PEM: Privacy-preserving Epidemiological Modeling

Marco Holz
Technical University of Darmstadt
holz@encrypto.cs.tu-darmstadt.de

Benjamin Judkewitz
Charité-Universitätsmedizin Berlin
benjamin.judkewitz@charite.de

Helen Möllering
Technical University of Darmstadt
moellering@encrypto.cs.tu-darmstadt.de

Benny Pinkas
Bar-Ilan University
benny@pinkas.net

Thomas Schneider
Technical University of Darmstadt
schneider@encrypto.cs.tu-darmstadt.de

***Abstract*—Modeling the spread of COVID-19 is crucial for any effort to manage the pandemic. However, detailed epidemiological simulations suffer from a scarcity of relevant empirical data, such as social contact graphs, because such data is inherently privacy-critical. Thus, there is an urgent need for a method to perform powerful epidemiological simulations on real-world contact graphs without disclosing privacy-critical information.**

In this work, we propose a practical framework for *privacy-preserving epidemiological modeling (PEM)* on contact information stored on mobile phones, like the ones collected by already deployed contact tracing apps. Unlike those apps, PEM allows for meaningful epidemiological simulations. This is enabled by a novel *Threshold-PIR-SUM* protocol to privately retrieve the sum of a fixed number of distinct values without revealing individual values. 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.

***Index Terms*—Decentralized Epidemiological Modeling, Privacy, Private Information Retrieval, COVID-19**

I. Introduction

Mathematical epidemiological modeling aims at predicting the spread of diseases. It can inform epidemic containment strategies and political interventions by anticipating the effects of different control measures. However, detailed epidemiological models of communicable diseases, including COVID-19, have long suffered from a lack of available social interaction data.

From a modelers 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 [61], [64], [66], [70]. 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.

Here, 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 epidemiological modeling while simultaneously protecting privacy. PEM involves simulating infections of participants, propagating *virtual* infections on the *real* contact graph [66] and analyzing the simulated spread. Many simulations may need to be run to compare the effect of different disease management scenarios (e.g., simulations including or excluding different types of contacts or assuming different likelihoods of person-to-person transmission). 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. §IV for details). 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. To thwart the spread of COVID-19, a large amount of contact tracing systems have been proposed providing different levels of privacy. These systems aim 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., COVIDSafe [24], PEPP-PT [71], StopCovid/ROBERT [48], [59], TraceTogether [44]), computations such as the generation of the exchanged identifiers are executed at a central party that may also store encounter information. In contrast, in decentralized approaches (e.g., DP-3T [76], Hashomer [72], PACT [17]), most computations are done locally at the participants' devices and information is typically only shared in the case of an infection. For more details and comparisons of different designs we refer the reader to [3], [8], [56], [77].

Beyond contact tracing, models for epidemiological diseases have been proposed that use real-world data, e.g., aggregated demographic data [80] or data extracted from social media platforms [70]. Thereby, the "lack of capacity for data collection or privacy concerns" was identified as the major challenge for using standard epidemiological models in [80].

FluPhone [83] aims at applying epidemiological models on real-world, non-aggregated contact graph data of individual participants. The project collects proximity data via Bluetooth LE and GPS location data, but, apart from user identifier anonymization, uses no privacy-preserving technologies to ensure privacy.

A system for disease hotspot detection using location data

stored at a mobile network operator or a mobile operating system manufacturer was proposed in [15]. An *Aggregated PIR* protocol correlates the mobile location data with the infected participants identified by their phone number. This aggregated PIR protocol is similar to our PIR-SUM protocol (cf. §V-A), but is based on single-server computational PIR which is substantially less efficient than our hybrid Threshold-PIR-SUM protocol (cf. §V-C).

In our work, we enable for the first time to simulate standard epidemiological models on contact information collected with mobile devices while preserving privacy and keeping efficiency in mind.

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

- In §III, we introduce privacy-preserving epidemiological modeling (PEM), its privacy requirements, its ideal functionality, and its components.
- In §IV, we provide a detailed evaluation of two non-trivial attacks on a PEM system that aim at extracting information about the contact graph.
- In §V, we introduce new protocols for variants of Private Information Retrieval (PIR): *Threshold PIR-SUM* allows a client to download the *sum* of t *distinct* blocks of a database from a server without learning the values of individual blocks and without revealing which blocks were requested. Additionally, we show how to embed it into an efficient hybrid PIR construction.
- In §VI, we introduce three PEM protocols offering different levels of trust that enable to use real-world contact data collected on the users' mobile devices. In our protocols, all data remains locally on the users' devices by distributing the simulation using cryptographic building blocks. Our three PEM protocols are based on trusted execution environments (TEE-PEM), mix-nets (MIX-PEM), and PIR (PIR-PEM), respectively. Additionally, we discuss in detail the strengths and weaknesses of our protocols.
- In §VII, we analyze privacy guarantees and complexities of our protocols and show that they are efficient and even scale up to millions of users.

The goal of our work is to extend the focus of the security and privacy research community from private contact tracing, which notifies users about potential exposures in the *past*, to privacy-preserving epidemiological modeling for predicting which implications control measures can have in the *future*.

II. Preliminaries

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

A. Epidemiological Modeling

We use the SEIR model as an example for a mathematical epidemiological compartmental model [35], [51] that is routinely used to model disease spread. The SEIR model assigns each individual of the population to one of the following four classes: Susceptible (S), Exposed (E), Infectious (I), Recovered (R). Initially, all individuals are susceptible, i.e., they have not been infected yet and are susceptible to the disease. If they meet an infectious person, they can get infected, i.e., exposed and become then infectious after an incubation time. Finally, they will recover and get healthy.

We focus on agent-based modeling (ABM) [47], [75] to capture the stochasticity of a heterogeneous population for modeling the spread of a disease in a realistic manner. In ABM, each individual of a population is modeled by a so-called agent that can have specific properties influencing its likelihood of becoming infected and its transitions between the classes. Additionally, parameters like the infectivity and recovery time can be adjusted based on the disease being modeled and newly gained insights about it from epidemiologic research. The simulation then predicts the spread of the disease in the population over several simulation steps by simulating physical interactions between the agents taking their and the disease's parameters into consideration. ABM enables flexible and detailed epidemiological simulations closely modeling a realistic population.

In the simplest implementation, the transitions of individuals between classes are simulated centrally based on collected contact data but sharing such data with a central party is not acceptable and severely infringes privacy. Our work introduces privacy-preserving protocols that enable the simulation of these transitions in a decentralized way on end-user devices and thereby keep the contact information collected on these devices private. Our protocols are *generic* w.r.t. the used compartmental model, i.e., others than SEIR can be used.

B. Cryptographic Building Blocks

Mix-Nets. Mix-nets [21] and protocols based on the dining cryptographer (DC) problem [20] were 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. Past research established a wide variety of oblivious shuffle protocols based on garbled circuits [38], [45], [79], homomorphic encryption [46], distributed point functions [1], switching networks [60], permutation matrices [55, §4.1], sorting algorithms [55, §4.2], and re-sharing [55, §4.3+4.4]. Verifiable shuffles [9], [40], [63] use zero-knowledge proofs to guarantee that the shuffled messages are a permutation of the input messages.

Anonymous Credentials. In identity management, the same identity was traditionally used for each transaction. However, this allows a service provider to link transactions to the same individual and leaks more information than necessary. For this reason, the idea of anonymous credentials was proposed by Chaum [19]. There, 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 can enable to certify specific properties like age. Several instantiations for anonymous credentials have been proposed, e.g., U-Prove [69] or Algebraic MACs [18].

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 [54]. Recent cPIR schemes [2], [5], [43] 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 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 [23], [25], [29], [30] or function secret sharing [13] making it significantly more efficient than cPIR. In its simplest form, two-server information theoretic PIR [23] works as follows: If a client C wants to retrieve block b from a database of N blocks it generates two queries q_1 and q_2 and sends q_1 to server S_1 and q_2 to server S_2. Query $q_1 \in_R \{0,1\}^N$ is chosen at random and q_2 is almost equal to q_1 but the b-th bit is flipped. The servers now XOR the blocks of the database specified by the 1-bits of the queries of length N bits 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. This protocol was optimized in RAID-PIR [29], [30].

Threshold Encryption and Secret Sharing. A threshold encryption scheme [12], [27], [32], [42] is a public-key encryption scheme where the secret key is split among X different parties. It requires at least τ (X) parties for decryption. In our Threshold-PIR (§V-B), the server instantiates a key key_{PIR} for a symmetric encryption scheme and secret shares this key with a τ-out-of-X secret sharing scheme like Shamir's secret sharing [74]. In Shamir's secret sharing, the server chooses a random polynomial f of degree τ 1 that evaluates to key_{PIR} on input 0 (i.e., $f_{SS}(0) = key_{\text{PIR}}$). Then, he provides each party $P_n \in [1, X]$ with the share $(n, f_{SS}(n))$. τ parties can jointly reconstruct the secret key_{PIR} with their shares using polynomial interpolation.

Garbled Cuckoo Tables. Garbled cuckoo tables (GCT) combine garbled bloom filters [36] with cuckoo hashing [52], [68]. In a GCT, each element is mapped to two locations using two different hash functions. Instead of simply storing the associated value v in one of the two locations (as in an ordinary cuckoo table), two XOR shares of the value v are stored at the two locations, respectively. If one of these locations is already in use, the XOR share for the other location is set to be the XOR of v and the data stored in the used location. By increasing the GCT's size, [73] guarantees that every element can be mapped to two locations to store the shares. We design a variant of GCT called *arithmethic garbled cuckoo table* (AGCT). In an AGCT, we use arithmetic sharing instead of XOR sharing. This is needed for combining the AGCT with our Threshold-PIR-SUM in §V.

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 [50] and ARM TrustZone [6] (often used on mobile platforms [65]). Using TEEs for private computation has been extensively investigated, e.g., [7], [10], [67]. 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 [4], [22], [49]. However, side-channel and cross-layer attacks [14], [33], [82] raise concerns about the security of TEEs.

III. Privacy-preserving Epidemiological Modeling

Contact tracing apps detect physical contacts to inform people about potential infections and store information about the encounters. If this information was combined at a central place, a full contact graph could be built usable for epidemiological simulations. However, contact data is highly sensitive information that must not be shared. A relatively efficient option to realize epidemiological modeling in a privacy-preserving manner 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 create the full contact graph and run simulations using multi-party computation (MPC) [31]. Even though such a non-collusion assumption is standard in the crypto community, the general public will struggle to trust a system that discloses all contact information if the servers collude. Hence, our system aims at distributing the trust by involving all participants who can keep their contact graph information completely locally and still simulate the spread of a disease by sending messages to each other in an anonymous fashion. Additionally, only an aggregated simulation result will be released to a research institute so that no data directly relating to a single identity is shared. Thus, given that a wide adoption of a PEM app in the population is needed to get a meaningful and sufficiently realistic contact graph, a distributed PEM is superior to using outsourced computation with MPC. We split PEM into two phases:

Collection phase: During a physical encounter, participants exchange data via Bluetooth LE to collect anonymous encounter information (cf. Figs. 1a and 1b). They locally store this along with additional information on the context of the encounter (for example, duration, proximity, time, and location). This information can later be used to include or exclude different encounters in the simulation phase, to model the effect of containment measures (e.g., restaurant closings, by excluding all encounters that happened in restaurants). The collection phase is independent of the simulation phase, i.e., no simulation-dependent infection information is exchanged. When PEM is integrated into a contact tracing app, the collection phase can be integrated into the normal activity

of this application recording physical contacts. (In theory, the collected information could be used to create a contact graph by matching sent/received information. But in PEM this it not possible because data remains locally on the clients' devices.)

