Epione: Lightweight Contact Tracing with Strong Privacy

Ni Trieu* Kareem Shehata† Prateek Saxena‡ Reza Shokri‡ Dawn Song¶

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, while protecting the privacy of users' contacts from both central services and other 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 two private sets without revealing 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 may have come into contact with a person that has been diagnosed with the disease, so they can be isolated and tested, and thus prevent the spread of the disease any further.

In the ongoing COVID-19 pandemic, recording of individuals in close proximity has been facilitated by mobile apps that detect nearby mobile phones using Bluetooth. Several countries have been developing contact tracing apps. Such large-scale collection of personal contact information is a significant concern for privacy [19, 46].

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. Indeed, in a fully untrusted environment one should expect any participant to behave adversarially with the goal to exploit others' personal information. Further, both end users of tracing apps as well as the authorities using the collected data can become victims of security attacks, which can allow powerful adversaries to misuse the information collected by the app.

*UC Bekerley, nitrieu@bekerley.edu

†kareem@shehata.ca

‡National University of Singapore, {prateeks, reza}@comp.nus.edu.sg

¶UC Bekerley and Oasis Labs, dawnsong@bekerley.edu

Multiple ways have been proposed to protect the users' private proximity data, offering different privacy guarantees and coming at different implementation costs. For instance, in the recently released BlueTrace protocol used by the Singapore Government [2], users are guaranteed privacy from each other, but the government agency responsible for the app is given an enormous amount of trust. We analyze BlueTrace/TraceTogether and other approaches in more detail in Section 2.1

We consider a model where the government authorities do not store any identifying information (e.g. phone numbers and social contacts) of the users. Such sensitive information databases are a lucrative target for cyber-attackers, and in many countries, collection of contact information conflicts with privacy regulations and public concerns, which may create a barrier to the usage of the tracing service. This is important, as privacy concerns may hinder adoption in some jurisdictions and contact tracing is expected to be effective only when participation is high (e.g. 60% or more of the population [28]).

In our model, the health authorities maintain a database of random tokens corresponding to users which have been diagnosed with the disease. The user's tracing app periodically checks an (untrusted) server to check if the user is potentially at risk in such a way that the server cannot deduce any information about the user which is not implied by the desired functionality and the user learns 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 will also be analyzed in Section 2.1. As we will see through that 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 matches the log on her phone. This could reveal the time, for example, when Alice and the user diagnosed positive with the disease were in close proximity, enabling her to identify Bob. Such identification is undesirable as people have been reported to harass individuals suspecting to be the source of exposure to the disease [6], leading to the so-called "vigilante" problem.

(2) *Infection status by server*: The server is able to determine which users have been diagnosed with the disease. In jurisdictions where the server is operated by the health authority which already knows this information, this may not be a concern. In jurisdictions where the server is operated by another party that does not or should not have this information, then 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 [4], or it is possible to infer the source of a sent token as in the case of [7], 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 linkage can also be used to track a user's movements via Bluetooth beacons. Bluetooth has protections against tracking users over time introduced in Bluetooth 4.0 Low Energy [48]. With solutions such as [1, 16] it is possible to link previously seen Bluetooth device (MAC) addresses despite these protections via exchanged contact tokens once the seeds used to generate the tokens are released[1], producing a similar attack to [11].

(4) *False claims by users*: Users may falsely claim to be diagnosed positive or may claim to be in contact with someone by modifying their app logs. For example, a user who has recovered after being diagnosed positive may threaten to retroactively add a user not currently diagnosed

[1] As internet users have already pointed out https://twitter.com/moxie/status/1248707315626201088

System	System Req. Trusted Server	#	Privacy Protection Against: Infection Status By User	Infection Status By Server	Social Graph	False-positive User	Client Comm. Cost
TraceTogether [2]	Yes	1	Yes	No	No	Yes	$O(n)$
Baseline*	**No**	1	No	No	Most	Some	$O(N)$
Private Messaging [19]		3	No	Yes	Yes	No	
Epione		2	**Yes**	**Yes**	**Yes**	**Yes**	$O(n \log(N))$

Table 1: Comparison of contact tracing systems with respect to security, privacy, required computational infrastructure, and client communication cost. **Baseline** systems include Private Kit [46], Covid-watch [4], CEN [3], DP-3 [5], and PACT's baseline system [16]. 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 encounter tokens from users diagnosed positive with the disease, n is the number of encounter 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 $n << N$).

with the disease to his contact log, unless he is paid a ransom. Similarly, a user not diagnosed may falsely claim to be in contact with a user diagnosed positive whose tokens have been revealed, for example, to qualify for a diagnostic test.

Table 1 provides a brief comparison of different contact tracing systems with respect to different security/privacy properties, required computational infrastructure and client's communication cost, all of which are important for real-world contact tracing. Details of the systems compared will be discussed in Section 2.1.

1.1 Our Contribution

In this work, we introduce Epione, a new system for decentralized contact tracing with stronger privacy protections than the strongest models currently found in related work. 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 server-side database and a small client-side database, as is the case for contact tracing applications. Our new PSI-CA constructions allow us to meet all of our privacy goals. With several other optimizations in our system design, we show that PSI-CA can make privacy-preserving contact tracing practical.

In summary, we make the following contributions:

- We design Epione, an efficient high-performance contact tracing system that provides strong privacy guarantees, specifically that user contact information is not revealed beyond what what is desired to any party and that the diagnosis status is revealed only to health authorities. The system prevents all important attacks, such as linkage attacks (e.g. infection status, social graph exposure), and false-positive claims, to which current models underpinning related work are vulnerable.

- We propose a new semi-honest private set intersection cardinality (PSI-CA) primitive for asymmetric sets. To the best of our knowledge, it is the first PSI-CA protocol which has communication complexity linear in the size of the smaller set (n), and logarithmic in the larger set size N. More precisely, if the set sizes are $n << N$, we achieve a communication overhead of $O(n \log N)$.

1.2 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 generically call "the server", which can collect the transmitted contact tokens from all users tested positive with the disease. The 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 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 when a healthcare provider (such as a hospital) is aware of the identity of the user who it diagnoses as positive. Thus, protecting the identity of the user diagnosed positive from 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 server, which only keeps the list of contact tokens of 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 hospital. Even then, the sent tokens are not useful for identifying any contacts or any other private information. Since tokens are randomly generated, the server would need to know which users received those tokens to re-identify them. We will show later that the 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 will 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)

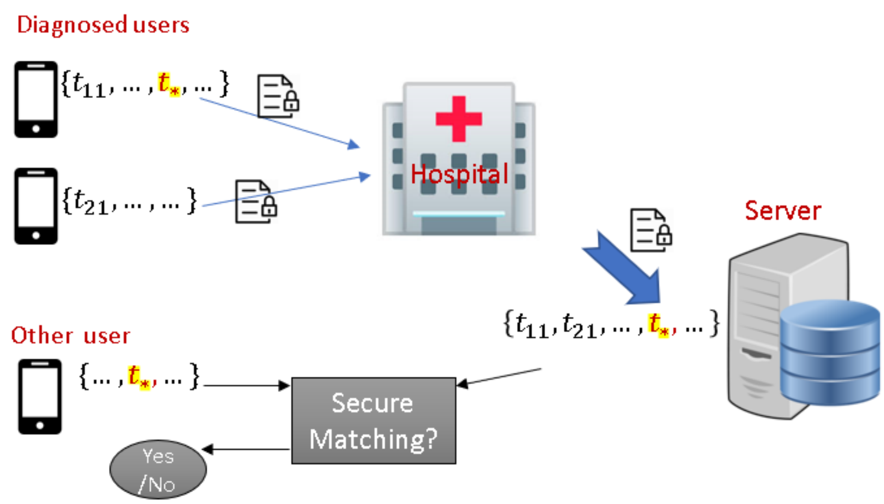

Figure 1: Overview of our Epione system. When a person is diagnosed with the disease, the hospital collects their encrypted sent tokens and forwards them to the server (actually, the PRG seed is collected to reduce communication costs). The tokens are encrypted under the server's public key. Using his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

rely on and expose data to a trusted third-party, such as TraceTogether [2], or (2) uses a decentralized/public list approach such as COVID-Watch [4], PACT [16], or Google/Apple [1] that allows users to infer linkages such as exposure sources. We divide existing Contact Tracing system into Trusted Central Authority and Untrusted Centralized / Distributed approaches.

