Epione: Lightweight Contact Tracing with Strong Privacy

Ni Trieu[1], Kareem Shehata, Prateek Saxena[2], Reza Shokri[2], and Dawn Song[1,3]
[1]UC Berkeley, [2]National University of Singapore, [3]Oasis Labs

Abstract

Contact tracing is an essential tool in containing infectious diseases such as COVID-19. Many countries and research groups have launched or announced mobile apps to facilitate contact tracing by recording contacts between users with some privacy considerations. Most of the focus has been on using random tokens, which are exchanged during encounters and stored locally on users' phones. Prior systems allow users to search over released tokens in order to learn if they have recently been in the proximity of a user that has since been diagnosed with the disease. However, prior approaches do not provide end-to-end privacy in the collection and querying of tokens. In particular, these approaches are vulnerable to either linkage attacks by users using token metadata, linkage attacks by the server, or false reporting by users.

In this work, we introduce Epione, a lightweight system for contact tracing with strong privacy protections. Epione alerts users directly if any of their contacts have been diagnosed with the disease, protects the privacy of users' contacts from both central services and users, and provides protection against false reporting. As a key building block, we present a new cryptographic tool for secure two-party private set intersection cardinality (PSI-CA), which allows two parties, each holding a set of items, to learn the intersection size of their sets without revealing the intersection items. We specifically tailor it to the case of large-scale contact tracing where clients have small input sets and the server's database of tokens is much larger.

1 Introduction

Contact tracing is an important method to curtail the spread of infectious diseases. The goal of contact tracing is to identify individuals that might have come into contact with a person that has been diagnosed with the disease, so they can be isolated and tested.

In the ongoing COVID-19 pandemic, contact tracing has been facilitated by mobile apps that detect nearby mobile phones using Bluetooth, and several countries / organizations have developed such apps. Such large-scale collection of personal contact information is a significant concern for privacy [1, 2, 3].

The main purpose of contact tracing applications—recording the fact that two or more individuals were near each other at a certain moment of time—seems to be at odds with their privacy. The app must record information about the individual's personal contacts and should be able to reveal this information (possibly, on demand) to some authorities.

Multiple ways have been proposed to protect user contact data, offering different privacy guarantees and coming at different implementation costs. For instance, in the recently released BlueTrace protocol used by the

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

Singapore Government [5], users are guaranteed privacy from each other, but this model places complete trust in certain operating entities for protecting user information.

We consider a model where governments (or operators) do not store any such sensitive user information. Not only are such databases lucrative targets for cyber-attackers, in many jurisdictions the collection of such information may raise public concerns or even conflict with privacy regulations. This is important not just for user privacy, but also because contact tracing is expected to be effective only when participation is high (e.g. 60% or more of the population [4]). Thus the overall success of the app could be limited if users are reluctant to use a contact tracing app due to privacy concerns.

In our model, the health authorities maintain a database of tokens corresponding to users which have been diagnosed with the disease. The tracing app periodically checks an untrusted server to determine if the user is potentially at risk. This is done in such a way that the server cannot deduce any information about the user which is not implied by the desired functionality. The users also learn no information beyond whether they may have been exposed to the disease.

Our model can also be contrasted to several other decentralized mobile contact tracing system/protocols, which we analyze in the full version of the paper [7]. As we see through our analysis, existing proposals or launched systems are vulnerable to one or more of the following privacy attacks:

(1) *Infection status / exposure source by users*: If tokens of users diagnosed positive are publicly released, Alice can determine which publicly-posted tokens match the log on her phone. This could reveal the time, for example, when Alice and the user diagnosed positive with the disease (Bob) were in close proximity, enabling her to identify Bob. Such identification is undesirable as people have been reported to harass individuals suspected to be the source of exposure to the disease [8], leading to the so-called "vigilante" problem.

(2) *Infection status by server*: If the server can determine which users have been diagnosed with the disease, this leaks the infection status of users to the server operator. This may not be a concern in jurisdictions where the server is operated by the health authority which already knows this information. However, in jurisdictions where the server is operated by another party that does not or should not have this information, this form of linkage can be a serious privacy threat.

(3) *Social graph exposure and user tracking*: If a central database is used to collect both sent and received tokens as in [9], or it is possible to infer the source of a sent token as in the case of [10], then the operator of this server is able to deduce all of the social connections of a user that is reported positive for the disease, including when and for how long each contact was made. This co-location information can also exacerbate the risk of users' location tracking [4, 3].

(4) *False-positive claims by users*: A user may claim to have been diagnosed with the disease when in reality, they are not. This would spread false information and panic other users, and reduce trust in the system.

Table 2 provides a brief comparison of different contact tracing systems with respect to security/privacy properties, required computational infrastructure and client's communication cost, all of which are important for a wide-scale real-world contact tracing. Details of the systems compared is discussed in the full version [7].