Simulation phase: In the simulation phase, typically many (e.g., 100 or more) simulations are executed. In each time step and simulation, each participant has a current simulated infection class (e.g., susceptible, infected, etc., cf. §II-A). Based on this class, participants calculate the likelihood of having transmitted infectious agents during each encounter, a quantity we call "encounter transmission intensity" (ETI). The ETI can, for example, be a binary variable (infected/not infected) or a continuous variable proportional to the number of transmitted particles (e.g., a product of participant infectiousness, encounter proximity, encounter duration, etc.). The participant then anonymously sends the encrypted ETI via the Internet to its contacts, who calculate the probability of having been infected during the last time step, and update their infection classes accordingly. Again, to model specific control measures in different simulations, some encounters might be excluded based on their recorded context s.t. these encounters do not trigger an ETI message (modeling that they would not have happened with these control measures in place). Fig. 1c shows the network of phones modeling the contact graph that can be created using the data collected in the collection phase. The simulated spread of the disease presented in Fig. 1d can be realized in a distributed but privacy-preserving using our PEM protocols in §VI. Simulations should typically be run during the night when devices are idle and, optimally, connected to a WLAN and a power source. Studies [78], [81] show that sleeping habits in multiple countries offer a time window of several hours every night that can be used for this purpose.

Integration into Contact Tracing. Our protocols are not designed for a specific contact tracing app. Instead, we aim at providing generic PEM protocols that can be added on top of existing contact tracing apps. However, they need to be decentralized (i.e., contact information stays locally on the device) and token-based (i.e., participants exchange information during an encounter via Bluetooth LE).

A. Terminology

We first introduce the underlying terminology.

Infection Class: In this work, we use as an example for an epidemiological model the agent-based SEIR model (cf. §II-A) with four *infection classes*: Susceptible (S), Exposed (E), Infectious (I), Recovered (R). In every simulation step, each participant locally determines to which class it is assigned. For the transition $S \rightarrow E$ in a simulation, a susceptible participant (S) has to be informed by its contacts if it had contact to an infectious (I) person. Whether it becomes exposed (E) then depends on parameters like the probability of disease transmission per contact, the duration of the contact, and the physical closeness during the meeting [11]. Additional parameters can be included. We privately realize this transition $S \rightarrow E$ with our PEM protocols in §VI. The subsequent transitions $E \rightarrow I$ and $I \rightarrow R$ can be computed locally by the participant and depend on the modeling of the disease, e.g., the incubation time ($E \rightarrow I$) can correspond to z simulation steps.

Participant: A participant is a person (i.e., an agent in the terminology of epidemiological modeling, cf. §II-A) using a contact tracing app extended by PEM.

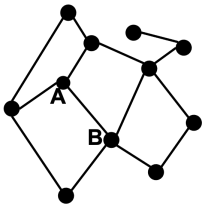
Research institute (RI): The research institute (RI) aims at investigating how different potential control measures, e.g., contact restrictions or closing/opening schools enforced by the government, would influence the spread of a disease. Furthermore, for new diseases where the infectiosity is unknown, different possible spreading behaviors of the disease can be modeled. Thus, it collects the aggregated number of participants in each class of the epidemiological model (e.g., SEIR). To start a simulation, the RI provides the simulation parameters $\mathsf{param}_{\mathsf{sim}}$ containing, e.g., an initial infection class per participant, control measures taken, the incubation time, and parameters influencing the spread of the virus.

Initialization. There are two ways how to initialize the participants' infection classes in a simulation: (a) uniform or (b) non-uniform at random. If the initialization is done uniformly at random, certain diseases' propagations cannot be modeled in a realistic manner. In contrast, when the initialization is done non-uniformly, e.g., only people from a specific part of the country are initialized exposed/infectious, the simulation outputs leak some information about the contacts of this group. We point out that this information leakage is tolerable as only the accumulated number of people per infection class is reported to the RI. However, some information about, e.g., the accumulated number of contacts, can still be inferred by observing how these numbers change. Hence, we leave the choice about how the initialization is done to epidemiologists and politics. Moreover, note that information leakage from the output does not conflict with the ideal functionality of PEM (cf. §III-D).

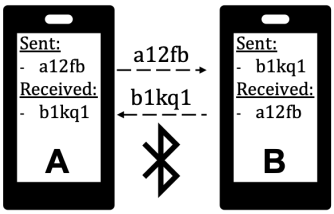
Simulation servers: These servers are used to anonymously exchange messages between the participants.

Simulation: A simulation $j \in [1, J]$, where J is the number of simulations, is a set of S simulation steps. Each simulation is instantiated with a public set of parameters $\mathsf{param}_{\mathsf{sim}}$ that determine the initial (simulated) infection class of the participants, the probability of infection, and other parameters required by each participant to privately calculate its infection class. A simulation simulates the spread of the disease over S time intervals (called simulation steps).

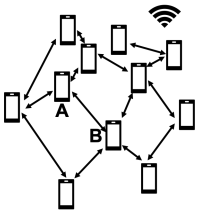
Simulation Step: In each simulation step $s \in [1, S]$, where S is the number of simulation steps in a simulation j, every participant updates its current infection class. It uses our PEM protocols from §VI to infer if it got exposed, i.e., to privately realize the $S \rightarrow E$ transition. A simulation step can, for example, simulate one day.



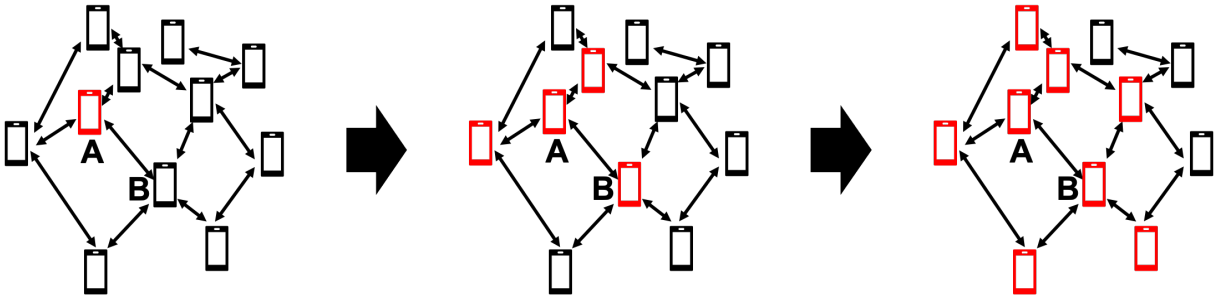
(a) Full contact graph (not known to any party).



(b) Exchange of secrets during physical encounters via BLE (similar to contact tracing apps).



(c) PEM enables privacy-preserving distributed simulation of disease spread on networks of phones via the Internet.



(d) Simulation Phase: Privacy-preserving propagation of simulated infection.

Fig. 1: PEM: System Overview (A=Alice/B=Bob).

B. Privacy Requirements

Encounter Anonymity: Participants must 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:

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. The second part cannot be avoided if the participants are able to assess their intermediate infection classes, as they will learn when they got exposed. Because this is only a simulated infection, we consider this as an acceptable leakage to improve the efficiency of our protocols. However, our protocols can be adapted to also hide this information. In TEE-PEM (§VI-A), the class is actually already securely updated in the TEE.

w.r.t. *simulation servers/RI*: Simulation servers and the RI must not learn anything about the contact graphs.

C. Security Model

In this work, we assume that all participants follow the protocol (i.e., semi-honest security) to achieve correctness of the simulation result, but seek to learn further information about the other participants. This assumption is reasonable as the participants are interested in contributing to the successful execution of the epidemiological study. It follows that they do not refuse to send (encrypted) messages about the infection likelihood to each other and do not tamper with their own (simulated) infection class. Again, we explicitly do not consider the scenario when the input data of the participants to the simulations is not reliable. To understand why it would not be a problem, let us analyse the causes and effects of unreliable data. We can think of the following two cases when the input data could be not completely correct: Firstly, considering the large number of participants involved, it is likely that a small fraction of them might behave deliberately dishonest and provide a wrong input to the experiment, e.g., a dishonest user changes its class to infected. 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. Secondly, considering a large-scale attack taking over a *large* fraction of possibly *millions* of users, e.g., by a state-level adversary, it would be detectable due to high quantities of unusual traffic and eventually strong deviations in the results.

We assume all servers to be semi-honest and the PIR servers to not collude. We discuss why this is different from running PEM with MPC in §VI-C. The servers could, for example, be run by governmental institutions and non-profit organisations like the EFF that are concerned with data security and privacy.

D. Ideal Functionality

Fig. 2 gives the ideal functionality of a simulation with a trusted third party (TTP). The research institute (RI) sends the simulation parameters $\mathsf{param}_{\mathsf{sim}}$ (e.g., initial infection classes, characteristics of the disease, and activated control measures) to the TTP (Step 1). In Step 2, the participants send the en-

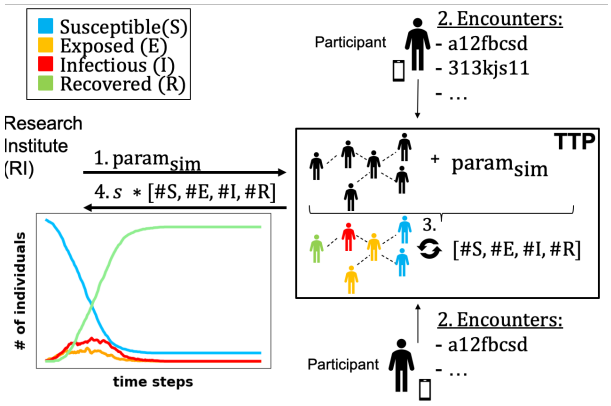


Fig. 2: Ideal functionality of private epidemiological modeling.

counter tokens received in the *collection phase*. The TTP then reconstructs the contact graph of the participants and calculates their new classes for S steps (Step 3). It sends the aggregated number of participants per class ($[\#S, \#E, \#I, \#R]$) for each simulation step s to the RI (Step 4).

A trivial solution to realize the ideal functionality would be to deploy the TTP in a TEE (cf. §II-B). However, the resource limitations of TEEs are a prohibiting factor considering the massive amount of data that needs to be handled in a large scale simulation with possibly millions of users. Additionally, as the TEE would contain the contract graph of the entire population, it would make it an attractive target for an attack on the known vulnerabilities of TEEs (cf. §II-B). In contrast, our protocols in §VI instantiate this ideal functionality such that no single TTP learns the contact graph.

E. Components

In the following, we describe the components of PEM later instantiated in our protocols in §VI.

Collection phase:

During an encounter, both participants exchange tokens via Bluetooth LE and use this connection to determine encounter parameters $\mathsf{param}_{\mathsf{enc}}$, i.e., details about the encounter like the duration, distance, and location (cf. §VI). First, in Emit, each participant sends a encounterToken to the other participant. Then, in Listen, each participant receives the token from the other participant. Note, that Emit and Listen can correspond to the activity of a contact tracing application.

