS primitive with Encode and Decode algorithms described in Section 3.3.
- Hashing parameters: a number of bins m, maximum bin sizes β for receiver's bins, a number of hash functions h.

INPUTS:

- Sender $\mathcal{S}$ has input $Y = \{y_1, \ldots, y_N\}$, where $y_i \in \{0,1\}^\kappa$ for $i \in [N]$.
- Receiver $\mathcal{R}$ has input $X = \{x_1, \ldots, x_n\}$, where $x_i \in \{0,1\}^\kappa$ for $i \in [n]$.
- Cloud server $\mathcal{C}$ has no input.

PROTOCOL:

1. The receiver $\mathcal{R}$ hashes items $x_i \in X$ into m bins using the Cuckoo hashing scheme with h hash functions. Let $B_{R[b]}$ denote the items in the receiver's bth bucket.
2. The sender $\mathcal{S}$ hashes items $y_i \in Y$ into m bins under h hash functions. Let $B_{S[b]}$ denote the set of items in the sender's bth bucket.
3. The receiver $\mathcal{R}$ chooses a random PRF key k and sends it to the sender $\mathcal{S}$
4. Upon receiving the key k from $\mathcal{R}$, the sender $\mathcal{S}$ generate a pseudo-random set $V = \{v_1, \ldots, v_m\}$ from PRG as $v_1 || \ldots || v_m \leftarrow PRG(k)$, where each v_i has κ-bit length.
5. For each bucket $b \in [m]$, the receiver $\mathcal{R}$ computes $x'_b \leftarrow F(k, x_b), \forall x_b \in B_{R[b]}$ or chooses a random value $x'_b \leftarrow \$$ for empty bin, and sends a set $X' = \{x'_1, \ldots, x'_m\}$ to the cloud server $\mathcal{C}$.
6. For each bucket $b \in [m]$:
   (a) The sender $\mathcal{S}$:
   - computes PRF values $y'_i = F(k, y_i), \forall y_i \in B_{S[b]}$
   - creates a set of points $P_b = \{(y'_i, H(y'_i) \oplus v_b)\}$, then pads P_b with dummy pairs to the maximum bin size β
   - encodes P_b into an OKVS table $T_b \leftarrow \mathsf{Encode}(P_b)$
   - sends T_b to the cloud server $\mathcal{C}$.

   (b) Upon receiving T_b from $\mathcal{S}$ and x'_b from $\mathcal{R}$, the cloud server $\mathcal{C}$ computes $w_b = \mathsf{Decode}(T_b, x'_b) \oplus H(x'_b)$
7. The cloud server $\mathcal{C}$ sends $W = \{w_{\pi(1)}, \ldots, w_{\pi(n)}\}$ to the receiver $\mathcal{R}$, where $\pi : ([n]) \to ([n])$ is a random shuffled function chosen by $\mathcal{C}$.
8. $\mathcal{R}$ generates a pseudo-random set $V = \{v_1, \ldots, v_m\}$ from PRG as $v_1 || \ldots || v_N \leftarrow PRG(k)$, and outputs $|V \cap W|$.

FIGURE 3 Our server-aided unbalanced private set intersection cardinality construction

- Sender also needs to generate the set of V which costs $m = O(n)$ AES calls and compute N PRF values y'. The sender has to hash all its N elements into the same m bins using those h hash functions, so basicall there are $h \cdot N$ AES calls. After the hashing, the sender needs to encode the items into an OKVS for each of m bins with a cost of $m \cdot |\mathtt{OKVS}|$. It should be noted that the computational cost of encoding/decoding OKVS is much smaller than that without the hash scheme. The sender computational complexity is $m + (h+1)N + m \cdot |\mathtt{OKVS}|$.
- The cloud sever decodes all the OKVS values of x' with the cost of $m \cdot |\mathtt{OKVS}|$

For the communication complexity, the sender sends the cloud server m OKVS tables, each encodes with $O(N/n)$ values on average. The receiver receives a κ-bit PRF key from the sender, sends m PRF values to the cloud server, and receives n decoded values from him.

Finally, it is easy to see that our server-aided unbalanced PSI-CA construction is 1-round.

5 | SecureCT SYSTEM

In this section, we describe the SecureCT system in detail. The PSI-CA protocol is used in the query CT process. We also propose an enhancement for token generation in Section 5.3, which allows SecureCT to eliminate the replay and relay attacks.

