Privacy-Preserving Epidemiological Modeling on Mobile Graphs

Daniel Günther*, Marco Holz*, Benjamin Judkewitz†, Hellen Möllering*, Benny Pinkas‡, Thomas Schneider*, Ajith Suresh§
*Technical University of Darmstadt, Germany †Charité-Universitätsmedizin, Germany ‡Bar-Ilan University, Israel §Technology Innovation Institute (TII), Abu Dhabi
{guenther, holz, moellering, schneider}@encrypto.cs.tu-darmstadt.de,
benjamin.judkewitz@charite.de, benny@pinkas.net, ajith.Suresh@tii.ae

***Abstract*—The latest pandemic COVID-19 brought governments worldwide to use various containment measures to control its spread, such as contact tracing, social distance regulations, and curfews. Epidemiological simulations are commonly used to assess the impact of those policies before they are implemented. Unfortunately, the scarcity of relevant empirical data, specifically detailed social contact graphs, hampered their predictive accuracy. As this data is inherently privacy-critical, a method is urgently needed to perform powerful epidemiological simulations on real-world contact graphs without disclosing any sensitive information.**

In this work, we present RIPPLE, a privacy-preserving epidemiological modeling framework enabling standard models for infectious disease on a population's real contact graph while keeping all contact information locally on the participants' devices. As a building block of independent interest, we present PIR-SUM, a novel extension to private information retrieval for secure download of element sums from a database. Our protocols are supported by a proof-of-concept implementation, demonstrating a 2-week simulation over half a million participants completed in 7 minutes, with each participant communicating less than 50 KB.

Keywords*—*epidemiological modeling, private information retrieval, secure multi-party computation

I. Introduction

The COVID-19 pandemic has profoundly impacted daily life, leading to heightened mental illness and domestic abuse cases [1]–[3]. Governments globally have taken steps, such as lockdowns and institution closures, to curb the virus while supporting the economy. Despite these measures, infections surged, and many lives were lost. Afterwards, new infectious diseases like monkeypox have spread, resulting in quarantines in Europe [4]–[6].

In the context of COVID-19, contact tracing apps were used all over the world to notify contacts of potential infections [7]–[14]. Unfortunately, there is a fundamental limitation to contact tracing: It only notifies contacts of an infected person *after* the infection has been detected, i.e., typically after a person develops symptoms, is tested, receives the test result, and can connect with contacts [15], [16]. Tupper et al. [15] report that in British Columbia in April 2021, this process ideally took five days, reducing new cases by only 8% compared to not using contact tracing. They conclude that contact tracing must be supplemented with multiple additional containment measures to control disease spread effectively.

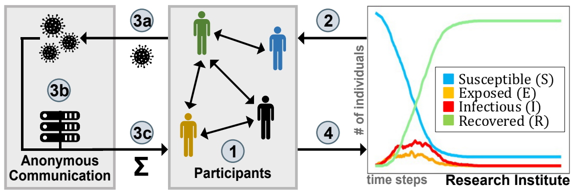
In contrast, we consider epidemiological modeling, which allows predicting the spread of an infectious disease in the *future* and has received a lot of attention [17]–[23]. It allows us to assess the effectiveness of containment measures by mathematically modeling their impact on the spread. As a result, it can be an extremely valuable tool for governments to select effective containment measures [22]. For example, Davis et al. [24] predicted in early 2020 that COVID-19 would infect 85% of the British population without any containment measures in place, causing a massive overload of the health system (13-80$\times$ the capacity of intensive care units). Their forecast also indicated that short-term interventions such as school closures, social distancing, and so on would not effectively reduce the number of cases. As a result, the British government decided to implement a lockdown in March 2020, effectively reducing transmissions and stabilising the health system [22].

Access to detailed information about a population's size, density, transportation, and health care system enables accurate epidemiological modeling to forecast disease transmission in various scenarios [25]. Precise, up-to-date data on movements and physical interactions is crucial for forecasting transmission and assessing the impact of control measures before implementation [26]. In practice, these simulations can quickly model the spread of a disease, project the number of infections based on specific actions, and predict how the disease might spread to specific areas.

However, data on personal encounters is scarce, limiting accurate assessments of containment measures' impacts [25]–[27]. This scarcity arises because encounter data is often obtained through surveys, which fail to capture the reality of random encounters in public places [26], [28]. Additionally, social interaction patterns can change rapidly, as seen with social distancing measures, making collected data quickly outdated. Therefore, existing data cannot realistically simulate person-to-person social contact graphs. Ideally, epidemiologists need a complete physical interaction graph of the population, but strict privacy regulations make accurate tracking of interpersonal contacts unacceptable.

To address the issue of preserving privacy while obtaining up-to-date contact data, we present RIPPLE, a practical framework for epidemiological modeling that allows precise disease spread simulations using current contact information, incorporating control measures without leaking information about individuals' contacts. RIPPLE provides a privacy-preserving method for collecting real-time physical encounters and can compute arbitrary compartment-based epidemiological models[1] on the latest contact graph of the previous days. RIPPLE is not only applicable to COVID-19, but to *any* infectious diseases. We anticipate that our framework's privacy guarantee will encourage more people to participate, allowing epidemiologists to compute more accurate simulations for developing effective containment strategies.

Our Contributions: This paper introduces RIPPLE (cf. Fig. 1), a framework for expanding the scope of privacy research from contact tracing to epidemiological modeling. While the former only warns about potential infections in the past, epidemiological modeling can predict the spread of infectious diseases in the future. Anticipating the effects of various control measures allows for the development of informed epidemic containment strategies and political interventions before their implementation.



① Mobile apps collect anonymous encounter tokens during interactions. ② Research Institute begins the simulation by providing initialization parameters. (3a) Participants securely upload infection likelihood to servers. (3b) Servers securely compute cumulative infection likelihood per participant. (3c) Participants retrieve their cumulative infection likelihood. ④ The aggregate results (#S,#E,#I,#R) are sent to the Research Institute.

Fig. 1: Overview of RIPPLE Framework.

RIPPLE uses a fully decentralised system similar to the federated learning paradigm [29] to achieve high acceptance and trust in the system and to motivate many participants to join the system to generate representative contact information. All participant data, such as encounter location, time, and distance, are kept locally on the participants' devices. Participants in RIPPLE communicate through anonymous communication channels enabled by a group of *semi-honest* central servers.

RIPPLE is instantiated with two methods for achieving privacy-preserving epidemiological modeling, each covering a different use case. The first is RIPPLE$_{\mathsf{TEE}}$, which assumes each participant's mobile device has a Trusted Execution Environment (TEE). The second method is RIPPLE$_{\mathsf{PIR}}$, which eliminates this assumption by utilising cryptographic primitives such as Private Information Retrieval (PIR). Along the way, we develop a multi-server PIR extension that allows a client to retrieve the sum of a set of elements (in our case, infection likelihoods) from a database without learning individual entries.

We assess the practicality of our methods by benchmarking core building blocks using a proof of concept implementation. Our findings indicate that, with adequate hardware, both protocols can scale up to millions of participants. For instance, a simulation of 14 days with 1 million participants can be completed in less than half an hour.

Our contributions are summarized as follows:

1) We present RIPPLE, the *first* privacy-preserving framework for epidemiological modeling on contact information stored on mobile devices.

2) RIPPLE formalises the notion of *privacy-preserving* epidemiological modeling and defines privacy requirements in the presence of both semi-honest and malicious participants.

3) We present two techniques – RIPPLE$_{\mathsf{TEE}}$ and RIPPLE$_{\mathsf{PIR}}$ – that combine anonymous communication techniques with either TEEs or PIR and anonymous credentials.

4) We propose PIR-SUM, an extension to existing PIR schemes, that allows a client to download the sum of τ distinct database entries without learning the values of individual entries or revealing which entries were requested.

5) We demonstrate the practicality of our framework by providing an open source implementation and a detailed performance evaluation of RIPPLE.

II. Related Work & Background Information

This section discusses related works addressing privacy challenges in the context of infectious diseases as well as necessary background information on contact tracing and epidemiological modeling (including a clarification of the differences between the two). An overview of the (cryptographic) primitives and other techniques used in this work is presented in §A.

A. Cryptography-based Solutions in the Context of Infectious Diseases

CrowdNotifier [13] notifies visitors of (large) events about an infection risk when another visitor reported SARS-CoV-2 positive after the event, even if they have not been in close proximity of less than 2 meters. To protect user privacy, it follows a distributed approach where location and time information is stored encrypted on the user's device. Bampoulidis et al. [30] introduce a privacy-preserving two-party set intersection protocol that

[1] The implementation of concrete simulation functions is outside the scope of this work and referred to medical experts. More details on epidemiological modeling are given in §II.

detects infection hotspots by intersecting infected patients, input by a health institute, with customer data from mobile network operators.

CoVault [31] is a data analytics platform based on secure multi-party computation techniques (MPC) and trusted execution environments. The authors discuss the usage of CoVault for storing location and timing information of people usable by epidemiologists to analyse (unique) encounter frequencies or linkages among two disease outbreak clusters while preserving privacy.

Al-Turjman and David Deebak [32] integrate privacy-protecting health monitoring into a Medical Things device that monitors the health status (heart rate, oxygen saturation, temperature, etc.) of users in quarantine with moderate symptoms. Only in the event of an emergency is medical personnel notified. Pezzutto et al. [33] optimize the distribution of a limited set of tests to identify as many positive cases as possible, which are then isolated. Their system can be deployed in a decentralized, privacy-preserving environment to identify individuals who are at high risk of infection. Barocchi et al. [34] develop a privacy-preserving architecture for indoor social distancing based on a privacy-preserving access control system. When users visit public facilities (e.g., a supermarket or an airport), their mobile devices display a route recommendation for the building that maximizes the distance to other people. Bozdemir et al. [35] suggest privacy-preserving trajectory clustering to identify typical movements of people and detect forbidden gatherings when contact restrictions are in place.

a) Contact Tracing.: A plethora of contact tracing systems has been introduced and deployed since the outbreak of the pandemic [7], [8], [36]. They either use people's location (GPS or telecommunication provider information) or measure proximity (via Bluetooth LE). Most systems can be categorized into centralized and decentralized designs [10]. In a centralized contact tracing system (e.g., [37], [38]), computations such as the generation of the tokens exchanged during physical encounters are done by a central party. This central party may also store some contact information depending on the concrete system design. In contrast, in decentralized approaches (e.g., [9], [12], [39]), computation and encounter information remain (almost completely) locally at the participants' devices.

Contact tracing focuses on determining contacts of infected people in the past. In contrast, epidemiological modeling, which we consider in this work, forecasts the spread of infectious diseases in the future. Thus, epidemiological modeling goes *beyond* established contact tracing systems. They share some technical similarities (specifically, the exchange of encounter tokens), but on top of anomalously recording the contact graph, simulations have to be run on it. Similarly, presence tracing and hotspot detection are concerned with "flattening the curve" in relation to infections in the past. In contrast, epidemiological modeling is a tool for decision-makers to evaluate the efficacy of containment measures like social distancing in the future, allowing them to "get ahead of the wave".

B. *Epidemiological Modeling*

a) Disease Modeling: There are various ways to model a disease mathematically [23], [40]–[46]. Popular compartment models use a few continuous variables linked by differential equations to capture disease spread. In the SEIR model [46], [47], individuals are assigned to four compartments: susceptible (S), exposed (E), infectious (I), and recovered (R). These models are useful for understanding macroscopic trends and are widely used in epidemiological research [48], [49]. However, they condense complex individual behaviors into a few variables, limiting predictive power [50]. In contrast, agent-based epidemiological models [51] simulate the spread by initializing numerous agents with individual properties (e.g., location, age) and interaction rules. This allows for more realistic disease transmission simulations by modeling individual behaviors. Combining agent-based models with compartment models enhances the realism and accuracy of disease forecasting. Simulations with varying parameters, like interaction reductions or targeted vaccinations, are run to predict the effects of different policy interventions.

A key challenge is modeling agents' contact behaviors. Older models used survey-based contact matrices to estimate average contacts within age ranges [26], which improved over uniform assumptions but still fell short. Aggregated network statistics can't replicate real network dynamics, including super-spreaders with numerous contacts [52]. Ideally, epidemiologists would like to use real-world contact graphs of all individuals, but this is often challenging due to privacy concerns.

b) Contact Tracing for Privacy-Preserving Epidemiological Modeling: If contact information collected through contact tracing apps was centralised, an up-to-date full contact graph could be constructed for epidemiological simulations. However, contact information is highly sensitive and should not be shared. Contact information collected via mobile phones can reveal who, when, and whom people meet, which is sensitive and must be protected. Beyond, such information also enables to derive indications about the financial situation [53] and personality [54]. One can think about many more examples: By knowing which medical experts are visited by a person, information about the health condition can be anticipated; contact with members of a religious minority as well as visits to places related to religion might reveal a religious orientation, etc. Thus, it would be ideal for enabling precise epidemiological simulations without leaking individual contact information.

One way to achieve privacy-preserving epidemiological modeling using contact tracing apps is by allowing each participant's device to share its contact information secretly with a set of non-colluding servers. These servers can then run simulations using secure multi-party computation (MPC). Araki et al. [55] demonstrated efficiently running graph algorithms on secret shared graphs via MPC. However, despite the common non-collusion assumption in the crypto community, public trust issues may arise if all contact information is disclosed once servers collude. To address this, RIPPLE distributes trust by enabling participants to keep their contact information local while anonymously sending messages to each other to simulate the disease spread. Only aggregated simulation results are shared with research institutes, ensuring no direct identity or contact data is disclosed. This method resembles Federated Learning [29] and the contact tracing designs by Apple and Google.[2] This distributed design can increase trust and facilitate broad adoption of the system.

To the best of our knowledge, RIPPLE is the first framework that allows executing any agent-based compartment model on the distributed real contact graph while maintaining privacy.

III. The RIPPLE Framework

RIPPLE's primary goal is to enable the evaluation of the impact of multiple combinations of potential containment measures defined by epidemiologists and the government, and to find a balance between the drawbacks and benefits of those measures, rather than to deploy the measures in "real-life" first and then analyse the impact afterwards. Such measures may include, for example, the requirement to wear face masks in public places, restrictions on the number of people allowed to congregate, the closure of specific institutions and stores, or even complete curfews and lockdowns within specific regions.

Participants in RIPPLE collect personal encounter data anonymously and store it locally on their mobile devices, similar to privacy-preserving contact tracing apps. For epidemiological modeling, RIPPLE must derive a contact graph without leaking sensitive personal information to simulate disease spread over a specific period, like two weeks. Most countries have a 6-hour period at night when most people are asleep, and their mobile devices are idle, connected to WiFi, and possibly charging—an ideal time for running RIPPLE simulations. Medical experts can then analyze the results to understand the disease better, and political decision-makers can identify the most effective containment measures to implement.

To acquire representative and up-to-date physical encounter data, widespread public usage of RIPPLE would be ideal. One way to encourage this is to piggyback RIPPLE on most countries' official contact tracing applications. On the other hand, politicians can motivate residents beyond the intrinsic incentive of supporting public health by coupling the use of RIPPLE with additional benefits such as discounted or free travel passes.

[2] https://covid19.apple.com/contacttracing

A. System and Threat Model

RIPPLE comprises of p participants, denoted collectively by $\mathcal{P}$, a research institute RI who is in charge of the epidemiological simulations, and a set of MPC servers $\mathcal{C}$ responsible for anonymous communication among the participants.

We assume that the research institute and MPC servers are semi-honest [56], meaning they follow protocol specifications correctly while attempting to gather additional information. These semi-honest MPC servers also establish an anonymous communication channel. We discuss the security of the anonymous communication channel in more detail in §B-C. A protocol is secure if nothing is leaked beyond what can be inferred from the output. While the semi-honest security model is not the strongest, it offers a good trade-off between privacy and efficiency, making it popular in practical privacy-preserving applications such as privacy-preserving machine learning [57]–[59], genome/medical research [60]–[62], and localization services [63], [64].. This model protects against passive attacks by curious administrators and accidental data leakage and often serves as a foundation for developing protocols with stronger privacy guarantees [65], [66]. We consider this a reasonable assumption, as the research institute and servers will be controlled by generally trusted entities such as governments or public medical research centers, potentially collaborating with NGOs like the EFF[3] or the CCC[4].

Given the widespread interest in discovering effective containment measures, we expect a high level of intrinsic motivation among participants for successful epidemiological modeling. However, assuming complete honesty from millions of potential participants is impractical. Therefore, we incorporate a client-malicious security model [67], [68] within $\mathcal{P}$, covering the possibility of some participants being malicious and deviating from the protocol to gain extra information. Malicious behavior could also disrupt or compromise the accuracy of the simulation. However, our focus here is on addressing deviations aimed at information gain. Tab. I summarises the notations used in this work.

B. Phases of RIPPLE

RIPPLE is divided into four phases as shown in Fig. 1: i) Token Generation, ii) Simulation Initialization, iii) Simulation Execution, and iv) Result Aggregation. While our framework can be applied to any compartment-based epidemiological modeling of any infectious disease (cf. §II-B), we explain RIPPLE using the prevalent Covid-19 virus and the SEIR model [47], [69] as a running example. For simplicity, we assume that an app that emulates RIPPLE is installed on each participant's mobile device and that the participants locally enter attributes such as workplace, school, regular eateries, and cafes in the app after installing the app.

[3] https://www.eff.org [4] https://www.ccc.de/en/

	Parameter	Description
Entities	$\mathcal{P}$	Set of all participants; $\mathcal{P} = \{\mathcal{P}_1, \ldots, \mathcal{P}_\mathsf{p}\}$
	RI	Research Institute
	$\mathcal{C}$	Communication Servers $\{\mathsf{S}_0, \mathsf{S}_1, \mathsf{S}_2\}$
Simulations	$\mathsf{param}_\mathsf{sim}$	simulation parameters defined by RI
	N_sim	# distinct simulations (executed in parallel)
	N_step	# steps per simulation
	$\mathsf{class}_\mathsf{inf}$	infection classes; $\mathsf{class}_\mathsf{inf} = \{\mathsf{class}_\mathsf{inf}^1, \ldots, \mathsf{class}_\mathsf{inf}^{N_\mathsf{inf}}\}$
	I_i^s	$\mathcal{P}_i$'s infection class in simulation step $\mathsf{s} \in [0, N_\mathsf{step}]$
	$\mathcal{E}_i$	Encounter tokens of $\mathcal{P}_i$
	E_i^max	#max. encounters by $\mathcal{P}_i$ in pre-defined time interval
	E^avg	average number of encounters
Protocols	κ	computational security parameter $\kappa = 128$
	r_e	Unique token for encounter $e \in [0, E^\mathsf{max}]$
	$\delta_i^{r_e}$	$\mathcal{P}_i$'s infection likelihood w.r.t token r_e
	Δ_i	$\mathcal{P}_i$'s cumulative infection likelihood
	m_i^e	metadata of an encounter e by $\mathcal{P}_i$
	$(\mathrm{pk}_i, \mathrm{sk}_i)$	$\mathcal{P}_i$'s public/private key pair
	$\sigma_i^e.$	$\mathcal{P}_i$'s signature on message about encounter e

TABLE I: Notations used in RIPPLE.

Fig. 2 summarises the phases of the RIPPLE framework in the context of a single simulation setting and we give details below. Multiple simulations can be executed in parallel. The concrete number of simulation runs with the same parameters or different parameters should be determined by epidemiologists. Note that simulations are run on collected data, e.g., from the last days, and not on real-time encounter information. This combines efficiency requirements with maximally up-to-date encounter information.

Protocol RIPPLE

① - Token Generation

- $\mathcal{P}_i \in \mathcal{P}$ executes $\mathcal{F}_\mathsf{gen}$ all the time (on its mobile device), collecting encounter data of the form (r_e, m_e) with $e < E_i^\mathsf{max}$.

② - Simulation Initialization

- $\mathcal{P}_i \in \mathcal{P}$ receives $\mathsf{param}_\mathsf{sim}$ from RI and locallysets $I_i^1 = I_i^\mathsf{init}$.

③ - Simulation Execution

For each simulation step $\mathsf{s} \in [N_\mathsf{step}]$, $\mathcal{P}_i \in \mathcal{P}$ execute:

- Filter out encounters using $\mathsf{param}_\mathsf{sim}$ to obtain encounter set $\mathcal{E}_i^\mathsf{s}$.
- For each token $r_e \in \mathcal{E}_i^\mathsf{s}$, compute the infection likelihood $\delta_i^{r_e}$ locally using the formula from RI.
- Invoke $\mathcal{F}_\mathsf{esim}$ with the input $\{\delta_i^{r_e}\}_{r_e \in \mathcal{E}_i^\mathsf{s}}$ and obtain $\Delta_i^\mathsf{s} = \sum_{r_e \in \mathcal{E}_i^\mathsf{s}} \hat{\delta}_i^{r_e}$.
- Update the infection class I_i^s using Δ_i^s and the guidelines from RI.