Simulation phase:

Upload: In each simulation step s, a participant ρ uses its user-specific parameters $\mathsf{param}_{\mathsf{user}}$ (e.g., its sensitivity towards an infection), its current infection class $\mathsf{class}_{\mathsf{inf}}$, the parameters $\mathsf{param}_{\mathsf{sim}}$ of the simulation j 2 [0, J] obtained from the RI, and the encounter parameters $\mathsf{param}_{\mathsf{enc}}$. With this information, ρ calculates the *ETI* (indicating the likelihood of an infection) it would have emitted during a specific encounter given the respective $\mathsf{param}_{\mathsf{enc}}$ and its current simulated $\mathsf{class}_{\mathsf{inf}}$. ρ does this for all encounters that it had in a specific time interval in the collection phase. The RI specifies the time interval for each simulation step s in $\mathsf{param}_{\mathsf{sim}}$ in advance before the start of the simulation phase such that all participants use the same encounters in s. Note, that in all J simulations, the same time interval of the collection phase is used for the s-th simulation step, but different time intervals are used for different simulation steps in the same simulation. If a simulation uses **filters**, e.g., it simulates that schools are closed, ρ will exclude all encounters it had in school and not send any messages to them. Location- or time-based filters can be introduced at any time (also after the collection phase), e.g., using location information such as coordinates. The participants only have to define which location(s) correspond(s) to "school" or other types of encounters s.t. the application can then automatically label all collected encounters. ρ sends all encrypted ETI values to the simulation server(s). We defer the concrete calculation of the ETI to the epidemiologists.

Shu e: The simulation servers apply operations on the uploaded data to provide anonymity. Fetch: In each simulation step s, each participant receives the aggregated sum of ETI infectionInfo to which it has been exposed.

Update: Each participant updates its infection class with the received sum of ETIs, the participant's previous $\mathsf{class}_{\mathsf{inf}}$, and its parameters $\mathsf{param}_{\mathsf{user}}$. We defer the concrete implementation to the epidemiologists.

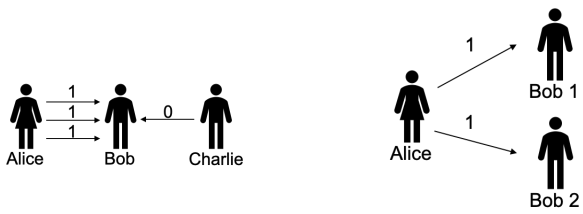
: After each simulation step s, the number of participants for each infection class is determined and output to the RI using a secure aggregation protocol .

IV. PEM Attack Scenarios

In this section, we describe the two non-trivial attacks we found to threaten the privacy of the users' contact graphs as they have important implications on the design of our PEM protocols in §VI.

A. Linking Identities Attack

When performing multiple simulations (with different simulation parameters) on the same contact graph, data from the *collection phase* gets re-used for every simulation. A participant Alice (anonymously) sends messages containing information about the infection risk (ETI, cf. §III) to another participant Bob. When Bob suspects that two different physical encounters were with the same person, Alice, he can verify this as the messages of these encounters in the simulations correlate with non-negligible probability. Concretely, Bob is even able to check correlations for all received messages to detect which messages are from the same Alice. For example, Bob could have met and identified Alice in a restaurant and later assumes that he might have met her again in the bus where he could not identify her because she was wearing a face mask. In this case, Bob receives Alice's infection likelihood represented by the ETI Alice emits (this can be a simple 0/1 for not infected/infected or a more complex representation, cf. §III) twice — once for each individual encounter. As



(a) Linking Identities Attack: If Bob has access to the individual messages sent by his encounters he can find correlations between the messages.

(b) Sybil Attack: If Bob creates multiple identities and records exactly one meeting with each identity, he can analyse correlations between the single messages.

Fig. 3: PEM Attacks (0/1= not infected/infected).

simulated "infections" will appear only for the minority of contacts, having two/multiple encounters that are infected 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. Then, 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 over multiple simulations.

To summarize, Bob can link meetings with the same Alice and with an increasing number of simulations Bob is able to validate his hypothesis with higher probability. We depict the attack in Fig. 3a. In order to mitigate this attack in PEM, Bob must not learn the infection class of individual encounters, but should also be able to determine his own risk of being exposed. Thus, our protocols in §VI obliviously sum up the messages from *all* encounters of Bob and he only receives the sum of the messages instead of the individual messages.

B. Sybil Attack

Because a participant Bob must not learn individual messages with the ETI sent by other participants (cf. §IV-A), our protocols only allow Bob to 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, and use one for each encounter. Note, that this still requires Bob to meet only a single Alice and no one else in the collection phase with each identity. By performing this sybil attack [37] with n identities, Bob would be able to receive n individual messages instead of the sum of the n messages originating from his n encounters, which again enables him to detect possible correlations between them. We depict the attack in Fig. 3b. 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. Thus, we need another mechanism to thwart sybil attacks: anonymous credentials to increase the costs to create (fake) identities (cf. §VI).

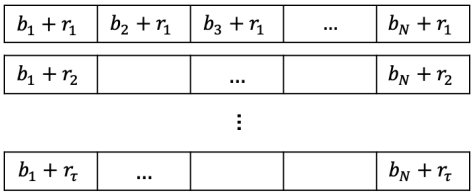


Fig. 4: Our PIR-Sum protocol. b_i is the message stored in the i-th database block (i 2 $[1, N]$). r_j is the random value used in all database blocks for blinding the j-th response (j 2 $[1, \tau]$).

V. Hybrid Threshold-PIR-SUM

In this section, we introduce two PIR constructions to (a) privately retrieve the sum of τ blocks in a database and to (b) ensure that a client retrieves τ *distinct* blocks of the database. We combine both protocols to achieve a scheme that requires a client to retrieve the *sum* of τ *distinct* blocks and embed the combination into an efficient hybrid PIR construction with multi-server [29], [30] and computational PIR [54]. In §VI-C, we show how to use this protocol to efficiently privately retrieve the sum of exactly τ distinct blocks of the database. To the best of our knowledge, this is the first multi-server Threshold-PIR-SUM protocol. We believe that our PIR constructions might also be of independent interest.[1]

A. PIR-SUM

Our first construction, depicted in Fig. 4, enables a client C to query τ blocks $(b_{\text{idx}_1}, ..., b_{\text{idx}}$) of a database DB containing N blocks, where $f\text{idx}_1, ..., \text{idx}\ g$ are the indices of the τ blocks of DB freely chosen by the client, and retrieve only the sum of these blocks.

Prior to any communication with the clients, the K PIR servers agree on a randomly chosen secret s_{PIR}. For each of the τ PIR queries from a client, the servers then derive τ fresh pseudo-random *arithmetic* shares from s_{PIR} that sum up to 0, i.e., $r_1 + ... + r\ = 0$.

For the j-th query of the τ queries from client C, the PIR servers add the same arithmetic share r_j to each element of DB, i.e., $b_i^0 = b_i + r_j$ where i 2 $[1, N]$, 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 τ PIR requests, the client receives τ arithmetically blinded blocks $b_{\text{idx}_1}^0, ..., b_{\text{idx}}^0$ where $b_{\text{idx}_j}^0 = b_{\text{idx}_j} + r_j$ for j 2 $[1, \tau]$. When summing up the blinded blocks, the shares added by the

[1]Note that all our PIR constructions are compatible with multi-block PIR [29], [30] where multiple blocks can be retrieved in a single query without increasing the query size.

$[\text{Enc}_{key_{PIR}}(b_1), k_1]$	$[\text{Enc}_{key_{PIR}}(b_2), k_2]$	...	$[\text{Enc}_{key_{PIR}}(b_N), k_N]$

Fig. 5: Our Threshold-PIR protocol. b_i is a message encrypted with key key_{PIR} and k_i is a share of key_{PIR} put into the i-th block ($i \in [0, N]$).

servers cancel out and the client retrieves the sum of database entries: $b_{\text{sum}} = \sum_{j=1} b^{0}_{\text{idx}_j} = \sum_{j=1}(b_{\text{idx}_j} + r_j) = \sum_{j=1} b_{\text{idx}_j}$. However, this alone does not guarantee that a client requests τ *different* blocks. Without any further measures, a client could query the same block in each of the τ PIR requests and divide the sum of the τ blocks by τ to receive block τ in plain.

B. Threshold-PIR

With our second PIR construction, depicted in Fig. 5 and called *Threshold-PIR*, we impose the client to request τ *distinct* blocks from the database, i.e., we require that $\forall i, j \in [1, \tau] : i \neq j \Rightarrow \text{idx}_i \neq \text{idx}_j$.

The PIR servers decide on a freshly created symmetric secret key key_{PIR} and a τ-out-of-N secret-sharing scheme (where N is the number of blocks in the PIR database) and secret-share this key. When instantiated with Shamir's secret sharing, the $\tau - 1$ random coefficients of the polynomial can be derived from s_{PIR}. Hence, this step does not require any interaction between the servers. Note, that a key can only be used once to request τ *distinct* blocks and cannot be re-used. Then, the servers symmetrically encrypt each entry of the database with key_{PIR} and append one share of the key to each block of the database. Only if the client privately retrieves at least τ distinct blocks, it is able to reconstruct the symmetric key key_{PIR} and can decrypt the τ received entries of the database. Dong et al. [36] use a similar construction to protect against malicious clients in a private set intersection protocol and reveal either one share of a secret key or one entry of a garbled Bloom filter via oblivious transfer.

C. Hybrid Combination

We now combine our PIR-SUM (cf. §V-A) and Threshold-PIR (cf. §V-B) to *Threshold-PIR-SUM* such that the client can only retrieve the *sum* of τ *distinct* blocks of the database. The servers agree on the arithmetic shares $r_1, ..., r$ as in our PIR-SUM and, for each of the τ PIR queries of the client, add the share r_j to each element of the database, where $j \in [1, \tau]$. Then, as in Threshold-PIR, the servers encrypt the resulting values using the key key_{PIR} for each of the τ PIR queries of the client. Once the client has retrieved τ of the resulting blocks, it can reconstruct the key key_{PIR}, decrypt the arithmetically blinded values and sum up the blinded values as in the PIR-SUM construction. The result is the sum of the τ blocks the client requested.

We cannot directly apply Threshold-PIR-SUM on a large PIR database DB with small blocks as needed in our PIR-PEM (§VI-C) because the PIR queries would get too large (572.21 MiB *per client* for each simulation step s with $u = 100K$ users and 55.88 GiB *per client* with $u = 10M$ users).

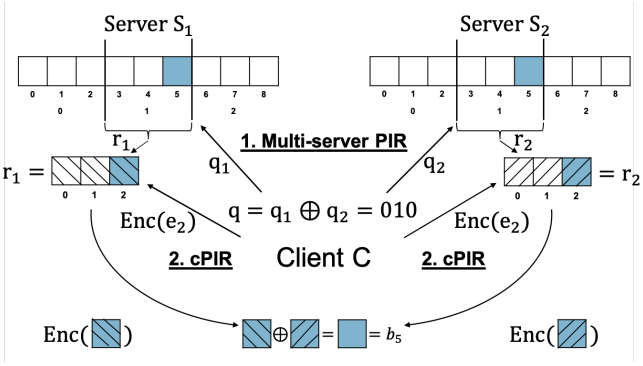


Fig. 6: Our Hybrid Threshold-PIR-SUM protocol. In order to receive the block b_5, the client first selects the second "superblock" from the database using multi-server PIR. Instead of sending the PIR responses r_1, r_2 back, the client performs a single-server computational PIR (cPIR) query on each of these. By combining the cPIR responses, the client recovers exactly block b_5.