5.1 | System's overview

We build a digital CT system aiming to identify and alert persons potentially exposed to an infected user. The framework of our SecureCT system is shown in Figure 4. Bluetooth low energy (BLE) is used here to detect whether people were in close proximity. The contact tracing systems comprise apps running on users' mobile devices, a cloud server, a backend server, and a health provider. We design this system following the idea described in Section 2.1.2. The working flow of the system goes like this. Users' apps use BLE to broadcast and receive anonymous tokens. Suppose there are two users, Alice and Bob, in close proximity. Alice stores the token broadcasted by Bob and vice versa. In this way, each user's app stores a list of tokens it has received from other users who have been in close proximity. When Bob is infected and tested positive, he uploads the seed used to generate the tokens, or all the tokens, to the backend server. Other users make the query through the cloud server to determine if they have come in contact with an infected user. The query is done by running the PSI-CA protocol shown in Figure 3 between the cloud server and backend server. Since Alice was in contact with Bob, she will be alerted because the intersection between the set of tokens she has received from other users and the set of tokens of infected users maintained by the server is non-zero.

The potential vulnerabilities associated with solely BLE-based contact tracing systems, including linkage and replay attacks, identification of diagnosed users, false reporting and false encounters, are covered in Section 3.7. Hence we propose the SecureCT system which is a secure, scalable, and efficient contact tracing system with strong privacy guarantees which is robust against these vulnerabilities. The framework takes a step further to aid in the prevention of contacts between the users and infected users. We also propose a token generation method containing the GPS information and timestamps to eliminate the replay and relay attack.

5.2 | End-to-end framework

Now we describe our SecureCT system design in detail. There is an app on the users' mobile devices to broadcast and receive the token. The cloud server and backend server can be assumed untrustworthy. The healthcare provider is needed for diagnosis and certification. There are mainly five phases for computing as follows:

1. **Initialisation**: During this phase, the cloud server randomly chooses a permutation function $\Pi : [N] \rightarrow [N]$, and provides it to the healthcare provider. The healthcare provider randomly chooses N certificates C_i and gives the

