Privacy-Preserving Epidemiological Modeling on Mobile Graphs

DANIEL GÜNTHER, MARCO HOLZ, ENCRYPTO, Technical University of Darmstadt, Germany
BENJAMIN JUDKEWITZ, Charité-Universitätsmedizin, Germany
HELEN MÖLLERING, ENCRYPTO, Technical University of Darmstadt, Germany
BENNY PINKAS, Bar-Ilan University, Israel
THOMAS SCHNEIDER, ENCRYPTO, Technical University of Darmstadt, Germany
AJITH SURESH, Technology Innovation Institute, Abu Dhabi

Since 2020, governments all over the world have used a variety of containment measures to control the spread of COVID-19, 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, their predictive accuracy is hampered by the scarcity of relevant empirical data, specifically detailed social contact graphs. As this data is inherently privacy-critical, there is an urgent need for a method to perform powerful epidemiological simulations on real-world contact graphs without disclosing sensitive information.

In this work, we present RIPPLE, a privacy-preserving epidemiological modeling framework that enables the execution of standard epidemiological models for infectious disease on a population's most recent real contact graph while keeping all contact information privately and locally on the participants' devices. As underlying building block, we present PIR-SUM, a novel extension to private information retrieval that allows users to securely download the sum of a set of elements from a database rather than individual elements. We provide a proof-of-concept implementation of our protocols demonstrating that a 2-week simulation over a population of half a million can be finished in 7 minutes, with each participant communicating less than 50 KB of data.

CCS Concepts: • **Security and privacy → Privacy-preserving protocols**; Social network security and privacy; Information accountability and usage control; **Privacy protections**.

ACM Reference Format:
Daniel Günther, Marco Holz, Benjamin Judkewitz, Helen Möllering, Benny Pinkas, Thomas Schneider, and Ajith Suresh. 2018. Privacy-Preserving Epidemiological Modeling on Mobile Graphs. In . ACM, New York, NY, USA, 36 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

The COVID-19 pandemic has profoundly affected people's daily lives, posing significant challenges such as increased mental illness, balancing childcare, homeschooling, and work, an increase in domestic abuse cases, and many more [91, 119, 128]. Governments all over the world have taken a variety of steps to restrict the spread of the virus to save human lives and keep the economic system working. Those range from closing institutions, such as schools, to country-wide lockdowns. Despite these courageous efforts, the global number of infections skyrocketed, and COVID-19 claimed far too many lives. Aside from highly lethal diseases like COVID-19, many other infectious diseases have emerged and have had a significant impact on human life over time. For example, since 2022 incidences of mpox (previously known as monkeypox) in Europe have increased to the point that quarantine measures have been implemented [23, 34, 56].

In the context of COVID-19, contact tracing apps are being used all over the world to notify contacts of potential infections [108, 113]. Unfortunately, contact tracing has a fundamental limitation: 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 previous contacts [85, 123]. Tupper et al. [123] 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 effectively control disease spread.

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 [59, 133]. Epidemiological modeling allows to assess the effectiveness of containment measures by mathematically modeling their impact on the spread, aiding governments in selecting effective strategies [120]. For example, Davis et al. [40] 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× 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 [120].

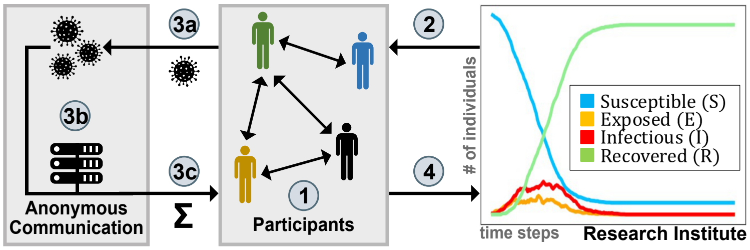
With access to detailed information about a population's size, density, transportation, and health care system, epidemiological modeling could accurately forecast disease transmission in a variety of situations [2]. Especially precise, up-to-date information about movements and physical interactions in space and time is crucial for precisely forecasting transmission as well as the impact of various control measures before being implemented [76]. In practice, these simulations may quickly model a future disease's spread, calculate the projected number of infections when specific actions are taken, and divert the spread to specific areas.

However, data on personal encounters in the real-world is very scarce and, thus the impact of containment measures can only be approximated so far [2, 53, 76]. This lack of data is primarily owing to the fact that encounter data has generally been acquired by surveys, which do not accurately reflect reality [48, 76], e.g., random encounters in public transport or shopping malls. Moreover, social interaction patterns change over time and sometimes even rapidly, as we have seen with social distancing measures, rendering collected contact information outdated. Hence, none of the existing data permits realistic simulations on the actual person-to-person social contact graph. Epidemiologists desire the full physical interaction graph of a population from a modeler's standpoint. Yet, strict data protection regulations such as in democratic states, honoring privacy rights, hinder accurate tracking of interpersonal contacts.

To address the issue of obtaining the most recent contact data while protecting individuals' privacy, we present RIPPLE, the first privacy-preserving framework for epidemiological modeling that allows precise simulations of disease spread based on current physical contact information while taking into account deployed control measures and without leaking any 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 most recent contact graph of the previous days in a privacy-preserving manner. RIPPLE can be used to investigate the effect of containment measures not only for COVID-19, but for *any* infectious diseases. We anticipate that our framework's privacy guarantee will encourage more people to participate, allowing epidemiologists to compute more accurate simulations that will eventually help to develop effective containment measures against diseases in the future.

[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 §2.

Our Contributions. This paper introduces RIPPLE (cf. Fig. 1), a framework that expands the scope of privacy research from contact tracing to epidemiological modeling. While contact tracing 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 prior to their implementation.



① Mobile app collects 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 [93], fostering trust and widespread participation, and encouraging participants to contribute representative contact information. All participant data, such as encounter location, time, and distance, are kept locally on the participants' devices. Communication among participants occurs through anonymous channels facilitated by a group of semi-honest central servers.

RIPPLE offers two methods for privacy-preserving epidemiological modeling, each covering different use cases. The first method, RIPPLE$_{\mathrm{TEE}}$, relies on the presence of a Trusted Execution Environment (TEE) on participants' mobile devices. The second method, RIPPLE$_{\mathrm{PIR}}$, eliminates this assumption by utilising cryptographic primitives like Private Information Retrieval (PIR). Along the way, we develop a multi-server PIR extension enabling clients to retrieve the sum of a set of elements (in our case, infection likelihoods) from a database without learning anything about individual entries.

We assess the practicality of our methods by benchmarking core components using a proof of concept implementation. Our results show that, with adequate hardware, both protocols scale up to millions of participants. For instance, simulating 14 days with 1 million participants takes less than 30 minutes to complete. We summarize our contributions as follows:

(1) We present RIPPLE, the *first* privacy-preserving framework to perform epidemiological modeling on contact information stored on mobile devices. RIPPLE formalises the notion of *privacy-preserving* epidemiological modeling and defines privacy requirements.
(2) For epidemiological simulations using real-world contact data acquired with participants' mobile devices, we present two techniques – RIPPLE$_{\mathrm{TEE}}$ and RIPPLE$_{\mathrm{PIR}}$ – that combine anonymous communication techniques with either TEEs or PIR and anonymous credentials.
(3) 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.
(4) We demonstrate the practicality of our framework by providing a detailed performance evaluation using our open source implementation of RIPPLE.

2 RELATED WORK & PRELIMINIARIES

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 used in this work is presented in §A.

2.1 Privacy-preserving Solutions in the Context of Infectious Diseases

CrowdNotifier [89] 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. [13] introduce a privacy-preserving two-party set intersection protocol that detects infection hotspots by intersecting infected patients, input by a health institute, with customer data from mobile network operators.

CoVault [42] is a data analytics platform based on secure multi-party computation techniques (MPC) and trusted execution environments (TEEs). 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 [4] 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. [107] 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-aware environment to identify individuals who are at high risk of infection. Barocchi et al. [14] 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. [19] suggest privacy-preserving trajectory clustering to identify typical movements of people allowing to detect forbidden gatherings when contact restrictions are in place.

Contact Tracing. A plethora of contact tracing systems has been introduced and deployed since the outbreak of the pandemic [3, 35, 113]. 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 [125]. In a centralized contact tracing system (e.g., [68, 118]), 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., [24, 108, 122]), 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 anonymously 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".

2.2 Epidemiological Modeling

There are several options to model a disease mathematically. The popular compartment models [20, 21, 55, 61, 64, 66, 114, 133] capture the spread with a few continuous variables linked by simple differential equations. A prominent example is the SEIR model [45, 66, 74, 133] with four compartments to which people are assigned, namely, susceptible (S), exposed (E), infectious (I), and recovered (R). For each simulated time interval, the number of people assigned to each class is computed. While such models are useful for capturing macroscopic trends and also used in state-of-the-art epidemiological research, e.g., [32, 117], the basic approaches condense complex individual behaviour into few variables, thus, limiting the simulation's predictive power [92, 104]. Agent-based epidemiological models [54], on the other hand, initialise a large number of agents with a set of individual properties (e.g., location or age). These agents then interact according to a set of interaction rules (e.g., location-based or age-based) to simulate disease spread. The simulations are carried out in many time steps. Combining both directions, i.e., using agents in a compartment model, allows for a more realistic model of individual behaviour for forecasting disease transmission in a population. Many such simulations with varying parameters are run in parallel to simulate the effect of various policy interventions (e.g., reducing interactions between agents of a certain age, capping the maximum number of allowed contacts, or vaccinating a selected group of agents). The aggregated number of agents assigned to the same "infection class" (e.g., susceptible, exposed, infectious, and recovered for the SEIR model) is then computed for each simulation step.

A crucial question is how to model the agents' individual contact behaviour. Older models relied on survey-based contact matrices, which included information such as the average number of contacts in a given age range [76]. This is already a significant improvement over treating all people the same. However, aggregated network statistics cannot recreate the dynamics of a real complex network graph, as evidenced by the prevalence of super-spreaders with far more contacts than the average [80]. Thus, using the real-world contact graph between all individual members of the population would be ideal from an epidemiological standpoint.

Privacy-Preserving Epidemiological Modeling from Contact Tracing. 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 information that should not be shared. Contact information collected via mobile phones can reveal who, when, and whom people meet, which is by itself sensitive information and must be protected. Moreover, such information also enables to derive indications about the financial situation [16, 90, 116], personality [97], life-partners [6], and ethnicity [6]. 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 to enable precise epidemiological simulations without leaking any individual contact information.

One way to achieve privacy-preserving epidemiological modeling from contact tracing apps is to let each participant (i.e., each device using the contact tracing app) secretly share its contact information between a set of non-colluding servers, which can then jointly run simulations using techniques like secure multi-party computation (MPC), cf. §A. In fact, Araki et al. [9] show how to run graph algorithms on secret shared graphs via MPC efficiently. Even though such a non-collusion assumption is common in the crypto community, the general public in some countries may have difficulty trusting a system in which all contact information is disclosed if the servers collude. In contrast, RIPPLE distributes trust among all participants in such a way that they can keep their own contact information local while simulating the spread of disease by sending messages to each other anonymously. Furthermore, only aggregated simulation results will be shared with a research institute, so no data directly relating to a single identity will be shared. This approach mimics the baseline idea of Federated Learning [93] and prominent contact tracing designs supported by Apple and Google.[2] The increased trust level of a distributed design fosters the crucial broad adoption of such a system in the population.

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

3 THE RIPPLE FRAMEWORK

RIPPLE's primary goal is to facilitate the assessment of various combinations of potential containment measures proposed by epidemiologists and the government. Rather than implementing measures in real-life and analyzing their impact afterwards, our focus is on finding a balance between the benefits and drawbacks of these measures. Examples of such measures include mandating face masks in public places, limiting the size of gatherings, closing specific institutions or stores, and even implementing curfews and regional lockdowns.

Participants in RIPPLE collect personal encounter data anonymously and locally store it on their mobile devices such as cell phones, similar to privacy-preserving contact tracing apps. However, for epidemiological modeling, RIPPLE must also derive a contact graph without leaking sensitive personal information in order to compute simulations of disease spread, which may involve multiple sets of containment measures for some time period, such as two weeks. In almost every country, we can find a 6-hour period during the night when the majority of the population sleeps and mobile devices are idle, connected to the Internet via WiFi, and possibly charging, i.e., an ideal time window for running RIPPLE simulations. The results can then be analysed by medical experts to learn more about the disease or by political decision-makers to determine the most promising containment measures to implement.

To acquire representative and up-to-date physical encounter data, widespread public usage of RIPPLE would be ideal, similar to contact tracing apps. One way to encourage this is to piggyback RIPPLE on the official contact tracing applications used by several countries. Politicians, on the other hand, 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.

3.1 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 C responsible for anonymous communication among the participants.