④ - Result Aggregation

For each simulation step $\mathsf{s} \in [N_\mathsf{step}]$, execute:

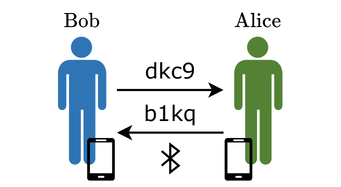
- $\mathcal{P}_i \in \mathcal{P}$ prepares $\{\mathsf{v}_\mathsf{i}^1, \ldots, \mathsf{v}_\mathsf{i}^{\mathsf{N}_\mathsf{inf}}\}^\mathsf{s}$ with $\mathsf{v}_\mathsf{i}^\mathsf{k} = 1$ if $I_i^\mathsf{s} = \mathsf{class}_\mathsf{inf}^\mathsf{k}$ and $\mathsf{v}_\mathsf{i}^\mathsf{k} = 0$ otherwise, for $k \in [N_\mathsf{inf}]$.
- Invoke $\mathcal{F}_\mathsf{agg}$ with inputs $\{\mathsf{v}_\mathsf{i}^1, \ldots, \mathsf{v}_\mathsf{i}^{\mathsf{N}_\mathsf{inf}}\}^\mathsf{s}$ to enable RI obtain the tuple $\{\mathsf{C}_\mathsf{inf}^1, \ldots, \mathsf{C}_\mathsf{inf}^{\mathsf{N}_\mathsf{inf}}\}^\mathsf{s}$, where $\mathsf{C}_\mathsf{inf}^\mathsf{k} = \sum_{\mathcal{P}_i \in \mathcal{P}} \mathsf{v}_\mathsf{i}^\mathsf{k}$ for $k \in [N_\mathsf{inf}]$.

Fig. 2: RIPPLE Framework (for one simulation setting).

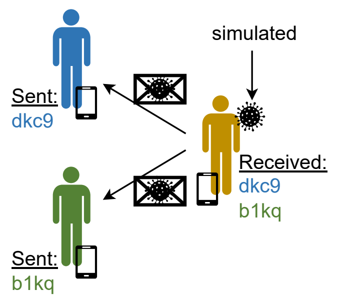
① - Token Generation: During a physical encounter, participants exchange data via Bluetooth LE to collect anonymous encounter information (Fig. 3a), similar to contact tracing [12], [70]. These tokens are stored locally on the users' devices and do not reveal any sensitive information (i.e., identifying information) about the individuals involved. In addition to these tokens, the underlying application will collect additional information on the context of the encounter known as "metadata" for simulation purposes. This varies depending on the underlying instantiation of the protocol and can include details such as duration, proximity, time, and location. The metadata can include or exclude different encounters in the simulation phase, allowing the effect of containment measures to be modelled (e.g., restaurant closings by excluding all encounters that happened in restaurants). The token generation phase is not dependent on the simulation phase, so no simulation-dependent infection data is exchanged. The token generation phase is modelled as an ideal functionality $\mathcal{F}_\mathsf{gen}$ that will be instantiated later in §IV.

Running Example: Assume that a participant, Alice, takes the bus to pick up her daughter from school. There are several other people on this bus – for simplicity, we call them $\mathsf{Bob}_1, \ldots, \mathsf{Bob}_\mathsf{x}$. As part of the token generation phase, Alice's phone exchanges unique anonymous tokens with the devices of the different Bobs. Now, two weeks later, it is night, and the national research institute (RI) wants to run a simulation covering 14 days to see how closing all schools would affect the spread of the disease. To accomplish this, the RI notifies all registered participants' applications to run a simulation using encounter data from the previous two weeks.

② - Simulation Initialization: The research institute RI initiates the simulation phase by sending a set of parameters, denoted by $\mathsf{param}_\mathsf{sim}$, to the participants in $\mathcal{P}$. The goal is to "spread" a fictitious infection across N_sim different simulation settings. To



(a) Token Generation



(b) Simulation

Fig. 3: Token Generation and Simulation phases in RIPPLE.

begin a simulation, each participant $\mathcal{P}_i$ is assigned to an infection class $I_i^{\mathsf{init}} \in \mathsf{class}_{\mathsf{inf}}$ (e.g., $\{\mathrm{S}\}$usceptible, $\{\mathrm{E}\}$xposed, $\{\mathrm{I}\}$nfectious, $\{\mathrm{R}\}$ecovered for the SEIR model) as specified in $\mathsf{param}_{\mathsf{sim}}$. For each individual simulation, $\mathsf{param}_{\mathsf{sim}}$ defines a set of containment measures, such as school closings and work from home, which the participants will use as filters to carry out the simulation in the next stage. In addition, RI publishes a formula to calculate the infection likelihood δ. The likelihood is determined by several parameters in the underlying modeling, such as encounter distance and time. For example, this likelihood might range from 0 (no chance of infection) to 100 (certain to get infected).

Running Example: Assume Alice is designated as infectious, while Bob_1 is designated as susceptible by RI. The other participants $\text{Bob}_2, \ldots,$ Bob_x are also assigned to an infection class (S, E, I, or R). To simulate containment measures, the RIPPLE-app now employs filters defined in $\mathsf{param}_{\mathsf{sim}}$. Using the information provided by the participants[5], the application may automatically filter out encounters that would not happen if a containment measure were in place, such as encounters in school while simulating school closings.

③ - Simulation Execution: Once RI initialises the simulation, N_{step} simulation steps (3a, 3b, 3c in Fig. 1) are performed for each of the N_{sim} simulation settings (e.g., $N_{\mathsf{step}} = 14$ days). Without loss of generality, consider the first simulation step and let $N_{\mathsf{sim}} = 1$. The simulation proceeds as follows:

1) Participant $\mathcal{P}_i \in \mathcal{P}$ filters out the relevant encounters based on the containment measures defined by RI. Let the set $\mathcal{E}_i$ represent the corresponding encounter tokens.

2) For each token $r_e \in \mathcal{E}_i$, $\mathcal{P}_i$ computes the infection likelihood $\delta_i^{r_e}$ using the formula from RI, i.e., the probability that $\mathcal{P}_i$ infects the participant met during the encounter with identifier token r_e.

3) Participants use the likelihood values δ obtained in the previous step to execute an ideal functionality called $\mathcal{F}_{\mathsf{esim}}$, which allows them to communicate the δ values anonymously through a set of MPC servers $\mathcal{C}$. Furthermore, it allows each participant $\mathcal{P}_j$ to receive a cumulative infection likelihood, denoted by Δ_j, based on all of the encounters they had on the day being simulated, i.e., $\Delta_j = \sum_{r_e \in \mathcal{E}_j} \hat{\delta}_j^{r_e}$. In this case, $\hat{\delta}_j^{r_e}$ denotes the infection likelihood computed by participant $\mathcal{P}_f$ and communicated to $\mathcal{P}_j$ for an encounter between $\mathcal{P}_f$ and $\mathcal{P}_j$ with identifier token r_e. As will be discussed later in §III-C, $\mathcal{F}_{\mathsf{esim}}$ must output the cumulative result rather than individual infection likelihoods because the latter can result in a breach of privacy.

4) Following the guidelines set by the RI, $\mathcal{P}_j$ updates its infection class I_j using the cumulative infection likelihood Δ_j acquired in the previous step.

These steps above are repeated for each of the N_{step} simulation steps in order and across all the N_{sim} simulation settings.

Running Example: Let the simulated containment measure be the closure of schools. As Alice is simulated to be infectious, Alice's phone computes the infection likelihood for every single encounter it recorded on the day exactly two weeks ago (Day 1) *except* those that occurred at her daughter's school. Then, Alice's phone combines the computed likelihood of each encounter with the corresponding unique encounter token to form tuples, which are then sent to the servers instantiating the anonymous communication channel. Using the encounter token as an address, the servers anonymously forward the likelihood to the person Alice has met, for example, Bob_1 (cf. Fig. 3b). Likewise, Bob_1 receives one message from each of the other participants he encountered and obtains the corresponding likelihood information. Bob_1 aggregates all likelihoods he obtained from his encounters on Day 1 and checks the aggregated result to a threshold defined by the RI to see if he has been infected in the simulation[6].

④ - Result Aggregation: For a given simulation setting, each participant $\mathcal{P}_i \in \mathcal{P}$ will have its infection class I_i^{s} updated at the end of every simulation step $\mathsf{s} \in [N_{\mathsf{step}}]$. This phase allows RI to obtain each simulated time step's aggregated number of participants per class (e.g., #S, #E, #I, #R). For this, we rely on a *Secure Aggregation* functionality, denoted by $\mathcal{F}_{\mathsf{agg}}$, which takes a N_{inf}-tuple of the form $\{\mathsf{v}_\mathsf{i}^1, \ldots, \mathsf{v}_\mathsf{i}^{N_{\mathsf{inf}}}\}^{\mathsf{s}}$ from each participant for every simulation step s and outputs the aggregate of this tuple over all the p participants to RI. In this case,

[5] This may also include location data obtained from the mobile app., e.g., Check In and Journal fields in the Corona-Warn contact tracing app.

[6] Bob_1 obtains the aggregated likelihood in the actual protocol.

v_i^k is an indicator variable for the k-th infection class, which is set to one if $I_i^s = \mathsf{class}_{\mathsf{inf}}^k$ and zero otherwise. Secure aggregation [71]–[73], [73] is a common problem in cryptography these days, particularly in the context of federated learning, and there are numerous solutions proposed for various settings, such as using TEEs, a semi-trusted server aggregating ciphertexts under homomorphic encryption, or multiple non-colluding servers that aggregate secret shares. In this work, we consider $\mathcal{F}_{\mathsf{agg}}$ a black box that can be instantiated using existing solutions compatible with our framework.

Running Example: All participants will know their updated infection class at the end of Day 1's simulation round, and they will prepare a 4-tuple of the form $\{\mathsf{v}^S, \mathsf{v}^E, \mathsf{v}^I, \mathsf{v}^R\}$ representing their updated infection class in the SEIR model. Participants will then engage in a secure aggregation protocol that determines the number of participants assigned to each infection class, which is then delivered to the RI. Then, the second simulation round begins, replicating the procedure but using encounters from 13 days ago, i.e., Day 2. The RI holds the aggregated number of participants per day per class after simulating all 14 days, i.e., a simulation of how the disease would spread if all schools had been closed in the previous 14 days (cf. graph in Fig. 1).

C. Privacy Requirements

A private contact graph necessitates that participants remain unaware of any unconscious interactions. This means they cannot determine if they had unconscious contact with the same person more than once or how often they did. We remark that an insecure variant of RIPPLE, in which each participant $\mathcal{P}_i$ receives the infection likelihood $\hat{\delta}_i^e$ for all its encounters $e \in E_i$ separately, will not meet this condition.

a) **Linking Identities Attacks.:** To demonstrate this, observe that when running multiple simulations (with different simulation parameters $\mathsf{param}_{\mathsf{sim}}$) on the same day, participants will use the same encounter tokens and metadata from the token generation phase in each simulation. If a participant $\mathcal{P}_i$ (Alice) can see the infection likelihood $\hat{\delta}_i$ of each of her interactions, $\mathcal{P}_i$ can look for correlations between those likelihoods to see if another participant $\mathcal{P}_j$ (Bob) was encountered more than once. We call this a *Linking Identities Attack* and depict it in Fig. 4, where, for simplicity, the infection likelihood accepts just two values: 1 for high and 0 for low infection likelihood.

Consider the following scenario to help clarify the issue: Alice and Bob work together in the same office. As a result, they have numerous conscious encounters during working hours. However, in their spare time, they may be unaware that they are in the same location (e.g., a club) and may not want the other to know. Their phones constantly collect encounters even if they do not see each other. Assume the RI sent the participants a simple infection likelihood formula that returns 0 (not infected) or 1 (infected). Furthermore, since the data is symmetric, Alice and Bob have the same metadata (duration, distance, etc.) about their conscious and unconscious encounters. Let Bob be modelled as infectious in the first simulation. As a result, he will send a 1 for each (conscious and unconscious) encounter he had (including those with Alice). If multiple simulations are run on the same day (i.e., with the same encounters), Alice will notice that some encounters, specifically all conscious and unconscious encounters with Bob, always have the same infection likelihood: If Bob is not infectious, all will return a 0; if Bob is infectious, all will return a 1. Thus, even if Alice had unconscious encounters with Bob, she can detect the correlations between the encounters and, as a result, determine which unconscious encounters were most likely with Bob.

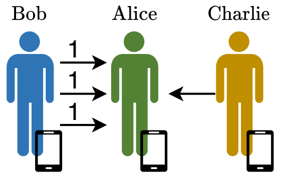


Fig. 4: Linking Identities Attack. Alice and Bob had several encounters, but Alice and Charlie only had one.

The more simulations she runs, the more confident she becomes. Since every participant knows the formula, this attack can also be extended to complex infection likelihood functions. While it may be more computationally expensive than the simple case, Alice can still identify correlations. This attack also works even if all of the encounters are unconscious. In such situations, Alice may be unable to trace related encounters to a single person (Bob), but she can infer that they were all with the same person (which is more than learning nothing). To avoid a Linking Identities attack, RIPPLE ensures that each participant receives an aggregation of all infection likelihoods of their encounters during the simulation. It cannot be avoided that participants understand that when "getting infected" some of their contacts must have been in contact with a (simulated) infectious participant. As this is only a simulated infection, we consider this leakage acceptable.

b) **Sybil Attack.:** While the Linking Identities Attack is already significant in the semi-honest security model, malicious participants may further circumvent aggregation mechanisms that prevent access to individual infection likelihoods. They could, for example, construct many *sybils*, i.e., multiple identities using several mobile devices, to collect each encounter one by one and then conduct a Linking Identities Attack with the information.

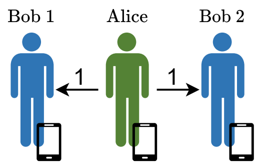


Fig. 5: Sybil Attack.

A registration system can be used to increase the costs of performing sybil attacks, i.e., to prevent an adversary from creating many identities. This assures that only legitimate users can join and participate in the simulation. In a closed ecosystem, such as a firm, this can be achieved by letting each member receive exactly one token to participate in the simulation. On a larger scale at the national level, one can let each citizen receive a token linked to a digital ID card. In such authentication mechanisms, anonymous credentials (cf. §A)can be used to ensure anonymity, and we leave the problem for future work.

c) **Inference Attacks.:** Note that although RIPPLE mimics the spirit of Federated Learning (FL) [29], it is not susceptible to so-called inference attacks [74], [75] in the same sense as FL. First, RIPPLE only reveals the final output (to a research institute RI) and no individual updates/results that ease information extraction. We, however, note that the analysis results provided to RI (cf. §III-A) contain information about the spread of the modeled disease in a specific population (otherwise it would be meaningless to run the simulation). The ideal functionality does not cover leakage from the final output but protects privacy during the computation. Thus, our security model does not consider anything that might be inferred from the output. We also argue that it is in the public interest to provide such aggregated information to the RI for deciding upon effective containment measures against infectious diseases.

IV. Instantiating $\mathcal{F}_{\mathsf{esim}}$

We propose two instantiations of $\mathcal{F}_{\mathsf{esim}}$ that cover different use cases and offer different trust-efficiency trade-offs. Our first design, RIPPLE$_{\mathsf{TEE}}$, is presented in the full version [76, §4.1] and assumes the presence of trusted execution environments (TEEs) such as ARM TrustZone on the mobile devices of the participants. In our second design, RIPPLE$_{\mathsf{PIR}}$ (§IV-B), we eliminate this assumption and provide privacy guarantees using cryptographic techniques such as PIR and anonymous communications.

A. *RIPPLE*$_{\mathsf{TEE}}$

The deployment of the entire operation in a single designated TEE would be a simple solution to achieving the ideal functionality $\mathcal{F}_{\mathsf{esim}}$. However, given the massive amount of data that must be handled in a large-scale simulation with potentially millions of users, TEE resource limitations are a prohibitive factor. Furthermore, since the TEE would contain the entire population's contact graph, it would be a single point of failure and an appealing target for an attack on TEE's known vulnerabilities. RIPPLE$_{\mathsf{TEE}}$ (Fig. 6), on the other hand, leverages the presence of TEEs in participants' mobile devices but in a decentralised manner, ensuring that each TEE handles only information related to the encounters made by the respective participant.

Before going into the details of RIPPLE$_{\mathsf{TEE}}$, we will go over the $\mathcal{F}_{\mathsf{anon}}$ functionality (cf. §B-C), which allows two participants, $\mathcal{P}_i$ and $\mathcal{P}_j$, to send messages to each other anonymously via a set of communication servers $\mathcal{C}$. The set $\mathcal{C}$ consists of one server acting as an entry node ($\mathcal{N}_{\mathtt{entry}}$), receiving messages from senders, and one server acting as an exit node ($\mathcal{N}_{\mathtt{exit}}$), forwarding messages to receivers. In $\mathcal{F}_{\mathsf{anon}}$, sender $\mathcal{P}_i$ does not learn to whom the message is sent, and receiver $\mathcal{P}_j$ does not learn who sent it. Similarly, the servers in $\mathcal{C}$ will be unable to relate receiver and sender of a message. Anonymous communication (cf. §A-A) is an active research area, e.g., [77]–[80], and $\mathcal{F}_{\mathsf{anon}}$ in RIPPLE$_{\mathsf{TEE}}$ can be instantiated using any of these efficient techniques.

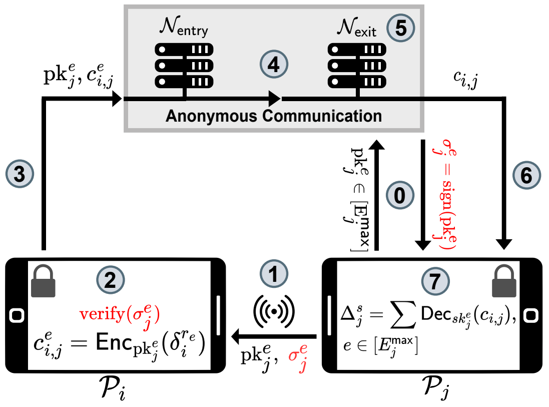

$\mathcal{N}_{\mathsf{entry}}$
$\mathcal{N}_{\mathsf{exit}}$
5
4
Anonymous Communication
$\mathrm{pk}_j^e, c_{i,j}^e$
$c_{i,j}$
3
$\mathrm{pk}_j^e \in [\mathrm{E}_j^{\mathsf{max}}]$
0
$\sigma_j^e = \mathrm{sign}(\mathrm{pk}_j^e)$
6
2
verify(σ_j^e)
$c_{i,j}^e = \mathsf{Enc}_{\mathrm{pk}_j^e}(\delta_i^{r_e})$
1
$\mathrm{pk}_j^e,\ \sigma_j^e$
7
$\Delta_j^s = \sum_{e \in [E_j^{\mathsf{max}}]} \mathsf{Dec}_{sk_j^e}(c_{i,j}),$
$\mathcal{P}_i$
$\mathcal{P}_j$

Fig. 6: RIPPLE$_{\mathsf{TEE}}$ Overview. Messages in red denote additional steps needed for malicious participants.

a) Token Generation: (steps ⓪ to ① in Fig. 6): During the pre-computation phase, the TEE of each participant $\mathcal{P}_i \in \mathcal{P}$ generates a list of fresh unique public/private key pairs $(\mathrm{pk}_i^e, \mathrm{sk}_i^e)$ for all possible encounters $e \in [E_i^{\mathsf{max}}]$. The keys, for example, can be generated and stored a day ahead of time. The newly generated public keys are then sent by $\mathcal{P}_i$'s TEE to the exit node $\mathcal{N}_{\mathtt{exit}}$ (step ⓪ in Fig. 6) to enable anonymous communication (cf. §B-C) via $\mathcal{F}_{\mathsf{anon}}$ later in the protocol's simulation part.

During a physical encounter e, $\mathcal{P}_i$ and $\mathcal{P}_j$ exchange two unused public keys pk_i^e and pk_j^e (step ① in Fig. 6). Simultaneously, both participants compute and record metadata m_e, such as the time, location, and duration of the encounter, and store this information alongside the received public key.