To optimize communication, we would like to pack $C = \hat{b}/|c|$ elements into a large superblock of size $\hat{b} = \sqrt{|DB|/K}$, where $|c|$ is the size of individual elements. However, since the blinding value r_j of our PIR-SUM protocol is the same for every element in the database for the j-th PIR query, a client would be able to learn some information about other elements in the retrieved superblock. Instead, we apply Threshold-PIR-SUM to each element in DB individually. But each PIR request would now directly leak C shares of key_{PIR} to the client (one for each of the C elements in the received block). After extracting τ shares from the superblocks, the client would be able to decrypt and sum up the elements as usual (cf. §V-C) although he might not have sent τ queries.

To improve communication while maintaining correctness and privacy properties, we apply a technique called hybrid PIR [34] which combines multi-server with single-server computational PIR (cPIR). Our resulting Hybrid Threshold-PIR-SUM protocol is shown in Fig. 6. We first run multi-server PIR on superblocks (Step 1), but the servers do not yet send back the responses. Instead, the client runs a cPIR protocol [54] with each of the PIR servers to recover only the single intended element from the superblock (Step 2).

VI. PEM Protocols

In this section, we introduce three privacy-preserving epidemiological modeling (PEM) protocols simulating transitions of infection classes requiring interaction between participants (cf. §III-A). For simplicity, we depict only the communication of two participants Alice and Bob who meet during the collection phase. Alice acts as the sender of a message $m_{i;j}$, where i is Bob's local counter for his encounters and j is the counter of the number of simulations in which he

participated. The message $m_{i;j}$ represents ETI of Alice during the encounter given her current (simulated) infection class, encounter distance, duration, etc. Bob receives the sum of ETIs from his encounters in simulation step s (including Alice's message) s.t. he can estimate if he got exposed and possibly update his infection class. Note that although we do not show it here for the sake of clarity, Bob also 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).

We assume the availability of an encrypted Bluetooth LE (BLE) channel between Alice and Bob using the trick from Apple's Find My protocol [16]: When using the elliptic curve P-224, 28 bytes are needed to represent a group element. These bytes can be encoded across the 6-byte randomized MAC address and 22 bytes of the payload. Clients can now announce a public key and run an elliptic-curve Diffie–Hellman key agreement to establish a shared secret. The shared secret can then be used to privately exchange more data, e.g., a keep-alive message to determine the encounter duration and the received signal strength indicator to estimate the physical distance between the devices. For more details on BLE for contact tracing we refer the reader to [26].

Anonymous Credentials against Sybils. To thwart sybil attacks (cf. §IV-B), i.e., to prevent an adversary from creating multiple identities, we propose to use anonymous credentials (cf. §II-B). 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. As indicated in our protocols in §VI-B and §VI-C, we require the message receiver Bob to use the anonymous credentials to anonymously authenticate with the central server(s).

Mix-nets for Sender-Anonymity. The messages encoding ETIs include an identifier used by another participant to download the message. If they are uploaded to a server, the server operator could place a Bluetooth device that collects encounter tokens and later re-identify participants that upload messages together with these tokens. Thus, a dishonest server could easily track participants. To thwart this attack, participants secret share their messages and identifiers and send a share to each L 1 of the L mix servers of a mix-net (cf. §II-B). The mix servers then obliviously shuffle them (cf. App. A) and reveal all shares in the shuffled database to either the exit node of the mix-net (TEE-PEM/MIX-PEM, cf. §VI-A/§VI-B) or the PIR servers (PIR-PEM, cf. §VI-C).

Complexity. The shuffling requires L communication rounds. In each round, one mix server sends a share of each of the N database entries to the L 1 other servers. Hence, communication complexity is $O(L$ $(L$ 1) $N)$.

A. TEE-PEM

In our first protocol, called TEE-PEM and depicted in Fig. 7, we assume that the mobile device of each participant is equipped with a TEE such as ARM TrustZone (cf. §II-B). Note, that this distributes trust over multiple TEEs which is more robust than a single TEE that holds the encounter information usable to construct the contact graph (cf. §III-D) and hence is an attractive target for attacks. We give pseudocode for TEE-PEM in App. C and proceed with a high-level description:

Prepare: For every time interval in the collection phase (corresponding to a simulation step s), Bob's TEE uploads a list of freshly created public keys PK_i^B, i 2 [1, e], where e is number of maximal possible encounters per simulation step, to the exit node of a mix-net (Step 1a, cf. §II-B) that returns signatures for each of them (Step 1b). The exit node only signs the public keys if it can anonymously verify that it is directly communicating with a non-tampered TEE via a secure channel.

Emit: During an encounter, Bob sends Alice a fresh public key created in Prepare together with the corresponding signature via Bluetooth LE (Step 2a).

Listen: Alice receives the key and corresponding signature from Bob. By verifying the signature, Alice can convince herself that the public key was indeed created by a TEE (Step 2b). Then, she adds the received information to the list REC.

Upload: In each simulation step, Alice's TEE encrypts her message $m_{i;j}$ with Bob's public key PK_i^B (Step 3a) that she received in the collection phase in encounter i. Then, the TEE creates L 1 shares of the encrypted message $c_{i;j}$ and PK_i^B and sends them to L 1 nodes of the mix-net (Step 3b).

Shu e: The mix-net shuffles all messages for this simulation step (Step 3c). Afterwards, the exit node combines all shares and verifies that PK_i^B is only used once per simulation (Step 3d).

Fetch: The exit node sends all $c_{i;j}$ to Bob's TEE via a secure channel using the PK_i^B from Step 1a as selection criterion (Step 3e). Bob's TEE decrypts and sums them up (Step 3f). As Bob committed to his public keys (Step 1a), he cannot lie about his encounters. Thus, he cannot infer correlations between encounters (cf. §IV-A)

Update: Based on the decrypted sum of ETIs, Bob's new infection class is determined in the TEE. This new class is the input for his messages in the next simulation step of simulation j.

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. §IV-A). However, this problem occurs only locally (only Bob's contacts are affected) and one has to corrupt many TEEs for a large scale attack.

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.

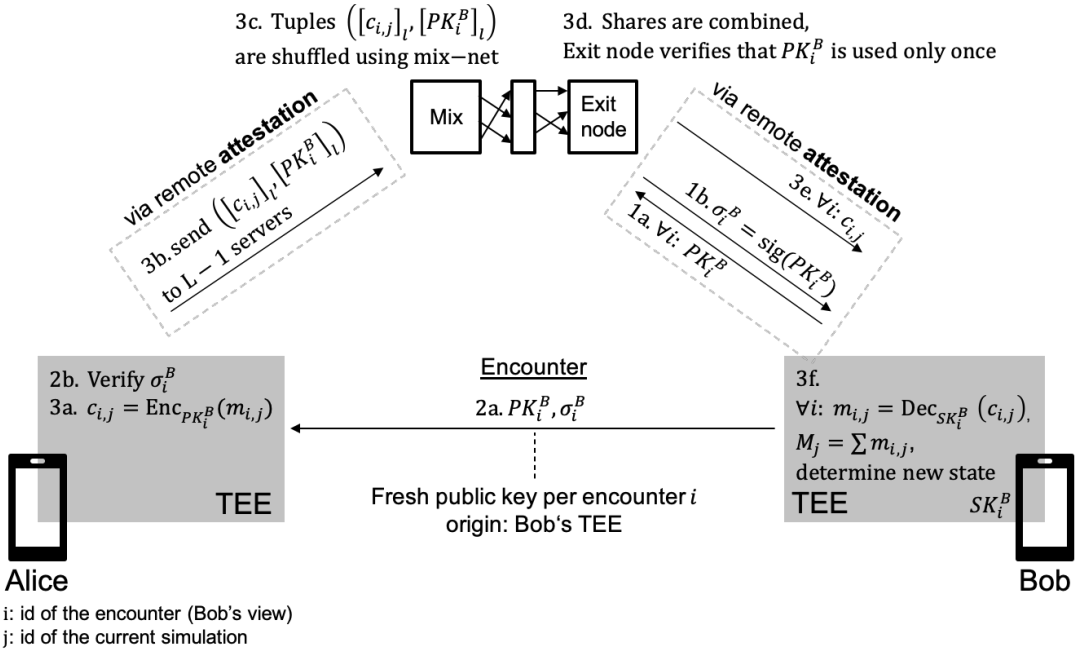


Fig. 7: TEE-PEM: Privacy-preserving epidemiological modeling using TEEs. Step 1a and 1b are done once per time interval in the collection phase. Step 2 is done per encounter in the collection phase. Steps 3a to 3f are done in every simulation step of a simulation. $[c_{i;j}]_l$ is the l-th share of the ciphertext $c_{i;j}$ and $[PK_i^B]_l$ is the l-th secret share of the public key PK_i^B sent to the l-th mix server.

B. MIX-PEM

Our second protocol is called MIX-PEM and shown in Fig. 8. It is purely based on secret sharing. We provide pseudocode for MIX-PEM in §C.

Prepare: Before the collection phase, Bob creates a set of e random blinding values r_i^B, where i 2 [1, e] and e is number of maximal possible encounters per time interval, and uploads a list of hashes of these blinding values to the exit node of a mix-net (Step 1).

Emit: During an encounter i, Bob sends one of the previously created random blinding values r_i^B.

Listen: Alice receives the random value from Bob and adds the received information to the list REC.

Upload: In each simulation j, Alice blinds her message with a hash of the random value: $c_{i;j}$ = $m_{i;j} + H(r_i^B jj j)$ (Step 3a). H denotes a publicly known cryptographic hash function. Alice then sends shares of this message alongside with shares of the hash of the blinding value r_i^B to L 1 of the L mix servers.

Shu e: After receiving shares of the messages from all participants, the mix-net shuffles them to provide sender privacy (Step 3c). Then, it reveals the shares to the exit node who combines them to receive the blinded messages and hashes. It verifies that $H(r_i^B)$ is used only once, i.e., that Bob did not sent r_i^B several times in different encounters (Step 3d). Then, it sums up the blinded messages for each participant using each participant's list of blinding values from Step 1 (Step 3e).

Fetch: The exit node sends the sum C_j and a list of all hashed blinding values corresponding to the messages included in the sum (to tolerate dropouts) to the respective participant Bob (Step 3f). By committing to the hash values before the collection phase (Step 1), Bob can only receive the sum messages of *all* of his encounters in the simulation (Step 3f). It follows that Bob cannot change the set of requested messages to infer differences between the received sums.

Update: Bob can now identify and remove the blinding values M_j = C_j $\sum H(r_i^B jj j)$ (Step 3g) and use the received sum of ETIs to update its infection class.