We assume that the research institute and MPC servers are semi-honest [60], which means they correctly follow protocol specifications while attempting to learn additional information. The semi-honest MPC servers are also used to establish an anonymous communication channel. We discuss the security of the anonymous communication channel in more detail in §B.3. A protocol is considered to be secure if nothing is leaked beyond what can be inferred from the output. Though the semi-honest security model is not the strongest security model, it provides a good trade-off between privacy and efficiency, which is why it is commonly used in the design of several practical privacy-preserving applications such as privacy-preserving machine learning [26, 94, 96, 105], genome/medical research [115, 121, 127], and localization services [71, 124]. It also protects against passive attacks by curious administrators and accidental data leakage. Furthermore, it is quite often the first step toward developing protocols with stronger privacy guarantees [11, 87]. We believe this is a reasonable assumption in our setting because the research institute and the servers will be controlled/run by generally trusted entities such as governments or (public) medical research centres, potentially in collaboration with NGOs such as the EFF[3] or the CCC[4].

[2] https://covid19.apple.com/contacttracing [3] https://www.eff.org [4] https://www.ccc.de/en/

Given the importance of effective containment measures, we expect motivated participants to contribute to epidemiological modeling. However, assuming complete honesty from all millions of potential participants is unrealistic. Therefore, we also consider a client-malicious security model [25, 84] for the participants in $\mathcal{P}$, in which some of the participants are malicious and may deviate from the protocol to gather additional information about their encounters. Malicious behaviour can actively try to hamper or even destroy the correctness of the simulation. However, in the scope of this work, we concentrate on the aforementioned deviations for additional information gain, leaving the problem of developing efficient countermeasures against correctness attacks to future work. Tab. 1 summarises the notations used in this work.

	Parameter	Description
Entities	$\mathcal{P}$	Set of all participants; $\mathcal{P} = \{\mathcal{P}_1, \ldots, \mathcal{P}_\mathsf{p}\}$
	RI	Research Institute
	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; $\mathsf{class}_\mathsf{inf} = \{\mathsf{class}^1_\mathsf{inf}, \ldots, \mathsf{class}^{\mathsf{N}_\mathsf{inf}}_\mathsf{inf}\}$
	I^s_i	$\mathcal{P}_i$'s infection class in simulation step $\mathsf{s} \in [0, N_\mathsf{step}]$
	$\mathcal{E}_i$	Encounter tokens of $\mathcal{P}_i$
	E^max_i	#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 $e \in [0, E^\mathsf{max}]$
	$\delta^{r_e}_i$	$\mathcal{P}_i$'s infection likelihood w.r.t. token r_e
	Δ_i	$\mathcal{P}_i$'s cumulative infection likelihood
	m^e_i	metadata of an encounter e by $\mathcal{P}_i$
	$(\mathrm{pk}_i, \mathrm{sk}_i)$	$\mathcal{P}_i$'s public/private key pair
	$\sigma^e_i.$	$\mathcal{P}_i$'s signature on message about encounter e

Table 1. Notations used in RIPPLE.

3.2 Phases of RIPPLE

RIPPLE consists of four phases shown in Fig. 1: i) Token Generation, ii) Simulation Initialization, iii) Simulation Execution, and iv) Result Aggregation. While RIPPLE can be applied to any compartment-based epidemiological modeling of infectious diseases (see §2.2), we will explain RIPPLE using the SEIR model [45, 74] and the COVID-19 virus as an example. For simplicity, we assume that each participant has installed an app that emulates RIPPLE on their mobile device, and they enter attributes like workplace, school, regular eateries, and cafes locally within the app (resp. the app could make suggestions for those based on the user's frequent locations).

Fig. 2 summarises the phases of the RIPPLE framework in the context of a single simulation setting. 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 with maximally up-to-date encounter information.

① - Token Generation: During a physical encounter, participants exchange data via Bluetooth LE to collect anonymous encounter information (Fig. 3a), similar to contact tracing [65, 108, 122]. These tokens are stored locally on the devices of the users 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. To generate the metadata, a set of standard labels (e.g., restaurants, bars, gyms) can be automatically assigned to a location derived from Google maps. The metadata can be used to 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 §4.

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 $\mathsf{param_{sim}}$ from RI and locally sets $I_i^1 = I_i^{\mathsf{init}}$.

③ - Simulation Execution

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

- Filter out encounters using $\mathsf{param_{sim}}$ 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 the following:

- $\mathcal{P}_i \in \mathcal{P}$ prepares $\{\mathsf{v}_\mathsf{i}^1, \ldots, \mathsf{v}_\mathsf{i}^{\mathsf{N_{inf}}}\}^{\mathsf{s}}$ 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 $\{\mathsf{v}_\mathsf{i}^1, \ldots, \mathsf{v}_\mathsf{i}^{\mathsf{N_{inf}}}\}^{\mathsf{s}}$ to enable RI obtain the tuple $\{\mathsf{C}_{\mathsf{inf}}^1, \ldots, \mathsf{C}_{\mathsf{inf}}^{\mathsf{N_{inf}}}\}^{\mathsf{s}}$, 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).

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 $\mathrm{Bob}_1, \ldots, \mathrm{Bob}_\mathrm{x}$. 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 what effect closing all schools would have on the disease's spread. 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 $\mathsf{param_{sim}}$, to the participants in $\mathcal{P}$. The goal is to "spread" a fictitious infection across N_{sim} different simulation settings. To begin a simulation, each participant $\mathcal{P}_i$ is assigned to an infection class $I_i^{\mathsf{init}} \in \mathsf{class_{inf}}$ (e.g., {S}usceptible, {E}xposed, {I}nfectious, {R}ecovered for the SEIR model) as specified in $\mathsf{param_{sim}}$. For each individual simulation, $\mathsf{param_{sim}}$ 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.[5] 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).

[5] Note, that potential alternatives (e.g., visiting a bar after restaurant closings) are not covered in this model.

Running Example: Assume Alice is designated as infectious, while Bob_1 is designated as susceptible by RI. The other participants $\mathrm{Bob}_2, \ldots, \mathrm{Bob}_\mathrm{x}$ 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[6], 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 the RI initialises the simulation, N_step simulation steps (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 corresponding encounter tokens be represented by the set $\mathcal{E}_i$.
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 respective participant he 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 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 §3.3, $\mathcal{F}_\mathsf{esim}$ 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. Following that, 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[7].

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 $\mathsf{s} \in [N_\mathsf{step}]$. The goal of this phase is to allow RI to obtain the aggregated number of participants per class (e.g., #S, #E, #I, #R) for each simulated time step. For this, we rely on a *Secure Aggregation* functionality, denoted by $\mathcal{F}_\mathsf{agg}$, which takes a N_inf-tuple of the form $\{\mathsf{v}_\mathsf{i}^1, \ldots, \mathsf{v}_\mathsf{i}^{\mathsf{N}_\mathsf{inf}}\}^\mathsf{s}$ 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, $\mathsf{v}_\mathsf{i}^\mathsf{k}$ is an indicator variable for the k-th infection class, which is set to one if $I_i^\mathsf{s} = \mathsf{class}_\mathsf{inf}^\mathsf{k}$ and zero otherwise. Secure

[6] 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. [7] Bob_1 obtains only the aggregated likelihood in the actual protocol.

aggregation [50, 81, 86, 86] is a well-studied building block 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 $\mathcal{F}_{\text{agg}}$ to be a black box that can be instantiated using multiple existing solutions.

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 $\{\mathsf{v}^{\mathsf{S}}, \mathsf{v}^{\mathsf{E}}, \mathsf{v}^{\mathsf{I}}, \mathsf{v}^{\mathsf{R}}\}$ 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, which replicates the procedure but this time using encounters from 13 days ago, i.e., Day 2. The RI obtains the aggregated number of participants for each of the simulated 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).

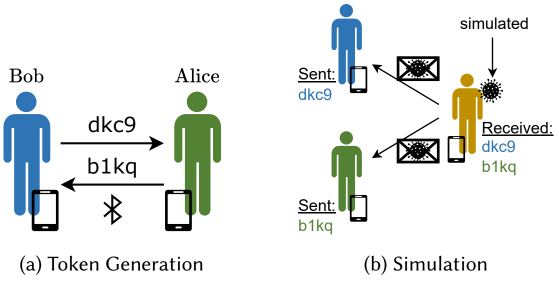


Fig. 3. Token Generation and Simulation phases in RIPPLE.

3.3 Privacy Requirements

Keeping the contact graph private requires that the participants remain unaware of any unconscious interactions. This means they cannot find out 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 of its encounters $e \in E_i$ separately (instead of the aggregation of all), will not meet this requirement as described next.

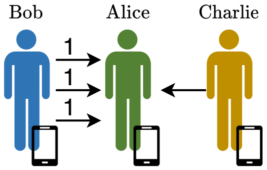


Fig. 4. Linking Identities Attack. Alice and Bob had several encounters, but Alice and Charlie only had one.

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 separately, $\mathcal{P}_i$ 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 is a high infection likelihood and 0 is a low one.

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. Even if they do not see each other, their phones constantly collect encounters. Assume the RI sent the participants a very simple infection likelihood formula that simply returns 0 (not infected) or 1 (infected). Furthermore, since the data is symmetric, both 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.

The more simulations she runs, the more confident she becomes because the infection state for multiple simulations is a unique fingerprint. Since every participant knows the formula, this attack can be extended to complex infection likelihood functions as well. While it may be more computationally expensive than the simple case, Alice is still able to identify correlations. This attack also works even if all of the encounters were unconscious. In such situations, Alice may not be able 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 in a simulation phase, each participant receives just an aggregation of all infection likelihoods of their encounters. It cannot be avoided that participants can understand that when "getting infected" someone 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.

Sybil Attack. While the Linking Identities Attack is already possible for semi-honest adversaries, malicious participants may go even further to circumvent the aggregation mechanism that prevents 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.

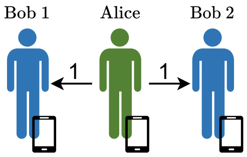


Fig. 5. Sybil Attack.

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 are allowed to join and participate in the simulation. In a closed ecosystem, such as a cpmpany, 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.

Inference Attacks. Note that although RIPPLE mimics the spirit of Federated Learning (FL) [93], it is not susceptible to so-called inference attacks [52, 99] in the same sense as FL. First of all, RIPPLE only reveals the final output (to a research institute RI) and no individual updates/results that ease information extraction. However, the analysis results provided to RI (cf. §3.1) 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, anything that might be inferred from the output is not considered in our security model. We 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.

4 INSTANTIATING $\mathcal{F}_{\text{esim}}$

In this section, we propose two instantiations of $\mathcal{F}_{\text{esim}}$ that cover different use cases and offer different trust-efficiency trade-offs. Our first design, RIPPLE$_{\text{TEE}}$ (§4.1), assumes the presence of trusted execution environments (TEEs) such as ARM TrustZone on the mobile devices of the participants. In our second design, RIPPLE$_{\text{PIR}}$ (§4.2), we eliminate this assumption and provide a software only solution using cryptographic techniques such as private information retrieval.

4.1 RIPPLE$_{\text{TEE}}$

The deployment of the entire operation in a single designated TEE would be a simple solution to achieving the ideal functionality $\mathcal{F}_{\text{esim}}$. 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$_{\text{TEE}}$ (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$_{\text{TEE}}$, we will go over the $\mathcal{F}_{\text{anon}}$ functionality (cf. §B.3) that we will use in our instantiation. We define it as follows: $\mathcal{F}_{\text{anon}}$ allows two participants, $\mathcal{P}_i$ and $\mathcal{P}_j$, to send messages to each other anonymously via a set of communication servers C. The set C consists of one server acting as an entry node ($\mathcal{N}_{\texttt{entry}}$), receiving messages from senders, and one server acting as an exit node ($\mathcal{N}_{\texttt{exit}}$), forwarding messages to receivers. In $\mathcal{F}_{\text{anon}}$, 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 C will be unable to link receiver and sender of a message. Anonymous communication (cf. §A) is an active research area, e.g., [1, 5, 51, 63], and $\mathcal{F}_{\text{anon}}$ in RIPPLE$_{\text{TEE}}$ can be instantiated using any of these efficient techniques.

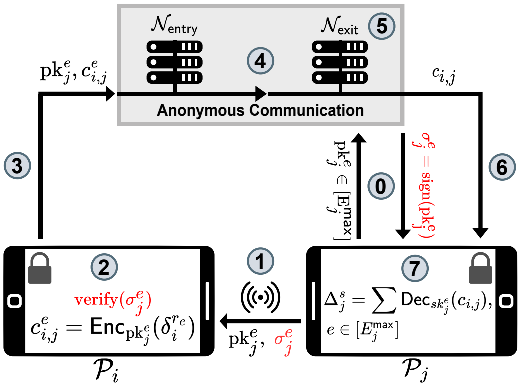

$\mathcal{N}_{\text{entry}}$
$\mathcal{N}_{\text{exit}}$
5
4
$\text{pk}_j^e, c_{i,j}^e$
$c_{i,j}$
Anonymous Communication
3
6
0
$\text{pk}_j^e \in [E_j^{\max}]$
$\sigma_j^e = \text{sign}(\text{pk}_j^e)$
2
$\text{verify}(\sigma_j^e)$
$c_{i,j}^e = \text{Enc}_{\text{pk}_j^e}(\delta_i^{r_e})$
1
$\text{pk}_j^e,\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

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

4.1.1 The RIPPLE$_{\text{TEE}}$ Protocol.

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 keys $(\text{pk}_i^e, \text{sk}_i^e)$ for all possible encounters $e \in [E_i^{\max}]$. The keys can be pre-generated and stored, e.g., on the day before. The newly generated public keys are then sent by $\mathcal{P}_i$'s TEE to the exit node $\mathcal{N}_{\texttt{exit}}$ (step ⓪ in Fig. 6) to enable anonymous communication (cf. §B.3) via $\mathcal{F}_{\text{anon}}$ 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 $\mathcal{N}_{\texttt{exit}}$ 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). 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).