2.1.1 Trusted Third-Party (Centralized) Approaches

The primary example of a trusted third party approach is TraceTogether by the Singaporean Government, released on March 20, 2020 [7]. Based on the Bluetrace Specification, the protocol works as follows. Let Alice and Bob be users of the app, and let Grace be the government server (or other central authority).

1. **Setup.** Alice and Bob both install the app on their smartphones. During the setup process, both Alice and Bob register a phone number with Grace and are each assigned a unique identifier (i.e., ID_A for Alice, and ID_B for Bob). Grace stores the phone numbers and ID numbers registered in a database.

2. **Token Generation** Grace assigns to Alice and Bob a set of encounter tokens[2] that are to be used at different times. Tokens are generated by concatenating the user's ID, the start time of the token, and the end time of the token, and then encrypting with AES-256-GCM with a key known only to Grace. Tokens are thus expressed as: $T_i = E_{AES}(k, ID||t_{start}||t_{end})$. Tokens are sent in batches along with validity times to users' devices in case they are not able to fetch

[2]In Bluetrace, tokens are called "TempIDs". We have kept the term token here for comparison with other systems.

new tokens on demand [3].

3. **Contact.** When Alice and Bob meet, they exchange tokens valid for the current time window. Both devices keep a list of received and sent tokens.

4. **Diagnosis.** Some time later, Bob is confirmed to have contracted the disease. Bob sends his received tokens list (i.e. the list that contains T_A received from Alice) to Grace. Grace verifies that Bob does in fact have a positive test result (this can be done either by matching Bob in a separate database of test results, or by providing Bob with a passcode that validates his positive result).

5. **Follow-up.** Grace then decrypts each token received from Bob using her key. This reveals the user ID and time of each contact along with other metadata. Grace can then look up each user's (e.g. Alice's) phone number and directly follow up with them.

Clearly, this system places a lot of trust in the government health authority (Grace). As soon as Bob is diagnosed with the disease and he submits his tokens, Grace learns all of Bob's social contacts, down to when and for how long they were in contact. If Grace's key is ever compromised, whether by a state actor or an attacker, this key could be used to track all users via Bluetooth (similar to [11]).

2.1.2 Untrusted Third-Party or Decentralized Approaches

Nearly all of the proposed or launched contact tracing schemes that do not rely on a trusted authority use a scheme as follows, with only minor variations:

1. Alice and Bob, two users of the contact tracing scheme, download and install the app.

2. Alice and Bob both generate contact tokens that rotate over time and cannot be used to reveal their identity directly or track them over time.

3. When Alice and Bob meet, they exchange contact tokens, for example via Bluetooth[4]. Alice and Bob both keep a list of received tokens and sent tokens.

4. Bob is later diagnosed with the disease. Bob submits his tokens to an untrusted server. This can be either the list of received tokens, the list of sent tokens, or both. Alternatively, Bob can submit the secret used to generate tokens from his sent list.

5. A list of tokens from users diagnosed with the disease is then maintained either in a private database to which users can submit queries or published in a public list so that users can check for intersections on their device.

6. If a user finds they have tokens in common with the public list, or via querying a central database, they may have been exposed to the disease and should be notified to seek appropriate next steps.

[3] Note that tokens are generated by Grace mainly to reduce computational load on client devices. This scheme is equivalent to having clients produce tokens by encrypting with a public key for Grace with the same information.

[4] in GPS or geolocation based systems, Alice and Bob independently generate tokens as a function of location and time, for example by hashing a grid square and time quantum.

Covid-watch [4], Private Kit [46][5], PACT's baseline design [16], and Google/Apple [1] 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, but this has no impact on privacy implications. All of these designs are susceptible to linkage attack by either users, the server, or both. Some offer protection against false positive claims.

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

2.1.3 Privacy Improvements

Private Messaging. To reduce the linkage information learned by server, [19] proposes to use two or more non-colluding servers operating as a private messaging network between users and the central server operated by the government (Grace). Concretely, assume that Grace stores a collection of mailboxes, one for each token that Alice and Bob exchange, and there are two non-colluding communication servers Frank and Fred. Frank forwards messages to/from Fred, and Fred forwards messages to/from Grace, all in such a way that Grace cannot know the source or destination of any messages. When Alice and Bob are in close contract, they exchange tokens. At fixed time points, both parties send a message which contains their current diagnosis status to each other via Frank, Fred, and Grace. For example, Bob addresses the message to Alice encrypted using Alice's public key, and gives the message to Grace (through Fred, who is received a forwarding messages from Frank), who puts it in Alice's mailbox. Alice checks all of the mailboxes through Frank and Fred to learn whether she has been exposed to the virus. Since Alice sees Bob's message in her mailbox, Alice might be able to infer who Bob is based on the time they are nearby. Moreover, Grace needs to maintain all tokens (messages) of all users, which requires storage.

Re-randomization of tokens. An alternative approach presented in PACT [16] aims to prevent a linkage attack by users by re-randomizing tokens. Their proposed solution is based on the DDH assumption, and works with the following changes compared to baseline system:

- Alice generates a token in the form $T_a = (g^{r_i}, g^{r_i s_A})$, where s_A is a key that Alice never shares, and r_i is a nonce used only for this token
- When Alice and Bob meet, Alice gives T_a to Bob
- When Bob is diagnosed positive, he chooses a random r'. If $T_a = (x, y)$, he submits the pair $T'_a = (x^{r'}, y^{r'})$ to the server
- Alice can determine whether she is at risk by checking all of the tokens in the public list to find one that satisfies $y = x^{s_A}$

While generating tokens is almost free in our Epione, PACT requires two group elements for each token's generation. Moreover, as mentioned by the authors, the privacy benefit inherently relies on each user re-using the same secret key (s_A in this case), and they cannot force a malicious user to comply. Using a different s_A for different encounters allows Alice to determine which encounter caused her exposure to the disease. In contrast, this malicious action does not happen in our Epione.

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

[5] 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.

information about the other's dataset. PSI has been motivated by many privacy-sensitive real-world applications. Consequently, a lot of work on efficient secure PSI computation has been done [17, 18, 32, 39, 40, 42, 43]. However, PSI only allows the computation of the intersection itself. In many scenarios it is preferable to compute some function of the intersection rather than reveal the elements in the intersection, such as whether intersection size is more than a given threshold f, as in contact tracing. Limited work has focused on this so-called f-PSI problem. Further, most prior works assume sets of comparable size that can be communicated in their entirety to the other party. In this section we focus on protocols [32, 34, 43–45] that support f-PSI as well as PSI-CA.