Malicious Exit Node. In MIX-PEM and TEE-PEM (cf. §VI-A), the exit node learns the hashes/public keys corresponding to the messages sent to Bob s.t. it is able to track Bob when a colluding participant gets Bob's messages via BLE. Moreover, the exit node could install Bluetooth sniffers to establish a mass surveillance system. The sniffers can collect the tokens from all people passing by while the exit node knows which tokens belong to the same participant. In this way, the exit node could construct a movement map per participant. 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. However, for the sake of completeness, we propose a protocol next that thwarts this attack at the cost of higher

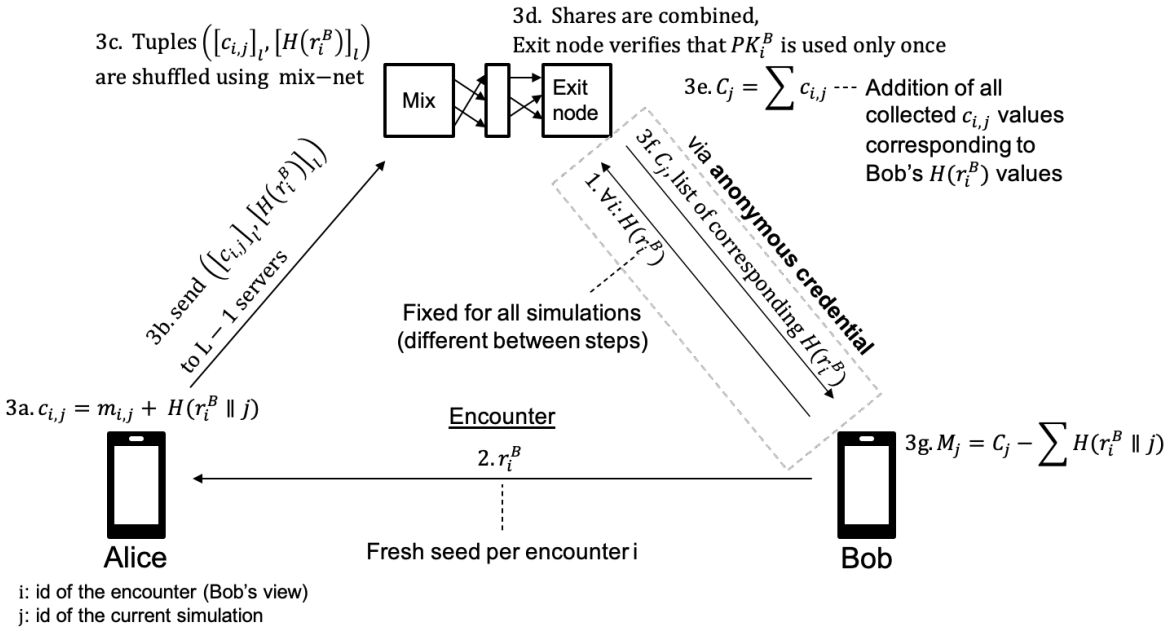


Fig. 8: MIX-PEM: Privacy-preserving epidemiological modeling using random blinding and mixing. Step 1a and 1b are done once per time interval in the collection phase. Step 2 is done per encounter in the collection phase. Steps 3a to 3g are done in every simulation step of a simulation. $[c_{i;j}]_l$ is the l-th share of the ciphertext $c_{i;j}$ and $[H(r_i^B)]_l$ is the l-th secret share of the hash of the random number r_i^B sent to the l-th mix server.

communication and computation.

C. PIR-PEM

In our third protocol, called PIR-PEM and depicted in Fig. 9, we use our novel Threshold-PIR-SUM (cf. §V) to query messages stored in an arithmetic garbled cuckoo table (AGCT, cf. §II-B) s.t. a malicious exit node cannot track participants (cf. §VI-B). We provide pseudocode for PIR-PEM in App. C.

PIR-PEM does not require any commitments prior to the collection phase. We therefore skip Step 1 and directly start with the encounter in the collection phase (Step 2).

Prepare: Bob creates e (e is number of max. possible encounters per simulation step) random values r_i^B.

Emit: In the i-th encounter in the collection phase, Bob sends r_i^B to Alice (Step 2).

Listen: Alice receives the random value r_i^B from Bob.

Upload: Alice uses the random value r_i^B to blind her message for Bob $c_{i;j} = m_{i;j} + H(r_i^B jj j)$ (Step 3a) and sends shares of $c_{i;j}$ and shares of the hash value $H(r_i^B)$ to L 1 mix servers (Step 3b).

Shu e: After all clients uploaded their blinded messages, the L mix servers obliviously shuffle the data in Step 3c. Then, assuming the PIR servers are equal to the mix servers, the mix servers combine their shares to create a joint database with all messages. They insert the ciphertexts $c_{i;j}$ at $f_{cuckoo} = 2$ positions of the AGCT, determined by the hash $H(r_i^B)$ (Step 3d). For this purpose, the servers initialize the hybrid Threshold-PIR-SUM with an 2γ-out-of-N secret sharing, where N is the size of the AGCT. The PIR servers allow Bob to fetch exactly 2γ blocks. The servers jointly fill up the remaining (untouched) locations with random values such that each server holds the same AGCT afterwards.[2]

Fetch: In Step 3e, Bob reveals the number of encounters γ he had to the PIR servers and creates 2γ PIR queries for our hybrid Threshold-PIR-SUM (cf. §V-C). Note that γ corresponds to τ in the general description of our PIR constructions in §V. The PIR queries privately request the two entries of the AGCT corresponding to $H(r_i^B)$ of each encounter. The clients' PIR queries for a simulation step are cached by the PIR servers and used in all later simulations, i.e., the PIR queries only have to be sent *once*. 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. §IV). In Step 3f, Bob retrieves the PIR responses from the PIR servers.

Update: In Step 3g, Bob reconstructs key_{PIR}, decrypts his PIR responses, and sums them up to get the sum of

[2] Alternatively, the operations can be done by a single server who distributes the AGCT to all other PIR servers, but this requires communication among the non-colluding PIR servers.

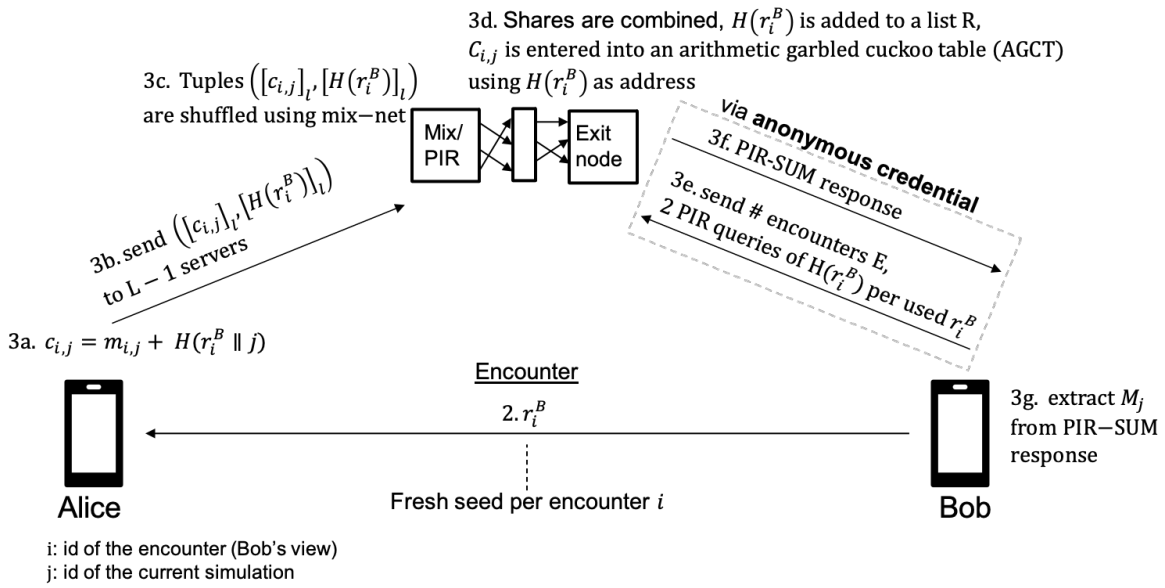


Fig. 9: PIR-PEM: Privacy-preserving epidemiological modeling using our novel Threshold-PIR-SUM (cf. §V). Step 2 is performed once per encounter in the collection phase. Steps 3a to 3f are performed for each time interval in multiple simulations. $[c_{i;j}]_l$ is the l-th share of the ciphertext $c_{i;j}$ and $[H(r_i^B)]_l$ is the l-th secret share of the hash of the random number r_i^B sent to the l-th mix server.

the blinded messages. Then, he unblinds it $M_j = C_j \sum H(r_i^B jjj)$ to receive the ETI.

Manipulation Attempts by Bob. We show that Bob gains nothing from lying about the number of encounters γ in Step 3e due to our Threshold-PIR-SUM: If Bob met less then γ participants in the collection phase, he is still forced to retrieve the sum of exactly 2γ distinct blocks from the AGCT. But, he cannot unblind messages that are not blinded with his r_i^B. If he requests a block that was not filled with a message, it contains a random value s.t. he cannot learn anything about the set of encounters from which he correctly requested the messages. If Bob met more than γ participants, he still only learns the sum of γ messages. Hence, this is equivalent to not meeting the not included participants. Note that Bob commits to the number γ for *all* simulations. Thus, he cannot change γ to extract information from differences between simulation results.

Hiding the number of encounters. A participant can prevent the simulation servers in all our PEM protocols from learning the number of encounters by sending "dummy" messages indicating no risk of infection to itself using the unused public keys/random values.

Having only one encounter. If Bob had only one physical encounter, the messages he would receive in the simulation phase of MIX-PEM (§VI-B) and PIR-PEM §VI-C) 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 were initialized as infected (cf. §III-A)), 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 with TEE-PEM (§VI-A).

Non collusion assumption. We argued in §III that doing PEM with MPC would not achieve a sufficient trust level for a broad adoption in the population due to the non collusion assumption. However, multi-server PIR as used in our hybrid Threshold-PIR-SUM also requires that the PIR servers do not collude. Let us analyze the information leakage when this assumption would break: The servers could learn which blocks were requested by Bob, but they could not extract the ETI per encounter as they do not hold the random values needed to compute the hashes used by Alice to blind the messages. Hence, colluding PIR servers cannot detect correlations between participants' encounters limiting the information leakage to knowing which messages were requested in a batch (with an anonymous communication channel). Thus, the attack of a malicious exit node colluding with a participant detailed in §VI-B would be possible if the PIR servers collude.

D. Result Aggregation

Secure aggregation is needed to determine the total number of participants per infection class (cf. §II-A) in each simulation step s and to reveal it to the research institute (cf. §III-A).

simulation j	users u	database size $1; ...; J$	per client upload 1	per client upload $2; ...; J$	per client download $1; ...; J$	per server from clients 1	per server from clients $2; ...; J$	per server to clients $1; ...; J$
TEE-PEM (§VI-A)	$100K$	915:53 MiB	21:89 KiB	18:75 KiB	12:50 KiB	2:09 GiB	1:79 GiB	1:19 GiB
MIX-PEM (§VI-B)	$100K$	305:18 MiB	7:81 KiB	6:25 KiB	1:58 KiB	0:75 GiB	0:60 GiB	0:15 GiB
PIR-PEM (§VI-C)	$100K$	366:21 MiB	9:36 MiB	6:25 KiB	25:00 KiB	913:70 GiB	0:60 GiB	2:38 GiB
TEE-PEM (§VI-A)	$10M$	89:40 GiB	21:89 KiB	18:75 KiB	12:50 KiB	208:62 GiB	178:81 GiB	119:21 GiB
MIX-PEM (§VI-B)	$10M$	29:80 GiB	7:81 KiB	6:25 KiB	1:58 KiB	74:51 GiB	59:60 GiB	15:05 GiB
PIR-PEM (§VI-C)	$10M$	35:76 GiB	93:45 MiB	6:25 KiB	25:00 KiB	891:24 TiB	59:60 GiB	238:42 GiB

TABLE I: Communication per simulation step. Beginning with the second simulation $j \in [2, ..., J]$, the upload communication drastically reduces due to the caching of client requests. u is the total number of users. The maximal number of encounters per user in a simulation step is $e = 100$. For simplicity, we omit the overhead of remote attestation and anonymous credentials. We assume (ECC) public keys with 256 bits, ECC ciphertexts/signatures with 512 bits, messages with 128 bits, and trim hash values to 128 bit.