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 $c_{i,j}^e = \text{Enc}_{\text{pk}_j^e}(\delta_i^{r_e})$ (step ② in Fig. 6).

- $\mathcal{P}_i$'s TEE establishes a secure channel with the entry node $\mathcal{N}_{\texttt{entry}}$ of C via remote attestation and uploads the tuple $(\mathrm{pk}^e_j, c^e_{i,j})$ (step ③ in Fig. 6).
- The tuple $(\mathrm{pk}^e_j, c^e_{i,j})$ traverses through the servers in 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.3).
- If the public key pk^e_i has already been used in this simulation step[8], $\mathcal{N}_{\texttt{exit}}$ discards the tuple (step ⑤ in Fig. 6).
- Otherwise, $\mathcal{N}_{\texttt{exit}}$ uses pk^e_j to identify the recipient $\mathcal{P}_j$ and sends the ciphertext $c^e_{i,j}$ 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).

4.1.2 Security of RIPPLE$_{\text{TEE}}$. 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 $\mathcal{F}_{\text{gen}}$ functionality, which captures the goal of several contact tracing apps in use.

The $\mathcal{F}_{\text{anon}}$ functionality, which implements an anonymous communication channel utilising the servers in 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 messages to it, 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$_{\text{TEE}}$ 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 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.

4.2 RIPPLE$_{\text{PIR}}$

In the following, we show how to get rid of RIPPLE$_{\text{TEE}}$'s assumption of each participant having a TEE on their mobile devices. If we simply remove the TEE part of RIPPLE$_{\text{TEE}}$ 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. §3.3). To get around this privacy issue, we need to find another way to aggregate the infection likelihoods so that only the sum, not individual values, can be derived by the participants.

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

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

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 §5.

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

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

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

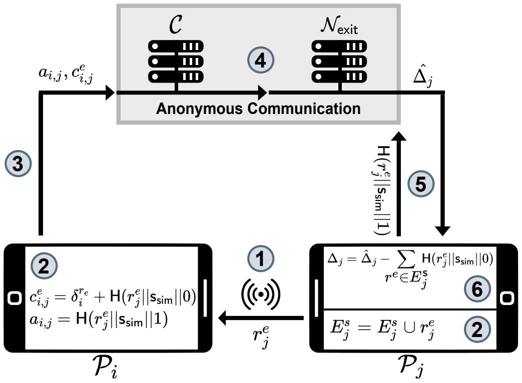
Output: $\mathcal{F}_{\mathsf{pirsum}}$ sends $\sum_{j=1}^{\tau} \mathsf{D}[q_j]$ to $\mathcal{P}_i$ as the output.

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

4.2.1 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 κ-bit random tokens denoted by r_i^e and r_j^e. Both participants, like in RIPPLE$_{\mathsf{TEE}}$, record the metadata m^e as well. 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 $E_i^{\mathsf{s}} = \{r_i^e\}_{e \in \mathcal{E}_i}$, where $\mathcal{E}_i$ is the complete (sent/received) set of encounters of $\mathcal{P}_i$, and a list of received tokens, denoted by $R_i^{\mathsf{s}} = \{r_j^e\}_{e \in \mathcal{E}_i}$. Looking ahead, these random tokens will be used as addresses for communicating the corresponding infection likelihood among the participants.

Simulation Execution (steps ② to ⑥ in Fig. 8): For local computations like encounter filtering and infection likelihood calculation, the steps for an encounter e between $\mathcal{P}_i$ and $\mathcal{P}_j$ are:



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

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

- $\mathcal{P}_i$ blinds each $\delta_i^{r_e}$ computed with the corresponding random token r_j^e received from $\mathcal{P}_j$ and obtains the ciphertext $c_{i,j}^e = \delta_i^{r_e} + \mathsf{H}(r_j^e||\mathsf{s}_{\mathsf{sim}}||0) \bmod 2^l$. Further, it computes the destination address for the ciphertext as $a_{i,j} = \mathsf{H}(r_j^e||\mathsf{s}_{\mathsf{sim}}||1)$. Here, 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) To prevent the exit node $\mathcal{N}_{\mathtt{exit}}$ from linking messages from different simulations, $\mathsf{s}_{\mathsf{sim}}$ is utilized in H() to generate unique (ciphertext, address) tuples for the same encounters across multiple simulation settings.
- $\mathcal{P}_i$ sends the tuple $(c_{i,j}^e, a_{i,j})$ anonymously to $\mathcal{N}_{\mathtt{exit}}$ with the help of the servers in C. $\mathcal{N}_{\mathtt{exit}}$ discards all the tuples with the same address field $(a_{i,j})$. (step ③ to ④ in Fig. 8, instantiation details for the anonymous communication channel are given in §B.3)

As a server in C, $\mathcal{N}_{\mathtt{exit}}$ locally creates the database D for the current simulation step using all of the $(a_{i,j}, c_{i,j}^e)$ tuples received (part of step ④ in Fig. 8). A naive solution of inserting $c_{i,j}^e$ 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 message receiver to precisely download the messages that were sent to them. Using simple hashing, this 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, cf. §4.2.3), 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 C based on the instantiation of $\mathcal{F}_{\mathsf{pirsum}}$ (cf. §5). Each $\mathcal{P}_j \in \mathcal{P}$ will then participate in an instance of $\mathcal{F}_{\mathsf{pirsum}}$ with the PIR servers C sharing a 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 the blinded 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 $\Delta_j = \hat{\Delta}_j - \sum_{r^e \in E^{\mathsf{s}}_j} \mathsf{H}(r^e||\mathsf{s}_{\mathsf{sim}}||0) \bmod 2^l$ concluding the current simulation step (step ⑥ in Fig. 8).

4.2.2 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$_{\mathsf{PIR}}$ cannot identify the message's destination from the address as it is only known by the receiving participant. Further, participants obtain their cumulative infection likelihood directly via the $\mathcal{F}_{\mathsf{pirsum}}$ functionality, ensuring that $\mathcal{N}_{\texttt{exit}}$ cannot infer the participant's encounter data 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 §3, we assume that malicious participants in our framework will not tamper with the correctness and will only try to learn additional information. A malicious participant could re-use the same encounter token for multiple encounters during token generation which would result in multiple tuples with the same address. However, as stated in the protocol, $\mathcal{N}_{\texttt{exit}}$ will discard all such tuples, effectively removing the malicious participant from the system. Another potential information leakage caused by a participant re-using encounter tokens is that the entry point of the anonymous communication channel will be able to deduce that multiple participants, say $\mathcal{P}_i$ and $\mathcal{P}_j$, had an encounter with 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.3), where all the servers except $\mathcal{N}_{\texttt{exit}}$ will not see any messages in the clear, but only see secret shares.

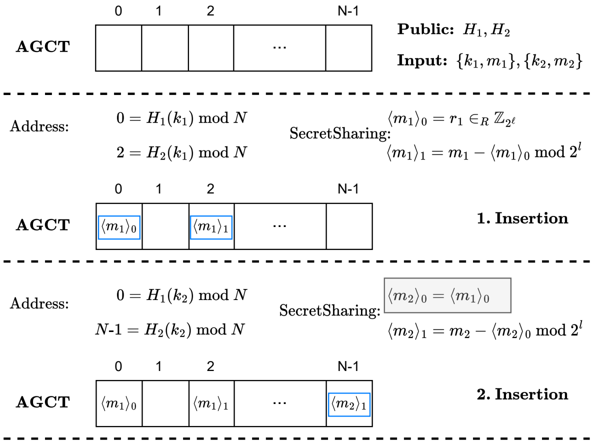

0
1
2
N-1
AGCT
···
Public: H_1, H_2
Input: $\{k_1, m_1\}, \{k_2, m_2\}$
Address:
$0 = H_1(k_1) \bmod N$
$2 = H_2(k_1) \bmod N$
SecretSharing:
$\langle m_1 \rangle_0 = r_1 \in_R \mathbb{Z}_{2^\ell}$
$\langle m_1 \rangle_1 = m_1 - \langle m_1 \rangle_0 \bmod 2^l$
$\langle m_1 \rangle_0$
$\langle m_1 \rangle_1$
1. Insertion
Address:
$0 = H_1(k_2) \bmod N$
$N\text{-}1 = H_2(k_2) \bmod N$
SecretSharing:
$\langle m_2 \rangle_0 = \langle m_1 \rangle_0$
$\langle m_2 \rangle_1 = m_2 - \langle m_2 \rangle_0 \bmod 2^l$
$\langle m_2 \rangle_1$
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 address of the data in the database.

4.2.3 Arithmetic Garbled Cuckoo Table (AGCT). We design a variant of garbled cuckoo tables ([109], cf. §A) that we term arithmetic garbled cuckoo table (AGCT) to reduce the size of the PIR database while ensuring a negligible collision probability. It uses arithmetic sharing instead of XOR-sharing to share database entries and the details are presented next.

Let's assume two key-message pairs $\{k_1, m_1\}$ and $\{k_2, m_2\}$[9] shall be added to database D with N bins and two hash function H_1 and H_2 to determine the insertion addresses. The insertion process works as follows:

1. Insertion of $\{k_1, m_1\}$:
   a) Compute $a_1 = \mathsf{H}_1(k_1) \bmod N$ and $a_2 = \mathsf{H}_2(k_1) \bmod N$.
   b) Check if bins a_1, a_2 are already occupied. Let's assume this is not the case.

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

c) Compute the arithmetic sharing of the message m_1: $\langle m_1 \rangle_0 = r_1 \in_R \mathbb{Z}_{2^\ell}$ and $\langle m_1 \rangle_1 = m_1 - \langle m_1 \rangle_0 \bmod 2^\ell$.

d) Insert $D[a_1] = \langle m_1 \rangle_0$, $D[a_2] = \langle m_1 \rangle_1$.

2. Insertion of $\{k_2, m_2\}$:

a) Compute $b_1 = \mathsf{H}_1(k_2) \bmod N$ and $b_2 = \mathsf{H}_2(k_2) \bmod N$.

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 $\langle m_2 \rangle_1 = m_2 - \langle m_2 \rangle_0 \bmod 2^\ell$.

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. [109] 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 [109, §5].

5 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 concentrate on instantiating our novel $\mathcal{F}_{\mathsf{pirsum}}$ functionality (Fig. 7) using three semi-honest MPC servers. 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 $Q = \{q_1, \ldots, q_\tau\}$ and wants to retrieve $\mathsf{res} = \sum_{q \in Q} \mathsf{D}[q] \bmod 2^\ell$. 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.1) 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 [9, 26, 105].

5.1 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 [18, 33], denoted by $\mathcal{F}_{\mathsf{pir}}^{\mathsf{2S}}$, and combine the responses to get the sum of the desired blocks as the output. $\mathsf{D}^m = \mathsf{D} + m \bmod 2^\ell$ 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 $\sum_{j=1}^{\tau} m_j = 0$.[10]
- $\mathsf{S}_1, \mathsf{S}_2$, 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 $j \in [\tau]$. The result obtained by $\mathcal{P}_i$ from the j-th $\mathcal{F}_{\mathsf{pir}}^{\mathsf{2S}}$ instance is denoted by res_j.
- $\mathcal{P}_i$ 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 [33] are provided in §C. The approach requires $\mathcal{P}_i$ to communicate $N \cdot \tau$ bits to the servers, which is further reduced in $\mathrm{RIPPLE}_{\mathsf{PIR}}$ as shown in §5.3.

[10] These masks are sampled for each participant.

Malicious participants. Recall from our threat model (cf. §3.1) that a malicious participant may deviate from the protocol to gain additional knowledge but does not try to harm correctness. For example, it 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 use a simple verification scheme over the $\mathcal{F}_{\mathsf{pir}}^{\mathsf{2S}}$ functionality to prevent these manipulations. Its details are presented next.

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) 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 further optimize this step. 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 for $j \in [\tau]$ and $u \in \{1, 2\}$.
- 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 [17, 18, 33].