Additional measures are required for malicious participants to ensure that the participants are exchanging public keys generated by the TEEs: After obtaining the new public keys from $\mathcal{P}_i$, the exit node $\mathcal{N}_{\mathtt{exit}}$ goes one step further: It signs them and returns the signatures to $\mathcal{P}_i$ after checking that it is connecting directly with a non-corrupted TEE (step ⓪ in Fig. 6 and §A). During a physical encounter, $\mathcal{P}_j$ will provide the corresponding signature, denoted by σ_j^e along with pk_j^e so that the receiver $\mathcal{P}_i$ can verify that the key was correctly generated by $\mathcal{P}_j$'s TEE (step ② in Fig. 6).

b) Simulation Execution: (steps ② to ⑦ in Fig. 6): All local computations, including infection likelihood calculation and infection class updates, will be performed within the participants' TEEs. In detail, for each encounter e involving participants $\mathcal{P}_i$ and $\mathcal{P}_j$, the following steps are executed:

– $\mathcal{P}_i$'s TEE computes $\delta_i^{r_e}$ and encrypts it using the public key pk_j^e of $\mathcal{P}_j$ obtained during the token generation phase. Let the ciphertext be $c_{i,j}^e = \mathsf{Enc}_{\mathrm{pk}_j^e}(\delta_i^{r_e})$ (step ② in Fig. 6).
– $\mathcal{P}_i$'s TEE establishes a secure channel with the entry node $\mathcal{N}_{\texttt{entry}}$ of $\mathcal{C}$ via remote attestation and uploads the tuple $(\mathrm{pk}_j^e, c_{i,j}^e)$ (step ③ in Fig. 6).
– The tuple $(\mathrm{pk}_j^e, c_{i,j}^e)$ traverses through the servers in $\mathcal{C}$ and reaches the exit node $\mathcal{N}_{\texttt{exit}}$ (step ④ in Fig. 6, instantiation details for the anonymous communication channel are given in §B-C).
– If the public key pk_i^e has already been used in this simulation step[7], $\mathcal{N}_{\texttt{exit}}$ discards the tuple (step ⑤ in Fig. 6).
– Otherwise, $\mathcal{N}_{\texttt{exit}}$ uses pk_j^e to identify the recipient $\mathcal{P}_j$ and sends the ciphertext $c_{i,j}^e$ to $\mathcal{P}_j$ (step ⑥ in Fig. 6).

After receiving the ciphertexts for all of the encounters, $\mathcal{P}_j$'s TEE decrypts them and aggregates the likelihoods to produce the desired output (step ⑦ in Fig. 6).

1) Security of RIPPLE$_{\mathsf{TEE}}$*.:* First, we consider the case of semi-honest participants. During the token generation phase, since the current architecture in most mobile devices does not allow direct communication with a TEE while working with Bluetooth LE interfaces, participant $\mathcal{P}_i$ can access both the sent and received public keys before they are processed in the TEE. However, unique keys are generated per encounter and do not reveal anything about an encounter's identities due to the security of the underlying $\mathcal{F}_{\mathsf{gen}}$ functionality, which captures the goal of several contact tracing apps in use.

The $\mathcal{F}_{\mathsf{anon}}$ functionality, which implements an anonymous communication channel utilising the servers in $\mathcal{C}$, aids in achieving *contact graph privacy* by preventing participants from learning to/from whom they are sending/receiving messages. While the entry node learns who sends a message, it does not learn who receives them. Similarly, the exit node $\mathcal{N}_{\texttt{exit}}$ has no knowledge of the sender but learns the recipient using the public key. Regarding *confidentiality*, participants in RIPPLE$_{\mathsf{TEE}}$ have no knowledge of the messages being communicated because they cannot access the content of the TEEs and the TEEs communicate directly to the anonymous channel. Furthermore, servers in $\mathcal{C}$ will not have access to the messages as they are encrypted.

For the case of malicious participants, they could send specifically crafted keys during the token generation phase instead of the ones created by their TEE. However, this will make the signature verification fail and the encounter will get discarded. Furthermore, a malicious participant may reuse public keys for multiple encounters. This manipulation, however, will be useless because the exit node $\mathcal{N}_{\texttt{exit}}$ checks that each key is only used once before forwarding messages to participants. During the simulation phase, all data and computation are handled directly inside the TEEs of the participants, so no manipulation is possible other than cutting the network connection, i.e., dropping out of the simulation, ensuring *correctness*. Dropouts occur naturally when working with mobile devices and have no effect on privacy guarantees.

B. *RIPPLE*$_{\mathsf{PIR}}$

In the following, we show how to get rid of RIPPLE$_{\mathsf{TEE}}$'s assumption of each participant having a TEE on their mobile devices. If we simply remove the TEE part of RIPPLE$_{\mathsf{TEE}}$ and run the same protocol, decryption and aggregation of a participant's received infection likelihoods would be under their control. Thus, the individual infection likelihoods of all encounters would be known to them, leaking information about the contact graph (cf. §III-C). To get around this privacy issue, we need to find a way to aggregate the infection likelihoods so that the participants can only derive the sum, not individual values.

Functionality $\mathcal{F}_{\mathsf{pirsum}}$

$\mathcal{F}_{\mathsf{pirsum}}$ interacts with M servers, denoted by $\mathcal{C}$, and participant $\mathcal{P}_i \in \mathcal{P}$.

Input: $\mathcal{F}_{\mathsf{pirsum}}$ receives τ indices denoted by $\mathcal{Q} = \{q_1, \ldots, q_\tau\}$ from $\mathcal{P}_i$ and a database D from $\mathcal{C}$.

Output: $\mathcal{F}_{\mathsf{pirsum}}$ sends $\sum_{j=1}^{\tau} \mathsf{D}[q_j]$ to $\mathcal{P}_i$ as the output.

Fig. 7: Ideal functionality for PIR-SUM (semi-honest).

Private Information Retrieval (PIR, cf. §A) is one promising solution for allowing participants to retrieve infection likelihoods sent to them anonymously. PIR enables the private download of an item from a public database D held by M servers without leaking any information to the servers, such as which item is queried or the content of the queried item. However, classical PIR is unsuitable for our needs because we need to retrieve the sum of τ items from the database rather than the individual ones. As a result, we introduce the ideal functionality $\mathcal{F}_{\mathsf{pirsum}}$ (Fig. 7), which is similar to a conventional PIR functionality but returns the sum of τ queried locations of the database as a result. For the remainder of this section, we consider $\mathcal{F}_{\mathsf{pirsum}}$ to be an ideal black-box and will discuss concrete instantiations in §VWe now detail the changes needed in the token generation phase to make it compatible with the rest of the RIPPLE$_{\mathsf{PIR}}$ protocol.

Token Generation (step ① in Fig. 8): During a physical encounter e among participants $\mathcal{P}_i$ and $\mathcal{P}_j$, they generate and exchange unique random tokens denoted by r_i^e and r_j^e. Both participants, like in

[7] This step is not required for semi-honest participants.

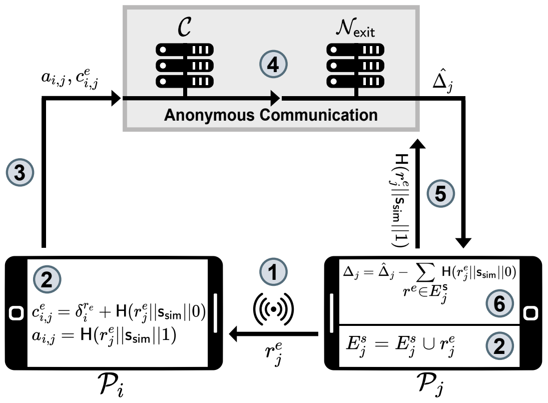

$\mathcal{C}$
$\mathcal{N}_{\text{exit}}$
4
Anonymous Communication
$a_{i,j}, c^e_{i,j}$
$\hat{\Delta}_j$
3
$\mathsf{H}(r^e_j||\mathsf{s}_{\mathsf{sim}}||1)$
5
2
$c^e_{i,j} = \delta_i^{r_e} + \mathsf{H}(r^e_j||\mathsf{s}_{\mathsf{sim}}||0)$
$a_{i,j} = \mathsf{H}(r^e_j||\mathsf{s}_{\mathsf{sim}}||1)$
1
r^e_j
$\Delta_j = \hat{\Delta}_j - \sum_{r^e \in E^{\mathsf{s}}_j} \mathsf{H}(r^e_j||\mathsf{s}_{\mathsf{sim}}||0)$
6
$E^s_j = E^s_j \cup r^e_j$
2
$\mathcal{P}_i$
$\mathcal{P}_j$

Fig. 8: RIPPLE$_{\mathsf{PIR}}$ Overview.

RIPPLE$_{\mathsf{TEE}}$, also record the metadata m^e. Thus, at the end of a simulation step $\mathsf{s} \in [N_{\mathsf{step}}]$ (e.g., a day), $\mathcal{P}_i$ holds a list of sent encounter tokens, denoted by $E^{\mathsf{s}}_i = \{r^e_i\}_{e \in \mathcal{E}_i}$, where $\mathcal{E}_i$ is the complete (sent/received) set of encounters of $\mathcal{P}_i$, and a list of received tokens, denoted by $R^{\mathsf{s}}_i = \{r^e_j\}_{e \in \mathcal{E}_i}$. Looking ahead, these random tokens will be used as addresses to communicate the corresponding infection likelihood among the participants.

Simulation Execution (steps ② to ⑥ in Fig. 8): Local computations such as encounter filtering and infection likelihood calculation proceed similarly to RIPPLE$_{\mathsf{TEE}}$ but without TEE protection. The steps for an encounter e among $\mathcal{P}_i$ and $\mathcal{P}_j$ are as follows:

- $\mathcal{P}_i$ blinds each infection likelihood $\delta_i^{r_e}$ computed with the corresponding random token r^e_j received from $\mathcal{P}_j$ and obtains the ciphertext $c^e_{i,j} = \delta_i^{r_e} + \mathsf{H}(r^e_j||\mathsf{s}_{\mathsf{sim}}||0)$. In addition, it computes the destination address for the ciphertext as $a_{i,j} = \mathsf{H}(r^e_j||\mathsf{s}_{\mathsf{sim}}||1)$. Here, $\mathsf{H}()$ is a cryptographic hash function and $\mathsf{s}_{\mathsf{sim}} \in [N_{\mathsf{sim}}]$ denotes the current simulation setting. (step ② in Fig. 8)
  - $\mathsf{s}_{\mathsf{sim}}$ is used in $\mathsf{H}()$ to ensure that distinct (ciphertext, address) tuples are generated for the same encounters across multiple simulation settings, preventing the exit node $\mathcal{N}_{\texttt{exit}}$ from potentially linking messages from different simulations.
- $\mathcal{P}_i$ sends the tuple $(c^e_{i,j}, a_{i,j})$ anonymously to $\mathcal{N}_{\texttt{exit}}$ with the help of the servers in $\mathcal{C}$. $\mathcal{N}_{\texttt{exit}}$ discards all the tuples with the same address field $(a_{i,j})$ (steps ③ to ④ in Fig. 8). The instantiation details for the anonymous communication channel are given in §B-C).

As a server in $\mathcal{C}$, $\mathcal{N}_{\texttt{exit}}$ locally creates the database D for the current simulation step using all of the $(a_{i,j}, c^e_{i,j})$ tuples received (part of step ④ in Fig. 8). A naive solution of inserting $c^e_{i,j}$ using a simple hashing of the address $a_{i,j}$ will not provide an efficient solution in our case since we require only one message to be stored in each database entry to have an injective mapping between addresses and messages. This is required for the receiver to download the messages sent to them precisely. Simple hashing would translate to a large database size to ensure a negligible probability of collisions. Instead, in RIPPLE$_{\mathsf{PIR}}$, we use a novel variant of a garbled cuckoo table that we call arithmetic garbled cuckoo table (AGCT, see below), with $a_{i,j}$ as the insertion key for the database.

Once the database D is created, $\mathcal{N}_{\texttt{exit}}$ sends it to the other servers in $\mathcal{C}$ based on the instantiation of $\mathcal{F}_{\mathsf{pirsum}}$ (cf. §V). Each $\mathcal{P}_j \in \mathcal{P}$ will then participate in an instance of $\mathcal{F}_{\mathsf{pirsum}}$ with the servers in $\mathcal{C}$ acting as PIR servers holding the database D. $\mathcal{P}_j$ uses the addresses of all its sent encounters from E^{s}_j, namely $\mathsf{H}(r^e||\mathsf{s}_{\mathsf{sim}}||1)$, as the input to $\mathcal{F}_{\mathsf{pirsum}}$ and obtains a blinded version of the cumulative infection likelihood, denoted by $\hat{\Delta}_j$, as the output (step ⑤ in Fig. 8). The cumulative infection likelihood, Δ_j, is then unblinded as

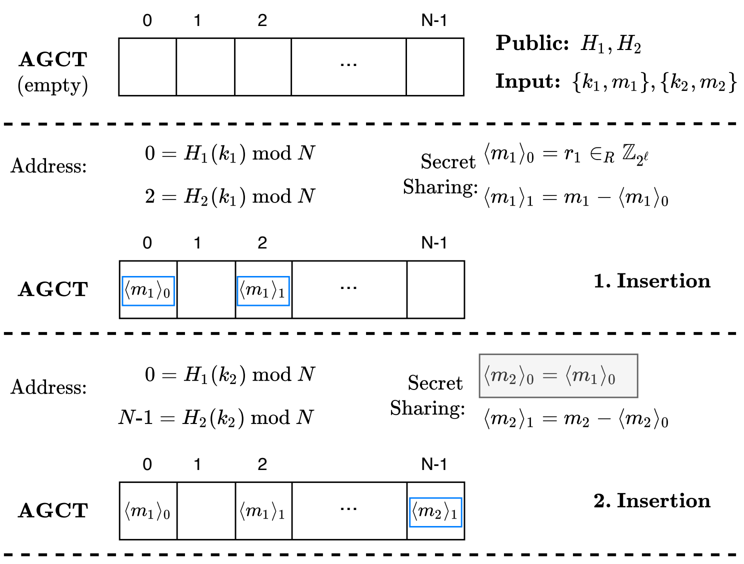
$$\Delta_j = \hat{\Delta}_j - \sum_{r^e \in \; E^{\mathsf{s}}_j} \mathsf{H}(r^e||\mathsf{s}_{\mathsf{sim}}||0)$$

concluding the current simulation step (step ⑥ in Fig. 8).

1) Security of RIPPLE$_{\mathsf{PIR}}$: Except for the database constructions at exit node $\mathcal{N}_{\texttt{exit}}$ and the subsequent invocation of the $\mathcal{F}_{\mathsf{pirsum}}$ functionality for the cumulative infection likelihood computation, the security guarantees for semi-honest participants in RIPPLE$_{\mathsf{PIR}}$ are similar to those of RIPPLE$_{\mathsf{TEE}}$. Unlike RIPPLE$_{\mathsf{TEE}}$, $\mathcal{N}_{\texttt{exit}}$ in RIPPLE$_{\mathsf{PIR}}$ will be unable to identify the message's destination from the address because it will be known only to the receiving participant. Furthermore, each participant obtains the cumulative infection likelihood directly via the $\mathcal{F}_{\mathsf{pirsum}}$ functionality, ensuring that $\mathcal{N}_{\texttt{exit}}$ cannot infer the participant's encounter details and, thus, *contact graph privacy*.

Malicious participants in RIPPLE$_{\mathsf{PIR}}$, as opposed to RIPPLE$_{\mathsf{TEE}}$,can tamper with the protocol's correctness by providing incorrect inputs. However, as stated in the threat model in §III, we assume that malicious participants in our framework will not tamper with the correctness and will only seek additional information. A malicious participant could re-use the same encounter token for multiple encounters during token generation, causing the protocol to generate multiple tuples with the same address. However, as the protocol states, $\mathcal{N}_{\texttt{exit}}$ will discard all such tuples, removing the malicious participant from the system. Another potential information leakage caused by the participant's aforementioned action is that the entry point of the anonymous communication channel can deduce that multiple participants encountered the same participant. This is not an issue in our protocol because we instantiate the $\mathcal{F}_{\mathsf{anon}}$ functionality using a 3-server oblivious shuffling scheme (cf. §B-C), where all the servers except $\mathcal{N}_{\texttt{exit}}$ will not see any messages in the clear, but only see secret shares.

2) Arithmetic Garbled Cuckoo Table (AGCT): We design a variant of garbled cuckoo tables ([81], cf. §A)that we term arithmetic garbled cuckoo table



AGCT (empty)
0 1 2 ... N-1
Public: H_1, H_2
Input: $\{k_1, m_1\}, \{k_2, m_2\}$
Address: $0 = H_1(k_1) \bmod N$
$2 = H_2(k_1) \bmod N$
Secret Sharing: $\langle m_1\rangle_0 = r_1 \in_R \mathbb{Z}_{2^\ell}$
$\langle m_1\rangle_1 = m_1 - \langle m_1\rangle_0$
AGCT $\langle m_1\rangle_0$ $\langle m_1\rangle_1$
1. Insertion
Address: $0 = H_1(k_2) \bmod N$
$N\text{-}1 = H_2(k_2) \bmod N$
Secret Sharing: $\langle m_2\rangle_0 = \langle m_1\rangle_0$
$\langle m_2\rangle_1 = m_2 - \langle m_2\rangle_0$
AGCT $\langle m_1\rangle_0$ $\langle m_1\rangle_1$ $\langle m_2\rangle_1$
2. Insertion

Fig. 9: Insertion into the Arithmetic Garbled Cuckoo Table (AGCT). H_1 and H_2 are two hash functions. $\{k_1, m_1\}$ and $\{k_2, m_2\}$ are key-value pairs where the key is used to determine the data address in the database.

(AGCT) to reduce the size of the PIR database while ensuring a negligible collision likelihood. It uses arithmetic sharing instead of XOR-sharing to share database entries and present the details next.

Assume two key-message pairs $\{k_1, m_1\}$ and $\{k_2, m_2\}$[8] are to be added to database D with N bins, using two hash function H_1 and H_2 to determine the insertion addresses.

1. Insertion of $\{k_1, m_1\}$:
   a) Compute $a_1 = \mathsf{H}_1(k_1) \mod N$ and $a_2 = \mathsf{H}_2(k_1) \mod N$.
   b) Check if bins a_1 and a_2 are already occupied. Let's assume this is not the case.
   c) Compute the arithmetic sharing of the message m_1: $\langle m_1\rangle_0 = r_1 \in_R \mathbb{Z}_{2^\ell}$ and $\langle m_1\rangle_1 = m_1 - \langle m_1\rangle_0$.
   d) Insert $D[a_1] = \langle m_1\rangle_0$ and $D[a_2] = \langle m_1\rangle_1$.
2. Insertion of $\{k_2, m_2\}$:
   a) Compute $b_1 = \mathsf{H}_1(k_2) \mod N$ and $b_2 = \mathsf{H}_2(k_2) \mod N$.
   b) Check if bins b_1 and b_2 are already occupied. Let's assume $b_1 = a_1$, i.e., the first bin is already occupied, but bin b_2 is free.
   c) Compute the arithmetic sharing m_2 with $\langle m_2\rangle_0 = \langle m_1\rangle_0$ as $b_1 = a_1$. Then, the other share is $\langle m_2\rangle_1 = m_2 - \langle m_2\rangle_0$.
   d) Insert $D[b_1] = \langle m_2\rangle_0$ and $D[b_2] = \langle m_2\rangle_1$.

Double Collision: Now the question is how to handle the insertion of a database entry if both addresses determined by the two hash functions are already occupied. An easy solution is to pick different hash functions s.t. no double collision occurs for all n elements that shall be stored in the database. Alternatively, Pinkas et al. [81] demonstrate for a garbled cuckoo table how to extend the database by $d + \lambda$ bins, where d is the upper bound of double collisions and λ is an error parameter, such that double collisions occur with a negligible likelihood. For details, please refer to [81, §5].

[8] k corresponds to a key and m to a message in our application.

V. PIR-SUM: Instantiating $\mathcal{F}_{\mathsf{pirsum}}$

So far, the discussion has focused on RIPPLE as a generic framework composed of multiple ideal functionalities that could be efficiently instantiated using state-of-the-art privacy-enhancing technologies. In this section, we will use three semi-honest MPC servers to instantiate our novel $\mathcal{F}_{\mathsf{pirsum}}$ functionality (Fig. 7). In particular, we have three servers $\mathsf{S}_0, \mathsf{S}_1$, and S_2, and we design the $\mathsf{PIR}_{\mathsf{sum}}$ protocol to instantiate the $\mathcal{F}_{\mathsf{pirsum}}$ functionality.

