# PEM: Privacy-preserving Epidemiological Modeling

Daniel Günther[1], Marco Holz[1], Benjamin Judkewitz[2], Helen Möllering[1], Benny Pinkas[3], and Thomas Schneider[1]

[1] Technical University of Darmstadt, Darmstadt, Germany
{lastname}@encrypto.cs.tu-darmstadt.de
[2] Charité-Universitätsmedizin, Berlin, Germany
benjamin.judkewitz@charite.de
[3] Bar-Ilan University, Ramat Gan, Israel
benny@pinkas.net

**Abstract.** To better manage the current pandemic, it would be beneficial to model the potential effect of measures, e.g., contact restrictions and school closings, on the spread of COVID-19. However, detailed epidemiological simulations suffer from a scarcity of relevant empirical data, such as social contact graphs. As this data is also 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 introduce the problem of *privacy-preserving epidemiological modeling (PEM)*. We propose a practical framework for PEM on contact information stored on mobile phones, like the ones collected by already deployed contact tracing apps. Unlike existing apps that focus on past events, PEM allows for meaningful epidemiological simulations about *future* developments. PEM protects the privacy of the users by not revealing sensitive data to the system operator or other participants, while enabling detailed predictive models of pandemic spread. Our protocols are combining mix-nets with either trusted execution environments or private information retrieval. We show that they achieve practical performance for deployments in large regions.

**Keywords:** Decentralized Epidemiological Modeling · Privacy · Private Information Retrieval · COVID-19

## 1 Introduction

In the current pandemic, governmental decision making processes closely resemble epidemiological simulations: They put measures in place, wait for a few weeks, and evaluate the outcome. If the effect is not sufficient different measures are put into place.

In contrast, epidemiological modeling can predict the spread of diseases for many scenarios anticipating the effects of different control measures in advance within a short time period. With these insights, it enables to create informed epidemic containment strategies and political interventions *before* putting them into place. Considering the massive restrictions people face to get the virus

under control, it is in everybody's interest to determine which measures are most promising and enable people to securely return to normality as soon as possible. However, detailed epidemiological models of communicable diseases, including COVID-19, have long suffered from a lack of available social interaction data [3, 27, 39]. From a modeler's perspective, epidemiologists would ideally like to have access to the complete physical interaction graph of a population. It has long been known that the topology of the interaction network has a large influence on the spread of diseases [44, 49]. Empirical contact graph data would permit detailed simulations of how a disease propagates through an interaction network, and what type of interventions might be most effective at containing it. Yet, direct access to the social contact graph of a population raises vast privacy concerns, rendering this direct modeling approach non-viable.

In this work, we present a practical framework for privacy-preserving epidemiological modeling (PEM) to overcome this barrier. Our goal is to leverage data about physical closeness collected by mobile devices to realise detailed (decentralized) epidemiological modeling on the real contact graph while simultaneously protecting privacy. Many simulations need to be run to compare the effect of different disease management scenarios or to anticipate potential manifestations of properties of a new unknown disease. However, when running multiple simulations on the real-world connectivity graph, even just the *simulated* infection status of other participants can leak information about the *real* contact graphs (cf. §4.1). Thus, PEM requires sophisticated approaches for protecting the privacy of *all* participants. Our PEM protocols can be integrated in already deployed decentralized contact tracing apps.

**Related Work.** A large amount of contact tracing systems have been proposed aiming at informing people about exposures with infectious persons such that they can be isolated and tested. They realize contact tracing either based on the location (via GPS or telecommunication provider information) or based on proximity (via Bluetooth LE). The systems can be split into centralized and decentralized approaches. In a centralized contact tracing system (e.g., StopCovid/ROBERT [32,43] and TraceTogether [30]), computations such as the generation of the exchanged identifiers are executed at a central party that may also receive encounter information of people. In contrast, in decentralized approaches (e.g., DP-3T [54] and PACT [12]), computation and encounter information remain (almost completely) locally at the participants' devices.

A system for privacy-preserving detection of infection hotspots using location data stored at a mobile network operator was proposed in [7]. While learning about areas of high risk of contamination also helps policymakers for taking countermeasures, PEM goes beyond by simulating the effect of various containment measures in advance.

FluPhone [58] aims at applying epidemiological models on real-world, non-aggregated contact graph data of individuals. The project collects proximity data via Bluetooth LE and GPS location data, but only anonymizes user identifiers. Also Biasse et al. [9] suggest to create an anonymized contact graph from contact information collected via Bluetooth-based decentralized contact tracing

apps, that can then be used for epidemiological modeling. However, even an anonymized contact graph without user identifiers and locations would have to include some time and distance information to enable the determination of the infection likelihood of contacts. Such information still allows to detect potentially sensitive behavior patterns, e.g., regularly occurring times of prayers common in some religions, that might even allow re-identification to a certain extent.

**Outline and our Contributions.** After giving related work (§1) and preliminaries (§2), we provide the following contributions:

- In §3, we introduce an extension to existing private information retrieval (PIR) schemes that we call *Threshold-PIR-SUM*. It allows a client to download the *sum* of $t$ *distinct* entries of a database without learning the values of individual entries and without revealing which entries were requested. Our idea for verifying properties of multi-server PIR queries via MPC might be of independent interest.
- In §4, we introduce privacy-preserving epidemiological modeling (PEM), its privacy requirements, ideal functionality, and components. Further, we provide a detailed evaluation of two attacks on PEM that aim at extracting information about the contact graph. Then, we introduce two PEM protocols with different trust assumptions for epidemiological simulations on real-world contact data collected with the users' mobile devices. In our protocols, all data remains locally on the devices by distributing the simulation using cryptographic building blocks. Our PEM protocols combine mix-nets with either trusted execution environments or PIR and anonymous credentials.
- In §5, we analyze privacy guarantees, communication costs, and give microbenchmarks for the runtimes of our protocols. We show that they are efficient and even scale up to thousands of users.

Our goal is to extend the focus of the privacy research community from private contact tracing, which notifies users about potential exposures in the *past*, to the problem of PEM for predicting the effect of containment measures in the *future*.

## 2 Preliminaries

In this section, we summarize epidemiological modeling and the cryptographic building blocks used in our work.

### 2.1 Epidemiological Modeling

State-based compartment models capture the spread of a disease with a few continuous variables linked by simple differential equations. A prominent example is the SEIR model [23, 37] with four *classes* to which people are assigned, namely, susceptible (S), exposed (E), infected (I), and recovered (R). While such models are useful for capturing macroscopic trends and also used in state-of-the-art epidemiological research, e.g., [16, 53], they often fail to capture the heterogeneity of a population. The reason is that they are condensing complex individual behaviour into few variables, thus, limiting the simulation's predictive power.

This is why finer-grained, so-called agent-based models have been developed by epidemiologists and governments to provide more accurate predictions [28]. Such simulations typically involve (1) initialising a large number of agents with a set of individual properties (e.g., location, age, etc.), (2) assigning initial infections to a small set of starter agents, (3) letting the agents interact according to a set of interaction rules (e.g., location-based, age-based, etc.), (4) spreading infections by changing the state of interacting agents, (5) calculating aggregate data such as the number of infected people at a given time and (6) running the simulation over many time steps to capture the evolution of a disease. Many such simulations with varying parameters are run in parallel (e.g., reducing interactions between agents of a certain age, capping the maximum number of allowed contacts, or setting a selected group of agents to vaccinated – to simulate the effect of various policy interventions). One of the most critical components of such models are the rules by which agents interact. Previous models used survey-based contact matrices, which provide aggregated data such as the average number of contacts between people in a certain age range [39].