Our Contributions. In this work, we introduce Epione, a new system for decentralized contact tracing with stronger privacy protections than the existing systems. As a key primitive enabling Epione, we introduce a new private set intersection cardinality or PSI-CA, which is used to check how many tokens held by a user (client) match the tokens in a set stored on a server, without the user revealing their tokens. More formally, PSI-CA allows two parties, each holding a private set of tokens, to learn the size of the intersection between their sets without revealing any additional information. Our PSI-CA primitive is designed to be efficient for a large

System	System Req.		Privacy Protection Against				Client
	Trusted Server	#	Infection Status		Social Graph	False-positive User	Comm. Cost
			By User	By Server			
TraceTogether [5]	Yes	1	Yes	No	No	Yes	$O(n)$
Baseline*	**No**	1	No	No	Most	Some	$O(N)$
Private Messaging [2]		3	No	Yes	Yes	No	
Epione		2	**Yes**	**Yes**	**Yes**	**Yes**	$O(n\log(N))$

Table 2: Comparison of contact tracing systems with respect to security, privacy, required computational infrastructure, and client communication cost. **Baseline** systems include Private Kit[13], Covid-watch [9], CEN [14], DP-3 [15], and PACT's baseline system [12]. Some of these systems provide a limited level of false-positive claim protection with an additional server (or healthcare provider), and most provide protection from social graph discovery. N is the total number of contact tokens from users diagnosed positive with the disease, n is the number of contact tokens recorded by an average user that need to be checked for disease exposure (Note that $\frac{N}{n}$ is typically the number of new positive diagnoses per day, thus $n << N$).

server-side database and a small client-side database, as is the case for contact tracing applications. Our new PSI-CA construction allows us to meet our privacy goals. With several other optimizations in our system design, we show that PSI-CA can make privacy-preserving contact tracing *practical*.

1.1 System Overview

Figure 1 shows an overview of the Epione system. Users of Epione want to be notified if any of the people they have been in contact with are later diagnosed with the disease. They do *not* want to reveal to other users their identity, reveal whether they have been diagnosed positive, be tracked over time, or reveal their contacts to any other organization.

We use a short-range network (such as Bluetooth) to detect when two users are within close range and exchange a randomly generated "contact token". All of the sent and received contact tokens are stored securely on the user's phone in the "sent token list" and "received token list", respectively. The received token list never leaves the user's phone in a form that can be used by anyone else, and the sent token list is only revealed to a healthcare provider on a positive diagnosis and with the user's consent. In Section 5, we explain in detail how to generate and store the tokens.

In Epione, we assume that there is an untrusted service provider, which we call the Epione Server, which can collect the transmitted contact tokens from all users tested positive with the disease. The Epione server allows users to check whether they have received a token from a user who has since been diagnosed with the disease, without revealing to the server their tokens (and thus their contacts) and without the server revealing any information to the user about the tokens of users diagnosed positive beyond the count of contact tokens in common. We use secure computation techniques, particularly PSI-CA, for private matching. This prevents the Epione server from inferring linkages between users, as well as preventing users from inferring the diagnosis status of other users, or the source of any exposure to the disease.

It is assumed that a healthcare provider (such as a hospital) is aware of the identity of the user whom it diagnoses as having the disease. Thus, exposing the identity of the user diagnosed positive to the provider is not considered as a threat. It is also assumed that the healthcare provider keeps a local database of positively diagnosed users to be able to verify if a user was legitimately diagnosed positive. The healthcare provider collects (with the user's consent) the list of "sent tokens" from a positively diagnosed user's app and sends it to the Epione server, which the latter adds to a database of contact tokens from such users.

Note that in this model the server does not know the identity of the user diagnosed positive. It is not hard to imagine collusion between the healthcare database and the backend server for Epione, say by a state actor or attacker within the healthcare provider. Even then, the sent tokens are not useful for identifying any contacts

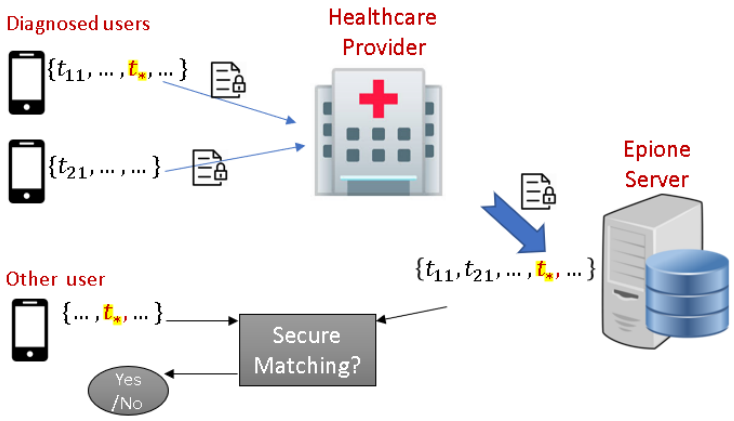

Figure 1: Overview of our Epione system. When a person is diagnosed with the disease, the healthcare provider collects their sent tokens and forwards them to the Epione server encrypted under the server's public key (actually, the PRG seed is collected to reduce communication costs). The Epione server decrypts the received ciphertexts from the healthcare provider and obtains the transmitted tokens of all patients diagnosed positive. Next, each user's app uses PSI-CA to compare their set of received tokens with the set of sent tokens stored on the Epione server. If the intersection size is more than zero, then the user is alerted that they may have been exposed to the disease.

or any other private information. Since tokens are randomly generated, the attacker would need to know which users received those tokens to re-identify them. Section 5 shows that the Epione server never learns the received tokens of any user and thus linkage is not possible.

2 Related Work

We begin by discussing previous approaches to contact tracing, and then current approaches to secure computation and private set intersection, which form the basis for our own PSI-CA used in Epione.