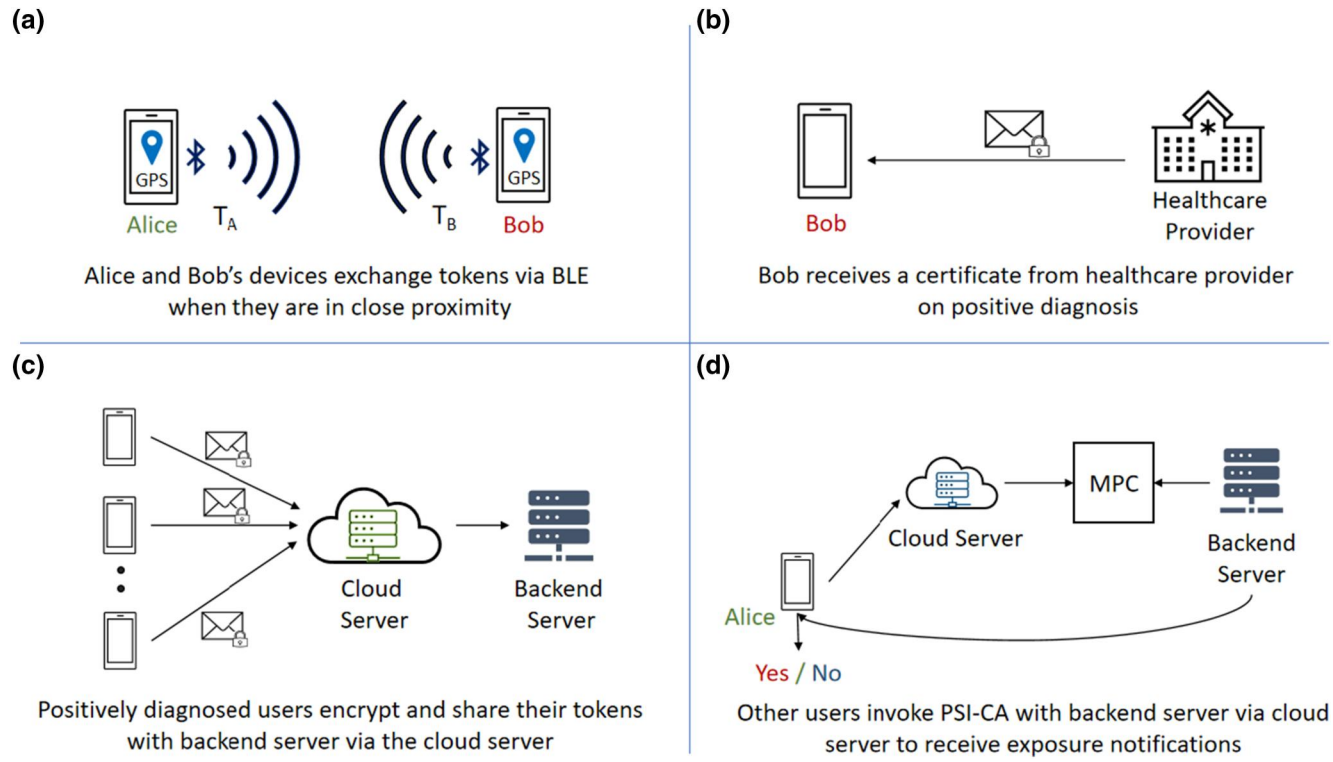

FIGURE 4 Our SecureCT framework. (a) At contact (b) At test (c) Diagnosis token collection (d) User's query

backend server $\Pi(C_i)$ in order. This can be done by randomly choosing a PRG seed c for generating valid certificates. The healthcare provider sends the seed to the backend server, which can locally compute the certificate $C_i \leftarrow PRG(c\|i)$. The backend server generates a public-private key pair (pk, sk) and sends the public key to every user. Each user/phone u_i randomly chooses a PRG seed s_i which is used to generate the Bluetooth tokens. As long as the server's configuration does not change, this phase does not need to be run more than once. Whenever a new user registers, they only need to generate their own PRG seed and receive the public key from the backend server.

2. **At Contact**: The BLE device is used to exchange tokens whenever users are in close proximity. The user can generate the τ tokens per day to be broadcast by using a PRG as $t_{i,1}\|\ldots\|t_{i,\tau} = PRG(s_i\|d)$, where s_i is the user's secret PRG seed, d is the current day, τ is an upper bound on the number of tokens needed for that day. Figure 4a illustrates this phase of token exchange and storage. In Section 5.3, we propose a method to add GPS information into the token and, here, we can consider the token is generated via a PRG function and a corresponding seed.
3. **At Test**: When a user u_i is diagnosed by the healthcare provider, the healthcare provider computes a certificate $C_i \leftarrow PRG(c\|i)$ using their own secret PRG seed, and gives it to the user u_i. The certificate validates that this user tested positive for the disease and is used to detect false-positive claims if any. Note that before adding the user's tokens to the infected tokens database, the backend server checks whether the certificate is valid. If not, the backend server has permission to ask the cloud server to reveal the identity (e.g. the IP address) of this nefarious user.
4. **Token Collection**: Figure 4c describes the process of collecting diagnosis tokens, which involves the computation and communication of every user, the cloud server, and the backend server. The goal is to have the backend server collect all diagnostic tokens in a privacy-preserving manner. This phase contains three steps as follows:
   a) At the beginning of the phase, every ith diagnosed user encrypts their PRG seed s_i together with their received certificate C_i using the public key pk of the backend server as $Enc(pk, s_i\|C_i)$ and sends it to the cloud server.
   b) After receiving the encrypted values from the diagnosed users, the cloud server permutes and then forwards them to the backend server.
   c) Using its secret key, the backend server decrypts ciphertexts to obtain plaintexts as $s_i\|C_i$. First, the backend server verifies whether C_i is valid. This can be done as follows. The backend server uses the PRG seed c of the healthcare provider, generates all possible certificates as $\mathbb{C} = \{C_i \leftarrow PRG(c\|i), \forall i \in [N]\}$, and checks whether $C_i \in \mathbb{C}$. If so, the backend server computes all diagnosis tokens as $t_{i,1}\|\ldots\|t_{i,n} = PRG(s_i\|d)$, for every d in the infection period, and adds them to the list of diagnosis tokens **T**. Otherwise, a false-positive claim is easily detected. A nefarious actor has been caught by communicating with the cloud server and could be held accountable to the law.
   The privacy of diagnosed users can be enhanced by allowingevery user, including those who have not tested positive yet,to send an encrypted zero value and an "empty" certificate as $Enc(pk, 0\|\perp)$ to the cloud server in Step a. Then, at Step c, thebackend server decrypts ciphertexts and removes all zero values,which belong to non-diagnosed users. By doing so, the cloudserver will not know whether a message it receives has comefrom a diagnosed user. We only require a random subset of thenon-diagnosed users, as large as the set of diagnosed users, tobe involved.
5. **Model Compute and Release**: Finally, the backend server holds the uploaded tokens from infected users **T** while the ith user holds the received tokens T_i obtained from the "contact" phase. The ith user can make the query by invoking the unbalanced PSI-CA protocol described in Figure 3 with the backend server with the help of the cloud server. The user plays the role of the receiver $\mathcal{R}$ with the input of the token from other users during the collection phase, the backend server plays the role of the sender $\mathcal{S}$ with the input of tokens **T**. If there is a match, the ith user was in close proximity to a user that has been diagnosed with the disease.