While such simulations represent an important improvement over earlier work that treat all individuals the same, they still treat all members of an age group the same. Yet, it is well known that aggregate network statistics often cannot recreate the dynamics of an actual complex network graph – a fact which is also highlighted by the prominence of super-spreaders with many more than average number of contacts. Thus, rather than using aggregate and highly reduced data, the best case scenario from an epidemiological perspective would be to use the real-world contact graph between all individual members of the population which we enable with our PEM protocols in §4.2.

### 2.2 Cryptographic Building Blocks

**Oblivious Shuffling.** Mix-nets [14] were one of the first approaches to anonymous messaging. A fundamental technique underlying mix-nets is oblivious shuffling that provides unlinkability between the messages. 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. Oblivious shuffling can, e.g., be based on distributed point functions [18] or MPC [2].

**Anonymous Credentials.** Chaum [13] 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 [48] or IBM Idemix [31],

**Multi-Party Computation (MPC).** Secure MPC [57] allows multiple parties to jointly compute an arbitrary function on their private information without leaking anything but the output. Thereby, at least one and potentially more parties are assumed to be non-colluding. In the last years, MPC techniques for various security models have been introduced, extensively studied, and improved,

e.g., in [20, 21, 42]. These advancements significantly enhance the efficiency of MPC making it more and more practical for real-world applications.

**Private Information Retrieval (PIR).** PIR enables a client $C$ to retrieve one or multiple blocks of a database $DB = [b_1, ..., b_N]$ of $N$ blocks without disclosing to the server which blocks were requested. The first computational single-server PIR (cPIR) scheme was introduced by Kushilevitz and Ostrovsky [41]. Recent cPIR schemes [5, 29] use homomorphic encryption (HE). However, single-server PIR suffers from significant computation overhead since compute intensive HE operations have to be computed on each block of the $DB$ for each PIR request. In contrast, multi-server PIR relies on a non-collusion assumption between multiple PIR servers and uses only XOR operations [10, 17, 19] making it significantly more efficient than cPIR. In its simplest form, two-server information theoretic PIR [17] works as follows: A client $C$ aiming to retrieve block $b$ from a database of $N$ blocks generates two random $N$-bit queries $q_1$ and $q_2$, which only differ in the $b$-th bit, and sends $q_1$ to server $\mathcal{S}_1$ and $q_2$ to server $\mathcal{S}_2$. The servers now XOR the blocks of the database specified by the 1-bits of the queries and send the result back to $C$. Since all blocks except the $b$-th block are included either zero or two times while block $b$ is in exactly one of the two responses, $C$ can now XOR both to retrieve block $b$. Boyle et al. [10] improve the linear communication complexity of this protocol to logarithmic communication by using function secret sharing (FSS), where the client shares a function between two PIR servers which outputs '1' at the index of block $b$ and '0' otherwise.

**Garbled Cuckoo Tables.** Garbled cuckoo tables (GCT) combine garbled Bloom filters [24] with cuckoo hashing [38, 47]. Instead of storing a value $v$ in one of $k$ locations determined by $k$ hash functions as in an ordinary cuckoo table, $k$ XOR shares of $v$ are stored at the $k$ locations. We design 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.

**Trusted Execution Environments (TEE).** TEEs are hardware-assisted environments providing secure storage and execution for sensitive data and applications isolated from the normal execution environment. Data stored in a TEE is secure even if the OS is compromised. Widely adopted TEEs are Intel SGX [34] and ARM TrustZone [6] (often used on mobile platforms [45]). Using TEEs for private computation has been extensively investigated, e.g., [8, 46]. 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 [15, 33]. However, side-channel and cross-layer attacks [11, 22] show that TEEs can be broken with some effort.

## 3 Threshold-PIR-SUM

For one of our PEM protocols in §4.2, we need a protocol where a client can privately retrieve the sum of $\tau$ *distinct* entries (corresponding to the infection likelihood of $\tau$ encounters). For this, we extend multi-server PIR to (a) privately

retrieve the sum of $\tau$ entries and to (b) ensure that a client retrieves $\tau$ *distinct* entries. Combining both yields the *sum* of $\tau$ *distinct* entries.

**PIR-SUM.** Our first construction (depicted in Fig. 3 in App. A) enables a client $C$ to query $\tau$ blocks $(b_{\text{idx}_1}, ..., b_{\text{idx}_\tau})$ of a database $DB$ containing $N$ blocks, where $\{\text{idx}_1, ..., \text{idx}_\tau\}$ are the indices of the $\tau$ blocks of $DB$ freely chosen by the client, and retrieve only the sum of these blocks. We assume that all arithmetic operations are performed in the ring $\mathbb{Z}_{2^\ell}$.

Prior to any communication with the clients, the $K$ PIR servers agree on a randomly chosen secret $s_{\text{PIR}}$. For $\tau$ PIR queries from a client, the servers then derive $\tau$ fresh pseudo-random *arithmetic* shares from $s_{\text{PIR}}$ that sum up to 0, i.e., $r_1 + ... + r_\tau \mod 2^\ell = 0$, where $\ell$ is the respective bit length.

For the $j$-th query of the $\tau$ queries from client $C$, the PIR servers add the same arithmetic share $r_j$ to each element of $DB$, i.e., $b'_i = b_i + r_j \mod 2^\ell$ where $i \in [1, N]$, and compute the PIR response in the usual way. Since the client first needs to XOR the retrieved PIR responses from all PIR servers, the arithmetic blinding must be applied to the database blocks and cannot be applied to the PIR responses (since XORing the responses would then impair the arithmetic sharing). After performing $\tau$ PIR requests, the client receives $\tau$ arithmetically blinded blocks $b'_{\text{idx}_1}, ..., b'_{\text{idx}_\tau}$ where $b'_{\text{idx}_j} = b_{\text{idx}_j} + r_j \mod 2^\ell$ for $j \in [1, \tau]$. When summing up the blinded blocks, the shares added by the servers cancel out and the client retrieves the sum of database entries: $b_{\text{sum}} = \sum_{j=1}^{\tau} b'_{\text{idx}_j} = \sum_{j=1}^{\tau} (b_{\text{idx}_j} + r_j) = \sum_{j=1}^{\tau} b_{\text{idx}_j} \mod 2^\ell$. However, this alone does not guarantee that a client requests $\tau$ *different* blocks. Without any further measures, a client could query the same block in each of the $\tau$ PIR requests and divide the sum by $\tau$ to receive block $\tau$ in plain.

**Threshold-PIR.** With our second construction, we force the client to request $\tau$ *distinct* blocks from the database.