The problem statement in our context is formally defined as follows: Participant $\mathcal{P}_i \in \mathcal{P}$ has a set of τ indices denoted by $\mathcal{Q} = \{q_1, \ldots, q_\tau\}$ and wants to retrieve $\mathsf{res} = \sum_{q \in \mathcal{Q}} \mathsf{D}[q]$. In this case, D is a database with N elements of ℓ-bits each that is held in the clear by both the servers S_1 and S_2. The server S_0 aids in the computation performed by the servers S_1 and S_2. Furthermore, we assume a one-time setup (cf. §B-A)among the servers and $\mathcal{P}_i$ that establishes shared pseudorandom keys among them to facilitate non-interactive generation of random values and, thus, save communication [55], [57], [59].

A. *Overview of* $\mathsf{PIR}_{\mathsf{sum}}$ *protocol*

At a high level, the idea is to use multiple instances of a standard 2-server PIR functionality [82], [83], denoted by $\mathcal{F}^{\mathsf{2S}}_{\mathsf{pir}}$, and combine the responses to get the sum of the desired blocks as the output. $\mathsf{D}^m = \mathsf{D} + m$ denotes a modified version of the database D in which every block is summed with the same ℓ-bit mask m, i.e., $\mathsf{D}^m[i] = \mathsf{D}[i] + m$ for $i \in [N]$. The protocol proceeds as follows:

- S_1 and S_2 non-interactively sample τ random mask values $\{m_1, \ldots, m_\tau\}$ such that $\sum_{j=1}^{\tau} m_j = 0$.
- $\mathsf{S}_1, \mathsf{S}_2$, and $\mathcal{P}_i$ execute τ instances of $\mathcal{F}_{\mathsf{pir}}^{\mathsf{2S}}$ in parallel, with servers using D^{m_j} as the database and $\mathcal{P}_i$ using q_j as the query for the j-th instance for $j \in [\tau]$.
- Let res_j denote the result obtained by $\mathcal{P}_i$ from the j-th $\mathcal{F}_{\mathsf{pir}}^{\mathsf{2S}}$ instance. $\mathcal{P}_i$ locally computes $\sum_{j=1}^{\tau} \mathsf{res}_j$ to obtain the desired result.

The details for instantiating $\mathcal{F}_{\mathsf{pir}}^{\mathsf{2S}}$ using the standard linear summation PIR approach [82] are provided in §C-A. The approach requires $\mathcal{P}_i$ to communicate $N \cdot \tau$ bits to the servers, which is further reduced in RIPPLE$_{\mathsf{PIR}}$ (cf. §V-C).

a) Malicious participants.: While it is simple to show that the above solution is adequate for semi-honest participants, malicious participants must be dealt with separately. A malicious participant, for example, could use the same query, say q_j, in all τ instances and retrieve only the block corresponding to q_j by dividing the result by τ. We present a simple verification scheme over the $\mathcal{F}_{\mathsf{pir}}^{\mathsf{2S}}$ functionality to prevent these manipulations.

For malicious participants, we want to ensure that $\mathcal{P}_i$ used a distinct vector $\vec{\mathsf{b}}$ (representing a PIR query q_j, cf. §C-A) during the τ parallel instances. One naive approach is to have S_1 and S_2 compute the bitwise-OR of all the τ bit query vectors $\vec{\mathsf{b}}_1, \ldots, \vec{\mathsf{b}}_\tau$, and then run a secure two-party computation protocol to compare the number of ones in the resultant vector to τ. We use the additional server S_0 to optimize this step further. S_1 and S_2 send randomly shuffled versions of their secret shared bit vectors to S_0, who reconstructs the shuffled vectors and performs the verification locally. This approach leaks no information to S_0 because it has no information about the underlying database D. The verification procedure is as follows:

– S_1 and S_2 non-interactively agree on a random permutation, denoted by π.

– S_u sends $\pi([\vec{\mathsf{b}}_j]_u)$ to S_0, $j \in [\tau]$, $u \in \{1, 2\}$.

– S_0 locally reconstructs $\pi(\vec{\mathsf{b}}_j) = \pi([\vec{\mathsf{b}}_j]_1) \oplus \pi([\vec{\mathsf{b}}_j]_2)$, for $j \in [\tau]$. If all the τ bit vectors are correctly formed and distinct, it sends `Accept` to S_1 and S_2. Else, it sends `abort`.

Note that the verification using $\mathcal{P}_0$ will incur a communication of $2\tau N$ bits among the servers. Furthermore, the above verification method can be applied to any instantiation of $\mathcal{F}_{\mathsf{pir}}^{\mathsf{2S}}$ that generates a boolean sharing of the query bit vector among the PIR servers and computes the response as described above, e.g., the PIR schemes of [82]–[84].

B. *Instantiating* $\mathcal{F}_{\mathsf{pirsum}}$

The formal protocol for PIR$_{\mathsf{sum}}$ in the case of malicious participants is provided in Fig. 10, assuming the presence of an ideal functionality $\mathcal{F}_{\mathsf{pir}}^{\mathsf{2S}}$ (as will be discussed in **HYB**$_2$ below). In PIR$_{\mathsf{sum}}$, the servers $\mathsf{S}_1, \mathsf{S}_2$ and the participant $\mathcal{P}_i$ run τ instances of $\mathcal{F}_{\mathsf{pir}}^{\mathsf{2S}}$ in parallel, one for each query $q \in \mathcal{Q}$. Following the execution, $\mathcal{P}_i$ receives $\mathsf{D}[q] + \mathsf{r}_q$ whereas S_u receives $\mathsf{r}_q, [q]_u$, for $u \in \{1, 2\}$ and $q \in \mathcal{Q}$. $\mathcal{P}_i$ then adds up the received messages to get a masked version of the desired output, i.e, $\sum_{q \in \mathcal{Q}} \mathsf{D}[q] + \mathsf{mask}_{\mathcal{Q}}$ with $\mathsf{mask}_{\mathcal{Q}} = \sum_{q \in \mathcal{Q}} \mathsf{r}_q$. $\mathsf{S}_1, \mathsf{S}_2$ compute $\mathsf{mask}_{\mathcal{Q}}$ in the same way.

Protocol PIR$_{\mathsf{sum}}$

Input(s): i) $\mathsf{S}_1, \mathsf{S}_2 : \mathsf{D}; |\mathsf{D}| = N$, ii) $\mathcal{P}_i : \mathcal{Q} = \{q_1, \ldots, q_\tau\}$, and iii) $\mathsf{S}_0 : \perp$.
Output: $\mathcal{P}_i : \mathsf{res} = \sum_{q \in \mathcal{Q}} \mathsf{D}[q]$ for distinct queries, else $\mathsf{res} = \perp$.

Computation

1. For each $q \in \mathcal{Q}$,
   a. $\mathsf{S}_1, \mathsf{S}_2$ and $\mathcal{P}_i$ invoke $\mathcal{F}_{\mathsf{pir}}^{\mathsf{2S}}$ (cf. **HYB**$_2$ in proof of Lemma 1) with the inputs D, q.
   b. Let $\mathsf{r}_q, [q]_u$ denote the output of S_u, for $u \in \{1, 2\}$ and $\mathsf{D}[q] + \mathsf{r}_q$ denote the output of $\mathcal{P}_i$.
2. $\mathcal{P}_i$ computes $\mathsf{res}' = \sum_{q \in \mathcal{Q}} (\mathsf{D}[q] + \mathsf{r}_q)$, while $\mathsf{S}_1, \mathsf{S}_2$ computes $\mathsf{mask}_{\mathcal{Q}} = \sum_{q \in \mathcal{Q}} \mathsf{r}_q$.
3. $\mathsf{S}_1, \mathsf{S}_2$ and S_0 invokes $\mathcal{F}_{\mathsf{vrfy}}$ on the secret shares of queries, denoted by $\{[q]_u\}_{q \in \mathcal{Q}, u \in \{1,2\}}$, to check the distinctness of the queries in $\mathcal{Q}$.
4. If $\mathcal{F}_{\mathsf{vrfy}}$ returns `Accept`, $\mathsf{S}_1, \mathsf{S}_2$ sends $\mathsf{mask}_{\mathcal{Q}}$ to $\mathcal{P}_i$, who computes $\mathsf{res} = \mathsf{res}' - \mathsf{mask}_{\mathcal{Q}}$. Otherwise, `abort`.

Fig. 10: PIR$_{\mathsf{sum}}$ Protocol.

The protocol could be completed by S_1 and S_2 sending $\mathsf{mask}_{\mathcal{Q}}$ to $\mathcal{P}_i$, then $\mathcal{P}_i$ unmasking its value to obtain the desired output. However, before communicating the mask, the servers must ensure that all queries in $\mathcal{Q}$ are distinct, as shown in $\mathcal{F}_{\mathsf{pirsum}}$ (Fig. 11). For this, $\mathsf{S}_1, \mathsf{S}_2$ use their share of the queries $q \in \mathcal{Q}$ and participate in a secure computation protocol with S_0. We capture this with an ideal functionality $\mathcal{F}_{\mathsf{vrfy}}$, which takes the secret shares of τ values from S_1 and S_2 and returns `Accept` to the servers if all of the underlying secrets are distinct. Otherwise, it returns `abort`.

1) Security of PIR$_{\mathsf{sum}}$ *Protocol:* Fig. 11 presents the ideal functionality for PIR$_{\mathsf{sum}}$ in the context of malicious participants. In this case, $\mathcal{F}_{\mathsf{pirsum}}$ first checks whether all the queries made by the participant $\mathcal{P}_i$ are distinct. If yes, the correct result is sent to $\mathcal{P}_i$; otherwise, $\perp$ is sent to $\mathcal{P}_i$.

Functionality $\mathcal{F}_{\mathsf{pirsum}}$

$\mathcal{F}_{\mathsf{pirsum}}$ interacts with servers in $\mathcal{C}$, and participant $\mathcal{P}_i \in \mathcal{P}$.

Input: $\mathcal{F}_{\mathsf{pirsum}}$ receives τ indices denoted by $\mathcal{Q} = \{q_1, \ldots, q_\tau\}$ from $\mathcal{P}_i$ and a database D from $\mathcal{C}$.

Computation: $\mathcal{F}_{\mathsf{pirsum}}$ sets $y = \sum_{j=1}^{\tau} \mathsf{D}[q_j]$ if all the queries in $\mathcal{Q}$ are distinct. Else, it sets $y = \perp$.
Output: $\mathcal{F}_{\mathsf{pirsum}}$ sends y to $\mathcal{P}_i$.

Fig. 11: PIR-SUM functionality (malicious participants).

Lemma 1. *Protocol* $\mathsf{PIR}_{\mathsf{sum}}$ *(Fig. 10) securely realises the* $\mathcal{F}_{\mathsf{pirsum}}$ *ideal functionality (Fig. 11) for the case of malicious participants in the* $\{\mathcal{F}_{\mathsf{pir}}^{\mathsf{2S}}, \mathcal{F}_{\mathsf{vrfy}}\}$*-hybrid model.*

Proof: The proof follows with a hybrid argument based on the three hybrids $\mathbf{HYB}_0$, $\mathbf{HYB}_1$, and $\mathbf{HYB}_2$ discussed below. Furthermore, any secure three-party protocol can be used to instantiate $\mathcal{F}_{\mathsf{vrfy}}$ in RIPPLE.

We use a standard 2-server PIR functionality, denoted by $\mathcal{F}_{\mathsf{pir}}^{\mathsf{2S}}$, to instantiate $\mathcal{F}_{\mathsf{pirsum}}$. The guarantees of $\mathcal{F}_{\mathsf{pir}}^{\mathsf{2S}}$, however, are insufficient to meet the security requirements of $\mathcal{F}_{\mathsf{pirsum}}$, so we modify $\mathcal{F}_{\mathsf{pir}}^{\mathsf{2S}}$ as a sequence of hybrids, denoted by $\mathbf{HYB}$. The modification is carried out in such a way that for a malicious participant $\mathcal{P}_i$, each hybrid is computationally indistinguishable from the one before it. As the first hybrid, $\mathcal{F}_{\mathsf{pir}}^{\mathsf{2S}}$ is denoted by $\mathbf{HYB}_0$.

$\mathbf{HYB}_0$: Let $\mathcal{F}_{\mathsf{pir}}^{\mathsf{2S}}$ denote a 2-server PIR ideal functionality for our case, with database holders S_1 and S_2, and client $\mathcal{P}_i$. For a database D held by S_1 and S_2 and a query q held by $\mathcal{P}_i$, $\mathcal{F}_{\mathsf{pir}}^{\mathsf{2S}}$ returns $\mathsf{D}[q]$ to $\mathcal{P}_i$, but S_1 and S_2 receive nothing.

$\mathbf{HYB}_1$: We modify $\mathcal{F}_{\mathsf{pir}}^{\mathsf{2S}}$ so that it returns $\mathsf{D}[q] + \mathsf{r}$ to $\mathcal{P}_i$, and $\mathsf{S}_1, \mathsf{S}_2$ receive r, where r is a random value from the domain of database block size. In other words, the modification can be thought of as the standard $\mathcal{F}_{\mathsf{pir}}^{\mathsf{2S}}$ being executed over a database $\mathsf{D}^{\mathsf{r}} = \mathsf{D} + \mathsf{r}$ rather than the actual database D. This modification leaks no additional information regarding the query to the servers because they will receive random masks that are independent of the query q. Furthermore, from the perspective of $\mathcal{P}_i$ with no prior knowledge of the database D, $\mathbf{HYB}_1$ will be indistinguishable from $\mathbf{HYB}_0$ because the values it sees in both cases are from the same distribution. As a result, $\mathbf{HYB}_0 \approx \mathbf{HYB}_1$.

$\mathbf{HYB}_2$: Looking ahead, in $\mathsf{PIR}_{\mathsf{sum}}$, servers $\mathsf{S}_1, \mathsf{S}_2$ and participant $\mathcal{P}_i$ run τ instances of $\mathcal{F}_{\mathsf{pir}}^{\mathsf{2S}}$ in parallel, one for each query $q \in \mathcal{Q}$. As shown in $\mathcal{F}_{\mathsf{pirsum}}$ (Fig. 11), the servers must ensure that all of the queries in $\mathcal{Q}$ are distinct. For this, we modify $\mathcal{F}_{\mathsf{pir}}^{\mathsf{2S}}$ in $\mathbf{HYB}_1$ to additionally output a secret share of the query q to each of S_1 and S_2. Because the servers S_1 and S_2 are assumed to be non-colluding in our setting, this modification will leak no information about the query q to either server. Since the output to $\mathcal{P}_i$ remains unchanged, $\mathbf{HYB}_1 \approx \mathbf{HYB}_2$ from $\mathcal{P}_i$'s perspective. ■

C. Reducing participant's communication

$\mathsf{PIR}_{\mathsf{sum}}$ is realized in $\mathsf{RIPPLE}_{\mathsf{PIR}}$ using two different approaches, each with its own set of trade-offs, with a goal of minimizing the communication at the participant's end. While the first approach, denoted by $\mathsf{PIR}_{\mathsf{sum}}^{\mathsf{I}}$ (Fig. 12), sacrifices computation for better communication, the second approach, denoted by $\mathsf{PIR}_{\mathsf{sum}}^{\mathsf{II}}$ (Fig. 13), reduces both the computational and communication overhead of the participant in $\mathsf{PIR}_{\mathsf{sum}}^{\mathsf{I}}$ with the help of additional server $\mathsf{S}_0 \in \mathcal{C}$.

1) $\mathsf{PIR}_{\mathsf{sum}}^{\mathsf{I}}$ *(Fig. 12):* In this approach, we instantiate $\mathcal{F}_{\mathsf{pir}}^{\mathsf{2S}}$ using PIR techniques based on Function Secret Sharing (FSS) [83]–[85]. To retrieve the q-th block from the database, $\mathcal{P}_i$ uses FSS on a Distributed Point Function (DPF) [86] that evaluates to a 1 only when the input q is 1 and to 0 otherwise. $\mathcal{P}_i$ generates two DPF keys k_1 and k_2 that satisfy the above constraint and sends one key to each of the servers S_1 and S_2. The servers S_1 and S_2 can then locally expand their key share to obtain their share for the bit vector $\vec{\mathsf{b}}$ and the rest of the procedure proceeds similarly to the naive linear summation method discussed in §V-A (more details on Linear Summation PIR are given in §C-A). The key size for a database of size N records using the optimised DPF construction in [83] is about $\lambda \log(N/\lambda)$ bits, where $\lambda = 128$ for an AES-based implementation. Fig. 12 provides the formal details of the $\mathsf{PIR}_{\mathsf{sum}}^{\mathsf{I}}$ protocol.

Security: For semi-honest participants, the security of our method directly reduces to that of the 2-server PIR protocol in [83]. However, as mentioned in [83], a malicious participant could generate incorrect DPF keys, risking the scheme's security and correctness. To prevent this type of misbehavior, Boyle et al. [83] present a form of DPF called "verifiable DPF", which can assure the correctness of the DPF keys created by $\mathcal{P}_i$ at the cost of an increased *constant* amount of communication between the servers.

While verifiable DPFs in $\mathsf{PIR}_{\mathsf{sum}}^{\mathsf{I}}$ ensure the validity of the τ bit vectors generated by $\mathcal{P}_i$, they do not ensure that bit vectors $\vec{\mathsf{b}}_1, \ldots, \vec{\mathsf{b}}_\tau$ correspond to τ distinct locations in the database D. However, we use the correctness guarantee of verifiable DPFs to reduce the communication cost for verification, as discussed in §V-A0a, §B-D, and §C-A1. In detail, all τ bit vectors $\vec{\mathsf{b}}_1, \ldots, \vec{\mathsf{b}}_\tau$ (PIR queries) are secret-shared between S_1 and S_2, each guaranteed to have exactly one 1 and the rest 0. To ensure distinctness, S_1 and S_2 XOR their respective τ shares locally to obtain the secret-share of a single vector $\vec{\mathsf{b}}_c = \oplus_{k=1}^{\tau} \vec{\mathsf{b}}_k$. The challenge is then to check if $\vec{\mathsf{b}}_c$ has exactly $\tau 1$ bits. This can be accomplished by having S_1 and S_2 agree on a random permutation π and reconstructing $\pi(\vec{\mathsf{b}}_c)$ to S_0 and allowing S_0 to perform the check, as in the naive approach (cf. §V-A0a).

Protocol $\mathsf{PIR}_{\mathsf{sum}}^{\mathsf{I}}$

Input(s): i) $\mathsf{S}_1, \mathsf{S}_2 : \mathsf{D}; |\mathsf{D}| = N$, ii) $\mathcal{P}_i : \mathcal{Q} = \{q_1, \ldots, q_\tau\}$, and iii) $\mathsf{S}_0 : \perp$.
Output: $\mathcal{P}_i : \mathsf{res} = \sum_{q \in \mathcal{Q}} \mathsf{D}[q]$

Computation S_1 and S_2 sample τ random mask values

$\{m_1, \ldots, m_\tau\} \in \mathbb{Z}_{2^\ell}^\tau$ such that $\sum_{j=1}^{\tau} m_j = 0$. For each $q \in \mathcal{Q}$, execute the following:

1. $\mathsf{S}_1, \mathsf{S}_2$ locally compute $\mathsf{D}^{m_q} = \mathsf{D} + m_q$.
2. Execute DPF protocol [83] (verifiable DPF for malicious participants) with $\mathcal{P}_i$ as client with input q. Server S_u obtains $[\vec{\mathsf{b}}_q]_u$ with $\mathsf{b}_q^j = 1$ for $j = q$ and $\mathsf{b}_q^j = 0$ for $j \neq q$, for $u \in \{1, 2\}$.

Verification Let $\{\vec{\mathsf{b}}_{q_1}, \ldots, \vec{\mathsf{b}}_{q_\tau}\}$ denote the bit vectors whose XOR-shares are generated during the preceding steps.

3. Servers verify correctness of q_j, $j \in [\tau]$, by executing the Ver algorithm of the verifiable DPF protocol [83] (cf. §B-D). It outputs `Accept` to S_1 and S_2 if q_j has exactly 1 one and $(N-1)$ zeroes. Else, it outputs `abort`.
4. S_u computes $[\vec{\mathsf{b}}_c]_u = \oplus_{q \in \mathcal{Q}} [\vec{\mathsf{b}}_q]_u$, for $u \in \{1, 2\}$.
5. S_1 and S_2 non-interactively agree on random permutation π.
6. S_u sends $\pi([\vec{\mathsf{b}}_c]_u)$ to S_0, for $u \in \{1, 2\}$.
7. S_0 locally reconstructs $\pi(\vec{\mathsf{b}}_c) = \pi([\vec{\mathsf{b}}_q]_1) \oplus \pi([\vec{\mathsf{b}}_q]_2)$. It sends `Accept` to S_1 and S_2, if $\pi(\vec{\mathsf{b}}_c)$ has exactly τ ones. Else, it sends `abort`.