5.3 | Resilient SecureCT with GPS

In this section, we describe a method to modify the SecureCT system with tokens containing GPS and timestamp information. The proposed decentralised BLE-based contact tracing scheme in Section 5.2 is used as a baseline. It becomes robust against a variety of attacks by utilising the GPS location and timestamp data. As mentioned in Section 3.7, linkage attacks exploit the fact that tokens broadcast by user devices can be captured and linked, to reveal the seed used to generate the tokens and thereby track a diagnosed user retroactively when the list of infected users' tokens is available. Also, replay and relay attack can be prevented.

Rather than only using tokens generated by a PRG with a seed, GPS and timestamp are also stored in a list when users are in contact. The app on the user's device continuously broadcasts anonymous tokens T_B that are rotated periodically. The app also listens for any tokens received T_R from other users within a valid range. In addition, the app logs the location loc and timestamp t of the user periodically. Suppose Alice and Bob are two users and they are in close proximity. Alice broadcasts T_{Alice} and Bob broadcasts T_{Bob}. They are at location loc at time t. Both Alice and Bob do the following:

1. Store $H(T_R + \text{loc} + t)$ in list L_R, where T_R is the received token, H is a public hash function and L_R is the list/table of tokens received. Alice stores $H(T_{\text{Bob}} + loc + t)$ and Bob stores $H(T_{\text{Alice}} + loc + t)$ in their respective L_R.
2. Store $H(T_B + \text{loc} + t)$ in list L_B, where T_B is the broadcast token, H is a hash function and L_B is the list/table of tokens

broadcast. Alice stores $H(T_{\text{Alice}} + \text{loc} + t)$ and Bob stores $H(T_{\text{Bob}} + \text{loc} + t)$ in their respective L_B.

Suppose Bob is positively diagnosed with the disease. He will follow the protocol to certificate the test result and upload all tokens in his list L_B to the backend server. Note that these tokens are the hash of broadcasted Bluetooth token, location, and timestamp combined so Bob has to upload the entire list rather than a single seed as mentioned in the SecureCT token collection phase. The list L_B may be prepared in two ways:

- Store $H(T_B + \text{loc} + t)$ in L_B with periodicity of location logs.
- Store {loc, t} entries in a separate table. Compute hash of T_B and {loc, t} entries and prepare L_B only when the user is positively diagnosed.

In the query phase, Alice can invoke the PSI-CA protocol to securely match tokens in her list L_R with the tokens stored by the backend server, and receive the number of her potential exposures.

The list of tokens can be maintained for a certain time period and then deleted, depending on the infectious period of the pathogen.

5.3.1 | Security discussion

The fact that location and timestamp features are incorporated along with the tokens makes it impossible for an adversary, whether the untrusted server or external, to capture and link broadcast Bluetooth tokens and attempt to track an infected user. Replay attacks by attempting to rebroadcast a captured Bluetooth token at another location to cause false exposure events are avoided as well since the location mismatch would result in an entirely different token that would not be uploaded to the backend server.

5.4 | Hotspots histogram computation

In addition, we proposed a protocol for secure histogram computation. This protocol determines geographical areas which are visited at least a threshold number of times by infected users. With the knowledge of such hotspots, users can avoid such areas to limit the spread through exposure prevention. The complete protocol specification is described in Figure 5. The protocol involves three parties—a client, cloud server, and backend server. The protocol involves each user's device maintaining a count vector V representing the number of times a user visited a location. The vector V associates each index with a predetermined location of interest. Additive secret sharing is used to distribute shares of V to the servers. The servers then aggregate the shares received from multiple users.

5.4.1 | Security discussion

From each server's view, it obtains a share of the count vector from each client. The share reveals nothing about the count vector, and hence, the server cannot learn an individual user's

PARAMETERS:

- Count vector V of size n
- A client $\mathcal{U}$, a backend server $\mathcal{S}$, and a cloud server $\mathcal{C}$
- Additive secret sharing scheme: If a is the original item, $[a]$ represents the set of shares. A two-out-of-two secret sharing scheme results in $[a] = \{a_1, a_2\}$ such that $a_1 + a_2 = a$

INPUTS:

- Client $\mathcal{U}$ has count vector V
- Backend server $\mathcal{S}$ and Cloud server $\mathcal{C}$ have no input.