After receiving all PIR queries in one simulation step, each PIR server locally $XOR$es the queries received from the same client. Then, the PIR servers engage in a MPC protocol (cf. §2.2) that privately $XOR$s the locally $XOR$ed queries for each client, computes the Hamming weight of the result, and checks that it is equal to $\tau$. This ensures that exactly $\tau$ different entries of the DB are queried by the client. In our performance evaluation in §5.2, we show that this verification is very efficient. It is run once per client and takes only 9.88 min with two PIR servers for a PIR database with $10^7$ entries.

## 4 Privacy-preserving Epidemiological Modeling (PEM)

Contact tracing apps detect physical contacts to inform people about potential infections. If this information was combined at a central place, a full contact graph could be built for epidemiological simulations. However, contact data is highly sensitive information that should not be shared. A relatively efficient option to realize privacy-preserving epidemiological modeling (PEM) would be that each participant (i.e., each device using the contact tracing app) secret shares its contact information between non-colluding servers that can then jointly run

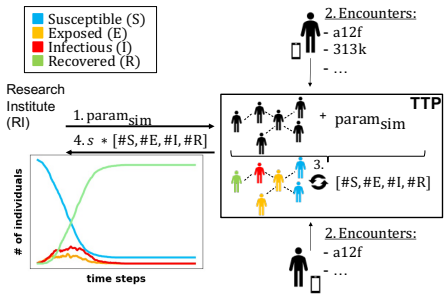


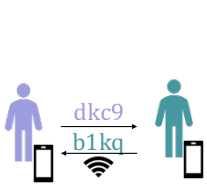

Fig. 1: Ideal functionality of private epidemiological modeling (PEM).

simulations using MPC (cf. §2.2). Even though such a non-collusion assumption is standard in the crypto community, the general public in some countries could struggle to trust a system where all contact information is disclosed if the servers collude. Hence, our system aims at distributing the trust by involving all participants such that they can keep their own contact information locally.

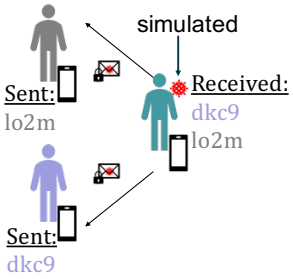
**Ideal Functionality.** Fig. 1 gives the ideal functionality of PEM with a trusted third party (TTP). A research institute (RI) sends simulation parameters $\mathsf{param}_{\mathsf{sim}}$ (e.g., initial class, characteristics of the disease, and activated control measures) to the TTP (Step 1). In Step 2, the clients send so-called encounter tokens to the TTP. These encounter tokens were exchanged via Bluetooth LE (similar to what is done in existing contact tracing apps) during a *collection phase* in the real-world when two people have a contact (cf. Fig. 2a). The TTP then reconstructs the contact graph of the participants and runs the *simulation phase*, i.e., epidemiological simulation for $s$ time steps (Step 3). It sends the aggregated number of participants per class (e.g., $[\#S, \#E, \#I, \#R]$) for each simulated time step to the RI (Step 4).

**Problem Statement.** To realize private epidemiological modeling on a real contact graph, the simulation phase must ideally be realized anonymously and distributedly. Thereby, each participant with its device represents an agent in the terminology of §2.1. To start a simulation, each participant is assigned to a class (e.g., S,E,I,R for the SEIR model). Then, in each simulation step, the user anonymously sends an encrypted message to its contacts via the Internet using the anonymous encounter tokens exchanged in the collection phase (cf. Fig. 2b) as address. These messages encode the likelihood of having infected the other contact during the encounter, a value we call *encounter transmission intensity (ETI)*. The ETI can, e.g., be a binary variable (infected/not infected) or a continuous variable based on several parameters, e.g., distance and duration of the encounter, transmission likelihood, etc. that is designed by epidemiologists. Location- or time-based filters that model containment measures can be integrated using the date, time, and coordinates of the respective encounter stored as metadata on the device. Then, as shown in Fig. 2c, each participant decrypts and aggregates the ETI it received and updates its simulated class accordingly.

**Security Model.** We assume that the participants in general follow the protocol (i.e., semi-honest security) to achieve correctness of the simulation result, but




(a) Collection Phase: Encounter token exchange in a physical encounter.




(b) Simulation phase: ETI sent by one participant in one simulation step.


(c) Simulation phase: Updating simulation class in a simulation step using total ETI from received messages.

Fig. 2: System Overview of privacy-preserving epidemiological modeling (PEM).

seek to learn further information about the other participants. This assumption is reasonable as, given the massive restrictions many people are facing because of COVID-19 nowadays, many people will be highly motivated to contribute to a successful epidemiological study, that determines which containment measures are most promising. However, considering the large number of participants involved, it is likely that few people might behave deliberately dishonestly and provide a wrong input to the experiment. But since the users can only affect people they were in contact with in the "real world", they cannot easily perform a large-scale attack to change sufficiently many inputs to alter the simulation result. We experimentally evaluate the effect of manipulations by a small fraction of the participants in App. B.

We also assume all involved servers to be semi-honest and to not collude. However, we discuss an attack by a single malicious server in §4.2 as it was also discussed in the context of contact tracing. The servers could, e.g., be run by governmental institutions and non-profit organisations like the EFF that are concerned with data security and privacy.

**Privacy Requirements.** The privacy requirements of PEM follow from the ideal functionality (cf. p.7), but for clarity we explicitly point them out:
*Encounter Anonymity:* PEM must ensure that participants do not learn to whom they unconsciously had contact.
*Re-Identification:* Participants must not learn if they unconsciously had contact with the same person twice. This includes *Infection Obscurity:* Exposed participants must not learn who of their encounters infected them (in the simulation).
*Contact Graph Privacy:* (a) w.r.t. *participants*: Participants must not be able to infer anything about contacts of other participants. The only exception is that exposed participants can infer that someone of their contacts must have been in contact with a (simulated) infectious participant. As this is only a simulated infection, we consider this as an acceptable leakage to improve the efficiency of our protocols; (b) w.r.t. *simulation servers/RI*: Servers in our system and the RI must not learn anything about the contact graphs.

### 4.1 PEM Attack Scenarios

Next, we describe two non-trivial attacks on the privacy of the contact graph that we mitigate in our PEM protocols.

**Linking Identities Attack.** When two participants Alice and Bob have met during the collection phase, Alice (anonymously) sends Bob messages containing the simulated infection risk (ETI) she exposed Bob to in the respective encounter in the simulation phase based on her current simulated infection class (and vice-versa). When Bob now suspects that two different physical encounters were with the same person, Alice, he can verify this as the messages of these encounters correlate with non-negligible probability. Concretely, Bob is even able to check correlations for all received messages to detect which messages are likely to be from the same person. In this case, Bob receives Alice's infection likelihood represented by the ETI Alice emits twice — once for each individual encounter. As simulated "infections" will appear only for the minority of contacts, having two/multiple encounters that are sending the same ETI in every simulation always at the same simulation steps makes it likely that these encounters have been with the same participant or closely related participants (e.g., flatmates). The problem persists if the ETI is the output of a function with inputs like the encounter's duration and distance. In this case, Bob might receive different ETIs from the same Alice for two different encounters, but they are still correlated and Bob knows the function for determining the ETI and its inputs such that he is able to detect correlations. In order to mitigate this attack, Bob must not learn the infection class of individual encounters, but should still be able to determine his own risk of exposure. Thus, our protocols in §4.2 obliviously sum up the messages from *all* encounters of Bob.