2.1 Contact Tracing Approaches

Due to the rapid spread of the COVID-19 pandemic and the importance of contact tracing, many research groups have been developing tools to improve contact tracing. Most schemes either (1) rely on and expose data to a trusted third-party, such as TraceTogether [5], or (2) uses a decentralized/public list approach such as COVID-Watch [9], PACT [12], or Google/Apple [11] that allows users to infer linkages such as exposure sources.

In this work, we focus on the latter approach. Covid-watch [9], Private Kit [13][1], PACT's baseline design [12], and Google/Apple [11] are all variations on this design. Some use pseudo-random number generators, and upload seeds for the sent token lists to reduce communication and storage costs at the expense of greater cost for comparisons. All of these designs are susceptible to linkage attack by either users, the server, or both. Some offer protection against false-positive claims. We refer the reader to the full verion of this paper [7] for additional discussion of decentralized contact tracing.

[1]PrivateKit claims that in V3 they will introduce strong privacy protections, but as of writing this paper the protocol to do so has not been announced.

In some of the above systems, each phone has to compare the publicly posted contact tokens against their own history, which requires to them download all public tokens. This requires significant bandwidth and places a burden on mobile devices.

2.2 Secure Computation and Private Set Intersection

Private set intersection (PSI) refers to a cryptographic protocol that allows two parties holding private datasets to compute the intersection of these sets without either party learning any additional information about the other's dataset. PSI has been motivated by many privacy-sensitive real-world applications. It does, however, reveal the intersection elements to at least one party. In many scenarios (such as contact tracing) it is preferable to compute some function of the intersection without revealing the elements in it, such as whether intersection size is more than a given threshold. Limited work has focused on this so-called f-PSI problem. The Diffie-Hellman homomorphic encryption approach in [16] is preferable in many real-world f-PSI applications[2], due to its more reasonable communication complexity.

3 Problem Statement and Security Goal

3.1 Problem Definition

We define the problem of contact tracing based on token exchange as follows. Various clients communicate with each other and with a contact tracing service. The service is provided by one or more servers. The overall system consists of the following procedures:

- $\texttt{Generate}(\kappa) \rightarrow t$: Client uses the `Generate` function to generate contact tokens, t, to be exchanged with other users. The function takes a security parameter κ as input.
- $\texttt{Exchange}(t_a) \rightarrow t_b$: The client (client A) uses the `Exchange` function to exchange tokens with another user (client B). Client A sends token t_a to B, and receives t_b from client B. Client A then stores t_b in the "received tokens list" ($\mathbf{T}_R$), and client B stores t_a in their "received tokens list".
- $\texttt{Query}(T_R, S) \rightarrow a$: With a set $\mathbf{T}_R$ of received tokens from `Exchange`, the client uses the `Query` function to query the server S and get an answer indicating how many of their tokens came from users currently diagnosed positive for the disease.

3.2 Security Goal

We consider a set of parties who have agreed upon a single function f to compute (such as contact tracing) and have also consented to give f's final result to some particular party. At the end of the computation, nothing is revealed by the computational process except the final output. In real-world execution, the parties often execute the protocol in the presence of an adversary $\mathcal{A}$ who corrupts a subset of the parties. In the ideal execution, the parties interact with a trusted party that evaluates the function f in the presence of a simulator Sim that corrupts the same subset of parties.

For simplicity, we assume there is an authenticated secure channel between each pair of clients, and client-server pair (e.g., with TLS). In this work, we consider a model with non-colluding servers. A desirable contract tracing system would make an honest user's actions perfectly indistinguishable from actions of all other honest users as well as servers. Thus, an ideal security system property would guarantee that executing the system in the real model is equivalent to executing this system in an ideal model with a trusted party. In particular,

[2]Google Security Blog, June 19, 2019 "Helping organizations do more without collecting more data"

Epione provably provides all of the functions of contact tracing while protecting against the linkage attacks and false-positive claims described in Section 1.

4 Preliminaries

In this work, the computational and statistical security parameters are denoted by κ, λ, respectively. For $n \in \mathbb{N}$, we write $[n]$ to denote the set of integers $\{1, \ldots, n\}$.

Definition 1: [17] Let $\mathcal{G}(\kappa)$ be a group family parameterized by security parameter λ. For every probabilistic adversary $\mathcal{A}$ run in polynomial time in λ, we define the advantage of $\mathcal{A}$ to be $|\Pr[\mathcal{A}(g, g^a, g^b, g^{ab}) = 1] - \Pr[\mathcal{A}(g, g^a, g^b, g^c) = 1]|$, where the probability is over a random choice G from $\mathcal{G}(\lambda)$, random generator g of G , random $a, b, c \in [|G|]$ and the randomness of $\mathcal{A}$. We say that the Decisional Diffie–Hellman assumption holds for G if for every such $\mathcal{A}$, there exists a negligible function ϵ such that the advantage of $\mathcal{A}$ is bounded by $\epsilon(\lambda)$.

Definition 2: [18] A pseudorandom number generator (PRG) is a function that, once initialized with some random value (called the seed), outputs a sequence that appears random, in the sense that an observer who does not know the value of the seed cannot distinguish the output from that of a (true) random bit generator.

Private Information Retrieval. Private Information Retrieval (PIR) allows a client to query information from one or multiple servers in a such way that the servers do not know which information the client requested. Recent PIR [19, 20, 21] reduces communication cost to logarithmic in the database size. In PIR, the server(s) hold a database DB of N strings, and the client wishes to read item $DB[i]$ without revealing i.