Output Transfer Send $\perp$ to $\mathcal{P}_i$ if verifiable DPF or S_0 generated `abort` during verification. Otherwise, proceed as follows:

8. S_u sends $[y_q]_u = \bigoplus_{j=1}^{N} [\mathsf{b}_q^j]_u \mathsf{D}^{m_q}[j]$ to $\mathcal{P}_i$, for $q \in \mathcal{Q}, u \in \{1, 2\}$.
9. $\mathcal{P}_i$ locally computes $\mathsf{res} = \sum_{q \in \mathcal{Q}} ([y_q]_1 \oplus [y_q]_2)$.

Fig. 12: $\mathsf{PIR}^{\mathsf{I}}_{\mathsf{sum}}$ Protocol.

Computation Complexity (#AES operations): In $\mathsf{PIR}^{\mathsf{I}}_{\mathsf{sum}}$, the participant $\mathcal{P}_i$ must perform $4 \cdot \log(N/\lambda)$ AES operations as part of the key generation algorithm for each of the τ instances of $\mathcal{F}^{2\mathsf{S}}_{\mathsf{pir}}$ over a database of size N, where $\lambda = 128$ for an AES-based implementation. Similarly, S_1 and S_2 must perform $\log(N/\lambda)$ AES operations for each of the N DPF evaluations. We refer to Table 1 in [83] for more specifics.

2) $\mathsf{PIR}^{\mathsf{II}}_{\mathsf{sum}}$ (Fig. 13): In this approach, we use the server S_0 to reduce the computation and communication of the participant $\mathcal{P}_i$ in $\mathsf{PIR}^{\mathsf{I}}_{\mathsf{sum}}$. The idea is that S_0 plays the role of $\mathcal{P}_i$ for the PIR protocol in $\mathsf{PIR}^{\mathsf{I}}_{\mathsf{sum}}$. However, $\mathcal{P}_i$ cannot send its query q to S_0 in clear because it would violate privacy. As a result, $\mathcal{P}_i$ selects random values $q', \theta_q \in [N]$ such that $q = q' + \theta_q$. In this case, q' is a *shifted version* of the index q, and θ is a *shift correction* for q. $\mathcal{P}_i$ sends q' to S_0 and θ_q to both S_1 and S_2. The rest of the computation until output retrieval will now occur solely among the servers.

Protocol $\mathsf{PIR}^{\mathsf{II}}_{\mathsf{sum}}$

Input(s): i) $\mathsf{S}_1, \mathsf{S}_2 : \mathsf{D}; |\mathsf{D}| = N$, ii) $\mathcal{P}_i : \mathcal{Q} = \{q_1, \ldots, q_\tau\}$, and iii) $\mathsf{S}_0 : \perp$.
Output: $\mathcal{P}_i : \mathsf{res} = \sum_{q \in \mathcal{Q}} \mathsf{D}[q]$

Computation S_1 and S_2 sample τ random mask values $\{m_1, \ldots, m_\tau\} \in \mathbb{Z}_{2^\ell}^\tau$ such that $\sum_{j=1}^{\tau} m_j = 0$. For each $q \in \mathcal{Q}$, execute the following:

1. $\mathsf{S}_1, \mathsf{S}_2$ locally compute $\mathsf{D}^{m_q} = \mathsf{D} + m_q$, i.e., $\mathsf{D}^{m_q}[j] = \mathsf{D}[j] + m_q$, for $j \in [N]$.
2. $\mathcal{P}_i, \mathsf{S}_1, \mathsf{S}_2$ sample random $\theta_q \in [N]$.
3. $\mathcal{P}_i$ computes and sends $q' = q - \theta_q$ to S_0.
4. Servers execute DPF protocol [83] with S_0 as client with input q'. Server S_u obtains $[\vec{\mathsf{b}}_{q'}]_u$ with $\mathsf{b}_{q'}^j = 1$ for $j = q'$ and $\mathsf{b}_{q'}^j = 0$ for $j \neq q'$, for $u \in \{1, 2\}$.
5. S_u locally applies θ_u on $[\vec{\mathsf{b}}_{q'}]_u$ to generate $[\vec{\mathsf{b}}_q]_u$, for $u \in \{1, 2\}$.

Verification Let $\{\vec{\mathsf{b}}_{q_1}, \ldots, \vec{\mathsf{b}}_{q_\tau}\}$ denote the bit vectors whose XOR-shares are generated during the preceding steps:

6. S_k computes $[\vec{\mathsf{b}}_c]_k = \oplus_{q \in \mathcal{Q}} [\vec{\mathsf{b}}_q]_k$, for $u \in \{1, 2\}$.
7. S_1 and S_2 non-interactively agree on random permutation π.
8. S_u sends $\pi([\vec{\mathsf{b}}_c]_u)$ to S_0, for $u \in \{1, 2\}$.
9. S_0 locally reconstructs $\pi(\vec{\mathsf{b}}_c) = \pi([\vec{\mathsf{b}}_q]_1) \oplus \pi([\vec{\mathsf{b}}_q]_2)$. It sends `Accept` to S_1 and S_2, if $\pi(\vec{\mathsf{b}}_c)$ has exactly τ ones. Else, it sends `abort`.

Output Transfer Send $\perp$ to $\mathcal{P}_i$ if S_0 generated `abort` during verification. Otherwise, proceed as follows:

10. S_u sends $[y_q]_u = \bigoplus_{j=1}^{N} [\mathsf{b}_q^j]_u \mathsf{D}^{m_q}[j]$ to $\mathcal{P}_i$, for $q \in \mathcal{Q}, u \in \{1, 2\}$.
11. $\mathcal{P}_i$ locally computes $\mathsf{res} = \sum_{q \in \mathcal{Q}} ([y_q]_1 \oplus [y_q]_2)$.

Fig. 13: $\mathsf{PIR}^{\mathsf{II}}_{\mathsf{sum}}$ Protocol.

The servers run a DPF instance [83] with S_0 acting as the client and input query q'. At the end of the computation, S_1 and S_2 obtain the bit vector $\vec{\mathsf{b}}_{q'}$, which corresponds to q'. However, as discussed in $\mathsf{PIR}^{\mathsf{I}}_{\mathsf{sum}}$, the servers require an XOR sharing corresponding to the actual query q to continue the computation. S_1 and S_2 do this by using the shift correction value θ_q received from $\mathcal{P}_i$. Both S_1 and S_2 will perform a right cyclic shift of their $\vec{\mathsf{b}}_{q'}$ shares by θ_q units. A negative value for θ_q indicates a cyclic shift to the left.

It is easy to see that the XOR shares obtained after the cyclic shift corresponds to the bit vector $\vec{\mathsf{b}}_q$. To further optimise $\mathcal{P}_i$'s communication, $\mathcal{P}_i$ and servers $\mathsf{S}_1, \mathsf{S}_2$ non-interactively generate random shift correction values θ_q first using the shared-key setup (cf. §B-A), and only the corresponding q' values are communicated to S_0. The rest of the protocol is similar to $\mathsf{PIR}^{\mathsf{I}}_{\mathsf{sum}}$, and the formal protocol is shown in Fig. 13. In terms of malicious participants, $\mathsf{PIR}^{\mathsf{II}}_{\mathsf{sum}}$ has an advantage over $\mathsf{PIR}^{\mathsf{I}}_{\mathsf{sum}}$ as there is no need to use a verifiable DPF to protect against malicious $\mathcal{P}_i$, because the semi-honest server S_0 generates the DPF key instead of $\mathcal{P}_i$.

Improving Verification Costs in $\mathsf{PIR}^{\mathsf{II}}_{\mathsf{sum}}$: A large amount of communication is used in the $\mathsf{PIR}^{\mathsf{II}}_{\mathsf{sum}}$ protocol to verify malicious participants. More specifically, in Step 8 of Fig. 13, $2N$ bits are communicated towards S_0 to ensure the distinctness of the queries made by the participant $\mathcal{P}_i$. We note that allowing

Stage	$\mathsf{PIR}^{\mathsf{I}}_{\mathsf{sum}}$	$\mathsf{PIR}^{\mathsf{II}}_{\mathsf{sum}}$
$\mathcal{P}_i$ to servers in $\mathcal{C}$	$2\tau(\lambda+2)\log(N/\lambda)+4\tau\lambda$	$\tau \log N$
Server to server	0	$2\tau(\lambda+2)\log(N/\lambda)+4\tau\lambda$
Servers in $\mathcal{C}$ to $\mathcal{P}_i$	$\tau \cdot 2\ell$	$\tau \cdot 2\ell$
+ Verification (mal.)	$2N+2+\delta$	$2N+2$

TABLE II: Summary of communication costs for $\mathsf{PIR}_{\mathsf{sum}}$. λ denotes the AES key size ($\lambda = 128$ in [84]), ℓ denotes the block size in bits ($\ell = 128$ in this work), and δ denotes the constant involved in the verifiable DPF approach [83] (cf. §C-A1).

a small amount of leakage to S_0 could improve this communication and is discussed next.

Consider the following modification to the $\mathsf{PIR}^{\mathsf{II}}_{\mathsf{sum}}$ protocol. Instead of sampling θ_q for each query $q \in \mathcal{Q}$ (cf. Step 2 in Fig. 13), $\mathcal{P}_i$, S_1, and S_2 sample only one random shift value θ and uses it for all τ instances. Since the queries must be distinct, $\mathcal{P}_i$ is forced to send distinct q' values to S_0 in Step 3 of Fig. 13. If not, S_0 can send `abort` to S_1 and S_2 at this step, eliminating the need for communication-intensive verification. The relative distance between the queried indices would be leaked to S_0 as a result of this optimization. In concrete terms, if we use the same θ value for any two queries $q_m, q_j \in \mathcal{Q}$, then $q_m - q_n = q'_m - q'_n$. Because S_0 sees all q' values in the clear, it can deduce the relative positioning of $\mathcal{P}_i$'s actual queries. However, since S_0 has no information about the underlying database D, this leakage may be acceptable for some applications.

3) Summary of communication costs: Tab. II summarises the communication cost for our two $\mathsf{PIR}_{\mathsf{sum}}$ approaches for instantiating $\mathcal{F}_{\mathsf{pirsum}}$ over a database of size N with τ PIR queries per client.

VI. Evaluation

In this section, we evaluate and compare the computation and communication efficiency of our two RIPPLE protocols presented in §IV. A fully-fledged implementation, similar to existing contact tracing apps, would necessitate collaboration with industry partners to develop a real-world scalable system for national deployment. Instead, we carry out a proof-of-concept implementation and provide micro benchmark results for all major building blocks.[9] We focus on the simulation phase for benchmarking, which is separate from the token generation phase. The simulations can ideally be done overnight while mobile phones are charging and have access to a high-bandwidth WiFi connection. According to studies [87], [88], sleeping habits in various countries provide a time window of several hours each night that can be used for this purpose.

[9] Note that we are not attempting to create the most efficient instantiation. More optimizations will undoubtedly improve efficiency, and our protocols can be heavily parallelized with many servers. Instead, our goal here is to demonstrate the viability of RIPPLE protocols for large-scale deployment.

a) Setup and Parameters.: We run the benchmarks on the server-side with three servers (two for FSS-PIR and one as a helper server as discussed in §V-C) with Intel Core i9-7960X CPUs@2.8 GHz and 128 GB RAM connected with 10 Gbit/s LAN and 0.1 s RTT. The client is a Samsung Galaxy S10+ with an Exynos 9820@2.73 GHz and 8GB RAM. As Android does not allow third-party developers to implement applications for Android's TEE Trusty [89], we use hardware-backed crypto operations already implemented by Android instead. We use the code of [90] to instantiate FSS-PIR. We implement the AGCT in C++ and follow previous work on cuckoo hashing [91] by using tabulation hashing for the hash functions.

We instantiate our protocols in RIPPLE with $\kappa = 128$ bit security. We use RSA-2048 as the encryption scheme in $\text{RIPPLE}_{\mathsf{TEE}}$ since Android offers a hardware-backed implementation. We omit the overhead of remote attestation for the sake of simplicity. For $\text{RIPPLE}_{\mathsf{PIR}}$, we use the FSS-PIR scheme of [83], [90] as the baseline. The addresses are hashed with SHA-256 and trimmed to $40 - 1 + \log_2(\mathsf{p} \cdot E^{\mathsf{avg}})$ bits, where p is the number of participants and E^{avg} represents the average number of encounters per participant per simulation step. We set $E^{\mathsf{avg}} = 100$ while benchmarking based on numbers provided by research on epidemiological modeling [92], [93]. To avoid cycles when inserting n messages into the AGCT (cf. §IV-B2), we set its size to $10n$. This can be further improved as discussed in §IV-B2 [81], [91], [94]. A typical simulation step corresponds to one day, such that 14 simulation steps can simulate two weeks.

A. Communication Complexity

Here, we look at the communication costs that our protocols incur. To analyse the scalability of our protocols, we consider p participants ranging from thousand (1K) to twenty million (20M). Tab. III summarises the communication costs of each participant as well as the communication servers ($\mathcal{C}$) for one simulation step in a specific simulation. One simulation step includes all protocol steps, beginning with participants locally computing their infection likelihood δ and ending with them obtaining their cumulative infection likelihood Δ for that step.

1) Participant Communication: As shown in Tab. III, a participant in $\text{RIPPLE}_{\mathsf{TEE}}$ requires just 16KB of total communication in every simulation

Entities	Protocol	Population (p)									
		1K	10K	50K	100K	500K	1M	2M	5M	10M	20M
Participants in $\mathcal{P}$ (in KB)	RIPPLE$_{\mathsf{TEE}}$ (§IV-A)	16.00	16.00	16.00	16.00	16.00	16.00	16.00	16.00	16.00	16.00
	RIPPLE$_{\mathsf{PIR}}$: PIR$^{\mathsf{I}}_{\mathsf{sum}}$ (§V-C1)	51.63	62.42	69.97	73.22	80.77	84.02	87.27	91.56	94.81	98.06
	RIPPLE$_{\mathsf{PIR}}$: PIR$^{\mathsf{II}}_{\mathsf{sum}}$ (§V-C2)	3.45	3.49	3.52	3.53	3.56	3.57	3.59	3.60	3.62	3.63
Servers in $\mathcal{C}$ (in GB)	RIPPLE$_{\mathsf{TEE}}$ (§IV-A)	0.02	0.19	0.96	1.92	9.60	19.20	38.40	96.00	192.00	384.00
	RIPPLE$_{\mathsf{PIR}}$ (§V)	0.01	0.10	0.48	0.96	4.80	9.60	19.20	48.00	96.00	192.00

TABLE III: Communication costs per simulation step in our RIPPLE instantiations.

step, and this is independent of the population on which the simulation is done. This is because each participant will only send and receive infection likelihood messages related to its encounters. While the value in the table corresponds to an average of 100 encounters ($E^{\mathsf{avg}} = 100$), we depict the participants' communication in Fig. 14a with varied E^{avg} ranging from 10 to 500 for a population of 10M. Note that a 2-week simulation with $E^{\mathsf{avg}} = 500$ can be completed by a participant in RIPPLE$_{\mathsf{TEE}}$ with roughly 1MB of communication.

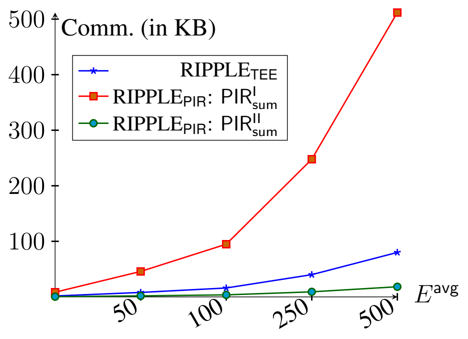
Unlike RIPPLE$_{\mathsf{TEE}}$, participant communication in both PIR$^{\mathsf{I}}_{\mathsf{sum}}$ and PIR$^{\mathsf{II}}_{\mathsf{sum}}$ increases for larger populations as the corresponding database size increases. The communication, however, is only sub-linear in the database size[10].

In particular, the participant's communication in PIR$^{\mathsf{I}}_{\mathsf{sum}}$ ranges from 51.63KB to 98.06KB, with the higher cost over RIPPLE$_{\mathsf{TEE}}$ attributed to the size of DPF keys used in the underlying FSS-PIR scheme [83], as discussed in §V. The communication in PIR$^{\mathsf{II}}_{\mathsf{sum}}$, on the other hand, is about 3.5KB for all participant sizes we consider. This reduced communication is due to the optimization in PIR$^{\mathsf{II}}_{\mathsf{sum}}$, which offloads the DPF key generation task to the helper server S_0 (cf. §V-C2). A participant in PIR$^{\mathsf{I}}_{\mathsf{sum}}$ must communicate approximately 7MB of data for a 2-week simulation for a 10M population with $E^{\mathsf{avg}} = 500$, whereas it is only 0.25MB in the case of PIR$^{\mathsf{II}}_{\mathsf{sum}}$.

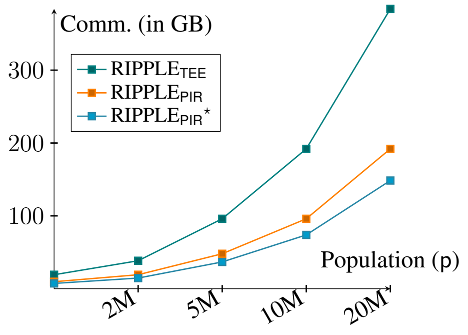
Tab. IV provides the communication per participant for multiple population sizes in RIPPLE$_{\mathsf{TEE}}$, PIR$^{\mathsf{I}}_{\mathsf{sum}}$, and PIR$^{\mathsf{II}}_{\mathsf{sum}}$, while varying the average number of encounters E^{avg} per simulation step from 10 to 500. The communication cost in RIPPLE$_{\mathsf{TEE}}$ is independent of the population size and grows linearly in E^{avg}. A similar trend can be seen in RIPPLE$_{\mathsf{PIR}}$ except that the cost increases sub-linearly with the population size due to using the FSS-based PIR scheme in RIPPLE$_{\mathsf{PIR}}$.

2) Server Communication: The servers' communication is primarily attributed to the anonymous communication channel they have established, which provides unlinkability and, thus, privacy to the participants' messages. Communicating M messages through the channel requires the servers to communicate $2M$ messages in RIPPLE$_{\mathsf{TEE}}$, and $3M$ messages in RIPPLE$_{\mathsf{PIR}}$. When it comes to concrete values, however, the server communication in RIPPLE$_{\mathsf{PIR}}$ is half that of RIPPLE$_{\mathsf{TEE}}$, as shown in Tab. III. This is due to the larger message size in RIPPLE$_{\mathsf{TEE}}$ due to the use of public-key encryption.

[10] DB size of $10n$ and communication costs of RIPPLE$_{\mathsf{PIR}}$ can be reduced by optimizing the database size by extending the database by only $d+\lambda$ bins, where d is the upper bound of double collisions and λ is an error parameter (cf. §IV-B2 and [81])



(a) Participant's communication with varying E^{avg} for a population of 10M.



(b) Servers' communication per simulation step for varying population. * denotes the results for optimized bit addresses in RIPPLE$_{\mathsf{PIR}}$ (cf. full version [76]).

Fig. 14: Communication Costs of RIPPLE.

For a population of 10M, the servers in RIPPLE$_{\mathsf{TEE}}$ must communicate 192GB of data among themselves, whereas RIPPLE$_{\mathsf{PIR}}$ requires 96GB.Setting the proper bit length for the address field in the messages can further reduce communication. For example, a population of 20M with $E^{\mathsf{avg}} = 100$ can be accommodated in a 70-bit address field. Using this optimization will result in an additional 23 % reduction in communication at the servers, as shown in Tab. V. Fig. 14b captures these observations better, and Tab. V and Tab. IV in the next subsection provide a detailed analysis of the concrete communication costs.