GC-based PSI. Huang, Katz, and Evans [32] presented techniques for using the generic garbled-circuit approach for f-PSI, which is based on their efficient sort-compare-shuffle circuit construction. Later Pinkas et al [43–45] improved a circuit-PSI using several hashing techniques.

The main bottleneck in the existing circuit-based protocols is the number of string comparisons and computing the statistics (e.g, count) of the associated values that are done inside a generic circuit-based secure two-party computation, which is communication-expensive.

HE-based PSI. The Diffie-Hellman Homomorphic encryption approach of [34] has by far the lowest communication complexity of f-PSI protocols. However, the protocol of [34] has communication complexity linear in set size, which is still communication-expensive in client-server settings where the server's dataset size is asymmetrically larger than that of the client.

3 Problem Statement and Security Goal

Here we define the problem of contact tracing that we intend to solve, and our security goals for Epione.

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 encounter 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 uses the `Exchange` function to exchange tokens with another user. The client's token t_a is sent to the other user, and the client received t_b from the other user. The client stores t_b in the "received tokens list". Similarly, the other user stores t_a in the "received tokens list".
- $\texttt{Query}(T_R, S) \rightarrow a$: With a set T_R of received tokens from `Exchange`, the client uses the `Query` function to query the server S and get answer a indicating how many of their tokens came from users diagnosed positive for the disease.

3.2 Security Definition and Goal

We consider a set of parties who have agreed upon a single functionality 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 the 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. There are two classical security models.

- Colluding model: This is modeled by considering a single monolithic adversary that captures the possibility of collusion between the dishonest parties. The protocol is secure if the joint distribution of those views can be simulated.
- Non-colluding model: This is modeled by considering independent adversaries, each captures the view of each independent dishonest party. The protocol is secure if the individual distribution of each view can be simulated.

There are also two adversarial models.

- Semi-honest model (or honest-but-curious): The adversary is assumed to follow the protocol, but attempts to obtain extra information from the execution transcript.
- Malicious model: The adversary may follow any arbitrary polynomial-time strategy to deviate from the protocol, such as supplying inconsistent inputs, or executing different computation.

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. We formally present the security definition considered in this work, which follows the definition of [37, 41].

Real-world execution. The real-world execution of protocol Π takes place between users $(P_1, \ldots, P_n)$, servers $(P_{n+1}, \ldots, P_N)$ and adversaries $(\mathcal{A}_1, \ldots, \mathcal{A}_m)$, where $m < N$. Let $H \subseteq [n]$ denote the honest parties, $I \subseteq [n]$ denote the set of corrupted and non-colluding parties and $C \in [n]$ denote the set of corrupted and colluding parties.

At the beginning of the execution, each user $P_{i\in[n]}$ receives its input x_i, an auxiliary input z_i and random tape r_i, while each server $P_{i\in[n+1,N]}$ receives only an auxiliary input z_i and random tape r_i. Each adversary $\mathcal{A}_{i\in[m-1]}$ receives an index $i \in I$ that indicates the party it corrupts, while adversary $\mathcal{A}_m$ receives C indicating the set of parties it corrupts.

For all $i \in H$, let OUT_i denote the output of honest party P_i and, let OUT'_i denote the view of corrupted party P_i for $i \in I \cup C$ during the execution of Π. The i^{th} partial output of a real-world execution of Π between parties $(P_1, \ldots, P_N)$ in the presence of adversaries $\mathcal{A} = (\mathcal{A}_1, \ldots, \mathcal{A}_m)$ is defined as

$$\mathsf{REAL}^i_{\Pi,\mathcal{A},I,C,z_i,r_i}(k, x_i) \stackrel{\mathsf{def}}{=} \{\mathsf{OUT}_j \mid j \in H\} \cup \mathsf{OUT}'_i$$

Ideal-world execution. In the ideal-world execution, all the parties interact with a trusted party that evaluates a function f. Similar to the real-world execution, the ideal execution begins with each user $P_{i\in[n]}$ receiving its input x_i, an auxiliary input z_i, and random tape r_i. Each server $P_{i\in[n+1,N]}$ receives only an auxiliary input z_i and random tape r_i.

Each user $P_{i\in[n]}$ sends x'_i to the trusted party, where x'_i is equal to x_i if this user is semi-honest, and is an arbitrary value if he is malicious. If any semi-honest server sends an abort message $(\perp)$, the trusted party returns $\perp$ to all users. The trusted party then returns $f(x'_1, \ldots, x'_n)$ to all the parties.

For all $i \in H$, let OUT_i denote the output returned to the honest party P_i by the trusted party, and let OUT'_i denote some value output by corrupted party P_i for $i \in I \cup C$. The i^{th} partial output of a ideal-world execution of Π between parties $(P_1, \ldots, P_N)$ in the presence of independent simulators $\mathcal{S} = (\mathcal{S}_1, \ldots, \mathcal{S}_m)$ is defined as

$$\mathsf{IDEAL}^i_{\Pi,\mathcal{S},I,C,z_i,r_i}(k, x_i) \stackrel{\mathsf{def}}{=} \{\mathsf{OUT}_j \mid j \in H\} \cup \mathsf{OUT}'_i$$

Definition 1. *(Security) Suppose f is a deterministic-time n-party functionality, and Π is the protocol. Let x_i be the parties' respective private inputs to the protocol. Let $I \in [N]$ denote the set of corrupted and non-colluding parties and $C \in [N]$ denote the set of corrupted and colluding parties. We say that protocol $\Pi(I, C)$ securely computes deterministic functionality f with abort in the presence of adversaries $\mathcal{A} = (\mathcal{A}_1, \ldots, \mathcal{A}_m)$ if there exist probabilistic polynomial-time simulators $\mathsf{Sim}_{i \in m}$ for $m < n$ such that for all $\bar{x}, \bar{z}, \bar{r} \leftarrow \{0,1\}^\star$, and for all $i \in [m]$,*

$$\{\mathit{REAL}^i_{\Pi,\mathcal{A},I,C,\bar{z},\bar{r}}(k, \bar{x})\} \cong \{\mathit{IDEAL}^i_{\Pi,\mathsf{Sim},I,C,\bar{z},\bar{r}}(k, \bar{x})\}$$

Where $\mathcal{S} = (\mathcal{S}_1, \ldots, \mathcal{S}_m)$ and $\mathcal{S} = \mathsf{Sim}_i(\mathcal{A}_i)$

Desirable Security/Privacy Properties. 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 as presented in the above definition 1.

Based on the above security definition, we consider following attacks, especially in the context of contract tracing.

- Linkage attacks: A linkage attack attempts to match anonymized records with non-anonymized records in a different dataset [27]. For contract tracing there are two types of linkage attacks: by users and by the server.

  The adversarial server aims to link users and re-identify their contact history by observing tokens it receives. For example, if the server is able to deduce that Alice and Bob had come in contact, regardless of frequency or duration, that is a linkage attack, referred to in the Introduction as a social graph exposure.

  Even without connecting two users, if the server operator is able to track a single user over time, say by using Bluetooth beacons, that would also constitute a linkage attack.

  On the user side, most users are aware of who they are in contact with for at least some amount of time, thus linkage to tokens is not useful. Instead, we consider any other use of the anonymized information, such as identifying other users they do not already know, finding out about other users' contacts, the infection status of other users, or the source of their own exposure to the disease. Note that if a user was only near one individual during the infection period, and if she gets an alert of having been in contact with a confirmed case, then she knows who it was. This case cannot be avoided while providing the functionality of the application.

- False-positive claim: A malicious 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.

As we will demonstrate in the following sections, Epione provably provides all of the functions of contact tracing while protecting against the attacks above.