Chor, et al. [22] define a variant of PIR called keyword PIR, in which the client has an item x, the server has a set S, and the client learns whether $x \in S$. In this paper, we are interested in Keyword PIR based on both 1-server PIR [20, 21] and 2-server PIR [23, 24] with different trade-offs.

Private Set Intersection Cardinality. Private set intersection cardinality (PSI-CA) is a two-party protocol that allows one party to learn the intersection size of their private sets without revealing any additional information.

5 Our Epione System

We now present the Epione system in detail, the construction of which closely follows the high-level overview presented in Section 1.1. Recall that Epione aims to alert any users who have, within the infection window, come into contact with another user who has been diagnosed positive with an infectious disease.

Epione's design combines several different cryptographic primitives. We refer the reader to Section 4 and Section 6 for more details on the cryptographic gadgets used here. The Epione system consists of four phases as follows.

Agreement and Setup Phase. The Epione server takes a security parameter λ as input, outputs a public-private key pair (pk, sk), and shares the public key with every user. Each user/client u_i generates a random PRG seed s_i which it uses to generate contact tokens in the next phase. As long as the server's configuration does not change, this phase does not need to be re-run. Whenever a new user registers with Epione, they only need to generate their own PRG seed, and the server shares the public key with the new user.

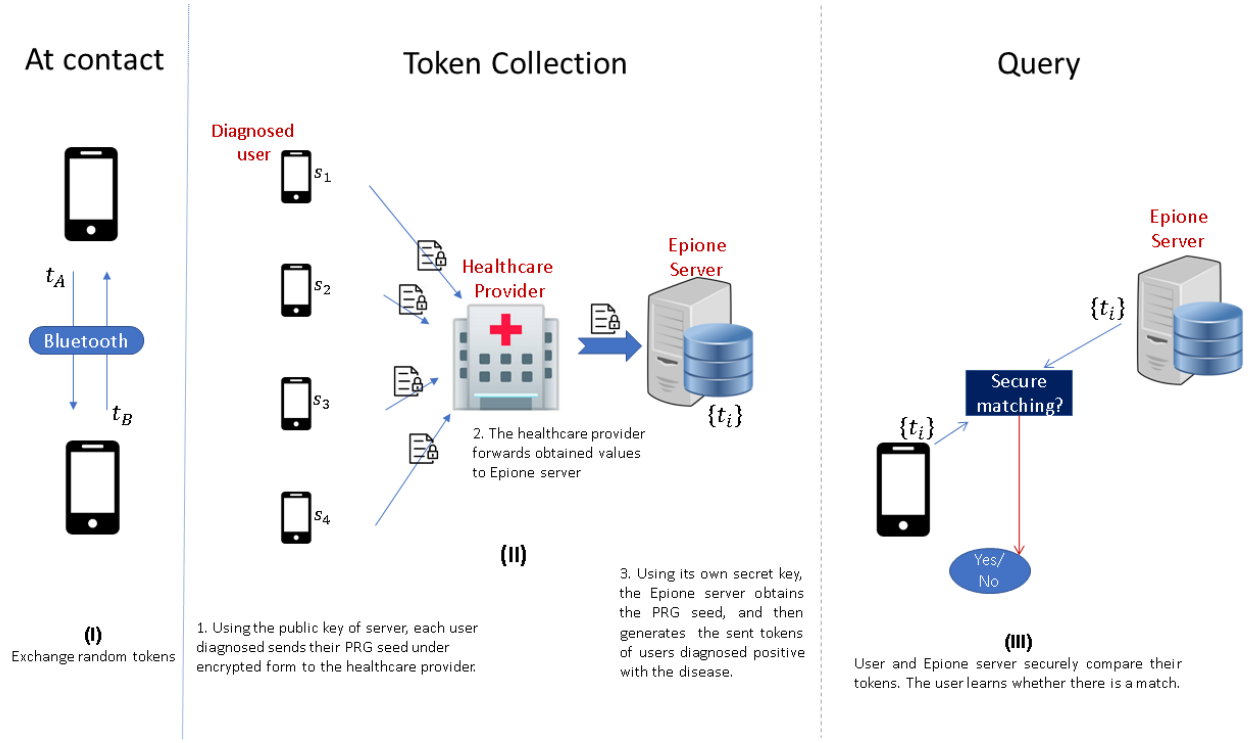

Figure 2: Epione System Design without Agreement and Setup Phase. (I) Tokens are exchanged when two users are in close proximity. (II) When a user is diagnosed with the disease, the user encrypts their PRG seed using the public key of the Epione server, and gives the encrypted value to the healthcare provider, who then transmits it to the Epione server. Using its private key, the Epione server decrypts the received ciphertexts and obtains the PRG seeds of diagnosed users. The Epione server generates the sent tokens of users diagnosed positive using the PRG. (III) Each user invokes a secure matching algorithm with the Epione server, where the user's input is their received tokens and the server's input is the database of tokens from users diagnosed with the disease. The user learns only whether (or how many) tokens there are in common between the two sets, while the Epione server learns nothing.

Token Generation. Similar to most recent contract tracing systems [2, 9, 14, 12], we use Bluetooth to exchange contact tokens whenever two users are in close proximity. The $\texttt{Generate}(s_u, d_i, j) \to t_{u,i,j}$ function is used to generate tokens of κ bits each to be sent by user u on day i and timeslot j. The precise details of the token generation are left as an implementation detail, so long as the following criteria are met:

- Tokens are indistinguishable from random by anyone not in possession of the user's seed s_u. In other words, the `Generate` function acts as a PRG as defined in Section 4.
- Tokens can be deterministically generated for the given day d_i, and time slot j using a secret seed, s_u, such that when a user gives their seed to the Epione server, the server is able to regenerate the tokens sent by the user.
- All users and the Epione server agree on the method used to generate tokens, the time intervals, and day numbering.