5.2 Instantiating $\mathcal{F}_{\mathsf{pirsum}}$

The formal protocol for $\mathsf{PIR}_{\mathsf{sum}}$ in the case of malicious participants is provided in Fig. 10 and is based on a variant of the standard 2-server PIR functionality $\mathcal{F}_{\mathsf{pir}}^{\mathsf{2S}}$ (as will be discussed in $\mathbf{HYB}_2$ below). In $\mathsf{PIR}_{\mathsf{sum}}$, 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 $q \in Q$. Following the execution, $\mathcal{P}_i$ receives $\mathsf{D}[q] + \mathsf{r}_q$ whereas S_u receives r_q, $[q]_u$, for $u \in \{1, 2\}$ and $q \in Q$. $\mathcal{P}_i$ then adds up the received messages to get a masked version of the desired output, i.e, $\sum_{q \in Q} \mathsf{D}[q] + \mathsf{mask}_Q$ with $\mathsf{mask}_Q = \sum_{q \in Q} \mathsf{r}_q$. $\mathsf{S}_1, \mathsf{S}_2$ compute mask_Q in the same way.

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

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

Computation

1. For each $q \in Q$,
   a. $\mathsf{S}_1, \mathsf{S}_2$ and $\mathcal{P}_i$ invoke $\mathcal{F}_{\mathsf{pir}}^{\mathsf{2S}}$ (cf. $\mathbf{HYB}_2$ in proof of Lemma 5.1) with the inputs D, q.
   b. Let 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 $\mathsf{res}' = \sum_{q \in Q}(\mathsf{D}[q] + \mathsf{r}_q)$, while $\mathsf{S}_1, \mathsf{S}_2$ computes $\mathsf{mask}_Q = \sum_{q \in Q} \mathsf{r}_q$.
3. $\mathsf{S}_1, \mathsf{S}_2$ and S_0 invokes $\mathcal{F}_{\mathsf{vrfy}}$ on the secret shares of queries, denoted by $\{[q]_u\}_{q \in Q, u \in \{1,2\}}$, to check the distinctness of the queries in Q.
4. If $\mathcal{F}_{\mathsf{vrfy}}$ returns `Accept`, $\mathsf{S}_1, \mathsf{S}_2$ sends mask_Q to $\mathcal{P}_i$, who computes $\mathsf{res} = \mathsf{res}' - \mathsf{mask}_Q$. Otherwise, `abort`.

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

The protocol could be completed by S_1 and S_2 sending mask_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 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 $q \in Q$ 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`.

5.2.1 Security of $\mathsf{PIR}_{\mathsf{sum}}$ *Protocol.* Fig. 11 presents the ideal functionality for $\mathsf{PIR}_{\mathsf{sum}}$ 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 C, and participant $\mathcal{P}_i \in \mathcal{P}$.

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

Computation: $\mathcal{F}_{\mathsf{pirsum}}$ sets $y = \sum_{j=1}^{\tau} \mathsf{D}[q_j]$ if all the queries in 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 5.1. *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* $\{\mathcal{F}^{\mathsf{2S}}_{\mathsf{pir}}, \mathcal{F}_{\mathsf{vrfy}}\}$*-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 $\mathcal{F}^{\mathsf{2S}}_{\mathsf{pir}}$, to instantiate $\mathcal{F}_{\mathsf{pirsum}}$. The guarantees of $\mathcal{F}^{\mathsf{2S}}_{\mathsf{pir}}$, however, are insufficient to meet the security requirements of $\mathcal{F}_{\mathsf{pirsum}}$, so we modify $\mathcal{F}^{\mathsf{2S}}_{\mathsf{pir}}$ 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. $\mathcal{F}^{\mathsf{2S}}_{\mathsf{pir}}$ is equal to the first hybrid $\mathbf{HYB}_0$. We use the hybrid $\mathbf{HYB}_2$ instead of $\mathcal{F}^{\mathsf{2S}}_{\mathsf{pir}}$, and we omit introducing a different notation for the same for simplicity.

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

$\mathbf{HYB}_1$: We modify $\mathcal{F}^{\mathsf{2S}}_{\mathsf{pir}}$ 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, such that addition of r to the database blocks respects the underlying distribution. In other words, the modification can be thought of as the standard $\mathcal{F}^{\mathsf{2S}}_{\mathsf{pir}}$ 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 $\mathsf{PIR}_{\mathsf{sum}}$, the servers $\mathsf{S}_1, \mathsf{S}_2$ and the participant $\mathcal{P}_i$ run τ instances of $\mathcal{F}^{\mathsf{2S}}_{\mathsf{pir}}$ in parallel, one for each query $q \in Q$. As shown in $\mathcal{F}_{\mathsf{pirsum}}$ (Fig. 11), the servers must ensure that all of the queries in Q are distinct. For this, we modify $\mathcal{F}^{\mathsf{2S}}_{\mathsf{pir}}$ 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. □

5.3 Reducing participant's communication

$\mathsf{PIR}_{\mathsf{sum}}$ in RIPPLE$_{\mathsf{PIR}}$ can be implemented using two approaches with different trade-offs to minimize participants' communication and computation. $\mathsf{PIR}^{\mathsf{I}}_{\mathsf{sum}}$ (Fig. 12) prioritizes low communication over computation, while $\mathsf{PIR}^{\mathsf{II}}_{\mathsf{sum}}$ (Fig. 13) reduces both the computational and communication overhead of the participant by involving an additional server $\mathsf{S}_0 \in C$.

5.3.1 $\mathsf{PIR}^{\mathsf{I}}_{\mathsf{sum}}$ *(Fig. 12).* In this approach, we instantiate $\mathcal{F}^{\mathsf{2S}}_{\mathsf{pir}}$ using PIR techniques based on Function Secret Sharing (FSS) [17, 18, 36]. To retrieve the q-th block from the database, $\mathcal{P}_i$ uses FSS on a Distributed Point Function (DPF) [58] 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 §5.1 (more details on Linear Summation PIR are given in §C). The key size for a database with N blocks using the optimised DPF construction in [18] is about $\lambda \log_2(N/\lambda)$ bits, where $\lambda = 128$ for an AES-based implementation. Fig. 12 provides the formal details of the $\mathsf{PIR}^{\mathsf{I}}_{\mathsf{sum}}$ protocol.

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

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

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

1. $\mathsf{S}_1, \mathsf{S}_2$ locally compute $\mathsf{D}^{m_q} = \mathsf{D} + m_q$.
2. Execute DPF protocol [18] (verifiable DPF for malicious participants) with $\mathcal{P}_i$ as client with input q. Server S_u obtains $[\vec{\mathsf{b}}_q]_u$ with $\mathsf{b}^j_q = 1$ for $j = q$ and $\mathsf{b}^j_q = 0$ 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 [18] (cf. §B.4). 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 $[\vec{\mathsf{b}}_c]_u = \oplus_{q \in Q} [\vec{\mathsf{b}}_q]_u$, 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 $\pi(\vec{\mathsf{b}}_c) = \pi([\vec{\mathsf{b}}_q]_1) \oplus \pi([\vec{\mathsf{b}}_q]_2)$, sends `Accept` to S_1 and S_2, if $\pi(\vec{\mathsf{b}}_c)$ has exactly τ ones, `abort` otherwise.

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

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

Security. For semi-honest participants, the security of protocol $\mathsf{PIR}^{\mathsf{I}}_{\mathsf{sum}}$ directly reduces to that of the 2-server PIR protocol in [18]. However, as mentioned in [18], a malicious participant could generate incorrect DPF keys, compromising the scheme's security and correctness. To prevent this type of misbehaviour, Boyle et al. [18] 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.

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

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

Computation S_1 and S_2 sample τ random mask values $\{m_1, \ldots, m_\tau\} \in \mathbb{Z}^{\tau}_{2^\ell}$ such that $\sum_{j=1}^{\tau} m_j = 0$. For each $q \in Q$, 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 [18] with S_0 as client with input q'. Server S_u obtains $[\vec{\mathsf{b}}_{q'}]_u$ with $\mathsf{b}^j_{q'} = 1$ for $j = q'$ and $\mathsf{b}^j_{q'} = 0$ 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 $[\vec{\mathsf{b}}_c]_k = \oplus_{q \in Q} [\vec{\mathsf{b}}_q]_k$, 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 $[y_q]_u = \bigoplus_{j=1}^{N} [\mathsf{b}^j_q]_u \mathsf{D}^{m_q}[j]$ to $\mathcal{P}_i$, for $q \in Q, u \in \{1, 2\}$.
11. $\mathcal{P}_i$ locally computes $\mathsf{res} = \sum_{q \in Q}([y_q]_1 \oplus [y_q]_2)$.

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

While using verifiable DPFs in $\mathsf{PIR}^{\mathsf{I}}_{\mathsf{sum}}$ ensures that the τ bit vectors generated by $\mathcal{P}_i$ are valid, it does not ensure that the bit vectors $\vec{\mathsf{b}}_1, \ldots, \vec{\mathsf{b}}_\tau$ correspond to τ distinct locations in the database D. However, we leverage the correctness guarantee of verifiable DPFs to reduce the communication cost for verification, as discussed in §5.1, §B.4, and §C. In detail, all τ bit vectors $\vec{\mathsf{b}}_1, \ldots, \vec{\mathsf{b}}_\tau$, i.e., the PIR queries, that are available in a secret-shared form among S_1 and S_2 are now guaranteed to have exactly one 1 in them, with the remaining bit positions being 0. To ensure distinctness, S_1 and S_2 XOR all 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 problem now boils down to determining whether or not $\vec{\mathsf{b}}_c$ has exactly τ bit positions set to 1. This can be accomplished by servers S_1 and S_2 agreeing 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. §5.1).

Computation Complexity (#AES operations). In $\mathsf{PIR}^{\mathsf{I}}_{\mathsf{sum}}$, the participant $\mathcal{P}_i$ must perform $4 \cdot \log_2(N/\lambda)$ AES operations as part of the key generation algorithm for each of the τ instances of $\mathcal{F}^{\mathsf{2S}}_{\mathsf{pir}}$ over a database of size N, where $\lambda = 128$ for an AES-based implementation. Similarly, S_1 and S_2 must perform $\log_2(N/\lambda)$ AES operations for each of the N DPF evaluations. We refer to Table 1 in [18] for more specifics.

5.3.2 $\mathsf{PIR}^{\mathsf{II}}_{\mathsf{sum}}$ *(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 remainder of the computation until output retrieval will now take place solely among the servers.

The servers run a DPF instance [18] 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 in order 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 positions. 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 correspond 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 a random shift correction values θ_q using the shared-key setup (cf. §B.1), and only the corresponding q' values are sent 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 both $\mathsf{PIR}_{\mathsf{sum}}$ protocols to protect against malicious participants. More specifically, in Step 8 of Fig. 13 (resp., Step 8 of Fig. 12), $2N$ bits are sent to S_0 to ensure the distinctness of the queries made by the participant $\mathcal{P}_i$. We note that allowing 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), $\mathcal{P}_i, \mathsf{S}_1$, and S_2 sample only one random shift value θ and use 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.

5.3.3 *Summary of communication costs.*

Tab. 2 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.

Stage	$\mathsf{PIR}^{\mathsf{I}}_{\mathsf{sum}}$	$\mathsf{PIR}^{\mathsf{II}}_{\mathsf{sum}}$
$\mathcal{P}_i$ to servers in C	$2\tau(\lambda+2)\log_2(N/\lambda)+4\tau\lambda$	$\tau \log_2 N$
Server to server	0	$2\tau(\lambda+2)\log_2(N/\lambda)+4\tau\lambda$
Servers in C to $\mathcal{P}_i$	$\tau \cdot 2\ell$	$\tau \cdot 2\ell$
+ Verification (mal.)	$2N+2+\delta$	$2N+2$

Table 2. Summary of communication costs in bits between participants $\mathcal{P}_i$ and a server $\mathsf{S}_j \in C$ for $\mathsf{PIR}_{\mathsf{sum}}$. λ denotes the AES key size ($\lambda = 128$ in [17]), ℓ denotes the block size in bits ($\ell = 128$ in this work), and δ denotes the constant involved in the verifiable DPF approach enabling malicious security [18] (cf. §C).

6 EVALUATION

In this section, we evaluate and compare the computation and communication efficiency of our two RIPPLE protocols presented in §4. A fully-fledged implementation, similar to existing contact tracing apps, would necessitate collaboration with industry partners to develop a real-world scalable system

Entities	Protocol	Population (p)									
		1K	10K	50K	100K	500K	1M	2M	5M	10M	20M
Participants in $\mathcal{P}$ (in KB)	RIPPLE$_{\mathsf{TEE}}$ (§4.1)	16.00	16.00	16.00	16.00	16.00	16.00	16.00	16.00	16.00	16.00
	RIPPLE$_{\mathsf{PIR}}$: $\mathsf{PIR}^{\mathsf{I}}_{\mathsf{sum}}$ (§5.3.1)	51.63	62.42	69.97	73.22	80.77	84.02	87.27	91.56	94.81	98.06
	RIPPLE$_{\mathsf{PIR}}$: $\mathsf{PIR}^{\mathsf{II}}_{\mathsf{sum}}$ (§5.3.2)	3.45	3.49	3.52	3.53	3.56	3.57	3.59	3.60	3.62	3.63
Servers in C (in GB)	RIPPLE$_{\mathsf{TEE}}$ (§4.1)	0.02	0.19	0.96	1.92	9.60	19.20	38.40	96.00	192.00	384.00
	RIPPLE$_{\mathsf{PIR}}$ (§5)	0.01	0.10	0.48	0.96	4.80	9.60	19.20	48.00	96.00	192.00

Table 3. Communication costs per simulation step in our RIPPLE instantiations.

for national deployment. Instead, we provide a proof-of-concept implementation and micro benchmark results for all major building blocks.[11] Moreover, we do not measure the speed of the communication link between the participants and the servers. 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 [129, 131], sleeping habits in various countries provide a time window of several hours each night that can be used for this purpose.