4 Preliminaries

This section introduces the notations and cryptographic primitives used in later sections.

4.1 Notation

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 $\{1, \ldots, n\}$. We use '$||$' to denote string concatenation. We use party to refer to either a server or a user in the system.

4.2 Cryptographic building blocks

4.2.1 Decisional Diffie–Hellman

Definition 2. *[24] Let $\mathcal{G}(\kappa)$ be a group family parameterized by security parameter λ. For every probabilistic adversary $\mathcal{A}$ that runs 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 $\epsilon(\lambda)$.

4.3 Pseudorandom Number Generator

Definition 3. *[38] 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.*

4.4 Discrete Log Zero-Knowledge Proof

Discrete Log Zero-Knowledge Proof (DLZK) is a cryptographic protocol, which allows Alice to convince Bob that she has k for known $y = g^k$ in the cyclic group $\mathbb{G} = \langle g \rangle$ without revealing the value of k. One of the simplest and frequently used proofs of knowledge for discrete log is Schnorr protocol, which incurs communication of 2 group elements, and computation of 3 modular exponentiations in a cyclic group.

4.5 Garbled Circuits

Garbled Circuit (GC) is currently the most common generic technique for two-party secure computation (2PC), which was first introduced by Yao [49] and Goldreich et al. [31]. The ideal functionality of GC is to take each party's inputs, x and y respectively, and compute some function f on them. We denote this garbled circuit by $z \leftarrow \mathcal{GC}(x, y, f)$. GC has seen dramatic improvements in recent years. Modern GC protocols [47, 50] evaluate two million AND gates per second on a 1Gbps LAN. In our protocols, we use the "subtraction" and "less than" circuits.

4.6 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 [8, 10, 15, 25, 30] 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.

4.6.1 1-Server PIR

In general, 1-server PIR construction [9, 10, 26] consists of the following algorithms:

- $\mathsf{PIR.Gen}(\kappa) \to (pk, sk)$: takes a security parameter and generates an additively homomorphic public and secret key pair (pk, sk).
- $\mathsf{PIR.Query}(pk, i) \to k$: a randomized algorithm that takes index $i \in [N]$ and public key pk as input and outputs a (short and unexpanded) key k of size $O(\log(N))$.
- $\mathsf{PIR.Expand}(pk, k) \to K$: takes a short key k and public key pk as input and outputs a long *expanded key* $K \in \{0, 1\}^N$.
- $\mathsf{PIR.Answer}(pk, K, DB) \to d$: takes an expanded key K, public key pk, and a database DB as input, returns an answer d encrypted under pk.
- $\mathsf{PIR.Extract}(sk, d) \to DB[i]$: takes a secret key sk and answer d as input, returns $DB[i]$.

With these algorithms defined, PIR generally proceeds as follows.

1. The client generates a public and secret key pair with PIR.Gen. This is generally done once at setup and then reused.
2. Client uses PIR.Query to generate a query key k for the desired item, and send both k and pk to the server.
3. The server uses PIR.Expand to expand k to the much larger K, and then uses PIR.Answer to generate the answer d, which is transmitted to the client.
4. The client then reconstructs $DB[i]$ using PIR.Extract.

PIR is generally implemented such that if $(k) \leftarrow \mathsf{PIR.Query}(i)$, then $K \leftarrow \mathsf{PIR.Expand}(k)$ is the encryption of a binary string of zero everywhere except for a 1 in the ith bit under public key pk. PIR.Answer then consists of iterating over all of the items in the database and computing $d \stackrel{\text{def}}{=} \bigoplus_{j=1}^{N} K[j] \cdot DB[j]$. Thus, d is the encryption of $DB[i]$ under the public key pk.

Most single-server PIR constructions [9, 10, 26, 29] have communication cost of $O(\log(N))$ bits or $O(d\lceil \frac{\sqrt[d]{N}}{p} \rceil)$ bits, where d is typically 2 or 3, and p is typically 2048 or 4096. Depending on the implementation chosen, the latter may actually be faster for the application due to the size of d and p. The single-server PIR requires computation of $O(N)$ additive homomorphic operations. There is also a tradeoff between communication and computation costs, as discussed in [9].

4.6.2 2-Server PIR

The $O(N)$ homomorphic operations required for single server PIR can be very expensive for large databases and for serving large userbases. In order to reduce the computational overhead on the server's side, some PIR schemes use multiple servers with the assumption that not all of them collude [13, 14].

So-called 2-server PIR replaces homomorphic encryption with symmetric encryption (typically AES) and bit operations, making the following changes to the single server scheme:

1. PIR.Query uses a PRF (such as AES) to produce two query keys, k_1, k_2, each of size $O(\log(N))$, which the client sends to server 1 and server 2 respectively.

PARAMETERS: Two parties: server and client; and upper bound on the input set size.
FUNCTIONALITY:

- Wait for input set X from the server
- Wait for input set Y from the client
- Give server nothing
- Give client $|X \cap Y|$

Figure 2: The PSI-CA Functionality.

2. Each server then expands their key by $K_i \leftarrow \mathsf{PIR.Expand}(k_i)$. An important property of the keys is that $K = K_1 \oplus K_2$ is zero everywhere except for position i which is 1. Since each server only has one of the two K_i values, and assuming the two servers do not collude, they cannot determine the value of K or i.
3. Both servers then locally compute the inner product $d_i \stackrel{\text{def}}{=} K_i \cdot DB = \bigoplus_{j=1}^{N} K_i[j] \cdot DB[j]$, and then send the result to the client.
4. The client can then reconstruct $d = d_1 \oplus d_2 = (K_1 \cdot DB) \oplus (K_2 \cdot DB) = (K_1 \oplus K_2) \cdot DB = DB[i]$

In 2-server PIR [13,14], the communication cost is $O(\log(N))$ bits and the computation requires $O(N)$ symmetric key operations which is much faster than additive homomorphic operations.

4.6.3 Keyword PIR

Chor, et al. [20] defined 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 $x \in S$. This variant of PIR has been used for the password checkup problem [9], where a client aims to check whether their password is contained in breached data, without revealing the password itself. One implementation of keyword PIR is based on PIR with Cuckoo hashing, which requires approximately three times the costs of regular PIR. Another solution relies on bucketing which we describe more detail in Section ??. In this paper, we are interested in Keyword PIR based on both 1-server PIR [9,10] and 2-server PIR [13,14].

4.7 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 [32,34,43–45]. The PSI-CA functionality is presented in Figure 2.

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 1.2. Recall that Epione aims to alert any users who have, within the infection window (e.g. 14 days for COVID-19), come into contact with another user who has been diagnosed positive with an infectious disease.

5.1 System Phases

Epione 's design combines several different cryptographic primitives. To explain the design clearly, in this section we only present the functionality of gadgets and how to use them. Section 4.2, and Section 6 discuss how Epione implements them. The Epione system consists of four phases as follows.