PROTOCOL:

1. Each user device maintains count vector V from location logs. If user visits location loc associated with index i in V, then increment $V[i]$
2. If user is positively diagnosed, obtain shares of V, $[V] = \{V_1, V_2\}$ such that $V_1[i] + V_2[i] = V[i], \forall i \in [1, \ldots, n]$. Send V_1 to backend server $\mathcal{S}$ and V_2 to cloud server $\mathcal{C}$
3. Each server, $\mathcal{S}$ and $\mathcal{C}$, maintains an aggregate of shares received, $V_{\mathcal{S}}$ and $V_{\mathcal{C}}$ respectively: $V_{\mathcal{S}} = V_{\mathcal{S}} + V_1$ and $V_{\mathcal{C}} = V_{\mathcal{C}} + V_2$
4. On aggregating a threshold number of shares, cloud server sends $V_{\mathcal{C}}$ to backend server $\mathcal{S}$. Backend server recovers and outputs hotspots histogram, the correct aggregate, by recombining the aggregate of shares as $V_{Hist} = V_{\mathcal{S}} + V_{\mathcal{C}}$

OUTPUT:
Hotspots histogram V_{Hist}

FIGURE 5 Hotspots histogram construction

location trace or visits. The aggregates of shares received from multiple users are uniformly random. The recombination of aggregate shares results in the correct aggregate of count vectors, the intended result of the protocol, available to the servers and clients. Neither the client nor the servers can deduce anything more than the histogram of hotspots, as the aggregate does not reveal the count vector of an individual or subset of clients.

6 | IMPLEMENTATION AND EVALUATION

We implement our contact tracing framework SecureCT and estimate the cost of PSI-CA based on the cost of polynomial operations. In this section, we begin with discussing relevant implementation considerations, algorithm choices, and parameter values for the protocols described in Sections 5.1. We then evaluate our SecureCT and report its performance in Section 6.2.

6.1 | Implementation details

The SecureCT system and the enhancement with GPS described in Sections 5.2 and 5.3 mainly involve three parties—client app on the mobile device, the cloud server, and the backend server. The client functionality for the protocols is developed in Java as an Android mobile application. The cloud server and the backend server are implemented in Java using the Spring framework.

6.1.1 | SecureCT system

We implement our SecureCT for testing and evaluation. The GPS enhancement is used for the implementation. The implementation builds on the legacy DP3T [1] Android app and the server whose code is available on Github.[1] The location module is inspired with ideas from the Safe Paths approach [6], whose code is also available on Github.[2]

In the implementation of SecureCT, the client application involves the following major modules:

- Bluetooth server—to broadcast rotating proximity identifiers/tokens, register handshakes.
- Bluetooth client—to scan for nearby devices, receive rotating proximity identifiers/tokens, and store associated metadata like signal strength, duration of the handshake.
- Sync—to sync with backend server periodically to get exposure alerts, either through a download of infected users' tokens, or using PSI-CA protocol, as well as to upload tokens when positively diagnosed with the disease.
- Cryptography—to help with key generation, rotation, pseudorandom generation (PRG), encryption and hashing.
- Database—to assist in creating, reading, and deleting data in relevant tables including tokens broadcast, tokens received, infected users' tokens, and associated metadata.
- Location—module to periodically log users' location and get associated geohashes.

Similar to the Safe Paths approach [6], when a user is at a given set of coordinates, there is a radius r within which another user is said to be in close proximity. Points in the circle of radius r may lie in a neighbouring geohash. Hence, for a given location, the geohash of the exact coordinates, as well as a set of neighbouring geohashes covering the circle of proximity, are determined. This is done by considering the set of nearby points at a distance r along the cardinal and ordinal directions and determining the geohash of these points as well.

The broadcast Bluetooth tokens are rotated every 15 min. Location logs are recorded every 5 min. When storing the hash of the received token with geohash and timestamp, AES is chosen for its efficiency. The resulting hash has 128 bits. The app syncs with the backend server every 2 h to receive the tokens of infected users (legacy approach). The app can instead invoke the PSI-CA protocol to securely match tokens and receive exposure alerts. The app deletes tokens broadcast and receives those that are older than 14 days, which is the infectious period of COVID-19.

The backend server exposes API endpoints, handling user requests to fetch and upload infected users' tokens. It maintains a database to store tokens, where tokens older than 14 days are deleted.

6.1.2 | Server-aided PSI-CA implementation

Amongst different OKVS constructions [24], we choose the polynomial-based construction for SecureCT as it is easy to deploy. We integrate the polynomial-based OKVS to our server-aided PSI-CA protocol.