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 §5.3) 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 [7], we use hardware-backed crypto operations already implemented by Android instead. We use the code of [73] to instantiate FSS-PIR. We implement the AGCT in C++ and follow previous work on cuckoo hashing [112] 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 RIPPLE$_{\mathsf{TEE}}$ since Android offers a hardware-backed implementation. We omit the overhead of remote attestation for the sake of simplicity. For RIPPLE$_{\mathsf{PIR}}$, we use the FSS-PIR scheme of [18, 73] as the baseline and the addresses are hashed with SHA-256 and trimmed to $40 - 1 + \log_2(\mathrm{p} \cdot E^{\mathrm{avg}})$ bits, where p is the number of participants and E^{avg} represents the average number of encounters per participant per simulation step. We set $E^{\mathrm{avg}} = 100$ while benchmarking based on numbers provided by research on epidemiological modeling [43, 98]. To avoid cycles when inserting n messages into the AGCT (cf. §4.2.3), we set its size to $10n$. This can be further improved as discussed in §4.2.3 [109, 111, 112]. A typical simulation step corresponds to one day, such that 14 simulation steps can simulate two weeks.

6.1 Communication Complexity

In this section, 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. 3 summarises the communication costs of each participant as well as the communication servers (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.

[11] 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 a large number of servers. Instead, our goal here is to demonstrate the viability of RIPPLE protocols for large-scale deployment.

6.1.1 Participant Communication. As shown in Tab. 3, a participant in RIPPLE$_{\mathsf{TEE}}$ requires just 16KB of total communication in every simulation step, and this is independent of the population size. 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 ($E^{\text{avg}} = 100$), we depict the participants' communication in Fig. 14 with varied average number of encounters E^{avg} ranging from 10 to 500 for a population of 10M. Note that a 2-week simulation with $E^{\text{avg}} = 500$ can be completed by a participant in RIPPLE$_{\mathsf{TEE}}$ with roughly 1MB communication.

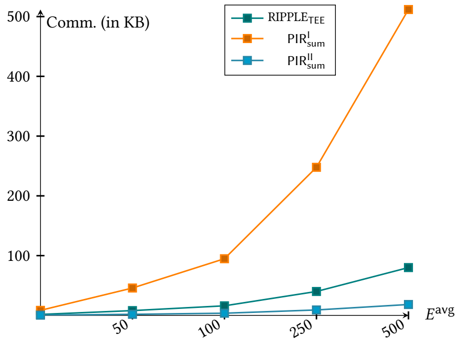


Fig. 14. Participant's communication with varying E^{avg} for a population of p =10M.

Unlike RIPPLE$_{\mathsf{TEE}}$, participant communication in both $\mathsf{PIR}^{\mathsf{I}}_{\mathsf{sum}}$ and $\mathsf{PIR}^{\mathsf{II}}_{\mathsf{sum}}$ increases for larger populations as the corresponding database size increases. The communication, however, is only sub-linear in the database size[12].

In particular, the participant's communication in $\mathsf{PIR}^{\mathsf{I}}_{\mathsf{sum}}$ 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 [18], as discussed in §5. The communication in $\mathsf{PIR}^{\mathsf{II}}_{\mathsf{sum}}$, on the other hand, is about 3.5KB for all participant sizes we consider. This reduced communication is due to the optimization in $\mathsf{PIR}^{\mathsf{II}}_{\mathsf{sum}}$, which offloads the DPF key generation task to the helper server S_0 (cf. §5.3.2). A participant in $\mathsf{PIR}^{\mathsf{I}}_{\mathsf{sum}}$ send approximately 7MB of data for a 2-week simulation for a 10M population with $E^{\text{avg}} = 500$, whereas it is only 0.25MB in the case of $\mathsf{PIR}^{\mathsf{II}}_{\mathsf{sum}}$.

Tab. 4 provides the communication cost for a participant for multiple population sizes in RIPPLE$_{\mathsf{TEE}}$, $\mathsf{PIR}^{\mathsf{I}}_{\mathsf{sum}}$, and $\mathsf{PIR}^{\mathsf{II}}_{\mathsf{sum}}$, 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$_{\mathsf{PIR}}$ with the exception that the cost increases sub-linearly with the population size due to the use of FSS-based PIR scheme in RIPPLE$_{\mathsf{PIR}}$.

Population p	Protocol	E^{avg}				
		10	50	100	250	500
100K	RIPPLE$_{\mathsf{TEE}}$ (§4.1)	1.60	8.00	16.00	40.00	80.00
	RIPPLE$_{\mathsf{PIR}}$: $\mathsf{PIR}^{\mathsf{I}}_{\mathsf{sum}}$ (§5.3.1)	6.24	34.99	73.22	193.79	403.83
	RIPPLE$_{\mathsf{PIR}}$: $\mathsf{PIR}^{\mathsf{II}}_{\mathsf{sum}}$ (§5.3.2)	0.35	1.76	3.53	8.87	17.81
1M	RIPPLE$_{\mathsf{TEE}}$ (§4.1)	1.60	8.00	16.00	40.00	80.00
	RIPPLE$_{\mathsf{PIR}}$: $\mathsf{PIR}^{\mathsf{I}}_{\mathsf{sum}}$ (§5.3.1)	7.32	40.38	84.02	220.78	457.81
	RIPPLE$_{\mathsf{PIR}}$: $\mathsf{PIR}^{\mathsf{II}}_{\mathsf{sum}}$ (§5.3.2)	0.35	1.78	3.57	8.98	18.01
10M	RIPPLE$_{\mathsf{TEE}}$ (§4.1)	1.60	8.00	16.00	40.00	80.00
	RIPPLE$_{\mathsf{PIR}}$: $\mathsf{PIR}^{\mathsf{I}}_{\mathsf{sum}}$ (§5.3.1)	8.40	45.78	94.81	247.77	511.79
	RIPPLE$_{\mathsf{PIR}}$: $\mathsf{PIR}^{\mathsf{II}}_{\mathsf{sum}}$ (§5.3.2)	0.36	1.80	3.62	9.08	18.22

Table 4. Communication (in KB)/participant/simulation step for varying average numbers of encounters E^{avg} and population sizes p.

6.1.2 Server Communication. The servers' communication is primarily attributed to the anonymous communication channel that they have established, which provides unlinkability and, thus, privacy to the messages of the participants. As discussed in §B.3, in order to communicate M messages through the channel, the servers must 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. 3. This is due to the larger message size in RIPPLE$_{\mathsf{TEE}}$ as a result of the use of public-key encryption.

[12] DB size of $10n$, where n is the number of messages, 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. §4.2.3 and [109]).

Stages of RIPPLE	Protocol	Population (p)									
		1K	10K	50K	100K	500K	1M	2M	5M	10M	20M
Message Generation by $\mathcal{P}_i \in \mathcal{P}$ (in KB)	RIPPLE$_{\mathsf{TEE}}$ (§4.1)[a]	12.80	12.80	12.80	12.80	12.80	12.80	12.80	12.80	12.80	12.80
	RIPPLE$_{\mathsf{PIR}}$: ● (§4.2)	3.20	3.20	3.20	3.20	3.20	3.20	3.20	3.20	3.20	3.20
	RIPPLE$_{\mathsf{PIR}}$: ◐ (§4.2)	2.30	2.34	2.38	2.39	2.41	2.43	2.44	2.45	2.46	2.48
Secure Shuffle by C (in GB)	RIPPLE$_{\mathsf{TEE}}$ (§4.1)	0.02	0.19	0.96	1.92	9.60	19.20	38.40	96.00	192.00	384.00
	RIPPLE$_{\mathsf{PIR}}$ - ● (§4.2)	0.01	0.10	0.48	0.96	4.80	9.60	19.20	48.00	96.00	192.00
	RIPPLE$_{\mathsf{PIR}}$ - ◐ (§4.2)	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)[b]	RIPPLE$_{\mathsf{TEE}}$ (§4.1)	6.40	6.40	6.40	6.40	6.40	6.40	6.40	6.40	6.40	6.40
	$\mathsf{PIR}^{\mathsf{I}}_{\mathsf{sum}}$ - ● (§5.3.1)	51.36	62.42	69.97	73.22	80.77	84.02	87.27	91.56	94.81	98.06
	$\mathsf{PIR}^{\mathsf{I}}_{\mathsf{sum}}$ - ◐ (§5.3.1)	26.48	32.64	37.69	39.82	44.77	47.05	49.38	52.33	54.77	57.26
	$\mathsf{PIR}^{\mathsf{II}}_{\mathsf{sum}}$ (§5.3.2)	3.45	3.49	3.52	3.53	3.56	3.57	3.59	3.60	3.62	3.63

● - 128-bit address for RIPPLE$_{\mathsf{PIR}}$ and ◐ - 40 − 1 + $\log_2(\mathsf{p} \cdot E^{\mathsf{avg}})$ bit address for RIPPLE$_{\mathsf{PIR}}$.

[a] Includes registration of public keys with the exit node $\mathcal{N}_{\mathtt{exit}}$. [b] includes message download, decryption/PIR queries, summation.

Table 5. Detailed communication costs per simulation step in 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. 5. Fig. 15 captures these observations better, and Tab. 5 and Tab. 4 in the next subsection provide a detailed analysis of the concrete communication costs.

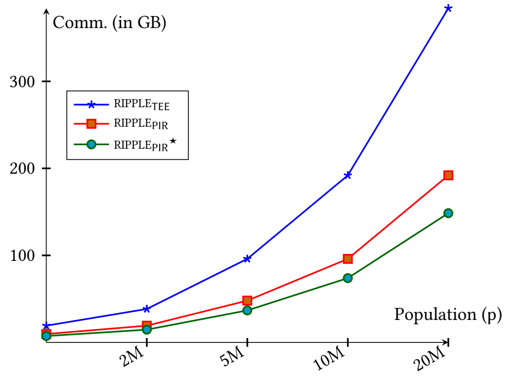


Fig. 15. Communication costs for servers per simulation step for varying population. ★ denotes the results for optimized bit addresses in RIPPLE$_{\mathsf{PIR}}$ (cf. Tab. 5).

6.1.3 Communication Micro Benchmarks. Tab. 5 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 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. 5 does not include costs for verification against malicious participants since they can be eliminated using server S_0 (cf. §5.3.2) or sketching algorithms similar to those in [18].

6.2 Computation Complexity

This section focuses on the runtime, which includes time for computation and communication between entities. Tab. 6 summarizes the computation time with respect to a participant $\mathcal{P}_i$ for a two-week simulation over a population of half a million. The longer computation time in RIPPLE$_{\mathsf{TEE}}$, as shown in Tab. 6, is due to the public key encryption

Stages of RIPPLE	Protocol	Population (p)					
		1K	10K	50K	100K	500K	1M
Message Generation by $\mathcal{P}_i \in \mathcal{P}$ (in sec)	RIPPLE$_{\mathsf{TEE}}$ (§4.1)	1.12	1.12	1.12	1.12	1.12	1.12
	RIPPLE$_{\mathsf{PIR}}$: (§4.2)	4.26e-3	4.26e-3	4.26e-3	4.26e-3	4.26e-3	4.26e-3
Secure Shuffle by C (in sec)	RIPPLE$_{\mathsf{TEE}}$ (§4.1)	0.70	5.20	25.38	60.77	211.47	493.33$^{\star a}$
	RIPPLE$_{\mathsf{PIR}}$ (§4.2)	0.78	6.65	32.36	71.17	386.68	1542.30*
Output Computation^{b} (in sec)	RIPPLE$_{\mathsf{TEE}}$ (§4.1)	44.66	44.66	44.66	44.66	44.66	44.66
	$\mathsf{PIR}^{\mathsf{I}}_{\mathsf{sum}}$ (§5.3.1)	32.31	32.33	32.34	32.35	32.36	32.37
	$\mathsf{PIR}^{\mathsf{II}}_{\mathsf{sum}}$ (§5.3.2)	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 7. Detailed computation costs per simulation ($N_{\text{step}} = 14$, i.e., 14 days) in RIPPLE.

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 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 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.

	Per Simulation Step			Per Simulation ($N_{\text{step}} = 14$)		
	Message Generation (in ms)	PIR Queries (in ms)	Output Computation (in ms)	Message Generation (in sec)	PIR Queries (in sec)	Output Computation (in sec)
RIPPLE$_{\mathsf{TEE}}$	80.00	-	3040.00	1.12	-	42.56
$\mathsf{PIR}^{\mathsf{I}}_{\mathsf{sum}}$	0.30	11.73	4.8e-2	4.26e-3	0.16	6.72e-4
$\mathsf{PIR}^{\mathsf{II}}_{\mathsf{sum}}$	0.30	3.0e-3	4.8e-2	4.26e-3	4.2e-5	6.72e-4

Table 6. Average participant computation times per simulation step distributed across various tasks. Values are obtained using a mobile for a population of p = 500K with $E^{\text{avg}} = 100$.