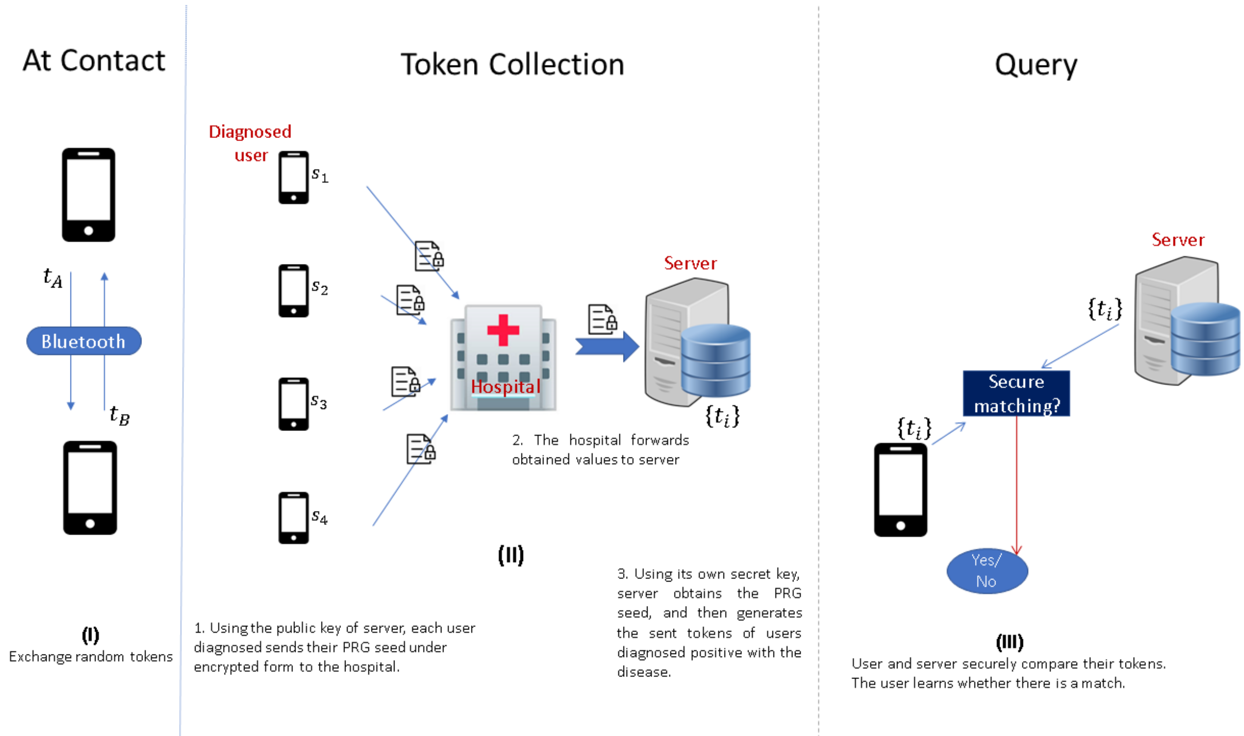

Figure 3: Epione System Design without Agreement and Setup Phase. (I) The tokens are exchanged when two users are in close proximity. (II) When a user is diagnosed with the disease, the user encrypts their PRG seed using the public key of the server, and gives the encrypted value to hospital who transmits it to the server. Using its private key, the server decrypts the received ciphertext and obtains the secret PRG seed of diagnosed users. The server generates the sent tokens of users diagnosed using PRG. (III) Each user invokes a secure matching algorithm with the server, where the user's input is their received tokens and server's input is the database of tokens from users diagnosed with the disease. The user learns whether (or how many) tokens there there are in common between the two sets only.

5.1.1 Agreement and Setup Phase

The first phase requires all parties (including users, hospital, and server) to agree to perform the objective function (e.g. contact tracing using our Epione) over their joint data, and security parameters for MPC. The parties should also agree to release the computed result to each user. This agreement might happen before user initializes Epione on their phone.

The server takes a security parameter λ as input, and 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 encounter 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 needs to generate their own PRG seed, and the server shares the public key with the new user.

5.1.2 Token Generation

Similar to most recent contract tracing systems [3,4,16,19], we use Bluetooth to exchange encounter tokens whenever two users are in close proximity. The `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 4.3.
- 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 server, the server is able to regenerate the tokens sent by the user.
- All users and the server agree on the method used to generate tokens, the time intervals, and day numbering.

5.1.3 Contact

As illustrated in Figure 3 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.

5.1.4 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 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 server. Using its private key, the server decrypts the received values to obtain the secret PRG seeds. The server can then generate all of the tokens sent by users diagnosed positive with the disease, $\widehat{\mathbf{T}}_S$. The token collection process is shown in part II of Figure 3.

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

Alternatively, the hospital could provide a token to the user that the user then provides the backend server when they upload their tokens to prove that they have a legitimate positive diagnosis. This would allow the server to verify that the user's claim is legitimate, but does not prevent the identity of the user from the server.

5.1.5 Query

Recall that each user u_i keeps a list of tokens received from other users they have been in contact with, $\mathbf{T}_{R,u_i}$ from the "contact" phase. The "query" phase aims to securely compare the user's received encounter tokens $\mathbf{T}_{R,u_i}$ with the 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 3.

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 discussed in Section 6.3.1.

5.2 Security Discussion

In this section we consider the security of Epione, starting from a general theorem and then considering specific protections and attacks as defined in Section 3.1.

5.2.1 Security Theorem

The security of Epione follows in a straightforward way from the security of its building blocks (e.g. PSI-CA) and the PRG scheme. Thus, we omit the proof of the following theorem.

Theorem 1. *The Epione construction securely implements the contact tracing functionality defined in Section 3.1 in the semi-honest setting, given the PSI-CA primitive described in Figure 2 and a secure pseudo-random number generator (PRG) as defined in Section 4.3.*

5.2.2 Defense against linkage attacks

As defined in Section 3.2, a linkage attack is any attempt to link an anonymized record with any identifying information. Epione successfully defends against all important linkage attacks.

Linkage attacks by server. In Epione, the server has only the tokens sent by users who have since been diagnosed with the disease. It cannot identify which users have been diagnosed with the disease without colluding with the healthcare provider (the latter is assumed to already know such information, and is responsible for verifying that positive diagnosis is legitimate). Similarly, the healthcare provider does not have access to even randomized encounter tokens without colluding with the server.

Even if the server is able to link a user to specific tokens or seeds in its database, the server does not gain any further information. This limits the amount of exposure in the event of collusion or a breach of the database.

By using PSI-CA, Epione prevents any kind of social graph exposure. The server has only a set of randomized sent encounter tokens, and does not know which users have received those tokens. With PSI-CA, users query for the presence of encounter tokens sent by users diagnosed positive within their own received tokens set, without the server gaining any knowledge of their received tokens.

Linkage attacks by users. Because users only learn the number of encounter tokens they have received from users since diagnosed with the disease and not the tokens themselves, semi-honest users cannot link their exposure to the disease to any particular user.

User tracking and identification. Because tokens appear random, users cannot be tracked using Bluetooth beacons, whether by other users or the Server. It's also impossible to identify a user based solely on a token received from that user without extra information.

5.2.3 Malicious User Queries

If a malicious user, Mallory, can deviate from the protocol by submitting arbitrary queries to the server, it is possible for her to craft queries in such a way as to perform a search on her tokens and find which token(s) in her set are also in the server's set. If Mallory also records the time, place, and who she was with for the tokens she has received, she can later use this information to glean which user(s) have been diagnosed with the disease.

There are several ways to mitigate the threat of arbitrary queries. First, we could require that users submit a cryptographic hash (e.g. by computing a Merkle root[6]) of their local token list periodically to the server, say once every day. When Mallory queries the server, her query includes the cryptographic hash of the set used in the query. The hash ensures that the entire set of tokens obtained by Mallory at a particular point in time is used in the query, and not some chosen subset. Because the query items are encrypted[7], the server cannot directly verify that the hash is correct. Instead, Mallory must also provide a zero knowledge proof that the query items correspond to the hash submitted, such as the SNARK method proposed in [12]. Using this, the server verifies that the query matches the hash.