Both cloud and backend servers are implemented in Java using the Spring framework. These servers expose RESTful APIs to communicate and consume services. The Cloud server exposes a getMatches API, used by the client device to provide its list of received tokens and to get count of matches/exposures in return. The backend server exposes a getPolynomials API used by the cloud server to provide the CHT hash functions and get the polynomial coefficients for each bin of the hash tables. The tokens are 128 bits long. To support polynomial interpolation and evaluation for such large data, the Java library implementations for polynomial interpolation using the Lagrange's algorithm, and polynomial evaluation using the Neville's algorithm, are modified and extended to support the Java BigDecimal data type. The Cuckoo hashing implementation utilises two hash functions to insert the user uploaded tokens into bins such that there is at the most one item per bin. The same two hash

[1] http://github.com/dp-3T/dp3t-sdk-backend and http://github.com/DP-3T/dp3t-sdk-android
[2] https://github.com/Path-Check/safeplaces-dct-app

functions are used by the backend server to insert infected tokens into the simple hash table. AES is used as the hash function algorithm.

6.2 | Performance

The performance of BT plus GPS-based contact tracing when carried out using the legacy approach to determine matches by downloading infected tokens from the backend server is shown. The major costs involve storage of tokens, upload and download of tokens, and time taken for matching tokens to get exposure alerts.

Parameters: If a user generates a new token every 15 min and runs the proximity tracing process for approximately 18 h a day, then each user sends 72 distinct, 128-bit tokens per day. Assuming that the user meets people and receives the same number of tokens, then each user device has a total of $n = 1008 \approx 1000$ tokens over a 14-day period. With 1000 new cases per day, the backend server will receive $N \approx 1000 \times 1000 = 10^6$ new tokens per day.

Token storage: Storing both broadcast and received tokens for a 14-day period requires ≈31 KB on the client device. Assuming the server stores tokens for 15 days to accommodate offline clients, the total storage needed is ≈0.25 GB for 1000 daily new cases and ≈1.25 GB for 5000 daily new cases. If the client uses the legacy approach to download new infected tokens uploaded for that day and match with received tokens on the device, the client incurs the download and storage costs.

Testing platform configuration: The client application is installed as a Java Android app on a Google Pixel 3 device with Snapdragon 845 processor, 4 GB RAM, and 64 GB storage. The backend server and cloud server are deployed on an AWS m5.2×large instance with 8 vCPUs, 32 GB memory, and upto 10 Gbps network bandwidth.

The SecureCT system performance is compared with other works, including the Google Apple approach, DP3T [1], PACT [27], Epione [14], Catalic [9], and PSI-WCA protocol in [8] with respect to security and privacy guarantees, infrastructure requirements and client side cost in terms of computation and communication. The comparison is presented in Table 1. The method of evaluation followed is as explained in [9], and outlined briefly here. The Google Apple approach, DP3T, and PACT publicly release tokens of the diagnosed users, and hence they are all vulnerable to the identification of the diagnosed user. In the Google Apple approach, keys or seeds used to generate the tokens are publicly available, hence allowing an adversary to learn the travel route of an infected user. Similar to Epione and Catalic, SecureCT keeps the tokens private and hence secure against these vulnerabilities.

Each user has $k = 144$ new tokens per day and receives a total of $n = 2^{11}$ tokens approximately over the 14-day infection window, according to the Google Apple approach. Also, with $K = 2^{15} = 32{,}768$ new cases per day, $N = 2^{26}$ new tokens are added daily.

In the Google Apple approach, the client device downloads 14 K keys per day. Each key is 128 bits long, resulting in 7.34 MB of communication cost. The device needs to compute $14Kk = 66, 060, 288$ AES operations, taking 0.33 s to complete the contact tracing query on a phone with a 1.99 GHz processor.

The DP3T approach utilises a Cuckoo filter to share the tokens of the diagnosed users. They store a 56-bit fingerprint with each item. With $N = 2^{26}$ new diagnosed tokens, the client incurs a communication cost which is $2^{26} \times 56 = 469.76$ MB when downloading the Cuckoo filter. For computation, the device computes $2n$ AES hash functions, taking 0.02 ms.

For the PACT approach, the client device downloads $2^{26} \times 128$ (bits) = 1073.74 MB for $N = 2^{26}$ new diagnosis tokens. Its running time is considered negligible as it does not carry out any cryptographic operations.