**Sybil Attack.** Because of the *Linking Identities Attack*, our protocols allow Bob to only receive the sum of his messages. This includes that he cannot change the set of requested messages in successive simulations to infer differences between the received sums. However, without further precautions, Bob could create multiple identities. Thus, for each of his identities he would then learn a sum which is equal to the ETI of one of his encounters. Note that this requires Bob to meet only a single Alice and no one else in the collection phase with each identity. With such a sybil attack [25] with $n$ identities, Bob would be able to receive $n$ individual messages. Trivially limiting the minimum number of encounters to $enc_{min}$ does not work: Bob could simply simulate $enc_{min} - 1$ additional encounters with identities created by himself and extract the single valid message obtained from the real encounter. To protect against such sybil attacks, i.e., to prevent an adversary from creating multiple identities, we use anonymous credentials that increase the costs to create (fake) identities: This can, e.g., be realized in a closed ecosystem like a company where each member gets exactly one token to join the simulation. This may also be deployed at a greater scale on a country level where every citizen receives a token coupled with a digital ID card. A client Bob can only receive his update for the infection class if he authenticates himself with his anonymous credentials.

### 4.2 PEM Protocols

We describe our two PEM protocols next. Let us assume two participants, Alice and Bob, who have met during the collection phase and are now taking part in the simulation phase. Alice acts as the sender of a message $m_{i,j}$, where $i$ is Bob's local id for this encounters and $j$ is the number of the current simulation. The message $m_{i,j}$ is the ETI of Alice during the encounter given her current (simulated) infection class, encounter distance, duration, etc. Bob receives the sum of the ETIs from his encounters in simulation step (including Alice's message) such that he can estimate if he got exposed and possibly update his infection class. For the sake of clarity, we do not discuss the analogous steps where Bob sends a message with his ETI to Alice, and all other participants that had encounters with Alice/Bob in the collection phase also send messages to them (and vice versa). Sending the messages between Alice and Bob is realized using central servers that must not learn anything about the contact graph as defined in the privacy requirements (cf. p. 8). The handling of dropouts is discussed in App. B.
*Mix-nets for Sender-Anonymity.* The messages encoding ETIs include the encounter token that can be used by the receiver (i.e., the contact) to download the message. If they are directly uploaded to a central server, the server could connect between participants and encounter tokens that were sent over Bluetooth (either collected by the server or by participants who collaborate with the server) and later re-identify participants that upload messages together with these tokens. Thus, a dishonest server could easily track participants. To thwart this attack, we use a mix-net to obliviously shuffle all messages. Then, the participants obliviously download the (shuffled) messages they are interested in.
*Hiding the number of encounters.* A participant can prevent the central servers in our PEM protocols from learning the number of encounters by sending "dummy" messages indicating no risk of infection to itself.
**TEE-PEM.** In our first protocol, called TEE-PEM, we assume that the mobile device of each participant is equipped with a TEE such as ARM TrustZone (cf. §2.2). Note that this distributes trust over multiple TEEs which is much more robust than a single TEE holding the complete contact graph (cf. §4) that would be an attractive target for attacks.

For every time step in the collection phase, e.g., a day, Bob's TEE uploads a list of freshly created public keys $PK_i^B$, $i \in [1, e]$ ($e$ is the number of max. possible encounters per simulation step) to the exit node of the mix-net that returns signatures for each of them. The exit node only signs the public keys if it can anonymously verify that it is directly communicating with a TEE via a secure channel. During an encounter, Bob then sends Alice a fresh (unused) public key together with the corresponding signature provided by the exit node via Bluetooth LE. By verifying the signature, Alice can convince herself that the public key was indeed created by a TEE. This concludes the collection phase. In the simulation phase in each simulation step, Alice's TEE encrypts her message $m_{i,j}$ with Bob's public key $PK_i^B$ that she received in the collection phase in encounter $i$. Then, the TEE sends the ciphertext and the public key to the mix-net. The mix-net shuffles all messages of this simulation step and forwards

them to the exit node. The exit node verifies that $PK_i^B$ is only used once per simulation. Next, the exit node sends all $c_{i,j}$ (where $j$ is the number of the current simulation) to Bob's TEE via a secure channel using the $PK_i^B$ from Step 1a to determine Bob as the receiver. Bob's TEE decrypts and sums all ETIs up. As Bob committed to his public keys before the collection phase, he cannot lie about his encounters. Thus, he cannot infer correlations between encounters (cf. §4.1). Based on the sum of all ETIs sent to him, Bob's new infection class is determined in the TEE.

*Corrupted TEE.* If Bob's TEE gets corrupted, he can access individual received messages in plain, i.e., there will not be any protection against the Linking Identities Attack (cf. §4.1). However, this problem occurs only locally (only Bob's contacts are affected). If Alice's TEE is corrupted, she can freely manipulate her messages without any restriction. Without the corruption Alice is only able to manipulate the information originating from her Bluetooth interface once, but these information are then fixed for all simulations.

*Malicious Exit Node.* In TEE-PEM, the exit node learns the public keys corresponding to the messages sent to Bob s.t. it is able to track Bob with the help of a colluding participant who previously received Bob's respective public key via BLE and, hence, know the location. Moreover, the exit node could install Bluetooth sniffers to establish a mass surveillance system. This attack requires to put physical devices in place and might be considered unrealistic if the exit node is run by a reasonably trustworthy governmental institution. Our protocol presented next thwarts this attack.

**PIR-PEM.** In our second protocol, called PIR-PEM, we use our novel Threshold-PIR-SUM protocol from §3 such that a malicious exit node cannot track participants. To run PIR with address identifiers, we adapt the approach of [4] that is based on Cuckoo hashing, but switch to an AGCT (cf. §2.2) to ensure that each participant knows the exact locations of the messages sent to him.

In PIR-PEM, Bob creates $e$ ($e$ is the number of max. possible encounters per simulation step) random values $r_i^B$. In the $i$-th encounter in the collection phase, Bob sends $r_i^B$ to Alice. Alice uses the random value $r_i^B$ to blind her message for Bob $c_{i,j} = m_{i,j} + H(r_i^B||j) \mod 2^\ell$, where $H$ is a cryptographic hash function and $j$ is the number of the current simulation. She sends the ciphertext $c_{i,j}$ and the address $a_i = H(r_i^B)$ to the mix-net. After all clients uploaded their messages, the mix-net obliviously shuffles them. The PIR servers (which can be equal to the exit nodes of the mix-net) insert the shuffled $c_{i,j}$ concatenated with $a_i$ into an arithmetic garbled cuckoo table (AGCT, cf. §2.2) using $a_i$ as insertion key. Once for all simulations with the same contacts, Bob sends $2\gamma$ PIR queries for our Threshold-PIR-SUM protocol (cf. §3) to each of the PIR servers, where $\tau = \gamma$ is the number of encounters Bob had (potentially including "dummy" messages for hiding the real number of encounters). The PIR queries privately request the two entries of the AGCT determined by $a_i$ of each encounter. The clients' PIR queries for a simulation step are cached by the PIR servers and re-used in all later simulations. This not only drastically improves communication in all later simulations, but also ensures that Bob cannot request different entries of

the AGCT in different simulations (thwarting the linkage of encounters by small variations between the requested sets, cf. §4.1). Additionally, the MPC protocol for the Threshold-PIR in Threshold-PIR-SUM must also be executed only once for each simulation step for all simulations with the same collection phase data. In the next step, Bob retrieves the PIR responses from the PIR servers, extracts the ciphertexts, and sums them up to get the sum of the blinded messages. Then, he unblinds it $M_j = C_j - \sum H(r_i^B||j) \mod 2^\ell$ to receive the total ETI. Note that he can use the concatenated addresses $a_i$ to determine which of his encounters participated in the simulation and, thus, which hashes he needs for unblinding the total ETI (cf. §B).