Periodic commitment to cryptographic hashes of a user's token set mitigates the threat of false claims as well. Users cannot retroactively add tokens to their local lists without being detected by the server.

Note that the contact tracing functionality itself reveals whether there is a match within Mallory's full set of tokens. This is not a linkage attack, but a direct implication of the desired functionality of the application. Mallory, or any benign user, may have had contact with only one other user during the infection window, in which case they can deduce the infection status of the other user when they query the server for intersection cardinality. This is fundamentally unavoidable as it is part of the benign functionality of the application.

A second complementary mitigation is rate limiting users to a few queries per day, and requiring a minimum number of tokens per query. That would only slow Mallory down, but not prevent the attack completely. With enough queries, Mallory will eventually deduce with high probability the source of her exposure.

Lastly, using Digital Rights Management (DRM) protocols such as Android SafetyNet and Apple DeviceCheck will make it much harder for Mallory to submit such queries and is highly recommended. When combined with rate limits and a minimum token set size, these protocols make crafting queries to find the exposure source impractical, though again not impossible. We will not cover protection from these attacks in this paper, as they cannot be distinguished from actual user queries and the protections above are the best possible defence.

[6]The following details ensure that the committed hash value is randomized and defeats any dictionary attacks by the server: We first permute the local tokens randomly, add a dummy random value in the list, and then compute a Merkle tree. The Merkle root is committed as the cryptographic hash of the list. The adversary does not know the dummy value or the permuted order and hence is unable to forge a Merkle proof.

[7]Either under DDH for token transformations, or homomorphic encryption for 1-server PIR, or AES for 2 server PIR, depending on at which step the check is done

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, as well as extensions to those tools.

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

In this section, we present PSI-CA construction, the functionality of which is described in Figure 2 and used as a core component of Epione.

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 [33] are still preferable in many real-world applications[8], due to their extremely low communication cost.

DH-based PSI Assume that server has input $X = \{x_1, \dots, x_N\}$ and client has input $Y = \{y_1, \dots, y_n\}$. 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 4.

PARAMETERS: cyclic group G of order p; random oracle H.

INPUTS: Server has input $X = \{x_1, \dots, x_N\}$; client has input $Y = \{y_1, \dots, y_n\}$.

PROTOCOL:

1. Client chooses $r \leftarrow \mathbb{Z}_p$ for each $y_i \in Y$ and 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 4: DH-based PSI protocol and extension to PSI-CA with changes highlighted.

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[9].

[8] https://security.googleblog.com/2019/06/helping-organizations-do-more-without-collecting-more-data.html

[9] Alternatively, the client can raise each $H(x_i)^k$ to the r power and compare to the $H(y_i)^{kr}$ values. However, the variant where the client raises $H(y_i)^{kr}$ to the $1/r$ power is compatible with further optimizations.

From DH-based PSI into 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 [33] and recently is incorporated into private intersection sum [34], 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 4 also shows the PSI-CA protocol with changes highlighted. The key idea to transform PSI into PSI-CA is that instead of sending m'_i in step 2 of Figure 4 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 into 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. There is a small handful of works [18, 23, 36] focused on PSI for asymmetric set sizes. However, to the best of our knowledge, ours is the first PSI-CA protocol which has communication complexity linear in the client's set (n), and logarithmic in the server's set (N).

We observe that the last two steps of Figure 4 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 4 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 5, following closely the description in the previous section. 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 will still be 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 5, we use a truncation function $\tau(z)$ which takes z as input and returns the most significant $\lambda + log(N)$ bits of z.

Theorem 2. *The PSI-CA construction of Figure 5 securely implements the PSI-CA functionality defined in Figure 2 in semi-honest setting, given the Multi-query Keyword PIR described in Section 4.6.*

Parameters: cyclic group G of order p; random oracle H, Multi-query Keyword PIR primitive (Section 4.6), 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 compute dataset $U = \{\tau\big(H(x_i)^k\big) \mid i \in [N]\}$. Server 1 sends U to Server 2
2. Client chooses $r \leftarrow \mathbb{Z}_p$ for each $y_i \in Y$ and sends $m_i = H(y_i)^r$ to the server 1.
3. Server 1 chooses a random permutation $\pi : [n] \to [n]$. 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 with 2 servers:
   - Server 1 acts as Keyword-PIR's server 1 with dataset U
   - Server 2 acts as Keyword-PIR's server 2 with dataset U
   - Client acts as Keyword-PIR's 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 5: Our semi-honest PSI-CA protocol for asymmetric sets, and extension to 2-server PIR based PSI-CA with changes highlighted

Sketch of proof:

The security of the PSI-CA protocol follows from the fact that $\{H(z_1)^k, \ldots, H(z_n)^k\}$ are indistinguishable from random, for *distinct* z_i, if H is a random oracle and the DDH assumption holds in G. To see why, consider a simulator that receives a DDH challenge $g^k, g^{a_1}, \ldots, g^{a_n}, g^{c_1}, \ldots, g^{c_n}$ where each c_i is either random or $c_i = a_i k$. The simulator programs the random oracle so that $H(z_i) = g^{a_i}$ and then simulates the outputs as $\{g^{c_1}, \ldots, g^{c_n}\}$. It is easy to see that these messages are distributed as specified by F if the c_i's are distributed as $a_i k$, but are distributed uniformly otherwise, with the difference being indistinguishable by the DDH assumption.

We consider two cases, corresponding to each party being corrupt.

- A corrupt server first sees $\{H(y_i)^r\}_{y_i \in Y}$. By our observation above, these values are pseudorandom. A corrupt server also sees PIR transcripts. Because pseudorandomness guarantees of PIR, the client's message to the server can be simulated as a random message.
- A corrupt client sees $\{H(y_{\pi(i)})^{rk} = (H(y_{\pi(i)})^k)^r\}_{y_{\pi(i)} \in Y}$ and PIR response (as the extra PSI message), along with its private randomness r and any random oracle queries/responses that it made. Consider modifying this view, replacing each $H(z)^k$ term with an independently random group element for each $z \in X \cup Y$ (each distinct, by definition). This change will be indistinguishable, by the reasoning above. Now it is not necessary to know the identities of

$x_i \in X \setminus Y$ as well as $x_i \in X \cup Y$, as their corresponding $H(x_i)^k$ values have been replaced with random group elements that are independent of everything else, and the secret permutation π hides the common items. In other words, this is a distribution that can be generated by the simulator, with knowledge of only Y and $|X \cap Y|$.

□

6.1.3 PSI-CA Costs

In our PSI-CA protocol, the communication cost is $O(n \log(N))$ while the client's computation is $O(n)$ and the server's computation is $O(nN)$. Concretely,

- The server and client must communicate (1) $O(n)$ group elements, (2) n homomorphically encrypted selection vectors each of size $O(\log(N))$ for keyword PIR, for a combined size of $O(n \log(N))$.
- The client's computation cost consists of: (1) $O(n)$ group elements, (2) $O(n)$ homomorphic encryptions for encoding the PIR queries and decoding the results.
- The server's computation cost consists of: (1) $O(n)$ group elements, (2) $O(nN)$ additive homomorphic encryption operations for finding the answer to the keyword PIR query.

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 5. 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 keyword PIR contains only symmetric-key operations. Concretely, it invokes roughly $2N$ PRF calls. Figure 5 also shows 2-server PIR based PSI-CA protocol with changes highlighted.

6.3 PSI-CA Extensions

Revealing the intersection size is acceptable in the contact tracing application we consider. However, it is possible that in other settings, knowing the size of the intersection is undesirable leakage.