In Epione, private set intersection using Private Information Retrieval is used, for which the client device incurs 1.79 MB and takes 394 ms. For Catalic, with 1 backend and 2

TABLE 1 Contact tracing system comparison: Comparison of SecureCT system with other contact tracing systems, in terms of privacy, infrastructure requirements, runtime and communication cost

	Linkage attack		System req.		Client	
Protocols	**Travel route**	**Infection status**	**#Rounds**	**#Servers**	**Runtime (ms)**	**Comm.Cost (MB)**
Google Apple	Yes	Yes	1/2	1	331.96	7.34
DP3T	No	Yes	1/2	1	0.02	469.76
PACT	No	Yes	1/2	1	Neg	1073.74
Epione	No	No	2	2	394.01	1.27
Catalic	No	No	1	3	0.86	0.095
PSI-WCA	No	No	1	2	0.064	2.048
SecureCT	No	No	1	2	208	0.032

Note: #Rounds is the number of interaction rounds between the client and server. Travel route refers to learning the travel route of the diagnosed user, while infection status refers to the identification of the diagnosed user. Each user has 2^{11} tokens. neg refers to negligible cost.

TABLE 2 Running time for interpolating all polynomials on the back-end server

# Bins (β)	# Server tokens (N)	Time (s)
40,000	500,000	19.2
80,000	500,000	14.2
80,000	1,000,000	37.7
100,000	1,000,000	34.7

Note: Performance of the polynomial interpolation is given by having different amount of bins (β) and different amount of tokens (N).

cloud servers, each running with a single thread, the protocol requires 0.86 ms 96 KB.

For the contact tracing system built upon the function secret sharing the PSI-WCA protocol in [8], the computation on the users' side is from generation n secret sharing point functions of the cost $n\lambda$ AES where λ is the security parameter. The communication cost is $n\lambda|\text{AES}|$. The estimation runtime and communication cost are 0.064 ms and 2.048 MB as shown in Table 1 with $\lambda = 128$.

For our SecureCT system using one cloud server and backend server, the client device has 1 round of interaction with the cloud server and backend server and is required to download n results from the cloud server. Thus the communication cost is 0.032 MB. The client device computes n AES hash functions to encrypt the tokens and generates $\beta = n$ secret values, where β is the number of bins, taking a total of 208 ms.

For the server performance, the major cost for the servers is the polynomial interpolation. We implement the polynomial-based OKVS structure in Java to estimate the performance of SecureCT if using our PSI-CA protocol. Table 2 summarises the time taken by the backend server, deployed on AWS m5.2 × large instance, to generate the polynomials for all bins, which is the major computation involved in the PSI-CA protocol. The polynomial interpolation for separate bins can be parallelised on more threads, resulting in a speedup for the performance. The code has been parallelised to run the polynomial interpolation on 7 threads. With 2 hash functions, the number of tokens in the hash table is doubled. The number of tokens per bin varies as per the hash function distribution. The performance is compromised because of the programming language, other languages like C++ may give a much faster result. A very similar implementation in C++ is given in Table 2 of [9]. We use the polynomial-based OKVS as the pack & unpack algorithm. The concrete running performance for the OKVS can be found in Appendix A of [24].

Overall, our SecureCT shows the best communication cost for clients among all the other contact tracing systems while still having a reasonable runtime on the clients' device. As for the server, our protocol has a similar performance with the state-of-the-art Catalic [9] system while their work requires at least two non-colluding cloud servers which is a much stronger system requirement.

ACKNOWLEDGEMENT

The authors are supported by the grant from the National Science Foundation #2031799, #2101052, and #2115075 and Google AI. Part of this work was done while the second author worked at ASU.

CONFLICT OF INTEREST

The authors declare that they have no conflicts of interest.

DATA AVAILABILITY STATEMENT

Research data are not shared.