*Manipulated Queries.* If Bob met less than $\gamma$ participants but requests $\gamma$ messages, he is still forced to retrieve the sum of exactly $\gamma$ distinct messages. However, he cannot unblind the sum as at least one messages was not blinded with one of his $r_i^B$ or is just random. If Bob met more than $\gamma$ participants, he still only learns the sum of $\gamma$ messages. Hence, this is equivalent to not meeting the not included participants. Note that Bob commits to the number $\gamma$ for *all* simulations. Thus, he cannot change $\gamma$ to infer information from differences between simulation results.

*Having only one encounter.* If Bob had only one physical encounter, the messages he would receive in the simulation phase of PIR-PEM would reveal if his contact was *simulated* infected by design. This is not avoidable as long as the sum of ETIs is revealed to Bob. However, as pointed out before, the infection class of a participant is not sensitive information as it is only simulated. Furthermore, having only the message from one contact does not allow Bob to infer correlations. Nonetheless, we point out that in combination with additional external information (e.g., about which groups of the population were initialized as infected), a message could reveal something about the relationship of the participant to this group. We consider this edge case to be acceptable and point out that it can be avoided using TEEs as in TEE-PEM.

*Non collusion assumption.* We argued in §4 that doing PEM with MPC does not generate sufficient trust in the population as colluding servers would leak the whole contact graph. Our PIR-PEM protocols leaks much less information if the servers collude: Colluding PIR servers would learn which blocks were requested by Bob, but they could not extract the ETI per encounter as they cannot unblind the message. Hence, colluding servers cannot detect correlations between participants' encounters limiting the information leakage to knowing which messages were requested in a batch.

**Result Aggregation.** After each simulation step, the total number of participants per infection class (e.g., $(\#S, \#E, \#I, \#R)$, cf. §2.1) is computed via secure aggregation and revealed to the research institute (cf. §4). Secure aggregation is a standard problem in privacy-preserving smart metering [26, 40] and there are several solutions, e.g., using TEE, or a semi-trusted server aggregating HE ciphertexts, or $N$ non-colluding servers that aggregate secret shares.

# 5 Evaluation

In this section, we evaluate our PEM protocols w.r.t to privacy and communication costs, and give microbenchmarks for the runtimes of their main components.

## 5.1 Privacy Discussion

We show that our PEM protocols in §4.2 achieve the privacy requirements (p. 8).
*Participants:* All messages are sent anonymously using a mix-net s.t. no participant learns anything about the identity of unconscious contacts (achieving *encounter anonymity*). In TEE-PEM, the simulated infection class is securely updated in a TEE offering protection against *re-identification* and achieving *contact graph privacy* w.r.t participants. In PIR-PEM, clients can infer their recent class and the sum of ETI, i.e., they learn that one of their contacts must have been infected when they get exposed. This also yields protection against *re-identification* and achieves *contact graph privacy* w.r.t participants.
*Servers:* Our protocols involve mix servers and the PIR servers. In TEE-PEM (§4.2), the non-colluding mix servers shuffle the encrypted messages and PKs that are then recevied by the exit node. Using an anonymous communication channel such as Tor, the identity of the participants can be hidden from the incoming mix servers and outgoing mix exit node. The only accessible information are the PKs available to the exit node. As these are unique per encounter, they do not leak any information about the contact graph. Additionally, using dummy messages as described in §4.2, the number of encounters can be hidden. Hence, TEE-PEM provides *contact graph privacy* w.r.t. the simulation servers. In PIR-PEM (§4.2), the same argumentation as before applies. However, a set of PIR servers replaces the mix exit node. As PIR guarantees that participants can download blocks of the database while hiding which blocks were requested, PIR-PEM provides *contact graph privacy* w.r.t. the simulation servers.
*RI:* The RI provides the simulation parameters and receives the simulation results. Thus, it is not involved in the simulation and does not learn anything beyond the output s.t. our protocols provide *contact graph privacy* w.r.t. the RI.

## 5.2 Performance

With our work, we want to initiate the discussion on PEM protocols and present two promising protocols that have feasible communication and computation costs. Similar to existing contact tracing apps, a fully-fledged implementation would require to team-up with industry partners to implement a real-world scalable system for deployment on a country level. To demonstrate the practicality of our PEM protocols, we give communication costs and provide microbenchmarks for their main components. Note that we do not aim at creating the most efficient instantiation. Surely more optimizations can further improve efficiency and our protocols can be heavily parallelized with many servers. Instead, our goal here is to show the practicality of our PEM protocols for a large scale deployment.

Table 1: Client communication per simulation step. Beginning with the second simulation $j \in [2, ..., J]$, the upload communication of PIR-PEM drastically reduces due to the caching of client requests (cf. §5.2).

| | | Per Client | | |
|---|---|---|---|---|
| | Users $u$ | Upload | Upload | Download |
| Simulation $j$ | | 1 | $2, ..., J$ | $1, ..., J$ |
| TEE-PEM (§4.2) | $100K$ | 50.00 KiB | 50.00 KiB | 25.00 KiB |
| PIR-PEM (§4.2) | | 287.53 KiB | 2.32 KiB | 1.56 KiB |
| TEE-PEM (§4.2) | $1M$ | 50.00 KiB | 50.00 KiB | 25.00 KiB |
| PIR-PEM (§4.2) | | 339.48 KiB | 2.36 KiB | 1.56 KiB |
| TEE-PEM (§4.2) | $10M$ | 50.00 KiB | 50.00 KiB | 25.00 KiB |
| PIR-PEM (§4.2) | | 391.42 KiB | 2.40 KiB | 1.56 KiB |

Since simulation is decoupled from the collection phase, the simulations can ideally be done overnight when mobile phones are charging and have access to a high-bandwidth wireless LAN connection. Studies [55, 56] show that sleeping habits across various countries offer a time window of several hours in every night that can be used for this purpose.

**Instantiation.** We instantiate our protocols for 128 bit security. For TEE-PEM, we use RSA-2048 as encryption scheme because Android offers a hardware-backed implementation. For PIR-PEM, we assume the use of FSS-PIR [10, 35] as baseline PIR scheme and two (non-colluding) PIR servers. In both protocols, addresses are hashed with SHA-256 and trimmed to $40 - 1 + log_2(\epsilon \cdot u)$ bits [52], where $u$ is the number of participants and $\epsilon$ is the average number of encounters per client per simulation step. For simplicity, we omit the overhead of remote attestation and anonymous credentials. A typical simulation step would be a day such that 14 simulation steps can simulate two weeks. For the communication costs and benchmarks, we assume that a participant has on average $\epsilon = 100$ encounters per simulation step, which is, e.g., realistic for a day. To avoid cycles when inserting $n$ messages into the AGCT, we set its size to $10n$. This can be further improved to $2.4n$ with negligible collision probability [50–52].