Population p	Protocol	E^{avg}				
		10	50	100	250	500
100K	$\text{RIPPLE}_{\mathsf{TEE}}$ (§IV-A)	1.60	8.00	16.00	40.00	80.00
	$\text{RIPPLE}_{\mathsf{PIR}}$: $\mathsf{PIR}^{\mathsf{I}}_{\mathsf{sum}}$	6.24	34.99	73.22	193.79	403.83
	$\text{RIPPLE}_{\mathsf{PIR}}$: $\mathsf{PIR}^{\mathsf{II}}_{\mathsf{sum}}$	0.35	1.76	3.53	8.87	17.81
1M	$\text{RIPPLE}_{\mathsf{TEE}}$ (§IV-A)	1.60	8.00	16.00	40.00	80.00
	$\text{RIPPLE}_{\mathsf{PIR}}$: $\mathsf{PIR}^{\mathsf{I}}_{\mathsf{sum}}$	7.32	40.38	84.02	220.78	457.81
	$\text{RIPPLE}_{\mathsf{PIR}}$: $\mathsf{PIR}^{\mathsf{II}}_{\mathsf{sum}}$	0.35	1.78	3.57	8.98	18.01
10M	$\text{RIPPLE}_{\mathsf{TEE}}$ (§IV-A)	1.60	8.00	16.00	40.00	80.00
	$\text{RIPPLE}_{\mathsf{PIR}}$: $\mathsf{PIR}^{\mathsf{I}}_{\mathsf{sum}}$	8.40	45.78	94.81	247.77	511.79
	$\text{RIPPLE}_{\mathsf{PIR}}$: $\mathsf{PIR}^{\mathsf{II}}_{\mathsf{sum}}$	0.36	1.80	3.62	9.08	18.22

TABLE IV: Communication (in KB) per participant in a simulation step for varying average numbers of encounters E^{avg} and population sizes p.

3) Communication Micro Benchmarks.: Tab. V details the communication costs per simulation step at various stages in our instantiations of RIPPLE. We find that a participant's communication costs are very low compared to the overall costs. In $\text{RIPPLE}_{\mathsf{TEE}}$, a participant communicates at most 268 KB and incurs a runtime of 92 seconds over a two-week simulation over a population of one million. In $\mathsf{PIR}^{\mathsf{II}}_{\mathsf{sum}}$, the cost is reduced to 100 KB and 40 seconds of runtime. Communication increases to 1.2 MB in $\mathsf{PIR}^{\mathsf{I}}_{\mathsf{sum}}$ due to the participant's handling of DPF keys.

Finally, Tab. V does not include costs for verification against malicious participants since they can be eliminated using server S_0 (cf. §V-C2) or sketching algorithms similar to those in [83].

B. Computation Complexity

This section focuses on the runtime, including computation time and communication between entities. Tab. VI summarizes the computation time with respect to a participant $\mathcal{P}_i$ for a two-week simulation over a half-million population. The longer computation time in $\text{RIPPLE}_{\mathsf{TEE}}$, as shown in Tab. VI, is due to the public key encryption and decryption that occurs within the mobile device's TEE. This cost, however, is independent of population size and scales linearly with the average number of encounters, denoted by E^{avg}. In particular, for a 14-day simulation with a population of half a million, $\mathcal{P}_i$ in $\text{RIPPLE}_{\mathsf{TEE}}$ needs approximately 43.7 seconds to perform the encryption and decryption tasks and may require additional time for the remote attestation procedure, which is not covered in our benchmarks. $\mathcal{P}_i$'s computation time in $\text{RIPPLE}_{\mathsf{PIR}}$, on the other hand, is significantly lower and is at most 5 milliseconds for $\mathsf{PIR}^{\mathsf{II}}_{\mathsf{sum}}$, while it increases to around 165 milliseconds for $\mathsf{PIR}^{\mathsf{I}}_{\mathsf{sum}}$. The increased computation time in $\mathsf{PIR}^{\mathsf{I}}_{\mathsf{sum}}$ is due to DPF key generation, which scales sub-linearly with population size.

In Fig. 15, we plot the overall runtime of our two instantiations in RIPPLE for a full simulation of 2 weeks over various populations ranging from 1K to 500K. After a population of 100K, the runtime of $\text{RIPPLE}_{\mathsf{PIR}}$ begins to exceed that of $\text{RIPPLE}_{\mathsf{TEE}}$ due to an increase in database size, which results in longer data transfer times. More details regarding computation time is presented in Tab. VII. Note that the runtimes in Fig. 15 include runtime for computation and communication of the secure shuffle among the servers for anonymous communication and among servers and clients for the PIR in $\text{RIPPLE}_{\mathsf{PIR}}$.

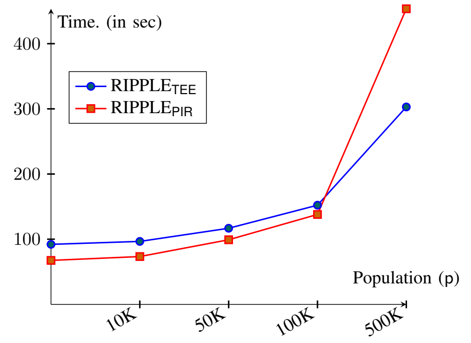


Fig. 15: Runtime per simulation in RIPPLE (14 days).

1) Computation Micro Benchmarks.: Tab. VII contains the computation costs per simulation at the different stages of our instantiations of RIPPLE's. As visible, data transfer time as part of anonymous communication through servers accounts for the majority of computation time and begins to affect overall performance as the population grows. Our system crashed due to memory constraints after a population of 500K while running the experiments. This will not be the case in a real-world deployment of powerful servers linked by high-speed networks. Similar as w.r.t. communication, participants' computation costs are very low in comparison to the overall costs.

2) Battery Usage.: The token generation phase in RIPPLE consumes the most amount of mobile battery as this phase is active throughout the day. This usage could be optimized by mobile OS providers like Apple and Google, as discussed by Vaudenay et al. [95] and Avitabile et al. [96] in the context of contact tracing apps. Their technology enables an app to run in the background, thus, significantly improving battery life, which is otherwise impossible for a standard third-party mobile application. Additionally, RIPPLE could offer users the choice to participate only in simulations while charging so as not to cause any unwanted battery drain.

3) Comparison to Related Work.: Note that no experimental comparison to related work is (and can be) done, as RIPPLE is the first distributed privacy-preserving epidemiological modeling system. Established contact tracing apps, such as the SwissCovid[11], the German Corona-Warn-App[12], or the Australian COVIDSafe[13] only record contacts for notifying contacts of infected people. Concretely, contact tracing basically relates to RIPPLE's to-

[11] https://github.com/SwissCovid
[12] https://www.coronawarn.app/en/ [13] https://www.health.gov.au/resources/apps-and-tools/covidsafe-app

Stages of RIPPLE	Protocol[a]		Population (p)									
			1K	10K	50K	100K	500K	1M	2M	5M	10M	20M
Message Generation by $\mathcal{P}_i \in \mathcal{P}$ (in KB)	$\text{RIPPLE}_{\text{TEE}}$	(§IV-A)[b]	12.80	12.80	12.80	12.80	12.80	12.80	12.80	12.80	12.80	12.80
	$\text{RIPPLE}_{\text{PIR}}$: ●	(§IV-B)	3.20	3.20	3.20	3.20	3.20	3.20	3.20	3.20	3.20	3.20
	$\text{RIPPLE}_{\text{PIR}}$: ◐	(§IV-B)	2.30	2.34	2.38	2.39	2.41	2.43	2.44	2.45	2.46	2.48
Secure Shuffle by $\mathcal{C}$ (in GB)	$\text{RIPPLE}_{\text{TEE}}$	(§IV-A)	0.02	0.19	0.96	1.92	9.60	19.20	38.40	96.00	192.00	384.00
	$\text{RIPPLE}_{\text{PIR}}$ - ●	(§IV-B)	0.01	0.10	0.48	0.96	4.80	9.60	19.20	48.00	96.00	192.00
	$\text{RIPPLE}_{\text{PIR}}$ - ◐	(§IV-B)	0.01	0.07	0.36	0.72	3.62	7.28	14.63	36.75	73.88	148.50
Output Computation by $\mathcal{P}_i \in \mathcal{P}$ (in KB)[c]	$\text{RIPPLE}_{\text{TEE}}$	(§IV-A)	6.40	6.40	6.40	6.40	6.40	6.40	6.40	6.40	6.40	6.40
	$\text{PIR}^{\text{I}}_{\text{sum}}$ - ●	(§V-C1)	51.36	62.42	69.97	73.22	80.77	84.02	87.27	91.56	94.81	98.06
	$\text{PIR}^{\text{I}}_{\text{sum}}$ - ◐	(§V-C1)	26.48	32.64	37.69	39.82	44.77	47.05	49.38	52.33	54.77	57.26
	$\text{PIR}^{\text{II}}_{\text{sum}}$	(§V-C2)	3.45	3.49	3.52	3.53	3.56	3.57	3.59	3.60	3.62	3.63

[a]● - 128-bit address for $\text{RIPPLE}_{\text{PIR}}$ and ◐ - $40 - 1 + \log_2(\mathsf{p} \cdot E^{\text{avg}})$ bit address for $\text{RIPPLE}_{\text{PIR}}$. [b]Includes registration of public keys with the exit node $\mathcal{N}_{\texttt{exit}}$. [c]includes message download, decryption/PIR queries, summation.

TABLE V: Detailed communication costs per simulation step in RIPPLE.

	Per Simulation Step / Simulation ($N_{\text{step}} = 14$)		
	Message Generation (in ms)	PIR Queries (in ms)	Output Computation (in ms)
$\text{RIPPLE}_{\text{TEE}}$	1.12 / 80.00	-	42.56 / 3040.00
$\text{PIR}^{\text{I}}_{\text{sum}}$	0.30 / 4.26	11.73 / 160	4.8e-2 / 6.72e-1
$\text{PIR}^{\text{II}}_{\text{sum}}$	0.30 / 4.26	3.0e-3 / 4.2e-2	4.8e-2 / 6.72e-1

TABLE VI: Average participant computation times per simulation step distributed across various tasks. Values are obtained using a mobile for a population of 500K with $E^{\text{avg}} = 100$.

ken generation phase, while the other three phases (simulation initialization, simulation execution, and result aggregation, cf. §III-B) are not covered by any contact tracing system. Crucially, the main contribution of our work is how to realize the simulation execution, which has never been done before. Hence, no meaningful comparison between the systems is possible due to differences in the fundamental functionalities.

4) Code availability: Available at `DOI: 10.5281/zenodo.6595448`.

a) Summary.: Our benchmarking using the proof-of-concept implementation demonstrated the RIPPLE framework's viability for real-world adaptation. One of the key benefits of our approaches is that participants have very little work to do. The system's efficiency can be further improved with appropriate hardware and optimized (non-prototype) implementations.

References

[1] N. Vindegaard and M. E. Benros, "COVID-19 pandemic and mental health consequences: Systematic review of the current evidence," *Brain, Behavior, and Immunity*, 2020.

[2] D. Maison, D. Jaworska, D. Adamczyk, and D. Affeltowicz, "The challenges arising from the COVID-19 pandemic and the way people deal with them. a qualitative longitudinal study," *PloS One*, 2021.

[3] A. Taub, "A new Covid-19 crisis: Domestic abuse rises worldwide," *The New York Times*, 2020.

[4] C. Caulcutt, "Belgium introduces quarantine for monkeypox cases," *Politico*, 2022, https://www.politico.eu/article/belgium-introduce-quarantine-monkeypox-case/.

[5] A. D. Christy Cooney, "High-risk monkeypox contacts advised to isolate," *BBC*, 2022, https://www.bbc.com/news/uk-61546480.

[6] E. C. for Disease Prevention and Control, "Epidemiological update: Monkeypox outbreak," 2022, https://www.ecdc.europa.eu/en/news-events/epidemiological-update-monkeypox-outbreak.

[7] L. Reichert, S. Brack, and B. Scheuermann, "Poster: Privacy-preserving contact tracing of covid-19 patients," *IEEE S&P*, 2021.

[8] N. Ahmed, R. A. Michelin, W. Xue, S. Ruj, R. Malaney, S. S. Kanhere, A. Seneviratne, W. Hu, H. Janicke, and S. K. Jha, "A survey of COVID-19 contact tracing apps," *IEEE Access*, 2020.

[9] C. Troncoso, M. Payer, J. Hubaux, M. Salathé, J. R. Larus, W. Lueks, T. Stadler, A. Pyrgelis, D. Antonioli, L. Barman, S. Chatel, K. G. Paterson, S. Capkun, D. A. Basin, J. Beutel, D. Jackson, M. Roeschlin, P. Leu, B. Preneel, N. P. Smart, A. Abidin, S. Gurses, M. Veale, C. Cremers, M. Backes, N. O. Tippenhauer, R. Binns, C. Cattuto, A. Barrat, D. Fiore, M. Barbosa, R. Oliveira, and J. Pereira, "Decentralized privacy-preserving proximity tracing," *IEEE Data Eng. Bull.*, 2020.

[10] S. Vaudenay, "Centralized or decentralized? the contact tracing dilemma," Cryptology ePrint Archive, Report 2020/531, 2020, https://ia.cr/2020/531.

[11] N. Trieu, K. Shehata, P. Saxena, R. Shokri, and D. Song, "Epione: Lightweight contact tracing with strong privacy," arXiv preprint arXiv:2004.13293, 2020, https://arxiv.org/abs/2004.13293.

[12] B. Pinkas and E. Ronen, "Hashomer–privacy-preserving bluetooth based contact tracing scheme for hamagen," *Real World Crypto and NDSS Corona-Def Workshop*, 2021.

[13] W. Lueks, S. F. Gürses, M. Veale, E. Bugnion, M. Salathé, K. G. Paterson, and C. Troncoso, "CrowdNotifier: Decentralized privacy-preserving presence tracing," *PoPETs*, 2021.

[14] K. Hogan, B. Macedo, V. Macha, A. Barman, X. Jiang *et al.*, "Contact tracing apps: Lessons learned on privacy,

Stages of RIPPLE	Protocol	Population (p)					
		1K	10K	50K	100K	500K	1M
Message Generation by $\mathcal{P}_i \in \mathcal{P}$ (in sec)	$\text{RIPPLE}_{\text{TEE}}$ (§IV-A)	1.12	1.12	1.12	1.12	1.12	1.12
	$\text{RIPPLE}_{\text{PIR}}$: (§IV-B)	4.26e-3	4.26e-3	4.26e-3	4.26e-3	4.26e-3	4.26e-3
Secure Shuffle by $\mathcal{C}$ (in sec)	$\text{RIPPLE}_{\text{TEE}}$ (§IV-A)	0.70	5.20	25.38	60.77	211.47	$493.33^{\star a}$
	$\text{RIPPLE}_{\text{PIR}}$ (§IV-B)	0.78	6.65	32.36	71.17	386.68	$1542.30^{\star}$
Output Computation^b (in sec)	$\text{RIPPLE}_{\text{TEE}}$ (§IV-A)	44.66	44.66	44.66	44.66	44.66	44.66
	$\text{PIR}^{\text{I}}_{\text{sum}}$ (§V-C1)	32.31	32.33	32.34	32.35	32.36	32.37
	$\text{PIR}^{\text{II}}_{\text{sum}}$ (§V-C2)	32.20	32.20	32.20	32.20	32.20	32.20

^{a*} denotes system crash due to memory. ^{b}includes message download, decryption/PIR queries, summation.

TABLE VII: Detailed computation costs per simulation ($N_{\text{step}} = 14$, i.e., 14 days) in RIPPLE.

autonomy, and the need for detailed and thoughtful implementation," *JMIR Medical Informatics*, 2021.

[15] P. Tupper, S. P. Otto, and C. Colijn, "Fundamental limitations of contact tracing for covid-19," *FACETS*, 2021.

[16] D. Lewis, "Where covid contact-tracing went wrong," *Nature*, 2020.

[17] G. Giordano, F. Blanchini, R. Bruno, P. Colaneri, A. Di Filippo, A. Di Matteo, and M. Colaneri, "Modelling the covid-19 epidemic and implementation of population-wide interventions in Italy," *Nature Medicine*, 2020.

[18] A. J. Kucharski, P. Klepac, A. J. Conlan, S. M. Kissler, M. L. Tang, H. Fry, J. R. Gog, W. J. Edmunds, J. C. Emery, G. Medley *et al.*, "Effectiveness of isolation, testing, contact tracing, and physical distancing on reducing transmission of SARS-CoV-2 in different settings: a mathematical modelling study," *The Lancet Infectious Diseases*, 2020.

[19] I. Cooper, A. Mondal, and C. G. Antonopoulos, "A SIR model assumption for the spread of COVID-19 in different communities," *Chaos, Solitons & Fractals*, 2020.

[20] P. C. Silva, P. V. Batista, H. S. Lima, M. A. Alves, F. G. Guimarães, and R. C. Silva, "COVID-ABS: an agent-based model of COVID-19 epidemic to simulate health and economic effects of social distancing interventions," *Chaos, Solitons & Fractals*, 2020.

[21] G. R. Shinde, A. B. Kalamkar, P. N. Mahalle, N. Dey, J. Chaki, and A. E. Hassanien, "Forecasting models for coronavirus disease (covid-19): A survey of the state-of-the-art," *SN Computer Science*, 2020.

[22] R. N. Thompson, "Epidemiological models are important tools for guiding covid-19 interventions," *BMC Medicine*, vol. 18, no. 1, p. 152, 2020.

[23] T. Šušteršič, A. Blagojević, D. Cvetković, A. Cvetković, I. Lorencin, S. B. Šegota, D. Milovanović, D. Baskić, Z. Car, and N. Filipović, "Epidemiological predictive modeling of covid-19 infection: Development, testing, and implementation on the population of the benelux union," *Frontiers in Public Health*, vol. 9, 2021.

[24] N. G. Davies, A. J. Kucharski, R. M. Eggo, A. Gimma, W. J. Edmunds, T. Jombart, K. O'Reilly, A. Endo, J. Hellewell, E. S. Nightingale *et al.*, "Effects of non-pharmaceutical interventions on covid-19 cases, deaths, and demand for hospital services in the UK: a modelling study," *The Lancet Public Health*, 2020.

[25] D. Adam, "Special report: The simulations driving the world's response to COVID-19." *Nature*, 2020.

[26] P. Klepac, A. J. Kucharski, A. J. Conlan, S. Kissler, M. L. Tang, H. Fry, and J. R. Gog, "Contacts in context: large-scale setting-specific social mixing matrices from the BBC pandemic project," *MedRxiv*, 2020.

[27] N. Ferguson, "What would happen if a flu pandemic arose in Asia?" *Nature*, 2005.

[28] W. J. Edmunds, C. O'callaghan, and D. Nokes, "Who mixes with whom? a method to determine the contact patterns of adults that may lead to the spread of airborne infections," *Proceedings of the Royal Society of London. Series B: Biological Sciences*, 1997.

[29] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Communication-efficient learning of deep networks from decentralized data," in *International Conference on Artificial Intelligence and Statistics*, 2017.

[30] A. Bampoulidis, A. Bruni, L. Helminger, D. Kales, C. Rechberger, and R. Walch, "Privately connecting mobility to infectious diseases via applied cryptography," *PoPETs*, 2022.

[31] R. De Viti, I. Sheff, N. Glaeser, B. Dinis, R. Rodrigues, J. Katz, B. Bhattacharjee, A. Hithnawi, D. Garg *et al.*, "Covault: A secure analytics platform," 2022, https://arxiv.org/pdf/2208.03784.pdf.

[32] F. Al-Turjman and B. D. Deebak, "Privacy-aware energy-efficient framework using the internet of medical things for COVID-19," *IEEE Internet Things Mag.*, vol. 3, no. 3, 2020.

[33] M. Pezzutto, N. B. Rosselló, L. Schenato, and E. Garone, "Smart testing and selective quarantine for the control of epidemics," *Annu. Rev. Control.*, vol. 51, pp. 540–550, 2021.

[34] P. Barsocchi, A. Calabrò, A. Crivello, S. Daoudagh, F. Furfari, M. Girolami, and E. Marchetti, "COVID-19 & privacy: Enhancing of indoor localization architectures towards effective social distancing," *Array*, vol. 9, 2021.

[35] B. Bozdemir, S. Canard, O. Ermis, H. Möllering, M. Önen, and T. Schneider, "Privacy-preserving density-based clustering," in *ASIACCS*, 2021.

[36] M. Ciucci and F. Gouardères, "National COVID-19 contact tracing apps," *EPRS: European Parliamentary Research Service*, 2020.

[37] H. Stevens and M. B. Haines, "Tracetogether: Pandemic response, democracy, and technology," 2020, https://www.tracetogether.gov.sg.

[38] Inria and Fraunhofer AISEC, *ROBust and privacy-presERving proximity Tracing protocol*, 2020, https://github.com/ROBERT-proximity-tracing/documents.