Contact. As illustrated in Figure 2 part I, when two users, say Alice and Bob, enter within close proximity, Epione detects this condition with a short range network such as Bluetooth, and then uses that network to

exchange tokens using the function `Exchange`. Alice generates token $t_a \leftarrow \texttt{Generate}(s_a, d_i, j)$, where s_a is Alice's private seed, d_i is the current day, and j is the current time slot. Similarly Bob generates token $t_b \leftarrow \texttt{Generate}(s_b, d_i, j)$. Alice sends t_a to Bob, and Bob send t_b to Alice.

Alice then adds the token received from Bob, t_b, to her set of received tokens, $\mathbf{T}_{R,A}$, and Bob adds t_a to $\mathbf{T}_{R,B}$. We use $\mathbf{T}_{S,A}$ to represent the set of tokens Alice has sent to other users (which includes t_a), though Alice does not actually store such a list since it can be regenerated at any time from her private seed. Alice and Bob discard received tokens that are older than the infection window (e.g. 14 days for COVID-19).

Positive Diagnosis and Token Collection. When a user (u_i in general) is diagnosed with the disease, the user encrypts their PRG seed using the public key of the Epione server and gives that to the healthcare provider (provided the user consents to this, of course). The healthcare provider gathers the seeds from several users diagnosed positive, shuffles them, and transmits the set of seeds over a secure channel to the Epione server. Using its private key, the Epione server decrypts the received values to obtain the secret PRG seeds. The Epione server can then generate all of the tokens for the infection window sent by users diagnosed positive with the disease, $\widehat{\mathbf{T}}_S$. The token collection process is shown in part II of Figure 2.

Two servers are used at this phase to prevent any one server from knowing both the diagnosis status of a user and their sent tokens. This is useful in the case that the Epione server is operated by an untrusted party, such as a commercial provider, that should not have access to sensitive information such as a user's diagnosis. If such protection is not needed, for example if the Epione server is operated by a health authority that already has access to the infection status of users and can be trusted not to try to discern a user's diagnosis status from the token collection process, then both services can be provided by the same server.

Alternatively, the healthcare provider could provide a token to the user that the user then provides the Epione server when they upload their tokens to prove that they have a legitimate positive diagnosis. This would allow the Epione server to verify that the user's claim is legitimate, but does not protect the user from the server linking them to a positive diagnosis.

Query. Recall from the contact phase that each user u_i keeps a list of tokens received from other users they have been in contact with within the infection window, $\mathbf{T}_{R,u_i}$. The query phase aims to securely compare the user's received contact tokens $\mathbf{T}_{R,u_i}$ with the Epione server's set of tokens sent by users diagnosed positive with the disease, $\widehat{\mathbf{T}}_S$. If there are any tokens in common, then user u_i has come into contact with an individual diagnosed positive within the infection window, and should be notified that they are at risk of having contracted the disease. This process is illustrated in part III of Figure 2.

The comparison of tokens is done by calling the `Query` function, which we implement using PSI-CA. We describe PSI-CA in detail in Section 6. Note that revealing the intersection size is acceptable in the contact tracing application we consider, however, it is possible hide the intersection size as we discuss in the full version [7].

6 Cryptographic Gadgets

This section provides more detail on the cryptographic tools we use to implement Epione, with a specific emphasis on our PSI-CA design and PIR. Extension to those tools is discussed in the full version [7].

6.1 PSI cardinality (PSI-CA) for asymmetric set sizes

6.1.1 Our technique

We start with a private set intersection (PSI) in the semi-honest setting, where two parties want to learn the intersection of their private set, and nothing else. The earliest protocols for PSI were based on the Diffie-Hellman

(DH) assumption in cyclic groups. Currently, DH-based PSI protocols [25] are still preferable in many real-world applications due to their low communication cost.

DH-based PSI. Assume that the server has input $X = \{x_1, \ldots, x_N\}$ and client has input $Y = \{y_1, \ldots, y_n\}$. Given a random oracle $H : \{0,1\}^* \to G$, and a cyclic group G in which the DDH assumption holds, the basic DH-based PSI protocol is shown in Figure 3. Intuitively, the client sends $\{H(y_i)^r\}_{y_i \in Y}$ for some random, secret exponent r. The server raises each of these values to the k power, and the client can then raise these results to the $1/r$ power to obtain $\{H(y_i)^k\}_{y_i \in Y}$ as desired.

PARAMETERS: cyclic group G of order p; random oracle H.
INPUTS: Server has input $X = \{x_1, \ldots, x_N\}$; client has input $Y = \{y_1, \ldots, y_n\}$.
PROTOCOL:

1. Client chooses $r \leftarrow \mathbb{Z}_p$, for each $y_i \in Y$ sends $m_i = H(y_i)^r$ to the server.
2. Server chooses $k \leftarrow \mathbb{Z}_p$ and for each $i \in [n]$, sends $m'_i = m_i^k$ to the receiver in a randomly permuted order.
3. For each $i \in [n]$, the client computes $v_i = (m'_i)^{1/r}$.
4. For each $x_i \in X$, the server computes $u_i = H(x_i)^k$ and sends $U = \{u_i \mid x_i \in X\}$ (randomly permuted) to the client.
5. [PSI-CA output] The client outputs $|\{i \in [n] \mid v_i \in U\}|$.