**Communication.** Tab. 1 shows the client communication per simulation step for $\{100K, 1M, 10M\}$ clients. Tab. 2 contains the respective communication costs per server (i.e., the MIX/PIR-servers and the exit node). In PIR-PEM, the upload communication is drastically reduced from the second simulation on, as clients' queries are re-used for later simulations with the same physical encounters. In the first, most expensive simulation with 10M clients, each client has to upload about 400 KiB per simulation step. In later simulations, the upload is reduced to 2.4 KiB per simulation step. TEE-PEM requires to upload 50 KiB per client. The PIR servers hold about 23 GiB in PIR-PEM with 10M clients while the exit node in TEE-PEM has to handle about 500 GiBs.[4]

[4] The DB size and communication costs of PIR-PEM in the first round can be reduced by optimizing the database size to $2.4n$ instead of $10n$ [50–52]. The upload in that round is then less than 100 KiB for 10M clients.

Table 2: Server communication per simulation step.

| | | | Per Server | | | |
|---|---|---|---|---|---|---|
| | Users $u$ | mix-net DB | Exit/PIR DB | from clients | from clients | to clients |
| Simulation $j$ | | $1, ..., J$ | $1, ..., J$ | 1 | $2, ..., J$ | $2, ..., J$ |
| TEE-PEM (§4.2) | $100K$ | 0.22 GiB | 2.21 Gib | 4.77 GiB | 4.77 GiB | 2.38 GiB |
| PIR-PEM (§4.2) | | 0.22 GiB | 0.52 GiB | 0.37 GiB | 0.22 GiB | 0.15 GiB |
| TEE-PEM (§4.2) | $1M$ | 47.68 GiB | 23.84 GiB | 47.68 GiB | 47.68 GiB | 23.84 GiB |
| PIR-PEM (§4.2) | | 2.25 GiB | 22.54 GiB | 3.74 GiB | 2.25 GiB | 1.49 GiB |
| TEE-PEM (§4.2) | $10M$ | 476.84 GiB | 238.42 GiB | 476.84 GiB | 476.84 GiB | 238.42 GiB |
| PIR-PEM (§4.2) | | 22.92 GiB | 229.22 GiB | 37.82 GiB | 22.92 GiB | 14.90 GiB |

**Runtimes.** To assess computation costs, we run microbenchmarks (averaged over 10 runs) for $u = \{10^3, 10^4, 10^5\}$ users on each component of TEE-PEM/PIR-PEM such that runtimes for more users can be interpolated.

*Setup and Hardware.* We run the benchmarks on the server-side with two PIR servers 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 [1], we use hardware-backed crypto operations already implemented by Android instead. We use the code of [35] to instantiate FSS-PIR and implement the Threshold-PIR (cf. §3) using the MPC framework ABY [21]. As the code of [35] uses Intel SSE intrinsics that are not available on smartphones, we benchmarked all PIR components on the servers and multiplied the runtime by $1000\times$ as safe estimate of the runtime on Android. We implement the AGCT in C++ and follow previous work on cuckoo hashing [52] by using tabulation hashing for the hash functions.

*Runtimes.* We approximate the runtimes for mixing the uploaded data using state-of-the-art benchmark results in Blinder [2]. Each client uploads in each simulation step $\epsilon \cdot u = 100u$ messages of size $2 \cdot 2048$ bits (TEE-PEM) or $128 + 40 - 1 + \log_2(\epsilon u) = 167 + \log_2(100u)$ bits (PIR-PEM) each. Blinder can mix $10^4$ 1KB messages with 5 non-colluding servers in about 16 seconds with a GPU. $10^4/10^5/10^6$ messages with 160Bits are mixed in about 0.2/0.5/8 minutes. Based on these results, we generously approximate the runtimes for mixing our $10^5/10^6/10^7$ messages in Tab. 3. As Blinder offers security against abort which is a stronger property than needed, efficiency can be further improved.

Tab. 3 shows the approximated server runtimes for mixing $100u$ messages from $u$ users with Blinder [2] next to the results of our microbenchmarks for inserting $\epsilon \cdot u = 100u$ messages into the AGCT, running a Threshold-PIR check of one client who sent $\epsilon = 100$ PIR queries, and calculating one PIR response. We run our experiments for $u = \{1K, 10K, 100K\}$ participants such that runtimes for more participants can also be interpolated.[5] Mixing $10^7$ messages for $100K$ users would take about 1.7h, but can be made more efficient by using a mix-

[5] The largest survey-based dataset from 2017/8 for epidemiological modeling has about 40 000 participants and is $4\times$ larger than the previous standard dataset from 2005/6 [39]. This also shows that such dataset are often not up-to-date.

Table 3: Approximation of server runtimes for mixing all messages based on Blinder [2] and microbenchmark results for inserting all messages into the AGCT, running one Threshold-Check per client, and creating one PIR response.

| Users $u$ | Per Server | | Per Client | |
|---|---|---|---|---|
| | Mix | Insertions AGCT | Threshold-PIR (cf. §3) | Response |
| Simulation $j$ | $1, ..., J$ | $1, ..., J$ | 1 | $1, ..., J$ |
| $1K$ | 40 s | 10.49 ms | 5.31 s | 0.90 s |
| $10K$ | 10 min | 0.11 s | 1.26 min | 13.09 s |
| $100K$ | 1.7 h | 1.53 s | 9.88 min | 1.85 min |

Table 4: Average client runtimes per simulation step: Creation of $\epsilon = 100$ messages, $2\epsilon = 200$ PIR queries, and extraction of the total ETI from the PIR responses. They are not affected by the number of clients.

| | Messages creation | PIR queries | ETI extraction |
|---|---|---|---|
| Simulation $j$ | $1, ..., J$ | 1 | $1, ..., J$ |
| TEE-PEM (§4.2) | 0.08 s | - | 3.04 s |
| PIR-PEM (§4.2) | 1.50 ms | 0.49 s | 4.90 s |

ing technique with weaker security guarantees than Blinder and parallelization as discussed before. Threshold-PIR must only be performed once per client as queries are re-used for later simulations. All other operations only take a few seconds. Additionally, parallelization using more servers is possible.

Tab. 4 gives our microbenchmark results for the client-side for an average of $\epsilon = 100$ encounters. As the number of participants $u$ taking part in the simulation does not change the average number of encounters a person has per time step in reality, the runtimes on the client side are not affected by $u$. As shown, each client has only a few seconds of computation per simulation step.

Based on our microbenchmarks, an entire PIR-PEM simulation with 14 simulation steps (i.e., 14 days) and $u = 100K$ clients would take about 59 657 hours with two servers. Thus, even without further optimizations, the whole simulation could be run on about 2 000 servers (or on 240 servers for $u = 10K$ users) during 6 hours in one night. In comparison, the bottleneck of TEE-PEM is only the mixing, resulting in about 24 hours or 8 servers for doing one simulation with $u = 100K$ users within 6 hours.