In Fig. 16, 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 RIPPLE$_{\mathsf{PIR}}$ begins to exceed that of RIPPLE$_{\mathsf{TEE}}$ due to an increase in database size, which results in longer data transfer times. More details regarding computation time are presented in Tab. 7. Note that the runtimes in Fig. 16 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 RIPPLE$_{\mathsf{PIR}}$.

6.2.1 Computation Micro Benchmarks. Tab. 7 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, which is due to the fact that our implementation requires to store the whole PIR database in the memory and its size increases linearly with the number of participants. This will not be the case in a real-world deployment of powerful servers, which are equipped with more internal memory and additionally can store parts of the database on hard disks. Similar as w.r.t. communication, participants' computation costs are very low in comparison to the overall costs.

6.2.2 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. [126] and Avitabile et al. [12] 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 not possible for a standard third-party mobile application. Additionally, RIPPLE could offer users the choice to only participate in simulations while charging in order to not cause any unwanted battery drain.

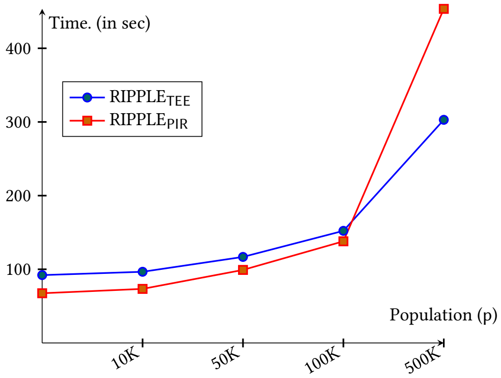


Fig. 16. Runtime per simulation in RIPPLE (14 days).

6.2.3 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[13], the German Corona-Warn-App[14], or the Australian COVIDSafe[15] only record contacts for notifying contacts of infected people. Concretely, contact tracing basically relates to RIPPLE's token generation phase, while the other three phases (simulation initialization, simulation execution, and result aggregation, cf. §3.2) 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, due to differences in the fundamental functionalities, no meaningful comparison between the systems is possible.

6.2.4 Code availability. Available at `DOI: 10.5281/zenodo.6599225`.

Summmary. 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 implementations.

ACKNOWLEDGMENTS

This project received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation program (grant agreement No. 850990 PSOTI). It was co-funded by the Deutsche Forschungsgemeinschaft (DFG) within SFB 1119 CROSSING/236615297 and GRK 2050 Privacy & Trust/251805230.

REFERENCES

[1] Ittai Abraham, Benny Pinkas, and Avishay Yanai. 2020. Blinder: MPC Based Scalable and Robust Anonymous Committed Broadcast. In *ACM CCS*.
[2] David Adam. 2020. Special report: The simulations driving the world's response to COVID-19. *Nature* (2020).
[3] Nadeem Ahmed, Regio A Michelin, Wanli Xue, Sushmita Ruj, Robert Malaney, Salil S Kanhere, Aruna Seneviratne, Wen Hu, Helge Janicke, and Sanjay K Jha. 2020. A Survey of COVID-19 Contact Tracing Apps. *IEEE Access* (2020).

[13] https://github.com/SwissCovid [14] https://www.coronawarn.app/en/ [15] https://www.health.gov.au/resources/apps-and-tools/covidsafe-app

[4] Fadi Al-Turjman and Bakkiam David Deebak. 2020. Privacy-Aware Energy-Efficient Framework Using the Internet of Medical Things for COVID-19. *IEEE Internet Things Mag.* 3, 3 (2020).

[5] Nikolaos Alexopoulos, Aggelos Kiayias, Riivo Talviste, and Thomas Zacharias. 2017. MCMix: Anonymous Messaging via Secure Multiparty Computation. In *USENIX Security*.

[6] Yaniv Altshuler, Nadav Aharony, Micky Fire, Yuval Elovici, and Alex Pentland. 2012. Incremental Learning with Accuracy Prediction of Social and Individual Properties from Mobile-Phone Data. In *International Conference on Privacy, Security, Risk and Trust and 2012 International Conference on Social Computing*.

[7] Android. 2020. Third-party Trusty applications. https://source.android.com/security/trusty.

[8] Sebastian Angel, Hao Chen, Kim Laine, and Srinath Setty. 2018. PIR with Compressed Queries and Amortized Query Processing. In *IEEE S&P*.

[9] Toshinori Araki, Jun Furukawa, Kazuma Ohara, Benny Pinkas, Hanan Rosemarin, and Hikaru Tsuchida. 2021. Secure Graph Analysis at Scale. In *ACM CCS*.

[10] ARM. 2009. ARM security technology building a secure system using TrustZone technology. https://developer.arm.com/documentation/genc009492/c.

[11] Yonatan Aumann and Yehuda Lindell. 2010. Security Against Covert Adversaries: Efficient Protocols for Realistic Adversaries. *Journal of Cryptology* (2010).

[12] Gennaro Avitabile, Vincenzo Botta, Vincenzo Iovino, and Ivan Visconti. 2020. Towards Defeating Mass Surveillance and SARS-CoV-2: The Pronto-C2 Fully Decentralized Automatic Contact Tracing System. https://eprint.iacr.org/2020/493

[13] Alexandros Bampoulidis, Alessandro Bruni, Lukas Helminger, Daniel Kales, Christian Rechberger, and Roman Walch. 2022. Privately Connecting Mobility to Infectious Diseases via Applied Cryptography. *PETs* (2022).

[14] Paolo Barsocchi, Antonello Calabrò, Antonino Crivello, Said Daoudagh, Francesco Furfari, Michele Girolami, and Eda Marchetti. 2021. COVID-19 & privacy: Enhancing of indoor localization architectures towards effective social distancing. *Array* 9 (2021).

[15] Sebastian P. Bayerl, Tommaso Frassetto, Patrick Jauernig, Korbinian Riedhammer, Ahmad-Reza Sadeghi, Thomas Schneider, Emmanuel Stapf, and Christian Weinert. 2020. Offline Model Guard: Secure and Private ML on Mobile Devices. *DATE* (2020).

[16] Joshua Blumenstock, Gabriel Cadamuro, and Robert On. 2015. Predicting poverty and wealth from mobile phone metadata. *Science* (2015).

[17] Dan Boneh, Elette Boyle, Henry Corrigan-Gibbs, Niv Gilboa, and Yuval Ishai. 2021. Lightweight Techniques for Private Heavy Hitters. In *IEEE S&P*.

[18] Elette Boyle, Niv Gilboa, and Yuval Ishai. 2016. Function Secret Sharing: Improvements and Extensions. In *ACM CCS*.

[19] Beyza Bozdemir, Sébastien Canard, Orhan Ermis, Helen Möllering, Melek Önen, and Thomas Schneider. 2021. Privacy-preserving Density-based Clustering. In *ASIACCS*.

[20] Fred Brauer. 2008. Compartmental models in epidemiology. In *Mathematical Epidemiology*.

[21] Fred Brauer, Carlos Castillo-Chavez, and Zhilan Feng. 2019. Simple Compartmental Models for Disease Transmission. In *Mathematical Models in Epidemiology*.

[22] Megha Byali, Harsh Chaudhari, Arpita Patra, and Ajith Suresh. 2020. FLASH: Fast and Robust Framework for Privacy-preserving Machine Learning. *PETS* (2020).

[23] Clea Caulcutt. 2022. Belgium introduces quarantine for monkeypox cases. *Politico* (2022). https://www.politico.eu/article/belgium-introduce-quarantine-monkeypox-case/.

[24] Justin Chan, Dean Foster, Shyam Gollakota, Eric Horvitz, Joseph Jaeger, Sham Kakade, Tadayoshi Kohno, John Langford, Jonathan Larson, Puneet Sharma, Sudheesh Singanamalla, Jacob Sunshine, and Stefano Tessaro. 2020. PACT: Privacy Sensitive Protocols and Mechanisms for Mobile Contact Tracing. https://arxiv.org/pdf/2004.03544.pdf.

[25] Nishanth Chandran, Divya Gupta, Sai Lakshmi Bhavana Obbattu, and Akash Shah. 2022. SIMC: ML Inference Secure Against Malicious Clients at Semi-Honest Cost. In *USENIX Security*.

[26] Harsh Chaudhari, Ashish Choudhury, Arpita Patra, and Ajith Suresh. 2019. ASTRA: High Throughput 3PC over Rings with Application to Secure Prediction. In *ACM CCSW@CCS*.

[27] Harsh Chaudhari, Rahul Rachuri, and Ajith Suresh. 2020. Trident: Efficient 4PC Framework for Privacy Preserving Machine Learning. In *NDSS*.

[28] David Chaum. 1985. Security without Identification: Transaction Systems to Make Big Brother Obsolete. *Commun. ACM* (1985).

[29] David Chaum. 1988. The Dining Cryptographers Problem: Unconditional Sender and Recipient Untraceability. *Journal of Cryptology* (1988).

[30] David L Chaum. 1981. Untraceable Electronic Mail, Return Addresses, and Digital Pseudonyms. *Commun. ACM* (1981).

[31] Guoxing Chen, Yinqian Zhang, and Ten-Hwang Lai. 2019. OPERA: Open Remote Attestation for Intel's Secure Enclaves. In *ACM CCS*.

[32] Yi-Cheng Chen, Ping-En Lu, Cheng-Shang Chang, and Tzu-Hsuan Liu. 2020. A Time-Dependent SIR Model for COVID-19 With Undetectable Infected Persons. *Transactions on Network Science and Engineering* (2020).

[33] Benny Chor, Oded Goldreich, Eyal Kushilevitz, and Madhu Sudan. 1995. Private Information Retrieval. In *FOCS*.

[34] Adam Durbin Christy Cooney. 2022. High-risk monkeypox contacts advised to isolate. *BBC* (2022). https://www.bbc.com/news/uk-61546480.

[35] Matteo Ciucci and Frédéric Gouardères. 2020. National COVID-19 contact tracing apps. *EPRS: European Parliamentary Research Service* (2020).

[36] Henry Corrigan-Gibbs, Dan Boneh, and David Mazières. 2015. Riposte: An Anonymous Messaging System Handling Millions of Users. In *IEEE S&P*.

[37] Henry Corrigan-Gibbs and Dmitry Kogan. 2020. Private Information Retrieval with Sublinear Online Time. In *EUROCRYPT*.

[38] Ivan Damgård, Marcel Keller, Enrique Larraia, Valerio Pastro, Peter Scholl, and Nigel P. Smart. 2013. Practical Covertly Secure MPC for Dishonest Majority – Or: Breaking the SPDZ Limits. In *ESORICS*.

[39] George Danezis and Len Sassaman. 2003. Heartbeat Traffic to Counter (n-1) Attacks: Red-Green-Black Mixes *(WPES'03)*.

[40] Nicholas G Davies, Adam J Kucharski, Rosalind M Eggo, Amy Gimma, W John Edmunds, Thibaut Jombart, Kathleen O'Reilly, Akira Endo, Joel Hellewell, Emily S Nightingale, et al. 2020. 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).

[41] Leo de Castro and Anitgoni Polychroniadou. 2022. Lightweight, Maliciously Secure Verifiable Function Secret Sharing. In *EUROCRYPT*.

[42] Roberta De Viti, Isaac Sheff, Noemi Glaeser, Baltasar Dinis, Rodrigo Rodrigues, Jonathan Katz, Bobby Bhattacharjee, Anwar Hithnawi, Deepak Garg, et al. 2022. CoVault: A Secure Analytics Platform. (2022). https://arxiv.org/pdf/2208.03784.pdf.

[43] Sara Y Del Valle, James M Hyman, Herbert W Hethcote, and Stephen G Eubank. 2007. Mixing patterns between age groups in social networks. *Social Networks* (2007).

[44] Daniel Demmler, Thomas Schneider, and Michael Zohner. 2015. ABY - A Framework for Efficient Mixed-Protocol Secure Two-Party Computation. In *NDSS*.

[45] Odo Diekmann, Hans Heesterbeek, and Tom Britton. 2012. *Mathematical Tools for Understanding Infectious Disease Dynamics*. Princeton University Press.

[46] Changyu Dong, Liqun Chen, and Zikai Wen. 2013. When private set intersection meets big data: an efficient and scalable protocol. In *ACM CCS*.

[47] Wenliang Du. 2001. A study of several specific secure two party computation problems. *USA: Purdue University* (2001).

[48] W John Edmunds, CJ O'callaghan, and DJ Nokes. 1997. 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).

[49] J. Ekberg, K. Kostiainen, and N. Asokan. 2014. The Untapped Potential of Trusted Execution Environments on Mobile Devices. In *IEEE S&P*.