Figure 3: DH-based PSI protocol and extension to PSI-CA with changes highlighted.

From DH-based PSI to PSI cardinality (PSI-CA). If the client uses the same r for every item, it is possible to extend the basic PSI algorithm to compute functions such as intersection set size (cardinality) without revealing the intersection items by having the server shuffle the items. This observation was suggested by [25] and recently was incorporated into private intersection sum [16], which allows two parties to compute the sum of payloads associated with the intersection set of two private datasets, without revealing any additional information. Clearly, PSI-CA is a special case of private intersection sum, where the payload is constant and equal to 1.

Figure 3 also shows the extension to PSI-CA with the highlighted changes. The key idea to transform PSI into PSI-CA is that instead of sending m'_i in step 2 of Figure 3 in order, the server shuffles the set in a randomly permuted order. Shuffling means the client can count how many items are in the intersection (PSI-CA) by checking whether $v_i \in U$, but learns nothing about which specific item was in common (e.g. which v_i corresponds to the item y_j). Thus, the intersection set is not revealed.

From PSI-CA to PSI-CA for asymmetric sets. In many applications, including contact tracing, the two parties (client and server) have sets of extremely different sizes. A typical client has less than 500 new tokens per day, while the server may have millions of tokens in its input set. In PSI, most work is optimized for the case where two parties have sets of similar size, and as such their communication and computation costs scale with the size of the larger set. For contact tracing, it is crucial that the client's effort (especially communication cost) be sub-linear in the server's set size. More practically, we aim for communication of at most a few megabytes in a setting where the client is a mobile device.

We observe that the last two steps of Figure 3 are similar to the function performed by keyword PIR, which is communication-efficient in the conventional client-server setting. Keyword PIR allows clients to check whether

their item is contained in a set held by a server, without revealing the actual item to the server. Therefore, step 4 and 5 of Figure 3 can be replaced by keyword PIR. Concretely, after step 3, the client has an input set $V = \{v_1, \ldots, v_n\}$ and the server has input set $U = \{u_1, \ldots, u_N\}$. The client sends a multi-query keyword PIR request with all of the elements in V to be queried against U on the server. From the PIR response, the client can count the number of $v_i \in U$ to find the set size, without revealing to the server the actual values in V and without the client learning any more information about U.

6.1.2 Protocol

Our semi-honest PSI-CA protocol is presented in Figure 4, following closely the description in the previous subsection. The client runs keyword PIR searches for each $v_{i\in[n]}$ in a set U held by the server. For communication and computation efficiency, the values of both u_i and v_i can be truncated, and the protocol is still correct as long as there are no spurious collisions. We can limit the probability of such a collision to $2^{-\lambda}$ by truncating to length $\lambda + log(N)$ bits. In Figure 4, we use a truncation function $\tau(z)$ which takes z as input and returns the most significant $\lambda + log(N)$ bits of z.

PARAMETERS: cyclic group G of order p; random oracle H, Multi-query Keyword PIR primitive (Section 4), a truncate function $\tau(z)$ takes z as input and returns first $\lambda + log(N)$ bits of z.
INPUTS: Server 1 has input $X = \{x_1, \ldots, x_N\}$; client has input $Y = \{y_1, \ldots, y_n\}$; Server 2 has no input
PROTOCOL:

1. Server 1 chooses $k \leftarrow \mathbb{Z}_p$, and computes dataset $U = \{\tau\big(H(x_i)^k\big) \mid i \in [N]\}$. Server 1 sends U to Server 2
2. Client chooses $r \leftarrow \mathbb{Z}_p$, and sends $m_i = H(y_i)^r$ for each $y_i \in Y$ to the server 1.
3. Server 1 chooses a random permutation $\pi : [n] \rightarrow [n]$. For each $i \in [n]$, sends $m'_i = (m_{\pi(i)})^k$ to the client.
4. For each $i \in [n]$, the client computes $v_i = \tau\big((m'_i)^{1/r}\big)$.
5. Parties invoke Multi-query Keyword-PIR:
   - Server 1 acts as Keyword-PIR server 1 with dataset U
   - Server 2 acts as Keyword-PIR server 2 with dataset U
   - Client acts as Keyword-PIR client with $V = \{v_i \mid i \in [n]\}$
   - Client learns whether $v_i \in U, \forall i \in [n]$
6. Client outputs $|V \cap U|$

Figure 4: Our semi-honest PSI-CA protocol for asymmetric sets, and extension to 2-server PIR based PSI-CA with changes highlighted

PSI-CA Cost. The server and client must communicate (1) $O(n)$ group elements, (2) $O(n)$ homomorphically encrypted selection vectors for Keyword-PIR. If Keyword-PIR uses $O(\log(N))$ bits for each vector[3], the total

[3]There is a tradeoff between communication and computation complexity in PIR/Keyword-PIR as discussed in [21]. Traditional PIR is $O(\log(N))$ or $O(polylog(N))$ for query vectors, but some schemes trade slightly higher communication complexity for reduced computational complexity.

communication cost is $O(n \log(N))$ bits. We provide more analysis of performance in Section 7 and the full version of the paper [7]. The client's computation is $O(n)$ and the server's computation is $O(nN)$.