## 6 Conclusion

In this work, we introduced the problem of privacy-preserving epidemiological modeling (PEM) which enables to privately simulate the spread of a disease on real contact graphs collected on mobile devices. We introduced two PEM protocols based on different trust assumptions and showed that they have practical performance. As building block of independent interest, we designed a PIR-based protocol for privately querying the sum of multiple distinct blocks. Our protocol uses MPC to verify properties of a multi-server PIR query. We hope to motivate further research towards highly efficient PEM.

## References


1. Third-party Trusty applications, https://source.android.com/security/trusty#third-party_trusty_applications
2. Abraham, I., Pinkas, B., Yanai, A.: Blinder – Scalable, Robust Anonymous Committed Broadcast. In: CCS (2020)
3. Adam, D.: Special report: The simulations driving the world's response to COVID-19. Nature (2020)
4. Ali, A., Lepoint, T., Patel, S., Raykova, M., Schoppmann, P., Seth, K., Yeo, K.: Communication-computation trade-offs in PIR. (2019), https://eprint.iacr.org/2019/1483.pdf
5. Angel, S., Chen, H., Laine, K., Setty, S.: PIR with compressed queries and amortized query processing. In: S&P (2018)
6. ARM: ARM security technology building a secure system using TrustZone technology (2009), https://developer.arm.com/documentation/genc009492/c
7. Bampoulidis, A., Bruni, A., Helminger, L., Kales, D., Rechberger, C., Walch, R.: Privately connecting mobility to infectious diseases via applied cryptography (2020), https://eprint.iacr.org/2020/522
8. Bayerl, S.P., Frassetto, T., Jauernig, P., Riedhammer, K., Sadeghi, A.R., Schneider, T., Stapf, E., Weinert, C.: Offline model guard: Secure and private ML on mobile devices. Design, Automation & Test in Europe Conference & Exhibition (2020)
9. Biasse, J.F., Chellappan, S., Kariev, S., Khan, N., Menezes, L., Seyitoglu, E., Somboonwit, C., Yavuz, A.: Trace-$\sigma$: a privacy-preserving contact tracing app (2020), https://eprint.iacr.org/2020/792
10. Boyle, E., Gilboa, N., Ishai, Y.: Function secret sharing: Improvements and extensions. In: CCS (2016)
11. Brasser, F., Müller, U., Dmitrienko, A., Kostiainen, K., Capkun, S., Sadeghi, A.R.: Software grand exposure: SGX cache attacks are practical. In: USENIX Security (2017)
12. Chan, J., Foster, D., Gollakota, S., Horvitz, E., Jaeger, J., Kakade, S., Kohno, T., Langford, J., Larson, J., Sharma, P., Singanamalla, S., Sunshine, J., Tessaro, S.: PACT: Privacy sensitive protocols and mechanisms for mobile contact tracing (2020), https://arxiv.org/pdf/2004.03544.pdf
13. Chaum, D.: Security without identification: Transaction systems to make big brother obsolete. In: Communications of the ACM (1985)
14. Chaum, D.L.: Untraceable electronic mail, return addresses, and digital pseudonyms. Communications of the ACM (1981)
15. Chen, G., Zhang, Y., Lai, T.H.: OPERA: Open Remote Attestation for Intel's Secure Enclaves. In: CCS (2019)
16. Chen, Y.C., Lu, P.E., Chang, C.S., Liu, T.H.: A time-dependent sir model for covid-19 with undetectable infected persons. Transactions on Network Science and Engineering (2020)
17. Chor, B., Goldreich, O., Kushilevitz, E., Sudan, M.: Private information retrieval. In: Annual Foundations of Computer Science (1995)
18. Corrigan-Gibbs, H., Boneh, D., Mazières, D.: Riposte: An anonymous messaging system handling millions of users. In: S&P (2015)
19. Corrigan-Gibbs, H., Kogan, D.: Private information retrieval with sublinear online time. In: EUROCRYPT (2020)
20. Damgård, I., Keller, M., Larraia, E., Pastro, V., Scholl, P., Smart, N.P.: Practical covertly secure MPC for dishonest majority – or: Breaking the SPDZ limits. In: ESORICS (2013)

21. Demmler, D., Schneider, T., Zohner, M.: ABY - A Framework for Efficient Mixed-Protocol Secure Two-Party Computation. In: NDSS (2015)
22. Dessouky, G., Gens, D., Haney, P., Persyn, G., Kanuparthi, A., Khattri, H., Fung, J.M., Sadeghi, A.R., Rajendran, J.: Hardfails: Insights into software-exploitable hardware bugs. In: USENIX Security (2019)
23. Diekmann, O., Heesterbeek, H., Britton, T.: Mathematical tools for understanding infectious disease dynamics. Princeton University Press (2012)
24. Dong, C., Chen, L., Wen, Z.: When private set intersection meets big data: an efficient and scalable protocol. In: CCS (2013)
25. Douceur, J.R.: The sybil attack. In: International Workshop on Peer-to-Peer Systems (2002)
26. Erkin, Z., Troncoso-pastoriza, J.R., Lagendijk, R.L., Perez-Gonzalez, F.: Privacy-preserving data aggregation in smart metering systems: an overview. In: Signal Processing Magazine (2013)
27. Ferguson, N.: What would happen if a flu pandemic arose in Asia? Nature (2005)
28. Ferguson, N.M., Cummings, D.A., Fraser, C., Cajka, J.C., Cooley, P.C., Burke, D.S.: Strategies for mitigating an influenza pandemic. Nature (2006)
29. Gentry, C., Halevi, S.: Compressible FHE with applications to PIR. In: TCC (2019)
30. Government of Singapor: TraceTogether, safer together (2020), `https://www.tracetogether.gov.sg/`
31. IBM: Specification of the Identity Mixer Cryptographic Library Version 2.3.0* (2010), IBM Research Report RZ 3730
32. Inria, Fraunhofer AISEC: ROBust and privacy-presERving proximity Tracing protocol (2020), `https://github.com/ROBERT-proximity-tracing/documents`
33. Intel: Attestation Service for Intel® Software Guard Extensions, `https://api.trustedservices.intel.com/documents/sgx-attestation-api-spec.pdf`
34. Intel: Intel® Software Guard Extensions Programming Reference (2014), `https://software.intel.com/sites/default/files/managed/48/88/329298-002.pdf`
35. Kales, D., Omolola, O., Ramacher, S.: Revisiting user privacy for certificate transparency. In: EuroS&P (2019), Code: `https://github.com/dkales/dpf-cpp`
36. Kales, D., Rechberger, C., Schneider, T., Senker, M., Weinert, C.: Mobile private contact discovery at scale. USENIX Security (2019)
37. Kermack, W.O., McKendrick, A.G.: Contributions to the mathematical theory of epidemics—i. In: Bulletin of Mathematical Biology (1991)
38. Kirsch, A., Mitzenmacher, M., Wieder, U.: More robust hashing: Cuckoo hashing with a stash. Journal on Computing (2010)
39. Klepac, P., Kucharski, A.J., Conlan, A.J., Kissler, S., Tang, M.L., Fry, H., Gog, J.R.: Contacts in context: large-scale setting-specific social mixing matrices from the bbc pandemic project. MedRxiv (2020)
40. Kursawe, K., Danezis, G., Kohlweiss, M.: Privacy-friendly aggregation for the smart-grid. In: PETS (2011)
41. Kushilevitz, E., Ostrovsky, R.: Replication is not needed: Single database, computationally-private information retrieval. In: FOCS (1997)
42. Lindell, Y., Pinkas, B., Smart, N.P., Yanai, A.: Efficient constant round multi-party computation combining BMR and SPDZ. In: CRYPTO (2015)
43. Ministère de l'Économie, des Finances et de la Relance: StopCovid (2020), `https://www.economie.gouv.fr/stopcovid`
44. Moreno, Y., Pastor-Satorras, R., Vespignani, A.: Epidemic outbreaks in complex heterogeneous networks. The European Physical Journal B-Condensed Matter and Complex Systems (2002)