[50] Z. Erkin, J. R. Troncoso-pastoriza, R. L. Lagendijk, and F. Perez-Gonzalez. 2013. Privacy-Preserving Data Aggregation in Smart Metering Systems: An Overview. In *Signal Processing Magazine*.

[51] Saba Eskandarian and Dan Boneh. 2022. Clarion: Anonymous Communication from Multiparty Shuffling Protocols. In *NDSS*.

[52] Hossein Fereidooni, Samuel Marchal, Markus Miettinen, Azalia Mirhoseini, Helen Möllering, Thien Duc Nguyen, Phillip Rieger, Ahmad-Reza Sadeghi, Thomas Schneider, Hossein Yalame, et al. 2021. SAFELearn: secure aggregation for private federated learning. In *IEEE Security and Privacy Workshops (SPW)*.

[53] Neil Ferguson. 2005. What would happen if a flu pandemic arose in Asia? *Nature* (2005).

[54] Neil M Ferguson, Derek AT Cummings, Christophe Fraser, James C Cajka, Philip C Cooley, and Donald S Burke. 2006. Strategies for mitigating an influenza pandemic. *Nature* (2006).

[55] Jesús Fernández-Villaverde and Charles I Jones. 2022. Estimating and Simulating a SIRD Model of COVID-19 for Many Countries, States, and Citie. *Journal of Economic Dynamics and Control* (2022).

[56] European Centre for Disease Prevention and Control. 2022. Epidemiological update: Monkeypox outbreak. (2022). https://www.ecdc.europa.eu/en/news-events/epidemiological-update-monkeypox-outbreak.

[57] Craig Gentry and Shai Halevi. 2019. Compressible FHE with Applications to PIR. In *TCC*.

[58] Niv Gilboa and Yuval Ishai. 2014. Distributed Point Functions and Their Applications. In *EUROCRYPT*.

[59] Giulia Giordano, Franco Blanchini, Raffaele Bruno, Patrizio Colaneri, Alessandro Di Filippo, Angela Di Matteo, and Marta Colaneri. 2020. Modelling the COVID-19 epidemic and implementation of population-wide interventions in Italy. *Nature Medicine* (2020).

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

[61] Alison Gray, David Greenhalgh, Liangjian Hu, Xuerong Mao, and Jiafeng Pan. 2011. A Stochastic Differential Equation SIS Epidemic Model. *SIAM J. Appl. Math.* (2011).

[62] Daniel Günther, Maurice Heymann, Benny Pinkas, and Thomas Schneider. 2022. GPU-accelerated PIR with Client-Independent Preprocessing for Large-Scale Applications. In *USENIX Security*.

[63] Thomas Haines and Johannes Müller. 2020. SoK: Techniques for Verifiable Mix Nets. In *CSF*.

[64] Tiberiu Harko, Francisco SN Lobo, and MK3197716 Mak. 2014. Exact analytical solutions of the Susceptible-Infected-Recovered (SIR) epidemic model and of the SIR model with equal death and birth rates. *Appl. Math. Comput.* (2014).

[65] Gary F Hatke, Monica Montanari, Swaroop Appadwedula, Michael Wentz, John Meklenburg, Louise Ivers, Jennifer Watson, and Paul Fiore. 2020. Using Bluetooth Low Energy (BLE) Signal Strength Estimation to Facilitate Contact Tracing for COVID-19. https://arxiv.org/ftp/arxiv/papers/2006/2006.15711.pdf.

[66] Shaobo He, Yuexi Peng, and Kehui Sun. 2020. SEIR modeling of the COVID-19 and its dynamics. *Nonlinear Dynamics* (2020).

[67] Yan Huang, David Evans, and Jonathan Katz. 2012. Private Set Intersection: Are Garbled Circuits Better than Custom Protocols?. In *NDSS*.

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

[69] Intel. 2014. Intel® Software Guard Extensions Programming Reference. https://software.intel.com/sites/default/files/managed/48/88/329298-002.pdf.

[70] Intel. unk. Attestation Service for Intel Software Guard Extensions. https://api.trustedservices.intel.com/documents/sgx-attestation-api-spec.pdf.

[71] Kimmo Järvinen, Helena Leppäkoski, Elena-Simona Lohan, Philipp Richter, Thomas Schneider, Oleksandr Tkachenko, and Zheng Yang. 2019. PILOT: Practical Privacy-Preserving Indoor Localization Using OuTsourcing. In *EuroS&P*.

[72] P. Jauernig, A. Sadeghi, and E. Stapf. 2020. Trusted Execution Environments: Properties, Applications, and Challenges. In *IEEE S&P*.

[73] Daniel Kales, Olamide Omolola, and Sebastian Ramacher. 2019. Revisiting User Privacy for Certificate Transparency. In *EuroS&P*.

[74] W. O. Kermack and A. G. McKendrick. 1991. Contributions to the mathematical theory of epidemics—I. In *Bulletin of Mathematical Biology*.

[75] Adam Kirsch, Michael Mitzenmacher, and Udi Wieder. 2010. More Robust Hashing: Cuckoo Hashing with a Stash. *Journal on Computing* (2010).

[76] Petra Klepac, Adam J Kucharski, Andrew JK Conlan, Stephen Kissler, Maria L Tang, Hannah Fry, and Julia R Gog. 2020. Contacts in context: large-scale setting-specific social mixing matrices from the BBC Pandemic project. *MedRxiv* (2020). https://www.medrxiv.org/content/10.1101/2020.02.16.20023754v2.full.pdf.

[77] Victor I. Kolobov, Elette Boyle, Niv Gilboa, and Yuval Ishai. 2022. Programmable Distributed Point Functions. In *CRYPTO*.

[78] Nishat Koti, Mahak Pancholi, Arpita Patra, and Ajith Suresh. 2021. SWIFT: Super-fast and Robust Privacy-Preserving Machine Learning. In *USENIX Security*.

[79] Nishat Koti, Arpita Patra, Rahul Rachuri, and Ajith Suresh. 2022. Tetrad: Actively Secure 4PC for Secure Training and Inference. In *NDSS*.

[80] Kai Kupferschmidt. 2020. Case clustering emerges as key pandemic puzzle. https://www.science.org/doi/full/10.1126/science.368.6493.808.

[81] Klaus Kursawe, George Danezis, and Markulf Kohlweiss. 2011. Privacy-friendly aggregation for the smart-grid. In *PETS*.

[82] Eyal Kushilevitz and Rafail Ostrovsky. 1997. Replication is NOT Needed: SINGLE Database, Computationally-Private Information Retrieval. In *FOCS*.

[83] Sven Laur, Jan Willemson, and Bingsheng Zhang. 2011. Round-Efficient Oblivious Database Manipulation. In *International Conference on Information Security*.

[84] Ryan Lehmkuhl, Pratyush Mishra, Akshayaram Srinivasan, and Raluca Ada Popa. 2021. MUSE: Secure Inference Resilient to Malicious Clients. In *USENIX Security*.

[85] Dyani Lewis. 2020. Where Covid contact-tracing went wrong. *Nature* (2020).

[86] F. Li, B. Luo, and P. Liu. 2010. Secure Information Aggregation for Smart Grids Using Homomorphic Encryption. In *International Conference on Smart Grid Communications*.

[87] Yehuda Lindell and Benny Pinkas. 2007. An Efficient Protocol for Secure Two-Party Computation in the Presence of Malicious Adversaries. In *EUROCRYPT*.

[88] Yehuda Lindell, Benny Pinkas, Nigel P Smart, and Avishay Yanai. 2015. Efficient Constant Round Multi-Party Computation Combining BMR and SPDZ. In *CRYPTO*.

[89] Wouter Lueks, Seda F. Gürses, Michael Veale, Edouard Bugnion, Marcel Salathé, Kenneth G. Paterson, and Carmela Troncoso. 2021. CrowdNotifier: Decentralized Privacy-Preserving Presence Tracing. *PETs* (2021).

[90] Shaojun Luo, Flaviano Morone, Carlos Sarraute, Matías Travizano, and Hernán A Makse. 2017. Inferring personal economic status from social network location. *Nature Communications* (2017).

[91] Dominika Maison, Diana Jaworska, Dominika Adamczyk, and Daria Affeltowicz. 2021. The challenges arising from the COVID-19 pandemic and the way people deal with them. A qualitative longitudinal study. *PloS One* (2021).

[92] Robert M. May and Alun L. Lloyd. 2001. Infection dynamics on scale-free networks. *Physical Review E* (2001).

[93] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. 2017. Communication-Efficient Learning of Deep Networks from Decentralized Data. In *International Conference on Artificial Intelligence and Statistics*.

[94] Pratyush Mishra, Ryan Lehmkuhl, Akshayaram Srinivasan, Wenting Zheng, and Raluca Ada Popa. 2020. Delphi: A Cryptographic Inference Service for Neural Networks. In *USENIX Security*.

[95] Payman Mohassel and Saeed Sadeghian. 2013. How to Hide Circuits in MPC an Efficient Framework for Private Function Evaluation. In *EUROCRYPT*.

[96] Payman Mohassel and Yupeng Zhang. 2017. SecureML: A System for Scalable Privacy-Preserving Machine Learning. In *IEEE S&P*.

[97] Yves-Alexandre de Montjoye, Jordi Quoidbach, Florent Robic, and Alex Sandy Pentland. 2013. Predicting Personality Using Novel Mobile Phone-Based Metrics. In *International Conference on Social Computing, Behavioral-Cultural Modeling, and Prediction*.

[98] Joël Mossong, Niel Hens, Mark Jit, Philippe Beutels, Kari Auranen, Rafael Mikolajczyk, Marco Massari, Stefania Salmaso, Gianpaolo Scalia Tomba, Jacco Wallinga, et al. 2008. Social Contacts and Mixing Patterns Relevant to the Spread of Infectious Diseases. *PLoS Medicine* (2008).

[99] Milad Nasr, Reza Shokri, and Amir Houmansadr. 2019. Comprehensive privacy analysis of deep learning: Passive and active white-box inference attacks against centralized and federated learning. In *IEEE S&P*.

[100] B. Ngabonziza, D. Martin, A. Bailey, H. Cho, and S. Martin. 2016. TrustZone Explained: Architectural Features and Use Cases. In *International Conference on Collaboration and Internet Computing*.

[101] Olga Ohrimenko, Felix Schuster, Cédric Fournet, Aastha Mehta, Kapil Vaswani, and Manuel Costa. 2016. Oblivious Multi-Party Machine Learning on Trusted Processors. In *USENIX Security*.

[102] Rasmus Pagh and Flemming Friche Rodler. 2004. Cuckoo Hashing. *Journal of Algorithms* (2004).

[103] Christian Paquin and Greg Zaveruch. 2013. *U-Prove Cryptographic Specification V1.1 (Revision 3)*. http://www.microsoft.com/uprove.

[104] Romualdo Pastor-Satorras and Alessandro Vespignani. 2002. Immunization of complex networks. *Physical Review E* (2002).

[105] Arpita Patra, Thomas Schneider, Ajith Suresh, and Hossein Yalame. 2021. ABY2.0: Improved Mixed-Protocol Secure Two-Party Computation. In *USENIX Security*.

[106] Arpita Patra and Ajith Suresh. 2020. BLAZE: Blazing Fast Privacy-Preserving Machine Learning. In *NDSS*.

[107] Matthias Pezzutto, Nicolás Bono Rosselló, Luca Schenato, and Emanuele Garone. 2021. Smart Testing and Selective Quarantine for the Control of Epidemics. *Annual Review of Control, Robotics, and Autonomous Systems* 51 (2021), 540–550.

[108] Benny Pinkas and Eyal Ronen. 2021. Hashomer–Privacy-Preserving Bluetooth Based Contact Tracing Scheme for Hamagen. *Real World Crypto and NDSS Corona-Def Workshop* (2021).

[109] Benny Pinkas, Mike Rosulek, Ni Trieu, and Avishay Yanai. 2020. PSI from PaXoS Fast, Malicious Private Set Intersection. In *EUROCRYPT*.

[110] Benny Pinkas, Thomas Schneider, Gil Segev, and Michael Zohner. 2015. Phasing: Private Set Intersection Using Permutation-based Hashing. In *USENIX Security*.

[111] Benny Pinkas, Thomas Schneider, Christian Weinert, and Udi Wieder. 2018. Efficient Circuit-Based PSI via Cuckoo Hashing. In *EUROCRYPT*.

[112] Benny Pinkas, Thomas Schneider, and Michael Zohner. 2018. Scalable Private Set Intersection based on OT Extension. *TOPS* (2018).

[113] Leonie Reichert, Samuel Brack, and Björn Scheuermann. 2021. Poster: Privacy-Preserving Contact Tracing of COVID-19 Patients. In *IEEE S&P*.

[114] Reinhard Schlickeiser and Martin Kröger. 2021. Analytical Modeling of the Temporal Evolution of Epidemics Outbreaks Accounting for Vaccinations. (2021).

[115] Thomas Schneider and Oleksandr Tkachenko. 2019. EPISODE: Efficient Privacy-PreservIng Similar Sequence Queries on Outsourced Genomic DatabasEs. In *ASIACCS*.