The two-server PIR model can be used to speed up the server side computation by avoiding homomorphic encryption operations.

6.2 PSI-CA with 2-server PIR

Recall that the client and server invoke Keyword PIR in step 5 of Figure 4. To speed up the computational overhead on the server side, we introduce a second, independently operated server. The primary server sends the dataset U to the second server after it has been computed. By DDH, the second server learns nothing about the item x_i from u_i.

The client sends PIR queries with keyword v_i to both servers, and learns whether $v_i \in U$ and nothing else. Neither PIR server learn anything about the client's query as long as the two servers do not collude.

With 2-server PIR, the computation cost of PIR contains only symmetric-key operations, using approximately $2N$ PRF calls, and the communication cost of PIR is $O(log(N))$ bits. The highlights in Figure 4 shows the changes in PSI-CA to go from single-server to 2-server PIR.

7 Implementation Choices and Performance Estimates

The main computation cost of our solution is PSI-CA algorithm, which itself is dominated by (1) token transforms (exponentiation) and (2) keyword PIR [20, 21]. We first propose the parameters of our estimation in the following subsection, then summarize the overall system performance. We refer reader to the full verion of the paper [7] for additional analysis of the Epione's performance.

7.1 Parameters and token storage

Assuming that a contact token is generated every 15 minutes for approximately 20 hours a day, then each user sends 80 distinct 128-bit tokens per day. If we assume that a user also receives approximately the same number of tokens and the infectious period is 14 days for COVID-19, then each client receives a total of $n = 1120$ over 14 days. If there are 5,000 new cases per day, the server receives $N = 1120 \times 5000 = 5.6 \times 10^6$ new tokens per day.

In Epione, the server maintains a list of tokens from positive patients for the duration of the infectious window. When a user is diagnosed positive for the disease, they give all of their sent contact tokens for the infection window (or the seeds to generate them) to the server. Rather than storing these by day they were exchanged, it is both more efficient and improves privacy for the server to store them by the day the server received the tokens. This way clients can query only for new tokens that have arrived since they last checked, rather than querying against the entire set.

If there are 5,000 new cases per day, the server receives 5.6×10^6 new tokens per day. Storing both sent and received tokens requires 35 KiB of storage on the client (this can be reduced to 18 KiB if sent tokens are generated with a PRG and only the seed needs to be stored). Assuming the server needs to keep 15 days of tokens in case clients are offline, the total storage for tokens is 1.25 GiB.

7.2 Implementation optimization: Database shape

The bottleneck for scaling PSI-CA to serve a large dataset to a large number of users is PIR. In order to scale up PIR, we propose using a bucket system similar to the password checkup design in [21]. First, the database is split into n_{shards} shards (sometimes referred to as megabuckets). Transformed tokens are grouped into buckets, each bucket holding the same number of tokens, with dummies added as needed. Rather than performing keyword PIR, normal PIR with a bucket address is used. Since tokens are expected to have a uniform distribution (both

before and after transformation), tokens should be uniformly distributed across shards and buckets. As such, the bucket addresses can simply be the first $\log_2(n_{shards}n_{buckets})$ most significant bits of the transformed token itself. Alternatively, a fast hash of the transformed token into the number of bits needed can be used. Recall that each transformed token is truncated to 74 bits before being stored in the database. We use the top bits of the token to be the shard index and bucket address, and only store the remaining bits.

For example, if there are 5.6 million tokens in the server's set, the database can be sharded into 8 sets each with approximately 700,000 tokens (again, assuming a uniform distribution of tokens). If each shard holds 2^{18} buckets, then each bucket holds $\lceil \frac{700{,}000}{2^{18}} \rceil = 3$ transformed tokens, with dummies added as necessary to pad buckets to the required length. The first three most significant bits of the transformed token are used as the shard number, and the following 18 bits of the transformed token are then the bucket address. Since each transformed token is stored as $74 - 3 - 18 = 53$ bits, each bucket has 20 bytes. More detail of the implementation is present in the full version [7].

7.3 Overall PSI-CA Performance Estimates

A major advantage of the Epione design is that the database shape described in the previous section can be tuned to fit the needs of the application and adjusted over time. If on a given day there is a spike in the number of tokens from users diagnosed with the disease, the number of database shards can be increased or the size of the buckets increased.

Using the parameters in Section 7.1 as a starting point and a few assumptions on the database shape, we estimate that single-server PIR-based PSI-CA will take approximately 35 seconds to complete a query. If the query is done in the background without the user waiting on a response, then the query can be done in the cloud as a lower-priority batch processing job, and server resources can be scaled up to meet the number of users required. This was an intentional tradeoff for network efficiency. If the server does some caching of the query keys, then only 37 MiB of network traffic is needed.

The 2-server approach reduces both server computational load and produces a large savings in network bandwidth, but requires an independent party and thus may increase infrastructure costs. Concretely, to complete a query, it requires 1.8 seconds and 679 KiB data transmitted.

We believe that the Epione solution proposed is feasible in practice. This will be studied further at implementation to determine the optimal configuration.

Acknowledgments

We thank Min Suk Kang, Ilya Sergey, Jun Han, Xiaoyuan Liu, Duong Hieu Phan, Jiaheng Zhang, Tiancheng Xie, and Lun Wang for helpful discussion. This material is in part based upon work supported by the National Science Foundation(NSF) under Grant No. TWC-1518899, DARPA under Grant No. N66001-15-C-4066, Center for Long-Term Cybersecurity (CLTC), and IC3 industry partners. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of NSF, DARPA, CLTC or IC3.