Secure aggregation is a standard problem in privacy-preserving smart metering [39], [41], [53] 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. Our focus in this work is on introducing PEM and designing efficient protocols for PEM. We refer to the related literature on secure aggregation for this building block.

E. Dropouts

PEM protocols must also be able to handle participant dropouts as mobile devices regularly lose connection or run out of battery. Generally, if messages are not received, this simulates that the respective encounter had no risk of infection. Assuming that the share of not transmitted messages is randomly distributed and just a small proportion of the overall messages, it does not significantly change the simulation result. Furthermore, our protocols are designed s.t. dropouts do not harm the functionality: In TEE-PEM (§VI-A), the exit node forwards all messages that it received for Bob identifying them by the PKs uploaded by Bob in Step 1a. Unused PKs are ignored. In MIX-PEM (§VI-B), the exit node identifies all messages that are meant for Bob by the hashes uploaded in Step 1. It sums these messages and forwards the sum with a list of the corresponding hashes s.t. Bob knows how to unblind the ETIs (and excludes hashes of dropped-out participants). In PIR-PEM (§VI-C), an additional step is needed to tolerate dropouts: before sending messages to his encounters ("Alices"), Bob has to send a "dummy" message to himself for each of its encounters encoding no exposure. The PIR servers later replace this message by the real message of Alice. However, if Alice has dropped out, the dummy message remains in the AGCT.

VII. Evaluation

In this section, we evaluate the privacy guarantees achieved by our PEM protocols and their complexities.

A. Privacy Protection

Participants: In our PEM protocols (§VI), all participants communicate anonymously using a mix-net s.t. they do not learn anything about the identity of unconscious contacts (achieving *encounter anonymity*). Only the sum of ETIs is revealed to the participants to protect against *re-identification* (§III-B).

Simulation servers: The simulation servers are the mix servers, the exit node, and the PIR servers. In TEE-PEM (§VI-A), the non-colluding mix servers receive secret shares of the encrypted messages and PKs. The exit node receives the ciphertexts and PKs. Using an anonymous communication channel, the identity of the participants can be hidden from the mix servers/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 §VI-C, the number of encounters can be hidden. Hence, TEE-PEM provides *contact graph privacy* w.r.t. the simulation servers (cf. §III-B). In MIX-PEM (§VI-B), the same argumentation as for TEE-TEM applies, just that the messages are blinded with hashes instead of being encrypted. Thus, MIX-PEM provides *contact graph privacy* w.r.t. the simulation servers. In PIR-PEM (§VI-C), the same argumentation as for the previous two protocols applies. However, a set of PIR servers replaces the 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 $\mathsf{param}_{\mathsf{sim}}$ and gets the simulation results. Thus, it is not involved in the simulation and does not learn anything beyond what can be derived from the output s.t. our protocols provide *contact graph privacy* w.r.t. the RI (§III-B).

B. Complexity Analysis

Database sizes. The database stores the (encrypted) messages from all participants, i.e., at most e messages for each participant, where e is the number of encounters each user can maximally have. In TEE-PEM (§VI-A), the database is significantly larger than in the other protocols due to the use of public key cryptography. The database size for PIR-PEM (§VI-B) is slightly larger than the database in MIX-PEM (§VI-C) because PIR-PEM uses an arithmetic garbled cuckoo table to store the database entries. In all protocols, the database size mainly depends on the number of users u and the maximum number of encounters e.

In Tab. I, we give concrete numbers for $u = 100K$ and $u = 10M$ users. We set the upper bound for the number of encounters per day $e = 100$ based on numbers provided by research on epidemiological modeling [28], [57], [58], [62]. Even for a nation-wide deployment with $u = 10M$ users, our largest database size of about 89 GiB for TEE-PEM is realistic. For $u = 100K$ users, the database size is only about 916 MiB. MIX-PEM even achieves a database size of about 305 MiB for $u = 100K$ users and about 30 GiB for $u = 10M$ users.

Communication. Tab. I shows the total client communication per simulation step. We assume the use of [29] as outer multi-server PIR scheme and the straightforward approach of [54] using EC ElGamal as our inner cPIR scheme (cf. §V-C). We justify the use of the simplistic cPIR scheme by the fact that cPIR is applied to only 9.57 KiB for $u = 100K$ resp. 95.68 KiB for $u = 100M$. Even for a nation-wide deployment with $u = 10M$ users, each user's total communication in PIR-PEM is 37.38 MiB. For the more realistic deployment of $u = 100K$ users, PIR-PEM has 3.76 MiB communication per user. Note that communication is drastically reduced in all subsequent simulations since the PIR queries only have to be uploaded once (cf. §VI-C). In all subsequent simulations, the total communication per client is only 31.25 KiB and independent of the number of users u. Hence, simulating various different spread behaviors becomes extremely inexpensive. Since simulation is decoupled from the collection phase, the first simulation can be done over night when people sleep [78], [81] s.t. mobile phones are charging and have access to a high-bandwidth wireless LAN connection.

In Tab. II in App. B, we depict the asymptotic complexities of our protocols. Each mix server secret shares the elements of its database to the other servers totaling $(L \quad 1)\ jDBj$ bytes. In the final step of the shuffling, $L \quad 1$ servers exchange their shares again to retrieve the plaintext database. The total communication of the mix-net with $L = 3$ servers equals $L \ (L \quad 1)\ jDBj + (L \quad 1)\ jDBj = 8\ jDBj$ which is reasonably practical.

Computation. Our protocols are computationally inexpensive as they mainly rely on information-theoretically secure primitives and symmetric cryptography. Apart from generating the PIR queries (which can be precomputed), client computation is *independent* of the total number of users u. Particularly, we do not employ expensive cryptographic primitives like homomorphic encryption in our PEM protocols.

Implementation. With our work, we want to initiate the discussion on PEM protocols and present three promising protocols that have feasible communication and computation complexities. Implementing our protocols is out of scope for now, as it would require to team-up with industry partners to implement a real-world scalable system for deployment on a country level.

VIII. Conclusion

In this work, we introduced the problem of privacy-preserving epidemiological modeling (PEM) which enables the distributed and private simulation of the spread of a disease on real contact graphs collected on mobile devices. We presented three PEM protocols based on different trust assumptions, discussed their weaknesses, and showed that they have manageable complexities to be implemented for Millions of users. As building block which might be of independent interest, we designed a communication-efficient PIR-based protocol for privately querying the sum of multiple distinct blocks. We hope to motivate further research and implementations towards highly efficient PEM.

References

[1] Ittai Abraham, Benny Pinkas, and Avishay Yanai. Blinder: MPC based scalable and robust anonymous committed broadcast. Cryptology ePrint Archive, Report 2020/248, 2020. https://eprint.iacr.org/2020/248.pdf.

[2] Carlos Aguilar-Melchor, Joris Barrier, Laurent Fousse, and Marc-Olivier Killijian. XPIR: Private information retrieval for everyone. *PETS*, 2016.

[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. A survey of COVID-19 contact tracing apps. In *Access*. IEEE, 2020.

[4] Ittai Anati, Shay Gueron, Simon P Johnson, and Vincent R Scarlata. Innovative technology for CPU based attestation and sealing. In *Workshop on Hardware and Architectural Support for Security and Privacy*. ACM, 2013.

[5] Sebastian Angel, Hao Chen, Kim Laine, and Srinath Setty. PIR with compressed queries and amortized query processing. In *S&P*. IEEE, 2018.

[6] ARM. *ARM security technology building a secure system using TrustZone technology*, 2009. https://developer.arm.com/documentation/genc009492/c, (accessed: 29.07.2020).

[7] Raad Bahmani, Manuel Barbosa, Ferdinand Brasser, Bernardo Portela, Ahmad-Reza Sadeghi, Guillaume Scerri, and Bogdan Warinschi. Secure multiparty computation from SGX. In *FC*. Springer, 2017.

[8] Lars Baumgärtner, Alexandra Dmitrienko, Bernd Freisleben, Alexander Gruler, Jonas Höchst, Joshua Kühlberg, Mira Mezini, Markus Miettinen, Anel Muhamedagic, Thien Duc Nguyen, Alvar Penning, Dermot Frederik Pustelnik, Filipp Roos, Ahmad-Reza Sadeghi, Michael Schwarz, and Christian Uhl. Mind the gap: Security & privacy risks of contact tracing apps, 2018. https://arxiv.org/abs/2006.05914.pdf.

[9] Stephanie Bayer and Jens Groth. Efficient zero-knowledge argument for correctness of a shuffle. In *Eurocrypt*. Springer, 2012.

[10] Sebastian P. Bayerl, Tommaso Frassetto, Patrick Jauernig, Korbinian Riedhammer, Ahmad-Reza Sadeghi, Thomas Schneider, Emmanuel Stapf, and Christian Weinert. Offline model guard: Secure and private ML on mobile devices. *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2020.

[11] G. V. Bobashev, D. M. Goedecke, Feng Yu, and J. M. Epstein. A hybrid epidemic model: Combining the advantages of agent-based and equation-based approaches. In *Winter Simulation Conference*. IEEE, 2007.

[12] Dan Boneh, Xavier Boyen, and Shai Halevi. Chosen ciphertext secure public key threshold encryption without random oracles. In *CT-RSA*. Springer, 2006.

[13] Elette Boyle, Niv Gilboa, and Yuval Ishai. Function secret sharing: Improvements and extensions. In *CCS*. ACM, 2016.

[14] Ferdinand Brasser, Urs Müller, Alexandra Dmitrienko, Kari Kostiainen, Srdjan Capkun, and Ahmad-Reza Sadeghi. Software grand exposure:SGX cache attacks are practical. In *USENIX*. USENIX Association, 2017.

[15] Alessandro Bruni, Lukas Helminger, Daniel Kales, Christian Rechberger, and Roman Walch. Privately connecting mobility to infectious diseases via applied cryptography. Cryptology ePrint Archive, Report 2020/522, 2020. https://eprint.iacr.org/2020/522.

[16] Ran Canetti, Yael Tauman Kalai, Anna Lysyanskaya, Ronald L Rivest, Adi Shamir, Emily Shen, Ari Trachtenberg, Mayank Varia, and Daniel J Weitzner. Privacy-preserving automated exposure notification. 2020.

[17] 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. PACT: Privacy sensitive protocols and mechanisms for mobile contact tracing, 2020. https://arxiv.org/pdf/2004.03544.pdf.

[18] Melissa Chase, Sarah Meiklejohn, and Greg Zaverucha. Algebraic MACs and keyed-verification anonymous credentials. In *CCS*. ACM, 2014.