[116] Vivek K Singh, Laura Freeman, Bruno Lepri, and Alex Sandy Pentland. 2013. Predicting Spending Behavior Using Socio-mobile Features. In *International Conference on Social Computing*.

[117] Michael Small and Chi K Tse. 2005. Small World and Scale free Model of Transmission of SARS. In *International Journal of Bifurcation and Chaos*.

[118] Hallam Stevens and Monamie Bhadra Haines. 2020. TraceTogether: Pandemic Response, Democracy, and Technology. https://www.tracetogether.gov.sg.

[119] Amanda Taub. 2020. A New Covid-19 Crisis: Domestic Abuse Rises Worldwide. *The New York Times* (2020).

[120] Robin N. Thompson. 2020. Epidemiological models are important tools for guiding COVID-19 interventions. *BMC Medicine* 18, 1 (2020), 152.

[121] Oleksandr Tkachenko, Christian Weinert, Thomas Schneider, and Kay Hamacher. 2018. Large-Scale Privacy-Preserving Statistical Computations for Distributed Genome-Wide Association Studies. In *ASIACCS*.

[122] Carmela Troncoso, Mathias Payer, Jean-Pierre Hubaux, Marcel Salathé, James R. Larus, Wouter Lueks, Theresa Stadler, Apostolos Pyrgelis, Daniele Antonioli, Ludovic Barman, Sylvain Chatel, Kenneth G. Paterson, Srdjan Capkun, David A. Basin, Jan Beutel, Dennis Jackson, Marc Roeschlin, Patrick Leu, Bart Preneel, Nigel P. Smart, Aysajan Abidin, Seda Gurses, Michael Veale, Cas Cremers, Michael Backes, Nils Ole Tippenhauer, Reuben Binns, Ciro Cattuto, Alain Barrat, Dario Fiore, Manuel Barbosa, Rui Oliveira, and José Pereira. 2020. Decentralized Privacy-Preserving Proximity Tracing. *IEEE Data Engineering Bulletin* (2020).

[123] Paul Tupper, Sarah P. Otto, and Caroline Colijn. 2021. Fundamental Limitations of Contact Tracing for COVID-19. *FACETS* (2021).

[124] Christopher van der Beets, Raine Nieminen, and Thomas Schneider. 2022. FAPRIL: Towards Faster Privacy-Preserving Fingerprint-Based Localization. In *SECRYPT*.

[125] Serge Vaudenay. 2020. Centralized or Decentralized? The Contact Tracing Dilemma. Cryptology ePrint Archive, Report 2020/531. https://ia.cr/2020/531.

[126] Serge Vaudenay and Martin Vuagnoux. 2020. *Analysis of SwissCovid*. Technical Report.

[127] Meilof Veeningen, Supriyo Chatterjea, Anna Zsófia Horváth, Gerald Spindler, Eric Boersma, Peter van der SPEK, Onno Van Der Galiën, Job Gutteling, Wessel Kraaij, and Thijs Veugen. 2018. Enabling Analytics on Sensitive Medical Data with Secure Multi-Party Computation. In *Medical Informatics Europe*.

[128] Nina Vindegaard and Michael Eriksen Benros. 2020. COVID-19 pandemic and mental health consequences: Systematic review of the current evidence. *Brain, Behavior, and Immunity* (2020).

[129] Olivia J Walch, Amy Cochran, and Daniel B Forger. 2016. A global quantification of "normal" sleep schedules using smartphone data. *Science Advances* (2016).

[130] Guan Wang, Tongbo Luo, Michael T Goodrich, Wenliang Du, and Zutao Zhu. 2010. Bureaucratic protocols for secure two-party sorting, selection, and permuting. In *ASIACCS*.

[131] Victoria Woollaston. 2015. *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*. https://www.dailymail.co.uk/sciencetech/article-3042230/Sleeping-habits-world-revealed-wakes-grumpy-China-best-quality-shut-eye-South-Africa-wakes-earliest.html.

[132] Andrew Chi-Chih Yao. 1986. How to Generate and Exchange Secrets. In *FOCS*.

[133] Tijana Šušteršič, Andjela Blagojević, Danijela Cvetković, Aleksandar Cvetković, Ivan Lorencin, Sandi Baressi Šegota, Dragan Milovanović, Dejan Baskić, Zlatan Car, and Nenad Filipović. 2021. Epidemiological Predictive Modeling of COVID-19 Infection: Development, Testing, and Implementation on the Population of the Benelux Union. *Frontiers in Public Health* 9 (2021).

A CRYPTOGRAPHIC PRIMITIVES USED

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

Anonymous Communication. To simulate the transmission of the modelled disease, RIPPLE requires anonymous messaging between participants. Mix-nets [30] and protocols based on the dining cryptographer (DC) problem [29] 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 [47, 67, 130], homomorphic encryption [67], distributed point functions [1], switching networks [95], permutation matrices [83, §4.1], sorting algorithms [83, §4.2], and re-sharing [83, §4.3+4.4]. Recently, the works of [9] and [51] proposed efficient oblivious shuffling schemes using a small number of mix net servers.

Trusted Execution Environment (TEE). RIPPLE$_{\mathsf{TEE}}$ (§4.1) requires a TEE on the mobile devices of participants. TEEs are hardware-assisted environments that provide secure storage and execution of code on sensitive data which is 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 [49, 72]. Widely adopted TEEs are Intel SGX [69] and ARM TrustZone [10] (often used on mobile platforms [100]). Using TEEs for private computation has been extensively investigated, e.g., [15, 101]. 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 [31, 70].

Private Information Retrieval (PIR). The first computational single-server PIR (cPIR) scheme was introduced by Kushilevitz and Ostrovsky [82]. Recent cPIR schemes [8, 57] 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 [17, 18, 33, 36, 37] making it significantly more efficient than cPIR.

Cuckoo Hashing. In RIPPLE$_{\mathsf{PIR}}$ (§4.2), 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 §4.2, RIPPLE$_{\mathsf{PIR}}$ requires that at most one element is stored in every database location which renders simple hashing impracticable [110]. 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, the element is inserted into a special bin called stash which is allowed to hold more than one element. Pinkas et al. [110] 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.

Garbled Cuckoo Table (GCT). As RIPPLE$_{\mathsf{PIR}}$ uses key-value pairs for the insertion into the database, a combination of garbled Bloom filters [46] with cuckoo hashing [75, 102], called Garbled Cuckoo Table [109], is needed. Instead of storing x elements in one bin as in an ordinary cuckoo table, 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 §4.2.3, we introduce a variant of GCT called *arithmethic 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. [109] show that the number of cycles is maximally $\log_2 n$ with high probability.

Secure Multi-Party Computation (MPC). MPC [132] 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 [38, 44, 88]. 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 [9, 22, 27, 78, 79, 106] 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.3) and to develop efficient $\mathsf{PIR}_{\mathsf{sum}}$ protocols (cf. §5).

Anonymous Credentials. To protect against sybil attacks (cf. §3.3), 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 [28] 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 [103].

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.

B.1 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 $C' = C \cup \{\mathcal{P}_i\}$ for a participant $\mathcal{P}_i \in \mathcal{P}$. The following PRF keys are established among the parties in C' in RIPPLE:

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

To sample a random value $r_{ij} \in_R \mathbb{Z}_{2^\ell}$ 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.2 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 $m = \mathrm{poly}(\kappa)$.

B.3 Anonymous Communication Channel

This section describes how to instantiate the $\mathcal{F}_{\mathsf{anon}}$ functionality used by RIPPLE for anonymous communication, as discussed in §4. 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 §4.2 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 [9, 51], 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 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. §4.2). 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) $\mathcal{P}_i, \mathrm{S}_0$ sample random $\langle\vec{m}\rangle_1 \in_R \mathbb{Z}^{\tau}_{2^\ell}$.
   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. $\mathrm{S}_0, \mathrm{S}_1$ perform a *re-sharing* of $\pi_{01}(\vec{m})$, denoted by $\vec{m_{01}}$, by jointly sampling a random $\vec{r_{01}} \in_R \mathbb{Z}^{\tau}_{2^\ell}$ 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 $\langle\vec{m_{02}}\rangle_1 \in_R \mathbb{Z}^{\tau}_{2^\ell}$ 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 $\vec{m_{02}} = \langle\vec{m_{02}}\rangle_1 + \langle\vec{m_{02}}\rangle_2$.

Anonymous Communication in RIPPLE$_{\mathsf{TEE}}$. As discussed in §4.1, 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 $\vec{m_{01}} = \pi_{01}(\vec{m})$.

Security Guarantees. As discussed in §3.1, we assume that the MPC servers $\mathrm{S}_i, i \in [2]$, 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 [25, 84], 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 by the exit servers, 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. Hence, considering the discussed aspects/assumptions, classical attacks on anonymous communication such as flooding [39] are not relevant for our model.

B.4 Distributed Point Functions (DPF)

Consider a point function $P_{\alpha,\beta} : \mathbb{Z}_{2^\ell} \longrightarrow \mathbb{Z}_{2^{\ell'}}$ 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) [36, 58] 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 §5.3. Formally, a $(2,1)$-DPF comprises of the following two functionalities:

- $\mathsf{Gen}(\alpha,\beta) \longrightarrow (k_1,k_2)$. Output two DPF keys k_1 and k_2, given $\alpha \in \mathbb{Z}_{2^\ell}$ and $\beta \in \mathbb{Z}_{2^{\ell'}}$.
- $\mathsf{Eval}(k,\alpha') \longrightarrow \beta'$. 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 $(k_1,k_2) \longleftarrow \mathsf{Gen}(\alpha,\beta)$, 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 [18, 36], a malicious participant could manipulate the Gen algorithm to generate incorrect DPF keys that do not correspond to any point function. While [36] used an external non-colluding auditor to circumvent this issue in the two server setting, [18] 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 $(k_1,k_2) \longleftarrow \mathsf{Gen}(\alpha,\beta)$, 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 [41, 77] might be an interesting direction for future work to further enhance the efficiency of our $\mathsf{RIPPLE}_{\mathsf{PIR}}$ construction.

Communication Complexity. Using the protocol of Boyle et. al. [18], 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 [18]. Furthermore, as stated in [18], 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 §5.3.1. We refer to [18] for more details.

C PIR-SUM PROTOCOL DETAILS

This section provides additional details of our $\mathsf{PIR}_{\mathsf{sum}}$ protocols introduced in §5.1. We begin by recalling the security guarantees of a 2-server PIR for our setting [33, 62]. 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 C.1.

Definition C.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 $view(\mathsf{S}_u, Q)$ denote the view of server $\mathsf{S}_u \in \{\mathsf{S}_1, \mathsf{S}_2\}$ with respect to a list of queries, denoted by Q. We require that for any database D, and for any two τ-length list of queries $Q = (q_1, \ldots, q_\tau)$ and $Q' = (q'_1, \ldots, q'_\tau)$, 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 Q ($\{view(\mathsf{S}_u, Q)\}_{u \in \{1,2\}}$), and the case of it using Q' ($\{view(\mathsf{S}_u, Q')\}_{u \in \{1,2\}}$).

Linear Summation PIR for $\mathcal{F}^{2\mathsf{S}}_{\mathsf{pir}}$ with optimized Communication. This section describes Chor et al.'s 2-server linear summation PIR protocol [33], as well as how to optimize communication using DPF techniques discussed in Appendix B.4. 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 $\vec{\mathsf{b}}_q = \{\mathsf{b}_q^1, \ldots, \mathsf{b}_q^N\}$ with $\mathsf{b}_q^j = 1$ for $j = q$ and $\mathsf{b}_q^j = 0$ 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., $\mathcal{P}_i$ and S_1 non-interactively sample the random $[\vec{\mathsf{b}}_q]_1 \in_R \{0,1\}^N$ 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 $[y]_u = \bigoplus_{j=1}^{N} [\mathsf{b}_q^j]_u \mathsf{D}[j]$ 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.

Optimizing Communication using DPFs. Several works in the literature [18, 36, 58, 62] have used DPFs (cf. Appendix B.4) as a primitive to improve the communication in multi-server PIR. The idea is to use a DPF 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., $(k_1, k_2) \longleftarrow \mathsf{Gen}(q, 1)$.
- $\mathcal{P}_i$ 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 $[\mathsf{b}_q^j]_u \longleftarrow \mathsf{Eval}(k_u, j)$ for $j \in [N]$.
- S_u, for $u \in \{1, 2\}$, sends $[y]_u = \bigoplus_{j=1}^{N} [\mathsf{b}_q^j]_u \mathsf{D}[j]$ 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 [18] and the key size is $O(\lambda \cdot \log(N/\lambda))$ bits, where $\lambda = 128$ is related to AES implementation in [18].

To prevent a malicious participant from sending incorrect or malformed keys to the servers, we use the verifiable DPF construction proposed in [18] for the case of malicious participants. This results only in a constant communication overhead over the semi-honest case. Furthermore, as noted in [18], 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.4 and refer the reader to [18] for all details.