References

[1] Center for disease control and prevention. Contact Tracing : Part of a Multipronged Approach to Fight the COVID-19 Pandemic. https://www.cdc.gov/coronavirus/2019-ncov/php/principles-contact-tracing.html

[2] H. Cho, D. Ippolito, and Y. W. Yu. Contact tracing mobile apps for covid-19: Privacy considerations and related trade-offs. *arXiv* 2003.11511 2020

[3] R. Shokri, G. Theodorakopoulos, J.Y. Le Boudec, and J.P. Hubaux. Quantifying location privacy. IEEE symposium on security and privacy 2011

[4] A.M. Olteanu, K. Huguenin, R. Shokri, M. Humbert, and J.P. Hubaux. Quantifying interdependent privacy risks with location data. IEEE Transactions on Mobile Computing 2016

[5] B. Jason, K. Joel, T. Alvin, S. H. Chai, Y. Lai, T. Janice, and T. A. Quy. Bluetrace: A privacy-preserving protocol for community-driven contact tracing across borders. `https://bluetrace.io/`

[6] L. Ferretti, C. Wymant, M. Kendall, L. Zhao, A. Nurtay, L. Abeler-Dorner, M. Parker, D. G. Bonsall, and C. Fraser. Quantifying sars-cov-2 transmission suggests epidemic control with digital contact tracing. *medRxiv*, 2020

[7] N. Trieu, K. Shehata, P. Saxena, R. Shokri, and D. Song . Epione: Lightweight contact tracing with strong privacy. *arXiv:2004.13293* 2020

[8] More scary than coronavirus: South korea's health alerts expose private lives. https://www.theguardian.com/world/2020/mar/06/ more-scary-than-coronavirus-south-koreas-health-alerts-expose-private-lives

[9] Covid-watch. https://www.covid-watch.org/.

[10] Tracetogether. https://www.tracetogether.gov.sg/.

[11] Apple and google partner on covid-19 contact tracing technology. `https://www.apple.com/newsroom/2020/04/apple-and-google-partner-on-covid-19-contact-tracing-technology/`

[12] J. Chan, S. Gollakota, E. Horvitz, J. Jaeger, S. Kakade, T. Kohno, J. Langford, J. Larson, S. Singanamalla, J. Sunshine, and S. Tessaro. Pact: Privacy sensitive protocols and mechanisms for mobile contact tracing. 2020

[13] R. Raskar, I. Schunemann, R. Barbar, K. Vilcans, J. Gray, P. Vepakomma, S. Kapa, A. Nuzzo, R. Gupta, A. Berke, D. Greenwood, C. Keegan, S. Kanaparti, R. Beaudry, D. Stansbury, B. B. Arcila, R. Kanaparti, V. Pamplona, F. M. Benedetti, A. Clough, R. Das, K. Jain, K. Louisy, G. Nadeau, V. Pamplona, S. Penrod, Y. Rajaee, A. Singh, G. Storm, and J. Werner. Apps gone rogue: Maintaining personal privacy in an epidemic. 2020

[14] Cen. https://github.com/Co-Epi/CEN

[15] Dp-3t. https://github.com/DP-3T/documents

[16] M. Ion, B. Kreuter, A. E. Nergiz, S. Patel, M. Raykova, S. Saxena, K. Seth, D. Shanahan, and M. Yung. On deploying secure computing commercially: Private intersection-sum protocols and their business applications. *Cryptology ePrint Archive*, Report 2019/723, 2019. https://eprint.iacr.org/2019/ 723.

[17] W. Diffie and M. Hellman. New directions in cryptography. *IEEE Trans.* Inf. Theor., September 2006

[18] F. Koeune. Pseudorandom Number Generator. *pp. 995–996. Boston, MA: Springer US*, 2011.

[19] C. Gentry and Z. Ramzan. Single-database private information retrieval with constant communication rate. in *ICALP* 2005 , vol. 3580 of LNCS, pp. 803–815, Springer, Heidelberg, July 2005

[20] S. Angel, H. Chen, K. Laine, and S. T. V. Setty. PIR with compressed queries and amortized query processing. in 2018 *IEEE Symposium on Security and Privacy*, pp. 962–979, IEEE Computer Society Press, May 2018.

[21] A. Ali, T. Lepoint, S. Patel, M. Raykova, P. Schoppmann, K. Seth, and K. Yeo. Communication– computation trade-offs in PIR. *Cryptology ePrint Archive*, Report 2019/1483, 2019. https:// eprint.iacr.org/2019/1483

[22] B. Chor, N. Gilboa, and M. Naor. Private information retrieval by keywords. *Cryptology ePrint Archive*, Report 1998/003, 1998. http://eprint.iacr.org/1998/003

[23] E. Boyle, N. Gilboa, and Y. Ishai. Function secret sharing. *EUROCRYPT* 2015. Springer, Heidelberg, April 2015.

[24] E. Boyle, N. Gilboa, and Y. Ishai. Function secret sharing: Improvements and extensions. *ACM CCS* 2016.

[25] B. A. Huberman, M. Franklin, and T. Hogg. Enhancing privacy and trust in electronic communities. in *Proceedings of the 1st ACM Conference on Electronic Commerce*, EC '99, pp. 78–86, ACM, 1999.