[19] David Chaum. Security without identification: Transaction systems to make big brother obsolete. In *Communications of the ACM*. ACM, 1985.

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

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

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

[23] Benny Chor, Oded Goldreich, Eyal Kushilevitz, and Madhu Sudan. Private information retrieval. In *Annual Foundations of Computer Science*. IEEE, 1995.

[24] Commonwealth of Australia. *COVIDSafe app*, 2020. https://www.health.gov.au/resources/apps-and-tools/covidsafe-app, (accessed: 21.08.2020).

[25] Henry Corrigan-Gibbs and Dmitry Kogan. Private information retrieval with sublinear online time. In *Eurocrypt*. Springer, 2020.

[26] Mathieu Cunche, Antoine Boutet, Claude Castelluccia, Cédric Lauradoux, and Vincent Roca. On using Bluetooth-Low-Energy for contact tracing. Research report, Inria Grenoble Rhône-Alpes ; INSA de Lyon, June 2020.

[27] Alfredo De Santis, Yvo Desmedt, Yair Frankel, and Moti Yung. How to share a function securely. In *STOC*. ACM, 1994.

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

[29] Daniel Demmler, Amir Herzberg, and Thomas Schneider. RAID-PIR: Practical multi-server PIR. In *ACM Workshop on Cloud Computing Security*, 2014.

[30] Daniel Demmler, Marco Holz, and Thomas Schneider. OnionPIR: Effective protection of sensitive metadata in online communication networks. In *ANCS*. Springer, 2017.

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

[32] Yvo G Desmedt. Threshold cryptography. *European Transactions on Telecommunications*, 1994.

[33] Ghada Dessouky, David Gens, Patrick Haney, Garrett Persyn, Arun Kanuparthi, Hareesh Khattri, Jason M Fung, Ahmad-Reza Sadeghi, and Jeyavijayan Rajendran. Hardfails: Insights into software-exploitable hardware bugs. In *USENIX*. USENIX Association, 2019.

[34] Casey Devet and Ian Goldberg. The best of both worlds: Combining information-theoretic and computational pir for communication efficiency. In *PETS*. Springer, 2014.

[35] Odo Diekmann, Hans Heesterbeek, and Tom Britton. *Mathematical tools for understanding infectious disease dynamics*. Princeton University Press, 2012.

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

[37] John R Douceur. The sybil attack. In *International Workshop on Peer-to-Peer Systems*. Springer, 2002.

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

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

[40] Jun Furukawa and Kazue Sako. An efficient scheme for proving a shuffle. In *CRYPTO*. Springer, 2001.

[41] Flavio D Garcia and Bart Jacobs. Privacy-friendly energy-metering via homomorphic encryption. In *International Workshop on Security and Trust Management*. Springer, 2010.

[42] P. Gemmell. An introduction to threshold cryptography. In *CT-RSA*. Springer, 1997.

[43] Craig Gentry and Shai Halevi. Compressible FHE with applications to PIR. In *TCC*. Springer, 2019.

[44] Government of Singapor. *TraceTogether, safer together*, 2020. https://www.tracetogether.gov.sg/, (accessed: 21.08.2020).

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

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

[47] Elizabeth Hunter, Brian Mac Namee, and John D. Kelleher. A comparison of agent-based models and equation based models for infectious disease epidemiology. In *Artificial Intelligence and Computer Science (AICS)*, 2018.

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

[49] Intel. *Attestation Service for Intel® Software Guard Extensions*. https://api.trustedservices.intel.com/documents/sgx-attestation-api-spec.pdf, (accessed: 29.07.2020).

[50] Intel. *Intel® Software Guard Extensions Programming Reference*, 2014. https://software.intel.com/sites/default/files/managed/48/88/329298-002.pdf, (accessed: 29.07.2020).

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

[52] Adam Kirsch, Michael Mitzenmacher, and Udi Wieder. More robust hashing: Cuckoo hashing with a stash. *Journal on Computing*, 2010.

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

[54] Eyal Kushilevitz and Rafail Ostrovsky. Replication is not needed: Single database, computationally-private information retrieval. In *Proceedings 38th annual symposium on foundations of computer science*. IEEE, 1997.

[55] Sven Laur, Jan Willemson, and Bingsheng Zhang. Round-efficient oblivious database manipulation. In *International Conference on Information Security*. Springer, 2011.

[56] Franck Legendre, Mathias Humbert, Alain Mermoud, and Vincent Lenders. Contact tracing: An overview of technologies and cyber risks, 2020. https://arxiv.org/ftp/arxiv/papers/2007/2007.02806.pdf.

[57] James M McCaw, Kristian Forbes, Paula M Nathan, Philippa E Pattison, Garry L Robins, Terence M Nolan, and Jodie McVernon. Comparison of three methods for ascertainment of contact information relevant to respiratory pathogen transmission in encounter networks. In *BMC infectious diseases*, 2010.

[58] Alessia Melegaro, Mark Jit, Nigel Gay, Emilio Zagheni, and W John Edmunds. What types of contacts are important for the spread of infections? using contact survey data to explore european mixing patterns. In *Epidemics*. Elsevier, 2011.

[59] Ministère de l'Économie, des Finances et de la Relance. *StopCovid*, 2020. https://www.economie.gouv.fr/stopcovid.

[60] Payman Mohassel and Saeed Sadeghian. How to hide circuits in mpc an efficient framework for private function evaluation. In *Eurocrypt*. Springer, 2013.

[61] Yamir Moreno, Romualdo Pastor-Satorras, and Alessandro Vespignani. Epidemic outbreaks in complex heterogeneous networks. *The European Physical Journal B-Condensed Matter and Complex Systems*, 26(4):521–529, 2002.

[62] Joël Mossong, Niel Hens, Mark Jit, Philippe Beutels, Kari Auranen, Rafael Mikolajczyk, Marco Massari, Stefania Salmaso, Gianpaolo Scalia Tomba, Jacco Wallinga, et al. Social contacts and mixing patterns relevant to the spread of infectious diseases. In *PLoS Med*, 2008.

[63] C Andrew Neff. A verifiable secret shuffle and its application to e-voting. In *CCS*. ACM, 2001.

[64] Mark EJ Newman. Spread of epidemic disease on networks. *Physical review E*, 66(1):016128, 2002.

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

[66] Cameron Nowzari, Victor M Preciado, and George J Pappas. Analysis and control of epidemics: A survey of spreading processes on complex networks. *IEEE Control Systems Magazine*, 36(1):26–46, 2016.

[67] Olga Ohrimenko, Felix Schuster, Cédric Fournet, Aastha Mehta, Sebastian Nowozin, Kapil Vaswani, and Manuel Costa. Oblivious multi-party machine learning on trusted processors. In *USENIX*. USENIX Association, 2016.

[68] Rasmus Pagh and Flemming Friche Rodler. Cuckoo hashing. *Journal of Algorithms*, 2004.

[69] Christian Paquin and Greg Zaveruch. *U-Prove Cryptographic Specification V1.1 (Revision 3)*, 2013. http://www.microsoft.com/uprove, (accessed: 20.08.2020).

[70] Romualdo Pastor-Satorras, Claudio Castellano, Piet Van Mieghem, and Alessandro Vespignani. Epidemic processes in complex networks. In *Reviews of modern Physics*. APS, 2015.
[71] PEPP-PT Team. Pepp-pt documentation, 2020. https://github.com/pepp-pt/pepp-pt-documentation.
[72] Benny Pinkas and Eyal Ronen. Hashomer-a proposal for a privacy-preserving bluetooth based contact tracing scheme for hamagen, 2020.
[73] Benny Pinkas, Mike Rosulek, Ni Trieu, and Avishay Yanai. PSI from PaXosS Fast, malicious private set intersection. In *EUROCRYPT*. Springer, 2020.
[74] Adi Shamir. How to share a secret. *Communications of the ACM*, 1979.
[75] Michael Small and Chi K Tse. Small world and scale free model of transmission of SARS. In *International Journal of Bifurcation and Chaos*. World Scientific, 2005.
[76] Carmela Troncoso, Mathias Payer, Jean-Pierre Hubaux, Marcel Salathé, James Larus, Edouard Bugnion, Wouter Lueks, Theresa Stadler, Apostolos Pyrgelis, Daniele Antonioli, et al. Decentralized privacy-preserving proximity tracing.
[77] Serge Vaudenay. Centralized or decentralized? the contact tracing dilemma. Cryptology ePrint Archive, Report 2020/531, 2020. https://eprint.iacr.org/2020/531.pdf.
[78] Olivia J Walch, Amy Cochran, and Daniel B Forger. A global quantification of "normal" sleep schedules using smartphone data. *Science advances*, 2016.
[79] Guan Wang, Tongbo Luo, Michael T Goodrich, Wenliang Du, and Zutao Zhu. Bureaucratic protocols for secure two-party sorting, selection, and permuting. In *AsiaCCS*. ACM, 2010.
[80] Marleen Werkman, Michael J. Tildesley, Ellen Brooks-Pollock, and Matt J. Keeling. Preserving privacy whilst maintaining robust epidemiological predictions. *Epidemics*, 2016.
[81] Victoria Woollaston. *Sleeping habits of the world revealed: The US wakes up grumpy, China has the best quality shut-eye and South Africa gets up the earliest*, 2015. https://www.dailymail.co.uk/sciencetech/article-3042230/Sleeping-habits-world-revealed-wakes-grumpy-China-best-quality-shut-eye-South-Africa-wakes-earliest.html, (accessed: 30.10.2020).
[82] Yuanzhong Xu, Weidong Cui, and Marcus Peinado. Controlled-channel attacks: Deterministic side channels for untrusted operating systems. In *S&P*. IEEE, 2015.
[83] Eiko Yoneki. Fluphone study: Virtual disease spread using haggle. In *Workshop on Challenged Networks*. ACM, 2011.

APPENDIX

A. Shuffling with Mix-Net

The oblivious shuffle protocol (cf. §II-B) for $L = 3$ mix servers works as follows: The L 1 mix servers S_1, S_2 that receive the shares of messages from the participants first agree on a permutation π and reorder their shares accordingly. Then, one of the servers, let's say S_2, again secret-shares its (permuted) shares among the remaining L 1 mix servers, i.e., S_1 and S_3. S_1 now adds the received shares to its own (permuted) shares of the database entries. Now, S_1 and S_3 hold shares that sum up to the plaintext messages. They can now agree on another permutation π' and repeat the shuffle procedure after which S_2 and S_3 hold the two times permuted shares of the database entries. Finally, S_2 and S_3 send their permuted shares to S_1 (or a special exit server) who reconstructs the permuted messages.

B. Communication complexities

In Tab. II, we give formulas for the communication costs of our three private epidemiological modeling protocols.

C. Pseudo-Code PEM

List. 1 provides an overview about the information flow and computations at a participant in our three PEM protocols. In this section, we provide details about the instantiations of each of the components that we described in §III-E for our TEE-PEM (§VI-A), MIX-PEM (§VI-B), and PIR-PEM (§VI-C).