ORCID

Jiahui Gao https://orcid.org/0000-0003-3821-2263

REFERENCES

1. Troncoso, C., et al.: Decentralized privacy-preserving proximity tracing. arXiv preprint arXiv:2005.12273 (2020)
2. TCN Coalition: Temporary contact numbers protocol. Retrieved September. 1, 2020 (2020)
3. Cho, H., Ippolito, D., Yun, W.Y.: Contact tracing mobile apps for covid-19: privacy considerations and related trade-offs. arXiv preprint arXiv:2003.11511 (2020)
4. Gvili, Y.: Security analysis of the covid-19 contact tracing specifications by Apple Inc. and Google Inc. IACR Cryptol. ePrint Arch. 2020, 428 (2020)
5. Pietrzak, K.: Delayed authentication: preventing replay and relay attacks in private contact tracing. IACR Cryptol. ePrint Arch. 2020, 418 (2020)
6. Berke, A., et al.: Assessing disease exposure risk with location histories and protecting privacy: a cryptographic approach in response to a global pandemic. arXiv preprint arXiv:2003.14412 (2020)
7. Otto, S.: BLE contact tracing sniffer PoC. https://github.com/oseiskar/corona-sniffer
8. Dittmer, S., et al.: Function secret sharing for PSI-CA: with applications to private contact tracing. arXiv preprint arXiv:2012.13053 (2020)
9. Duong, T., Phan, D.H., Ni, T.: Catalic: delegated PSI cardinality with applications to contact tracing. In: Moriai, S., Wang, H. (eds.) ASIACRYPT 2020, Part III, Volume 12493 of LNCS, pp. 870–899. Springer, Heidelberg (2020)
10. Vaudenay, S.: Analysis of DP3T. Cryptology ePrint Archive, Report 2020/399 (2020). https://eprint.iacr.org/2020/399
11. Morton, G.M.: A computer oriented geodetic data base and a new technique in file sequencing. IBM Ltd, Ottawa, Ontario, Canada (1966)
12. Trivedi, A., et al.: WiFitrace: network-based contact tracing for infectious diseases using passive WiFi sensing. Proc. ACM Interact. Mob. Wearable Ubiquitous Technol. 5(1), 1–26 (2021). https://doi.org/10.1145/3448084
13. Raskar, R., et al.: Apps gone rogue: maintaining personal privacy in an epidemic. arXiv preprint arXiv:2003.08567 (2020)
14. Ni, T., et al.: Epione: lightweight contact tracing with strong privacy. arXiv preprint arXiv:2004.13293 (2020)
15. Chase, M., Miao, P.: Private set intersection in the internet setting from lightweight oblivious PRF. In: Micciancio, D., Ristenpart, T. (eds.) CRYPTO 2020, Part III, Volume 12172 of LNCS, pp. 34–63. Springer, Heidelberg (2020)
16. Rindal, P., Schoppmann, P.: VOLE-PSI: fast OPRF and circuit-PSI from vector-OLE. In: Canteaut, A., Standaert, F.-X. (eds.) Advances in Cryptology – EUROCRYPT 2021, pp. 901–930. Springer International Publishing, Cham (2021)
17. Pinkas, B., et al.: SpOT-light: lightweight private set intersection from sparse OT extension. In: Boldyreva, A., Micciancio, D. (eds.) CRYPTO 2019, Part III, Volume 11694 of LNCS, pp. 401–431. Springer, Heidelberg (2019)

18. Huberman, B.A., Franklin, M., Hogg, T.: Enhancing privacy and trust in electronic communities. In: Proceedings of the 1st ACM Conference on Electronic Commerce, EC '99, pp. 78–86. ACM (1999)
19. Chor, B., Gilboa, N., Naor, M.: Private Information Retrieval by Keywords. Cryptology ePrint Archive, Report 1998/003 (1998). https://eprint.iacr.org/1998/003
20. Demmler, D., et al.: PIR-PSI: scaling private contact discovery. Proc. Priv. Enh. Technol. 2018(4), 159–178 (2018). https://doi.org/10.1515/popets-2018-0037
21. Rabin, M.O.: How to exchange secrets with oblivious transfer. Cryptology ePrint Archive, Report 2005/187 (2005). https://eprint.iacr.org/2005/187
22. Boyle, E., Gilboa, N., Ishai Y.: Function secret sharing. In: Oswald, E., Fischlin, M. (eds.) EUROCRYPT 2015, Part II, Volume 9057 of LNCS, pp. 337–367. Springer, Heidelberg (2015)
23. Boyle, E., Gilboa, N., Ishai Y.: Function secret sharing: improvements and extensions. In: Weippl, E.R., et al. (eds.) ACM CCS 2016, pp. 1292–1303. ACM Press (2016)
24. Garimella, G., et al.: Oblivious key-value stores and amplification for private set intersection. In: Malkin, T., Peikert, C. (eds.) Advances in Cryptology – CRYPTO 2021, pp. 395–425. Springer International Publishing, Cham (2021)
25. Kolesnikov, V., et al.: Efficient batched oblivious PRF with applications to private set intersection. In: Weippl, E.R., et al. (eds.) ACM CCS 2016, pp. 818–829. ACM Press (2016)
26. Pinkas, B., et al.: Phasing: private set intersection using permutation-based hashing. In: Jung, J., Holz, T. (eds.) USENIX Security 2015, pp. 515–530. USENIX Association (2015)
27. Chan, J., et al.: Pact: privacy sensitive protocols and mechanisms for mobile contact tracing. arXiv preprint arXiv:2004.03544 (2020)

How to cite this article: Gao, J., Surana, C., Trieu, N.: Secure contact tracing platform from simplest private set intersection cardinality. IET Inf. Secur. 16(5), 346–361 (2022). https://doi.org/10.1049/ise2.12070