6.3.1 Potential approach from PSI-CA into threshold PSI-CA (t-PSI)

In general, threshold PSI-CA is an extension of PSI where parties learn the intersection size (or even the intersection items) if it is greater than a given threshold. In our t-PSI definition, the client learns whether two input sets have any common items and nothing else (e.g. $t = 0$).

A simple solution is to pad both input sets with dummy elements. The two parties decide on a pseudo-random generator function (PRG) and seed s which are used to generate common dummy elements. Each party randomly chooses a number of dummy elements, n' and N' for client and server respectively such that $n' > N'$. This step can be done by performing the following steps:

1. Parties randomly choose n' and N'
2. Parties invoke a garbled circuit to check whether $n' > N'$
3. Repeat this process until the “if" condition is true

The client and server then use the agreed PRG and seed s to generate n' and N' fake items, respectively, and add them to their input sets. The resulting intersection set size over the original and common dummy elements will be $\sigma = |X \cap Y| + N'$. The client learns σ at the end of PSI-CA, but does not know N', and thus has limited knowledge of $|X \cap Y|$.

The parties then invoke a garbled circuit to securely remove the term N' in σ and check whether $|X \cap Y| > t$. The circuit takes as input N' from the server and σ from the client, computes $f = \big(\sigma - N'\big) > t?$, and returns the result to the client.

There remains an important concern in this approach: how to choose n' and N', so that σ informationally hides the actual intersection size. Since the range of intersection size is from 0 to n, the bound of information leakage is $O(\log(n))$ bits. Therefore, it is sufficient to choose n' and N' to be $O(2^{\log(n)})$, which is essentially $O(n)$. However, it is not clear what the coefficient value behind the big O need to be to prevent leakage of the actual intersection size. For example, the client can infer the lower bound of the intersection size by knowing that $|X \cap Y| = \sigma - N' > \sigma - n'$. Further analysis needs to be done to prove if this method provides sufficient bits to prevent information leakage.

6.3.2 Potential approach for extending to malicious client

In the context of contact tracing, a malicious client seeks to obtain information about the server's database (set X in this case). Thus, they attempt to compute $m_i = H(y_i)^r$ incorrectly, since in step 3 of Figure 5 the server returns $m'_i = (m_{\pi(i)})^k$ and it may be possible to learn part or all of the value of k. With that, the client can determine which values of v_i map to which values of u_j, and thus learn which items from its set exist on the server.

One solution to prevent this is to augment the protocol with a zero-knowledge proof [22,35] that the m_i's were computed correctly. This adds the following step to the protocol:

(2a) Client performs a zero-knowledge proof of knowledge of r such that $\forall i \in [n] : m_i = (y_i)^r$. The server aborts if the proof does not verify.

This modification is enough to guarantee security against a malicious client for the DH-based PSI protocol.

Achieving a PSI-CA or t-PSI protocol resilient to a malicious client requires more work because the construction of the client's message involves more building blocks, namely keyword PIR and garbled circuit. The solution is to use versions of these building blocks resilient to such one-side malicious attacks. Moreover, to verify that the client uses the correct values for v_i for step 5 of Figure 5 in the keyword PIR query, we employ a consistent check such as MACs from the SPDZ protocol [21]. Using all of these techniques we achieve security from a malicious client. We will explore more detail in this direction.

Acknowledgments

We thank Min Suk Kang, Ilya Sergey, Jun Han, Xiaoyuan Liu, and Duong Hieu Phan for helpful discussion. This material is in part based upon work supported by Center for Long-Term Cybersecurity (CLTC), Brandeis, and National Science Foundation (NSF) Cryptocurrency.

References

[1] 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/.

[2] Bluetrace. https://bluetrace.io/.

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

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

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

[6] '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.

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

[8] Carlos Aguilar-Melchor, Joris Barrier, Laurent Fousse, and Marc-Olivier Killijian. Xpir : Private information retrieval for everyone. *Proceedings on Privacy Enhancing Technologies*, 2016(2):155 – 174, 2016.

[9] Asra Ali, Tancrède Lepoint, Sarvar Patel, Mariana Raykova, Phillipp Schoppmann, Karn Seth, and Kevin Yeo. Communication–computation trade-offs in PIR. Cryptology ePrint Archive, Report 2019/1483, 2019. https://eprint.iacr.org/2019/1483.

[10] Sebastian Angel, Hao Chen, Kim Laine, and Srinath T. V. Setty. PIR with compressed queries and amortized query processing. In *2018 IEEE Symposium on Security and Privacy*, pages 962–979. IEEE Computer Society Press, May 2018.

[11] Johannes Becker, David Li, and David Starobinski. Tracking anonymized bluetooth devices. *Proceedings on Privacy Enhancing Technologies*, 2019:50–65, 07 2019.

[12] Eli Ben-Sasson, Alessandro Chiesa, Eran Tromer, and Madars Virza. Succinct non-interactive zero knowledge for a von neumann architecture. In Kevin Fu and Jaeyeon Jung, editors, *USENIX Security 2014*, pages 781–796. USENIX Association, August 2014.

[13] Elette Boyle, Niv Gilboa, and Yuval Ishai. Function secret sharing. In Elisabeth Oswald and Marc Fischlin, editors, *EUROCRYPT 2015, Part II*, volume 9057 of *LNCS*, pages 337–367. Springer, Heidelberg, April 2015.

[14] Elette Boyle, Niv Gilboa, and Yuval Ishai. Function secret sharing: Improvements and extensions. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *ACM CCS 2016*, pages 1292–1303. ACM Press, October 2016.

[15] Christian Cachin, Silvio Micali, and Markus Stadler. Computationally private information retrieval with polylogarithmic communication. In Jacques Stern, editor, *EUROCRYPT'99*, volume 1592 of *LNCS*, pages 402–414. Springer, Heidelberg, May 1999.

[16] Justin Chan, Shyam Gollakota, Eric Horvitz, Joseph Jaeger, Sham Kakade, Tadayoshi Kohno, John Langford, Jonathan Larson, Sudheesh Singanamalla, Jacob Sunshine, and Stefano Tessaro. Pact: Privacy sensitive protocols and mechanisms for mobile contact tracing, 2020.

[17] Hao Chen, Zhicong Huang, Kim Laine, and Peter Rindal. Labeled PSI from fully homomorphic encryption with malicious security. In David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang, editors, *ACM CCS 2018*, pages 1223–1237. ACM Press, October 2018.

[18] Hao Chen, Kim Laine, and Peter Rindal. Fast private set intersection from homomorphic encryption. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *ACM CCS 2017*, pages 1243–1255. ACM Press, October / November 2017.

[19] Hyunghoon Cho, Daphne Ippolito, and Yun William Yu. Contact tracing mobile apps for covid-19: Privacy considerations and related trade-offs, 2020.

[20] Benny Chor, Niv Gilboa, and Moni Naor. Private information retrieval by keywords. Cryptology ePrint Archive, Report 1998/003, 1998. http://eprint.iacr.org/1998/003.

[21] Ivan Damgαrd, Valerio Pastro, Nigel P. Smart, and Sarah Zakarias. Multiparty computation from somewhat homomorphic encryption. In Reihaneh Safavi-Naini and Ran Canetti, editors, *CRYPTO 2012*, volume 7417 of *LNCS*, pages 643–662. Springer, Heidelberg, August 2012.

[22] Emiliano De Cristofaro, Jihye Kim, and Gene Tsudik. Linear-complexity private set intersection protocols secure in malicious model. In Masayuki Abe, editor, *ASIACRYPT 2010*, volume 6477 of *LNCS*, pages 213–231. Springer, Heidelberg, December 2010.