[39] J. Chan, D. Foster, S. Gollakota, E. Horvitz, J. Jaeger,

S. Kakade, T. Kohno, J. Langford, J. Larson, P. Sharma, S. Singanamalla, J. Sunshine, and S. Tessaro, "PACT: Privacy sensitive protocols and mechanisms for mobile contact tracing," 2020, https://arxiv.org/pdf/2004.03544.pdf.

[40] F. Brauer, "Compartmental models in epidemiology," in *Mathematical Epidemiology*, 2008.

[41] F. Brauer, C. Castillo-Chavez, and Z. Feng, "Simple compartmental models for disease transmission," in *Mathematical Models in Epidemiology*, 2019.

[42] T. Harko, F. S. Lobo, and M. Mak, "Exact analytical solutions of the susceptible-infected-recovered (SIR) epidemic model and of the SIR model with equal death and birth rates," *Applied Mathematics and Computation*, 2014.

[43] R. Schlickeiser and M. Kröger, "Analytical modeling of the temporal evolution of epidemics outbreaks accounting for vaccinations," 2021.

[44] J. Fernández-Villaverde and C. I. Jones, "Estimating and simulating a sird model of covid-19 for many countries, states, and citie," *Journal of Economic Dynamics and Control*, 2022.

[45] A. Gray, D. Greenhalgh, L. Hu, X. Mao, and J. Pan, "A stochastic differential equation sis epidemic model," *SIAM Journal on Applied Mathematics*, 2011.

[46] S. He, Y. Peng, and K. Sun, "Seir modeling of the covid-19 and its dynamics," *Nonlinear Dynamics*, 2020.

[47] W. O. Kermack and A. G. McKendrick, "Contributions to the mathematical theory of epidemics—i," in *Bulletin of Mathematical Biology*, 1991.

[48] M. Small and C. K. Tse, "Small world and scale free model of transmission of SARS," in *International Journal of Bifurcation and Chaos*, 2005.

[49] Y.-C. Chen, P.-E. Lu, C.-S. Chang, and T.-H. Liu, "A time-dependent SIR model for covid-19 with undetectable infected persons," *Transactions on Network Science and Engineering*, 2020.

[50] R. M. May and A. L. Lloyd, "Infection dynamics on scale-free networks," *Phys. Rev. E*, 2001.

[51] N. M. Ferguson, D. A. Cummings, C. Fraser, J. C. Cajka, P. C. Cooley, and D. S. Burke, "Strategies for mitigating an influenza pandemic," *Nature*, 2006.

[52] K. Kupferschmidt, "Case clustering emerges as key pandemic puzzle," 2020, https://www.science.org/doi/full/10.1126/science.368.6493.808.

[53] S. Luo, F. Morone, C. Sarraute, M. Travizano, and H. A. Makse, "Inferring personal economic status from social network location," *Nature Communications*, 2017.

[54] Y.-A. d. Montjoye, J. Quoidbach, F. Robic, and A. S. Pentland, "Predicting personality using novel mobile phone-based metrics," in *International conference on social computing, behavioral-cultural modeling, and prediction*, 2013.

[55] T. Araki, J. Furukawa, K. Ohara, B. Pinkas, H. Rosemarin, and H. Tsuchida, "Secure graph analysis at scale," in *ACM CCS*, 2021.

[56] O. Goldreich, *Foundations of Cryptography: Volume 2, Basic Applications*. Cambridge University Press, 2009.

[57] H. Chaudhari, A. Choudhury, A. Patra, and A. Suresh, "ASTRA: High Throughput 3PC over Rings with Application to Secure Prediction," in *ACM CCSW@CCS*, 2019.

[58] P. Mishra, R. Lehmkuhl, A. Srinivasan, W. Zheng, and R. A. Popa, "Delphi: A cryptographic inference service for neural networks," in *USENIX Security*, 2020.

[59] A. Patra, T. Schneider, A. Suresh, and H. Yalame, "ABY2.0: Improved Mixed-Protocol Secure Two-Party Computation," in *USENIX Security*, 2021.

[60] M. Veeningen, S. Chatterjea, A. Z. Horváth, G. Spindler, E. Boersma, P. van der SPEK, O. Van Der Galiën, J. Gutteling, W. Kraaij, and T. Veugen, "Enabling analytics on sensitive medical data with secure multi-party computation," in *Medical Informatics Europe*, 2018.

[61] O. Tkachenko, C. Weinert, T. Schneider, and K. Hamacher, "Large-scale privacy-preserving statistical computations for distributed genome-wide association studies," in *ASIACCS*, 2018.

[62] T. Schneider and O. Tkachenko, "EPISODE: efficient privacy-preserving similar sequence queries on outsourced genomic databases," in *ASIACCS*, 2019.

[63] K. Järvinen, H. Leppäkoski, E.-S. Lohan, P. Richter, T. Schneider, O. Tkachenko, and Z. Yang, "PILOT: practical privacy-preserving indoor localization using outsourcing," in *EUROS&P*, 2019.

[64] C. van der Beets, R. Nieminen, and T. Schneider, "FAPRIL: towards faster privacy-preserving fingerprint-based localization," in *SECRYPT*, 2022.

[65] Y. Lindell and B. Pinkas, "An efficient protocol for secure two-party computation in the presence of malicious adversaries," in *EUROCRYPT*, 2007.

[66] Y. Aumann and Y. Lindell, "Security against covert adversaries: Efficient protocols for realistic adversaries," *Journal of Cryptology*, 2010.

[67] R. Lehmkuhl, P. Mishra, A. Srinivasan, and R. A. Popa, "MUSE: Secure inference resilient to malicious clients," in *USENIX Security*, 2021.

[68] N. Chandran, D. Gupta, S. L. B. Obbattu, and A. Shah, "SIMC: ML inference secure against malicious clients at semi-honest cost," in *USENIX Security*, 2022.

[69] O. Diekmann, H. Heesterbeek, and T. Britton, *Mathematical Tools for Understanding Infectious Disease Dynamics*. Princeton University Press, 2012.

[70] G. F. Hatke, M. Montanari, S. Appadwedula, M. Wentz, J. Meklenburg, L. Ivers, J. Watson, and P. Fiore, "Using bluetooth low energy (BLE) signal strength estimation to facilitate contact tracing for covid-19," 2020, https://arxiv.org/ftp/arxiv/papers/2006/2006.15711.pdf.

[71] Z. Erkin, J. R. Troncoso-pastoriza, R. L. Lagendijk, and F. Perez-Gonzalez, "Privacy-preserving data aggregation in smart metering systems: An overview," in *Signal Processing Magazine*, 2013.

[72] K. Kursawe, G. Danezis, and M. Kohlweiss, "Privacy-friendly aggregation for the smart-grid," in *PETS*, 2011.

[73] F. Li, B. Luo, and P. Liu, "Secure information aggregation for smart grids using homomorphic encryption," in *International Conference on Smart Grid Communications*, 2010.

[74] M. Nasr, R. Shokri, and A. Houmansadr, "Comprehensive privacy analysis of deep learning: Passive and active white-box inference attacks against centralized and federated learning," in *SP*, 2019.

[75] H. Fereidooni, S. Marchal, M. Miettinen, A. Mirhoseini, H. Möllering, T. D. Nguyen, P. Rieger, A.-R. Sadeghi, T. Schneider, H. Yalame *et al.*, "SAFELearn: secure aggregation for private federated learning," in *IEEE Security and Privacy Workshops (SPW)*, 2021.

[76] D. Günther, M. Holz, B. Judkewitz, H. Möllering, B. Pinkas, T. Schneider, and A. Suresh, "Privacy-Preserving Epidemiological Modeling on Mobile Graphs," arXiv preprint, 2022, https://arxiv.org/abs/2206.00539.

[77] N. Alexopoulos, A. Kiayias, R. Talviste, and T. Zacharias, "MCMix: Anonymous messaging via secure multiparty computation," in *USENIX Security*, 2017.

[78] T. Haines and J. Müller, "Sok: Techniques for verifiable mix nets," in *CSF*, 2020.

[79] S. Eskandarian and D. Boneh, "Clarion: Anonymous communication from multiparty shuffling protocols," in *NDSS*, 2022.

[80] I. Abraham, B. Pinkas, and A. Yanai, "Blinder: MPC

based scalable and robust anonymous committed broadcast," in *ACM CCS*, 2020.

[81] B. Pinkas, M. Rosulek, N. Trieu, and A. Yanai, "PSI from PaXoS Fast, Malicious Private Set Intersection," in *EUROCRYPT*, 2020.

[82] B. Chor, O. Goldreich, E. Kushilevitz, and M. Sudan, "Private information retrieval," in *FOCS*, 1995.

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

[84] D. Boneh, E. Boyle, H. Corrigan-Gibbs, N. Gilboa, and Y. Ishai, "Lightweight techniques for private heavy hitters," in *IEEE S&P*, 2021.

[85] H. Corrigan-Gibbs, D. Boneh, and D. Mazières, "Riposte: An anonymous messaging system handling millions of users," in *IEEE S&P*, 2015.

[86] N. Gilboa and Y. Ishai, "Distributed point functions and their applications," in *EUROCRYPT*, 2014.

[87] O. J. Walch, A. Cochran, and D. B. Forger, "A global quantification of "normal" sleep schedules using smartphone data," *Science advances*, 2016.

[88] V. Woollaston, *Sleeping habits of the world revealed: The US wakes up grumpy, China has the best quality shut-eye and South Africa gets up the earliest*, 2015, https://www.dailymail.co.uk/sciencetech/article-3042230/Sleeping-habits-world-revealed-wakes-grumpy-China-best-quality-shut-eye-South-Africa-wakes-earliest.html.

[89] Android, "Third-party Trusty applications," unk, https://source.android.com/security/trusty.

[90] D. Kales, O. Omolola, and S. Ramacher, "Revisiting user privacy for certificate transparency," in *EuroS&P*, 2019.

[91] B. Pinkas, T. Schneider, and M. Zohner, "Scalable private set intersection based on OT extension," *TOPS*, 2018.

[92] J. Mossong, N. Hens, M. Jit, P. Beutels, K. Auranen, R. Mikolajczyk, M. Massari, S. Salmaso, G. S. Tomba, J. Wallinga *et al.*, "Social contacts and mixing patterns relevant to the spread of infectious diseases," *PLoS Medicine*, 2008.

[93] S. Y. Del Valle, J. M. Hyman, H. W. Hethcote, and S. G. Eubank, "Mixing patterns between age groups in social networks," *Social Networks*, 2007.

[94] B. Pinkas, T. Schneider, C. Weinert, and U. Wieder, "Efficient circuit-based PSI via cuckoo hashing," in *EUROCRYPT*, 2018.

[95] S. Vaudenay and M. Vuagnoux, "Analysis of swisscovid," Tech. Rep., 2020.

[96] G. Avitabile, V. Botta, V. Iovino, and I. Visconti, "Towards defeating mass surveillance and SARS-CoV-2: The Pronto-C2 fully decentralized automatic contact tracing system," 2020. [Online]. Available: https://eprint.iacr.org/2020/493

[97] D. L. Chaum, "Untraceable electronic mail, return addresses, and digital pseudonyms," *Communications of the ACM*, 1981.

[98] D. Chaum, "The dining cryptographers problem: Unconditional sender and recipient untraceability," *Journal of Cryptology*, 1988.

[99] W. Du, "A study of several specific secure two party computation problems," *USA: Purdue University*, 2001.

[100] G. Wang, T. Luo, M. T. Goodrich, W. Du, and Z. Zhu, "Bureaucratic protocols for secure two-party sorting, selection, and permuting," in *ASIACCS*, 2010.

[101] Y. Huang, D. Evans, and J. Katz, "Private set intersection: Are garbled circuits better than custom protocols?" in *NDSS*, 2012.

[102] P. Mohassel and S. Sadeghian, "How to hide circuits in MPC an efficient framework for private function evaluation," in *EUROCRYPT*, 2013.

[103] S. Laur, J. Willemson, and B. Zhang, "Round-efficient oblivious database manipulation," in *International Conference on Information Security*, 2011.

[104] J. Ekberg, K. Kostiainen, and N. Asokan, "The untapped potential of trusted execution environments on mobile devices," in *S&P*, 2014.

[105] P. Jauernig, A. Sadeghi, and E. Stapf, "Trusted execution environments: Properties, applications, and challenges," in *S&P*, 2020.

[106] Intel, "Intel® software guard extensions programming reference," 2014, https://software.intel.com/sites/default/files/managed/48/88/329298-002.pdf.

[107] ARM, "ARM security technology building a secure system using trustzone technology," 2009, https://developer.arm.com/documentation/genc009492/c.

[108] B. Ngabonziza, D. Martin, A. Bailey, H. Cho, and S. Martin, "TrustZone explained: Architectural features and use cases," in *International Conference on Collaboration and Internet Computing*, 2016.

[109] O. Ohrimenko, F. Schuster, C. Fournet, A. Mehta, K. Vaswani, and M. Costa, "Oblivious multi-party machine learning on trusted processors," in *USENIX Security*, 2016.

[110] S. P. Bayerl, T. Frassetto, P. Jauernig, K. Riedhammer, A.-R. Sadeghi, T. Schneider, E. Stapf, and C. Weinert, "Offline model guard: Secure and private ML on mobile devices," *DATE*, 2020.

[111] G. Chen, Y. Zhang, and T.-H. Lai, "OPERA: Open Remote Attestation for Intel's Secure Enclaves," in *CCS*, 2019.

[112] Intel, "Attestation service for intel software guard extensions," unk, https://api.trustedservices.intel.com/documents/sgx-attestation-api-spec.pdf.

[113] E. Kushilevitz and R. Ostrovsky, "Replication is NOT needed: SINGLE database, computationally-private information retrieval," in *FOCS*, 1997.

[114] S. Angel, H. Chen, K. Laine, and S. Setty, "PIR with Compressed Queries and Amortized Query Processing," in *IEEE S&P*, 2018.

[115] C. Gentry and S. Halevi, "Compressible FHE with Applications to PIR," in *TCC*, 2019.

[116] H. Corrigan-Gibbs and D. Kogan, "Private information retrieval with sublinear online time," in *EUROCRYPT*, 2020.

[117] B. Pinkas, T. Schneider, G. Segev, and M. Zohner, "Phasing: Private set intersection using permutation-based hashing," in *USENIX Security*, 2015.

[118] C. Dong, L. Chen, and Z. Wen, "When private set intersection meets big data: an efficient and scalable protocol," in *ACM CCS*, 2013.

[119] R. Pagh and F. F. Rodler, "Cuckoo hashing," *Journal of Algorithms*, 2004.

[120] A. Kirsch, M. Mitzenmacher, and U. Wieder, "More robust hashing: Cuckoo hashing with a stash," *Journal on Computing*, 2010.

[121] A. C.-C. Yao, "How to Generate and Exchange Secrets," in *FOCS*, 1986.

[122] I. Damgård, M. Keller, E. Larraia, V. Pastro, P. Scholl, and N. P. Smart, "Practical covertly secure MPC for dishonest majority – or: Breaking the SPDZ limits," in *ESORICS*, 2013.

[123] Y. Lindell, B. Pinkas, N. P. Smart, and A. Yanai, "Efficient constant round multi-party computation combining BMR and SPDZ," in *CRYPTO*, 2015.

[124] D. Demmler, T. Schneider, and M. Zohner, "ABY - A Framework for Efficient Mixed-Protocol Secure Two-Party Computation," in *NDSS*, 2015.

[125] A. Patra and A. Suresh, "BLAZE: Blazing Fast Privacy-Preserving Machine Learning," in *NDSS*, 2020.

[126] M. Byali, H. Chaudhari, A. Patra, and A. Suresh, “FLASH: Fast and Robust Framework for Privacy-preserving Machine Learning,” *PETS*, 2020.

[127] H. Chaudhari, R. Rachuri, and A. Suresh, “Trident: Efficient 4PC Framework for Privacy Preserving Machine Learning,” in *NDSS*, 2020.

[128] N. Koti, M. Pancholi, A. Patra, and A. Suresh, “SWIFT: Super-fast and Robust Privacy-Preserving Machine Learning,” in *USENIX Security*, 2021.

[129] N. Koti, A. Patra, R. Rachuri, and A. Suresh, “Tetrad: Actively Secure 4PC for Secure Training and Inference,” in *NDSS*, 2022.

[130] D. Chaum, “Security without identification: Transaction systems to make big brother obsolete,” in *Communications of the ACM*, 1985.

[131] C. Paquin and G. Zaveruch, *U-Prove Cryptographic Specification V1.1 (Revision 3)*, 2013, http://www.microsoft.com/uprove.

[132] G. Danezis and L. Sassaman, “Heartbeat traffic to counter (n-1) attacks: Red-green-black mixes,” ser. WPES ’03, 2003.

[133] L. de Castro and A. Polychroniadou, “Lightweight, maliciously secure verifiable function secret sharing,” in *EUROCRYPT*, 2022.

[134] V. I. Kolobov, E. Boyle, N. Gilboa, and Y. Ishai, “Programmable distributed point functions,” in *CRYPTO*, 2022.

[135] D. Günther, M. Heymann, B. Pinkas, and T. Schneider, “GPU-accelerated PIR with client-independent preprocessing for large-scale applications,” in *USENIX Security*, 2022.

Appendix A
Related Primitives

In the following, we provide an overview about the (cryptographic) primitives and other techniques used in this work.

A. Anonymous Communication

To simulate the transmission of the modelled disease, RIPPLE requires anonymous messaging between participants. Mix-nets [97] and protocols based on the dining cryptographer (DC) problem [98] were the first approaches to anonymous messaging. A fundamental technique underlying mix-nets is the execution of an oblivious shuffling algorithm that provides unlinkability between the messages before and after the shuffle. In a mix-net, so-called mix servers jointly perform the oblivious shuffling so that no single mix server is able to reconstruct the permutation performed on the input data. Past research established a wide variety of oblivious shuffle protocols based on garbled circuits [99]–[101], homomorphic encryption [101], distributed point functions [80], switching networks [102], permutation matrices [103, §4.1], sorting algorithms [103, §4.2], and re-sharing [103, §4.3+4.4]. Recently, the works of [55] and [79] proposed efficient oblivious shuffling schemes using a small number of mix net servers.

B. Trusted Execution Environment (TEE)

RIPPLE$_{\mathsf{TEE}}$ (§IV-A) assumes the availability of TEEs on the mobile devices of participants. TEEs are hardware-assisted environments providing secure storage and execution for sensitive data and applications isolated from the normal execution environment. Data stored in a TEE is secure even if the operating system is compromised, i.e., it offers confidentiality, integrity, and access control [104], [105]. Widely adopted TEEs are Intel SGX [106] and ARM TrustZone [107] (often used on mobile platforms [108]). Using TEEs for private computation has been extensively investigated, e.g., [109], [110]. A process called remote attestation allows external parties to verify that its private data sent via a secure channel is received and processed inside the TEE using the intended code [111], [112].

C. Private Information Retrieval (PIR)

The first computational single-server PIR (cPIR) scheme was introduced by Kushilevitz and Ostrovsky [113]. Recent cPIR schemes [114], [115] use homomorphic encryption (HE). However, single-server PIR suffers from significant computation overhead since compute intensive HE operations have to be computed on each of the database block for each PIR request. In contrast, multi-server PIR relies on a non-collusion assumption between multiple PIR servers and uses only XOR operations [82]–[85], [116] making it significantly more efficient than cPIR.

D. Cuckoo Hashing

In RIPPLE$_{\mathsf{PIR}}$ (§IV-B), messages of participants have to be stored in a database D. To do so, a hash function H can be used to map an element x into bins of the database: $D[H(x)] = x$. However, as we show in §IV-B, RIPPLE$_{\mathsf{PIR}}$ requires that at most one element is stored in every database location which renders simple hashing impracticable [117]. Cuckoo hashing uses h hash functions $H_1, \ldots, H_h$ to map elements into bins. It ensures that each bin contains exactly one element. If a collision occurs, i.e., if a new element is to be added into an already occupied bin, the old element is removed to make space for the new one. The evicted element, then, is placed into a new bin using another of the h hash functions. If the insertion fails for a certain number of trials, an element is inserted into a special bin called stash which is the only one that is allowed to hold more than one element. Pinkas et al. [117] show that for $h = 2$ hash functions and $n = 2^{20}$ elements inserted to $2.4n$ bins, a stash size of 3 is sufficient to have a negligible error probability.