45. Ngabonziza, B., Martin, D., Bailey, A., Cho, H., Martin, S.: TrustZone explained: Architectural features and use cases. In: International Conference on Collaboration and Internet Computing (2016)
46. Ohrimenko, O., Schuster, F., Fournet, C., Mehta, A., Vaswani, K., Costa, M.: Oblivious multi-party machine learning on trusted processors. In: USENIX Security (2016)
47. Pagh, R., Rodler, F.F.: Cuckoo hashing. Journal of Algorithms (2004)
48. Paquin, C., Zaveruch, G.: U-Prove Cryptographic Specification V1.1 (Revision 3) (2013), `http://www.microsoft.com/uprove`
49. Pastor-Satorras, R., Castellano, C., Van Mieghem, P., Vespignani, A.: Epidemic processes in complex networks. In: Reviews of modern Physics (2015)
50. Pinkas, B., Rosulek, M., Trieu, N., Yanai, A.: PSI from PaXosS Fast, malicious private set intersection. In: EUROCRYPT (2020)
51. Pinkas, B., Schneider, T., Weinert, C., Wieder, U.: Efficient circuit-based PSI via cuckoo hashing. In: EUROCRYPT (2018)
52. Pinkas, B., Schneider, T., Zohner, M.: Scalable private set intersection based on OT extension. TOPS (2018)
53. Small, M., Tse, C.K.: Small world and scale free model of transmission of SARS. In: International Journal of Bifurcation and Chaos (2005)
54. Troncoso, C., Payer, M., Hubaux, J.P., Salathé, M., Larus, J., Bugnion, E., Lueks, W., Stadler, T., Pyrgelis, A., Antonioli, D., et al.: Decentralized privacy-preserving proximity tracing (2020), `https://arxiv.org/pdf/2005.12273.pdf`
55. Walch, O.J., Cochran, A., Forger, D.B.: A global quantification of "normal" sleep schedules using smartphone data. Science advances (2016)
56. Woollaston, V.: Sleeping habits of the world revealed: The US wakes up grumpy, China has the best quality shut-eye and South Africa gets up the earliest (2015), `https://www.dailymail.co.uk/sciencetech/article-3042230/Sleeping-habits-world-revealed-wakes-grumpy-China-best-quality-shut-eye-South-Africa-wakes-earliest.html`
57. Yao, A.C.C.: How to Generate and Exchange Secrets. In: FOCS (1986)
58. Yoneki, E.: Fluphone study: Virtual disease spread using haggle. In: Workshop on Challenged Networks (2011)

## A SUM-PIR

Fig. 3 shows the idea of our PIR-SUM protocol presented in §3.

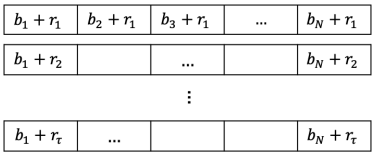



Fig. 3: Our PIR-SUM protocol. $b_i$ is the message stored in the $i$-th database block ($i \in [1, N]$). $r_j$ is the random value used in all database blocks for blinding the $j$-th response ($j \in [1, \tau]$).

## B Dropouts and Manipulations

PEM protocols must tolerate that participants drop out while the simulation is running as mobile devices regularly lose connection or run out of battery. Both our protocols (§4.2) can cope with such dropouts: In TEE-PEM, the exit node identifies all messages for Bob based on the PKs uploaded by Bob in the collection phase and forwards these to Bob. Unused PKs (corresponding to dropouts) are ignored. In PIR-PEM, we add an additional step: In each simulation step of the first simulation, Alice publishes on a bulletin board the value $H'(r_i^B)$, where $H'$ is a cryptographic hash function and $r_i^B$ is the encounter token received from Bob in encounter $i$. By downloading the entire bulletin board[6], these values allow the receiver Bob (and only him as only he and Alice know their shared encounter tokens) to determine which of his encounters take part in the simulation. He sets the number of encounters per simulation step, $\tau$, to the number of encounters that take part in the simulation and creates his PIR queries accordingly. After the first simulation, the PIR servers know for which addresses $a_i$ they should receive messages as the addresses remain constant over all simulations and they have seen them in the first simulation. If one of the messages does not arrive (because Alice dropped out), the servers insert $0 + r_i||0 \cdots 0$ into the AGCT instead of the real message $(c_{i,j} + r_i)||a_i$. If Bob downloads a message of the form $r_i||0 \cdots 0$, he can use $r_i$ to correctly unblind his total ETI.

**Experiments.** For simulating the deviations caused by dropouts and manipulated inputs, we implemented a simple plaintext SEIR model. The effects were measured by the normalized differences between the number of people assigned to a class per simulation step averaged over 1 000 runs.

First, we simulated that $\{0.1\%, 0.5\%, 1\%, 2\%, 3\%, 4\%, 5\%\}$ of 1000 clients randomly drop out per simulation step. The experiments showed that the dropout of $x\%$ clients causes on average about $0.5x\%$ of the clients to be assigned to different classes (S,E,I,R). The dropouts, however, do not change the simulation's trend at all, but create a small parallel shift. We also assessed the effect of $\{0.1\%, 0.5\%, 1\%\}$ of 1000 clients manipulating the simulation by deliberately indicating that they are infectious. The change of the number of clients per class is significantly stronger compared to the dropouts. Concretely, with $\{0.1\%, 0.5\%, 1\%\}$ of the clients doing the manipulation, the number of people per class changes by about $\{4.5\%, 14\%, 21\%\}$. While with 0.1% of the clients, the overall trend of a not manipulated simulation is preserved, the disease's spread is significantly speed up with 0.5%/1% manipulating clients. These results are transferable to the opposite case where people indicate to be always healthy. We assume that only very few people would deliberately try to harm simulations considering that it is in most people's interest to find effective containment measures. However, if a significant proportion is suspected to potentially manipulate the simulation, stronger PEM protocols than ours need to be designed.

[6] The hashes can also be trimmed to $40 - 1 + log_2(\epsilon \cdot u)$ bits [52]. The $n = \epsilon \cdot u$ hashes in the bulletin board can be compressed to $n/l + n \cdot v$ bits [36], i.e., about 42 MB for $u = 100K$ clients with tag size $t = 32$ and load factor $l = 66\%$.