[23] Daniel Demmler, Peter Rindal, Mike Rosulek, and Ni Trieu. Pir-psi: Scaling private contact discovery. *PoPETs*, 2018:159–178, 2018.

[24] W. Diffie and M. Hellman. New directions in cryptography. *IEEE Trans. Inf. Theor.*, 22(6):644–654, September 2006.

[25] Changyu Dong and Liqun Chen. A fast single server private information retrieval protocol with low communication cost. In Miroslaw Kutylowski and Jaideep Vaidya, editors, *ESORICS 2014, Part I*, volume 8712 of *LNCS*, pages 380–399. Springer, Heidelberg, September 2014.

[26] Changyu Dong and Liqun Chen. A fast single server private information retrieval protocol with low communication cost. In Mirosław Kutyłowski and Jaideep Vaidya, editors, *Computer Security - ESORICS 2014*, pages 380–399, Cham, 2014. Springer International Publishing.

[27] Cynthia Dwork and Aaron Roth. The algorithmic foundations of differential privacy. *Found. Trends Theor. Comput. Sci.*, 9(3–4):211–407, August 2014.

[28] Luca Ferretti, Chris Wymant, Michelle Kendall, Lele Zhao, Anel Nurtay, Lucie Abeler-Dorner, Michael Parker, David G Bonsall, and Christophe Fraser. Quantifying sars-cov-2 transmission suggests epidemic control with digital contact tracing. *medRxiv*, 2020.

[29] Craig Gentry and Zulfikar Ramzan. Single-database private information retrieval with constant communication rate. In *Proceedings of the 32nd International Conference on Automata, Languages and Programming*, ICALP'05.

[30] Craig Gentry and Zulfikar Ramzan. Single-database private information retrieval with constant communication rate. In Luís Caires, Giuseppe F. Italiano, Luís Monteiro, Catuscia Palamidessi, and Moti Yung, editors, *ICALP 2005*, volume 3580 of *LNCS*, pages 803–815. Springer, Heidelberg, July 2005.

[31] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or A completeness theorem for protocols with honest majority. In Alfred Aho, editor, *19th ACM STOC*, pages 218–229. ACM Press, May 1987.

[32] Yan Huang, David Evans, and Jonathan Katz. Private set intersection: Are garbled circuits better than custom protocols? In *NDSS 2012*. The Internet Society, February 2012.

[33] Bernardo A. Huberman, Matt Franklin, and Tad Hogg. Enhancing privacy and trust in electronic communities. In *Proceedings of the 1st ACM Conference on Electronic Commerce*, EC '99, pages 78–86. ACM, 1999.

[34] Mihaela Ion, Ben Kreuter, Ahmet Erhan Nergiz, Sarvar Patel, Mariana Raykova, Shobhit Saxena, Karn Seth, David Shanahan, and Moti 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.

[35] Stanislaw Jarecki and Xiaomin Liu. Fast secure computation of set intersection. In Juan A. Garay and Roberto De Prisco, editors, *SCN 10*, volume 6280 of *LNCS*, pages 418–435. Springer, Heidelberg, September 2010.

[36] Daniel Kales, Christian Rechberger, Thomas Schneider, Matthias Senker, and Christian Weinert. Mobile private contact discovery at scale. In Nadia Heninger and Patrick Traynor, editors, *USENIX Security 2019*, pages 1447–1464. USENIX Association, August 2019.

[37] Seny Kamara, Payman Mohassel, and Ben Riva. Salus: A system for server-aided secure function evaluation. Cryptology ePrint Archive, Report 2012/542, 2012. http://eprint.iacr.org/2012/542.

[38] François Koeune. *Pseudorandom Number Generator*, pages 995–996. Springer US, Boston, MA, 2011.

[39] Vladimir Kolesnikov, Ranjit Kumaresan, Mike Rosulek, and Ni Trieu. Efficient batched oblivious PRF with applications to private set intersection. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *ACM CCS 2016*, pages 818–829. ACM Press, October 2016.

[40] Vladimir Kolesnikov, Naor Matania, Benny Pinkas, Mike Rosulek, and Ni Trieu. Practical multi-party private set intersection from symmetric-key techniques. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *ACM CCS 2017*, pages 1257–1272. ACM Press, October / November 2017.

[41] Goldreich Oded. *Foundations of Cryptography: Volume 2, Basic Applications*. Cambridge University Press, USA, 1st edition, 2009.

[42] Benny Pinkas, Mike Rosulek, Ni Trieu, and Avishay Yanai. SpOT-light: Lightweight private set intersection from sparse OT extension. In Alexandra Boldyreva and Daniele Micciancio, editors, *CRYPTO 2019, Part III*, volume 11694 of *LNCS*, pages 401–431. Springer, Heidelberg, August 2019.

[43] Benny Pinkas, Thomas Schneider, Gil Segev, and Michael Zohner. Phasing: Private set intersection using permutation-based hashing. In Jaeyeon Jung and Thorsten Holz, editors, *USENIX Security 2015*, pages 515–530. USENIX Association, August 2015.

[44] Benny Pinkas, Thomas Schneider, Oleksandr Tkachenko, and Avishay Yanai. Efficient circuit-based PSI with linear communication. In Yuval Ishai and Vincent Rijmen, editors, *EUROCRYPT 2019, Part III*, volume 11478 of *LNCS*, pages 122–153. Springer, Heidelberg, May 2019.

[45] Benny Pinkas, Thomas Schneider, Christian Weinert, and Udi Wieder. Efficient circuit-based PSI via cuckoo hashing. In Jesper Buus Nielsen and Vincent Rijmen, editors, *EUROCRYPT 2018, Part III*, volume 10822 of *LNCS*, pages 125–157. Springer, Heidelberg, April / May 2018.

[46] Ramesh Raskar, Isabel Schunemann, Rachel Barbar, Kristen Vilcans, Jim Gray, Praneeth Vepakomma, Suraj Kapa, Andrea Nuzzo, Rajiv Gupta, Alex Berke, Dazza Greenwood, Christian Keegan, Shriank Kanaparti, Robson Beaudry, David Stansbury, Beatriz Botero Arcila, Rishank Kanaparti, Vitor Pamplona, Francesco M Benedetti, Alina Clough, Riddhiman Das, Kaushal Jain, Khahlil Louisy, Greg Nadeau, Vitor Pamplona, Steve Penrod, Yasaman Rajaee, Abhishek Singh, Greg Storm, and John Werner. Apps gone rogue: Maintaining personal privacy in an epidemic, 2020.

[47] Xiao Wang, Alex J. Malozemoff, and Jonathan Katz. Faster secure two-party computation in the single-execution setting. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *EUROCRYPT 2017, Part III*, volume 10212 of *LNCS*, pages 399–424. Springer, Heidelberg, April / May 2017.

[48] Martin Woolley. Bluetooth technology protecting your privacy. https://www.bluetooth.com/blog/bluetooth-technology-protecting-your-privacy/, 2015.

[49] Andrew Chi-Chih Yao. How to generate and exchange secrets (extended abstract). In *27th FOCS*, pages 162–167. IEEE Computer Society Press, October 1986.

[50] Samee Zahur, Mike Rosulek, and David Evans. Two halves make a whole - reducing data transfer in garbled circuits using half gates. In Elisabeth Oswald and Marc Fischlin, editors, *EUROCRYPT 2015, Part II*, volume 9057 of *LNCS*, pages 220–250. Springer, Heidelberg, April 2015.