E. Garbled Cuckoo Table (GCT)

As RIPPLE$_{\mathsf{PIR}}$ uses key-value pairs for the insertion into the database, a combination of garbled Bloom filters [118] with cuckoo hashing [119], [120], called Garbled Cuckoo Table [81], is needed. Instead of storing x elements in one bin as in an ordinary cuckoo table (cf. §A-D), in a GCT h XOR shares of x are stored at the h locations determined by inputting k into all h hash functions. E.g., with $h = 2$, if one of these two locations is already in use, the XOR share for the other (free) location is set to be the XOR of x and the data stored in the used location. In §IV-B2, we introduce a variant of GCT called *arithmetic garbled cuckoo table* (AGCT) that uses arithmetic sharing over the ring $\mathbb{Z}_{2^\ell}$ instead of XOR sharing. For a database with $2.4n$ entries where n is the number of elements inserted, Pinkas et al. [81] show that the number of cycles is maximally $\log n$ with high probability.

F. Secure Multi-Party Computation (MPC)

MPC [121] allows a set of mutually distrusting parties to jointly compute an arbitrary function on their private inputs without leaking anything but the output. In the last years, MPC techniques in various security models have been introduced, extensively studied, and improved, e.g., in [122]–[124]. These advancements significantly enhance the efficiency of MPC making it more and more practical for real-world applications. Due to the practical efficiency it can provide, various works [55], [125]–[129] have recently concentrated on MPC for a small number of parties, especially in the three and four party honest majority setting tolerating one corruption. In RIPPLE, we employ MPC techniques across three servers to enable an anonymous communication channel (cf. §B-C) and to develop efficient $\mathsf{PIR}_{\mathsf{sum}}$ protocols (cf. §V).

G. Anonymous Credentials

To protect against sybil attacks (cf. §III-C), i.e., to hinder an adversary from creating multiple identities that can collect encounter information to detect correlations among unconscious encounters, we suggest to use anonymous credentials such that only registered participants can join RIPPLE. In this manner, the registration process can, for example, be linked to a passport. Such a registration system increases the cost to create (fake) identities. Chaum [130] introduced anonymous credentials where a client holds the credentials of several unlinkable pseudonyms. The client can then prove that it possesses the credentials of pseudonyms without the service provider being able to link different pseudonyms to the same identity. Additionally, anonymous credentials allow to certify specific properties like the age. Several instantiations for anonymous credentials have been proposed, e.g., Microsoft U-Prove [131].

Appendix B
Building Blocks in RIPPLE

This section contains details about the building blocks used in the RIPPLE framework, such as shared-key setup, collision-resistant hash functions, anonymous communication channels, and Distributed Point Functions.

A. Shared-Key Setup

Let $F : \{0,1\}^\kappa \times \{0,1\}^\kappa \rightarrow X$ be a secure pseudo-random function (PRF), with co-domain X being $\mathbb{Z}_{2^\ell}$ and $\mathcal{C}' = \mathcal{C} \cup \{\mathcal{P}_i\}$ for a participant $\mathcal{P}_i \in \mathcal{P}$. The following PRF keys are established among the parties in $\mathcal{C}'$ in RIPPLE:

– k_{ij} among every $P_i, P_j \in \mathcal{C}'$ and $i \neq j$.
– k_{ijk} among every $P_i, P_j, P_k \in \mathcal{C}'$ and $i \neq j \neq k$.
– $k_{\mathcal{C}'}$ among all the parties in $\mathcal{C}'$.

To sample a random value $r_{ij} \in \mathbb{Z}_{2^\ell}$ non-interactively, each of P_i and P_j can invoke $F_{k_{ij}}(id_{ij})$. In this case, id_{ij} is a counter that P_i and P_j maintain and update after each PRF invocation. The appropriate sampling keys are implied by the context and are, thus, omitted.

B. Collision Resistant Hash Function

A family of hash functions $\{\mathsf{H} : \mathcal{K} \times \mathcal{L} \rightarrow \mathcal{Y}\}$ is said to be *collision resistant* if, for all probabilistic polynomial-time adversaries $\mathcal{A}$, given the description of H_k, where $k \in_R \mathcal{K}$, there exists a negligible function $negl()$ such that $\Pr[(x, x') \leftarrow \mathcal{A}(k) : (x \neq x') \wedge \mathsf{H}_k(x) = \mathsf{H}_k(x')] = negl(\kappa)$, where $x, x' \in_R \{0,1\}^m$ and $m = \text{poly}(\kappa)$.

C. Anonymous Communication Channel

This section describes how to instantiate the $\mathcal{F}_{\mathsf{anon}}$ functionality used by RIPPLE for anonymous communication, as discussed in §IV. We start with the protocol for the case of $\mathsf{RIPPLE}_{\mathsf{PIR}}$ and then show how to optimize it for the use in the $\mathsf{RIPPLE}_{\mathsf{TEE}}$ protocol. Recall from §IV-B that in $\mathsf{RIPPLE}_{\mathsf{PIR}}$, participants in $\mathcal{P}$ upload a set of messages from which a database D must be constructed at the end by S_1 and S_2. The anonymous communication is required to ensure that neither S_1 nor S_2 can link the source of the message even after receiving all messages in clear, which may not be in the same order. To tackle this problem, we use an approach based on oblivious shuffling inspired by [55], [79], which is formalised next.

Problem Statement. Consider the vector $\vec{m} = \{m_1, \ldots, m_\tau\}$ of τ messages with $m_j \in \mathbb{Z}_{2^\ell}$ for $j \in [\tau]$. We want servers S_1 and S_2 to obtain $\pi(\vec{m})$, where $\pi()$ denotes a random permutation that neither S_1 nor S_2 knows. Furthermore, an attacker with access to a portion of the network and, hence, the ability to monitor network data should not be able to gain any information about the permutation $\pi()$.

In $\mathsf{RIPPLE}_{\mathsf{PIR}}$, the vector $\vec{m}$ corresponds to the infection likelihood messages of the form $(a_{i,j}, c^e_{i,j})$ that each participant $\mathcal{P}_i \in \mathcal{P}$ sends over the network (cf. §IV-B). W.l.o.g., we let $\mathcal{P}_i$ have the complete $\vec{m}$ with them. The protocol makes use of the third server S_0 in our setting and proceeds as follows:

1. $\mathcal{P}_i$ generates an additive sharing of $\vec{m}$ among S_0 and S_1:
   a) $\mathcal{P}_i, \mathsf{S}_0$ sample random $\langle \vec{m} \rangle_1 \in \mathbb{Z}^\tau_{2^\ell}$.

b) $\mathcal{P}_i$ computes and sends $\langle\vec{m}\rangle_2 = \vec{m} - \langle\vec{m}\rangle_1$ to S_1.

2. S_0 and S_1 agree on a random permutation π_{01} and locally apply π_{01} to their shares. Let $\pi_{01}(\vec{m}) = \pi_{01}(\langle\vec{m}\rangle_1) + \pi_{01}(\langle\vec{m}\rangle_2)$.
3. $\mathsf{S}_0, \mathsf{S}_1$ perform a *re-sharing* of $\pi_{01}(\vec{m})$, denoted by $\vec{m_{01}}$, by jointly sampling a random $\vec{r_{01}} \in \mathbb{Z}_{2^\ell}^{\tau}$ and setting $\langle\vec{m_{01}}\rangle_1 = \pi_{01}(\langle\vec{m}\rangle_1) + \vec{r_{01}}$ and $\langle\vec{m_{01}}\rangle_2 = \pi_{01}(\langle\vec{m}\rangle_2) - \vec{r_{01}}$.
4. S_1 sends $\langle\vec{m_{01}}\rangle_2$ to S_2. Now, $(\langle\vec{m_{01}}\rangle_1, \langle\vec{m_{01}}\rangle_2)$ forms an additive sharing of $\vec{m_{01}}$ among S_0 and S_2.
5. S_0 and S_2 agree on a random permutation π_{02} and apply π_{02} to their shares. Let $\pi_{02}(\vec{m_{01}}) = \pi_{02}(\langle\vec{m_{01}}\rangle_1) + \pi_{02}(\langle\vec{m_{01}}\rangle_2)$.
6. S_0 sends $\pi_{02}(\langle\vec{m_{01}}\rangle_1)$ to S_2, who reconstructs $\pi_{02}(\vec{m_{01}})$.
7. S_2 generates an *additive-sharing* of $\pi_{02}(\vec{m_{01}})$, denoted by $\vec{m_{02}}$, among S_1 and S_2, by jointly sampling $\langle\vec{m_{02}}\rangle_1 \in \mathbb{Z}_{2^\ell}^{\tau}$ with S_1 and locally setting $\langle\vec{m_{02}}\rangle_2 = \pi_{02}(\vec{m_{01}}) - \langle\vec{m_{02}}\rangle_1$.
8. S_2 sends $\langle\vec{m_{02}}\rangle_2$ to S_1, who locally compute the output as $\vec{m_{02}} = \langle\vec{m_{02}}\rangle_1 + \langle\vec{m_{02}}\rangle_2$.

a) Anonymous Communication in RIPPLE$_{\mathsf{TEE}}$*.:* As discussed in §IV-A, the server S_2 is only required to have the complete set of messages in the clear but in an unknown random order. As a result, in the case of RIPPLE$_{\mathsf{TEE}}$, only the first permutation (π_{01} in Step 2) is sufficient and steps 5-8 are no longer required. Furthermore, in addition to the communication by S_1 in step 4, S_0 sends its share of $\vec{m_{01}}$ to S_2, who can then reconstruct $\vec{m_{01}} = \pi_{01}(\vec{m})$.

b) Security Guarantees.: As discussed in §III-A, we assume that the MPC servers $\mathsf{S}_i, i \in [2]$, that also instantiate the anonymous communication channel are semi-honest. We claim that the protocol described above will produce a random permutation of the vector $\vec{m}$ that neither S_1 nor S_2 is aware of. To see this, note that

$$\vec{m_{02}} = \pi_{02}(\vec{m_{01}}) = \pi_{02}(\pi_{01}(\vec{m}))$$

and both S_1 and S_2 know only one of the two permutations π_{01} and π_{02}, but not both. Furthermore, the re-sharing performed in step 3 and the generation of additive shares in step 6 above ensures that an attacker observing the traffic cannot relate messages sent and received.

As we also consider a client-malicious security model [67], [68], where some clients might deviate from the protocol to gain additional information, we also have to take into consideration how the clients could manipulate the communication to break anonymity. For RIPPLE$_{\mathsf{TEE}}$, this is trivial: The TEE ensures that clients' messages are correctly generated and uploaded. For RIPPLE$_{\mathsf{PIR}}$, a malicious client could manipulate how many messages it uploads. However, messages with addresses that are already used will be dropped, i.e., effectively removing the malicious client from the system. A receiver will never fetch messages with unknown, random addresses. Furthermore, the servers use secure communication channels and even send freshly re-shared shares. We also consider a global attacker being able to monitor the full network traffic to be unrealistic. Hence, considering the discussed aspects/assumptions, classical attacks on anonymous communication such as flooding [132] are not relevant for our model.

D. Distributed Point Functions (DPF)

Consider a point function $P_{\alpha,\beta} : \mathbb{Z}_{2^\ell} \to \mathbb{Z}_{2^{\ell'}}$ such that for all $\alpha \in \mathbb{Z}_{2^\ell}$ and $\beta \in \mathbb{Z}_{2^{\ell'}}$, $P_{\alpha,\beta}(\alpha) = \beta$ and $P_{\alpha,\beta}(\alpha') = 0$ for all $\alpha' \neq \alpha$. That is, when evaluated at any input other than α, the point function $P_{\alpha,\beta}$ returns 0 and when evaluated at α it returns β.

An (s,t)-distributed point function (DPF) [85], [86] distributes a point function $P_{\alpha,\beta}$ among s servers in such a way that no coalition of at most t servers learns anything about α or β given their t shares of the function. We use $(2,1)$-DPFs in RIPPLE to optimize the communication of PIR-based protocols, as discussed in §V-C. Formally, a $(2,1)$-DPF comprises of the following two functionalities:

– $\mathsf{Gen}(\alpha,\beta) \to (k_1,k_2)$. Output two DPF keys k_1 and k_2, given $\alpha \in \mathbb{Z}_{2^\ell}$ and $\beta \in \mathbb{Z}_{2^{\ell'}}$.
– $\mathsf{Eval}(k,\alpha') \to \beta'$. Return $\beta' \in \mathbb{Z}_{2^{\ell'}}$, given key k generated using Gen, and an index $\alpha' \in \mathbb{Z}_{2^\ell}$.

A $(2,1)$-DPF is said to be *correct* if for all $\alpha, x \in \mathbb{Z}_{2^\ell}$, $\beta \in \mathbb{Z}_{2^{\ell'}}$, and $(k_1,k_2) \leftarrow \mathsf{Gen}(\alpha,\beta)$, it holds that

$$\mathsf{Eval}(k_1,x) + \mathsf{Eval}(k_2,x) = (x = \alpha)\ ?\ \beta : 0.$$

A $(2,1)$-DPF is said to be *private* if neither of the keys k_1 and k_2 leaks any information about α or β. That is, there exists a polynomial time algorithm that can generate a computationally indistinguishable view of an adversary $\mathcal{A}$ holding DPF key k_u for $u \in \{1,2\}$, when given the key k_u.

As mentioned in [83], [85], a malicious participant could manipulate the Gen algorithm to generate incorrect DPF keys that do not correspond to any point function. While [85] used an external non-colluding auditor to circumvent this issue in the two server setting, [83] formalised this issue and proposed an enhanced version of DPF called Verifiable DPFs. In addition to the standard DPF, a verifiable DPF has an additional function called Ver that can be used to ensure the correctness of the DPF keys. In contrast to Eval, Ver in a $(2,1)$-verifiable DPF is an interactive protocol between the two servers, with the algorithm returning a single bit indicating whether the input DPF keys k_1 and k_2 are valid.

A verifiable DPF is said to be *correct* if for all $\alpha \in \mathbb{Z}_{2^\ell}$, $\beta \in \mathbb{Z}_{2^{\ell'}}$, keys $(k_1,k_2) \leftarrow \mathsf{Gen}(\alpha,\beta)$, the verify protocol Ver outputs 1 with probability 1. Ver should ensure that no additional information about α or β is disclosed to the party in possession of one of the DPF keys. Furthermore, the probability that

Ver outputs 1 to at least one of the two servers for a given invalid key pair (k'_1, k'_2) is negligible in the security parameter κ.

Recent results in the area of (verifiable) DPFs [133], [134] might be an interesting direction for future work to further enhance the efficiency of our RIPPLE$_{\mathsf{PIR}}$ construction.

a) Communication Complexity.: Using the protocol of Boyle et. al. [83], a $(2,1)$-DPF protocol for a point function with domain size N has key size $(\lambda+2)\cdot\log(N/\lambda)+2\cdot\lambda$ bits, where $\lambda = 128$ for an AES based implementation. The additional cost in the case of verifiable DPF is for executing the Ver function, which has a constant number of elements in [83]. Furthermore, as stated in [83], the presence of additional non-colluding servers can improve the efficiency of Ver, and we use S_0 in the case of $\mathsf{PIR}^{\mathsf{I}}_{\mathsf{sum}}$, as discussed in §V-C1. We refer to [83] for more details regarding the scheme.

Appendix C
PIR-SUM Protocol Details

This section provides additional details of our $\mathsf{PIR}_{\mathsf{sum}}$ protocols introduced in §V-A. We begin by recalling the security guarantees of a 2-server PIR for our setting [82], [135]. Informally in a two-server PIR protocol, where the database D is held by two non-colluding servers S_1 and S_2, a single server $\mathsf{S}_u \in \{\mathsf{S}_1, \mathsf{S}_2\}$ should not learn any information about the client's query. The security requirement is formally captured in Definition 1.

Definition 1. *(Security of 2-server PIR) A PIR scheme with two non-colluding servers is called secure if each of the servers does not learn any information about the query indices.*

Let $view(\mathsf{S}_u, \mathcal{Q})$ denote the view of server $\mathsf{S}_u \in \{\mathsf{S}_1, \mathsf{S}_2\}$ with respect to a list of queries, denoted by $\mathcal{Q}$. We require that for any database D, and for any two τ-length list of queries $\mathcal{Q} = (q_1, \ldots, q_\tau)$ and $\mathcal{Q}' = (q'_1, \ldots, q'_\tau)$, no algorithm whose run time is polynomial in τ and in computational parameter κ can distinguish the view of the servers S_1 and S_2, between the case of participant $\mathcal{P}_i$ using the queries in $\mathcal{Q}$ ($\{view(\mathsf{S}_u, \mathcal{Q})\}_{u\in\{1,2\}}$), and the case of it using $\mathcal{Q}'$ ($\{view(\mathsf{S}_u, \mathcal{Q}')\}_{u\in\{1,2\}}$).

A. Linear Summation PIR for $\mathcal{F}^{\mathsf{2S}}_{\mathsf{pir}}$ with optimized Communication.

This section describes the 2-server linear summation PIR protocol in [82], as well as how to optimize communication using DPF techniques discussed in Appendix B-D. To retrieve the q-th block from database D of size N, the linear summation PIR proceeds as follows:

- Participant $\mathcal{P}_i$ prepares an N-bit string $\vec{\mathsf{b}}_q = \{\mathsf{b}^1_q, \ldots, \mathsf{b}^N_q\}$ with $\mathsf{b}^j_q = 1$ for $j = q$ and $\mathsf{b}^j_q = 0$ and $j \neq q$, for $j \in [N]$.
- $\mathcal{P}_i$ generates a Boolean sharing of $\vec{\mathsf{b}}_q$ among S_1 and S_2, i.e., $\mathcal{P}_i, \mathsf{S}_1$ non-interactively sample the random $[\vec{\mathsf{b}}_q]_1 \in \{0,1\}^N$ and $\mathcal{P}_i$ sends $[\vec{\mathsf{b}}_q]_2 = \vec{\mathsf{b}}_q \oplus [\vec{\mathsf{b}}_q]_1$ to S_2.
- S_u, for $u \in \{1,2\}$, sends $[y]_u = \bigoplus_{j=1}^{N} [\mathsf{b}^j_q]_u \mathsf{D}[j]$ to $\mathcal{P}_i$.
- $\mathcal{P}_i$ locally computes $\mathsf{D}[q] = [y]_1 \oplus [y]_2$.

The linear summation PIR described above requires communication of $N + 2\ell$ bits, where ℓ denotes the size of each data block in D.

1) Optimizing Communication using DPFs.: Several works in the literature [83], [85], [86], [135] have used DPFs (cf. Appendix B-D) as a primitive to improve the communication in multi-server PIR. The idea is to use a DPF function to allow the servers S_1 and S_2 to obtain the XOR shares of an N-bit string $\vec{\mathsf{b}}$ that has a zero in all positions except the one representing the query q. Because DPF keys are much smaller in size than the actual database size, this method aids in the elimination of N-bit communication from $\mathcal{P}_i$ to the servers, as in the aforementioned linear summation PIR.

To query the q-th block from a database D of size N,

- Participant $\mathcal{P}_i$ executes the key generation algorithm with input q to obtain two DPF keys, i.e., $(k_1, k_2) \leftarrow \mathsf{Gen}(q, 1)$.
- $\mathcal{P}_i$ sends k_u to S_u, for $u \in \{1,2\}$.
- S_u, for $u \in \{1,2\}$, performs a DPF evaluation at each of the positions $j \in [N]$ using key k_u and obtains the XOR share corresponding to bit vector $\vec{\mathsf{b}}_q$.
    - S_u expands the DPF keys as $[\mathsf{b}^j_q]_u \leftarrow \mathsf{Eval}(k_u, j)$ for $j \in [N]$.
- S_u, for $u \in \{1,2\}$, sends $[y]_u = \bigoplus_{j=1}^{N} [\mathsf{b}^j_q]_u \mathsf{D}[j]$ to $\mathcal{P}_i$.
- $\mathcal{P}_i$ locally computes $\mathsf{D}[q] = [y]_1 \oplus [y]_2$.

For the case of semi-honest participants, we use the DPF protocol of [83] and the key size is $O(\lambda \cdot \log(N/\lambda))$ bits, where $\lambda = 128$ is related to AES implementation in [83].

To prevent a malicious participant from sending incorrect or malformed keys to the servers, we use the verifiable DPF construction proposed in [83] for the case of malicious participants. This results only in a constant communication overhead over the semi-honest case. Furthermore, as noted in [83], we use the additional server S_0 for a better instantiation of the verifiable DPF, removing the need for interaction with the participant $\mathcal{P}_i$ for verification. We provide more information in Appendix B-D and refer the reader to [83] for all details.