	TEE-PEM (§VI-A)
database size	$u \cdot \quad \cdot (512 + 256)$
client upload	$\cdot 256 + \quad \cdot (L-1) \cdot (512 + 256)$
client download	$\cdot (512 + 512)$
	MIX-PEM (§VI-B)
database size	$u \cdot \quad \cdot (128 + 128)$
client upload	$\cdot 128 + \quad \cdot (L-1) \cdot (128 + 128)$
client download	$128 + \quad \cdot 128$
	PIR-PEM (§VI-C)
database size	$((2 + \text{''}) \cdot u \cdot \quad + \log_2(u \cdot \quad) + \quad) \cdot 128$
client upload	$\cdot ((L-1) \cdot (128 + 128) + 2 \cdot \frac{\lvert DB \rvert}{\hat{b}} +$ $K \cdot \quad + 2 * K * \frac{\hat{b}}{128} * 512)$
client download	$\cdot 2 \cdot K \cdot 512$

TABLE II: Communication costs in bits for one simulation step s of our protocols in §VI. u is the total number of users, γ is an upper bound for the number of encounters each user had, $L = 3$ is the number of mix servers, $K = 2$ is the number of PIR servers, $\hat{b} = \sqrt{|DB|/K}$ is the optimal block size of the multi-server PIR scheme, $\varepsilon = 0.4$ is the a Cuckoo hash parameter from [73], $\lambda = 40$ bit is the statistical security parameter, and $\kappa = 128$ bit is the computational security parameter. We assume (ECC) public keys of size 256 bits, ECC ciphertexts and signatures of size 512 bit, messages of size 128 bit and trim hash values to 128 bit.

Listing 1: *f*TEE,MIX,PIR*g*-PEM: Participant

```
Input: list mixServers, timeInterval, param_user, identityInformation, (for PIR-PEM: PIRServers)
// setup -> only for MIX-PEM and PIR-PEM
anonymCredentialsData = getCredentials(credentialIssuer,identityInformation)

global list REC, SENT = ;
L = |mixServers|
exitNode = mixServers[L-1]

// collection phase
for each timeInterval:
  Emit()
  list receivedData.add(receiveBLE())
  for k 2 1...receivedData.length:
    Listen(receivedData[k])

// simulation phase
J = getNumberOfSimulations(from = RI)
class_inf = getInitialInfectionClass(from = RI)

for j 2 1...J:
  param_sim = getSimulationParameters(from = RI, j)
  S = getSimSteps(param_sim)
  for s 2 1...S:
    // iterate through B's (received) encounter information
    for (encounterToken, param_enc) 2 filter(REC, param_sim, day = s):
      c_{k;j} = Upload(class_inf,param_user, param_sim, param_enc, encounterToken)
      list sharesEncryption, sharesReceiverInfo = ;
      sharesEncryption = XORSharing.create(c_{k;j},L-1)
      // create shares of the PK or hash of the random value sent during the encounter
      sharesReceiverInfo = XORSharing.create(encounterToken.getReceiverInformation(),L-1)
      for l 2 1...L):
        // 3b: send shares to mix-net
        send(mixServers[l], sharesEncryption[l], sharesReceiverInfo[l])

    infectionInfo = Fetch(filter(SENT, param_sim, day = s))
    class_inf = Update(infectionInfo,class_inf,param_user)
```

Listing 2: Mix-net Server l (Shuffle)

```
Input: list participants, list otherMixServers, exitNode
// simulation phase
J = getNumberOfSimulations(from = RI)

for j 2 1...J:
  param_sim = getSimulationParameters(from = RI, j)
  S = getSimSteps(param_sim)
  for s 2 1...S:
    D_l = receive(participants)
    // 3c: jointly shuffle the received data
```

```
    DB_l = shuffle(otherMixServers, D_l)
    send(exitNode, DB_l)
```

Pseudo-Code TEE-PEM. List. 3 to 7 show the workflow of TEE-PEM (§VI-A) at a participant B. Note that all operations need to be executed in a TEE, i.e., the exit node will only send the ciphertexts to B in List. 7 if it can verify that it is communicating with a valid TEE via secure attestation (cf. §II-B). Similarly, the signature verification in List. 5 ensures that the exit node has verified that it received the PKs from a valid TEE before signing them. List. 8 presents the exit node's activities.

Listing 3: TEE-PEM: Prepare

```
function Prepare(e) {
  global list keypairs = ;
  list signatures = ;

  // generate e key pairs
  for i 2 1...e:
    PK_i^B, SK_i^B = generateECCKeyPair()
    keypairs.add((PK_i^B, SK_i^B))

    // 1a & 1b: send all public keys to the exit node and receive signatures
    _i^B = send(exitNode, PK_i^B)
    signatures.add(_i^B)

  return signatures
}
```

Listing 4: TEE-PEM: Emit

```
function Emit() {
  signatures = Prepare(getMaxNumberEncounter())
  counter i = 0
  for each encounter:
    encounterToken = (keypairs[i].pk, signatures[i])
    // 2a: send a PK and the respective signature to other participant in the BLE range (via
        encrypted BLE channel)
    sendBLE(encounterToken)
    param_enc = (getLocation(), getEncounterDuration(), getEncounterDistance())
    SENT.add((encounterToken, param_enc))
    i++
}
```

Listing 5: TEE-PEM: Listen

```
function Listen(encounterToken) {
  PK_k^{A_k}; _k^{A_k} = unpack(encounterToken)
  // 2b
  if verifySignature(_k^{A_k}, PK_k^{A_k}):
    param_enc = (getLocation(), getEncounterDuration(), getEncounterDistance())
    // stores PKs + signatures from other participants A_k in the BLE range
    REC.add(encounterToken, param_enc)
}
```

Listing 6: TEE-PEM: Upload

```
function Upload(class_inf; param_user; param_sim; param_enc; encounterToken) {
  PK_k^{A_k}; _k^{A_k} = unpack(encounterToken)
  m_{k;j} = getETI(class_inf; param_user; param_sim; param_enc)
  // 3a: encrypt message
  c_{k;j} = Enc_{PK_k^{A_k}}(m_{k;j})
  return c_{k;j}
}
```

Listing 7: TEE-PEM: Fetch

```
function Fetch(encounterToken, param_enc) {
  // 3e
  list ciphertexts = receive(exitNode)

  ETIsum = 0
  for i 2 0...ciphertexts.length:
    // 3f
    m_{i;j} = Dec_{keypairs[i]:sk}(ciphertexts[i])
    ETIsum += m_{i;j}

  return ETIsum
}
```

Listing 8: TEE-PEM: ExitNode (Shuffle)

```
Input: mixNet, participants,
// collection phase
for each timeInterval:
  for p 2 participants:
    authenticationInformation, PKs = receivePKs(p)
    // check that valid TEE is used
    if verify(authenticationInformation):
      // 1b: send signatures
      send([sign(key) for key in PKs])

// simulation phase
J = getNumberOfSimulations(from = RI)
class_inf = getInitialInfectionClass(from = RI)

for j 2 1...J:
  param_sim = getSimulationParameters(from = RI, j)
  S = getSimSteps(param_sim)
  for s 2 1...S:
    sharesCiphertexts, sharesPKs = receiveData(mixNet)
    ciphertexts, PKs = XORSharing.combine(sharesCiphertexts, sharesPKs)
    // 3d: check that hashes are used only once
    checkUsage(PKs)
    // 3e: send ciphertexts
    send(participants,ciphertexts)
```

Pseudo-Code MIX-PEM. List. 9 to 13 show the workflow of MIX-PEM (§VI-B) at a participant. Note that the steps 1) and 3f) in List. 9, List. 13, and List. 14 are only executed by the exit node if the participant can authenticate itself with anonymous credentials (§II-B). List. 14 presents the exit node's activities.

Listing 9: MIX-PEM: Prepare

```
function Prepare(e) {
  global list blindingValues = ;

  // generate e random values
  for i 2 1...e:
    r_i^B = generateRandomValue()
    blindingValues.add(r_i^B)
    // 1
    sent(exitNode, anonymCredentialsData, H(r_i^B))
}
```

Listing 10: MIX-PEM: Emit

```
function Emit() {
Prepare(getMaxNumberEncounter())
  counter i = 0
  for each encounter:
    encounterToken = blindingValues[i]
    // 2: send random value to other participant in the BLE range (via encrypted BLE channel)
    sendBLE(encounterToken)
    param_enc = (getLocation(), getEncounterDuration(), getEncounterDistance())
    SENT.add((encounterToken, param_enc))
    i++
}
```

Listing 11: MIX-PEM: Listen

```
function Listen(encounterToken) {
  r_k^{A_k} = encounterToken
  param_enc = (getLocation(), getEncounterDuration(), getEncounterDistance())
  // stores random values and encounter parameters from other participants A_k in the BLE range
  REC.add(r_k^{A_k}, param_enc)
}
```

Listing 12: MIX-PEM: Upload

```
function Upload(class_inf; param_user; param_sim; param_enc; encounterToken) {
  r_k^{A_k} = encounterToken
  m_{k;j} = getETI(class_inf; param_user; param_sim; param_enc)
  // 3a: blind message
  c_{k;j} = m_{k;j} + H(r_k^{A_k}jjj)
  return c_{k;j}
}
```

Listing 13: MIX-PEM: Fetch

```
function Fetch(encounterToken,param_enc) {
  // 3f
  sumShares, hashes = receive(exitNode)

  for i 2 1...hashes.length:
    // determine blinding value
    r^B = match(hashes[i], blindingValues)
```

```
    sumShares -= H(r^B_jjj)
  return sumShares
g
```

Listing 14: MIX-PEM: ExitNode (Shuffle)

```
Input: mixNet, participants
list collectedHashes = ;
// collection phase
for each timeInterval:
  for p 2 participants:
    collectedHashes.append(receiveHashes(p))

// simulation phase
J = getNumberOfSimulations(from = RI)
class_inf = getInitialInfectionClass(from = RI)

for j 2 1...J:
  param_sim = getSimulationParameters(from = RI, j)
  S = getSimSteps(param_sim)
  for s 2 1...S:
    sharesMessage, sharesHashes = receiveData(mixNet)
    // 3d: combine shares
    messages, hashes = XORSharing.combine(sharesMessage, sharesHashes)
    checkUsage(hashes) //3d: check that hashes are used only once
    // 3e: adds messages that have the same receiver (hashes came from the same sender -> stored in
        collectedHashes)
    sumMessages = sum(collectedHashes, messages, hashes)
    // 3f: send sums to participants + verify anonymous credentials
    send(participants, sumMessages, hashes)
```

Pseudo-Code PIR-PEM. List. 15 to 19 show the workflow of PIR-PEM (§VI-C) at a participan. Note that the steps 3e) and 3f) in List. 19 and List. 20 are only executed by the exit node if the participant can authenticate itself with anonymous credentials (§II-B). List. 20 presents the exit node's activities.

Listing 15: PIR-PEM: Prepare

```
function Prepare(e)f
  global list blindingValues = ;

  // generate e random valuesg
  for i 2 1..e:
    r^B_i = generateRandomValue()
    blindingValues.add(r^B_i)
g
```

Listing 16: PIR-PEM: Emit

```
function Emit(i)f
  Prepare(getMaxNumberEncounter())
  counter i = 0
  for each encounter:
    encounterToken = blindingValues[i]
    // 2: send random value to other participant in the BLE range (via encrypted BLE channel)
    sendBLE(encounterToken)
    param_enc = (getLocation(), getEncounterDuration(), getEncounterDistance())
    SENT.add((encounterToken, param_enc))
    i++
g
```

Listing 17: PIR-PEM: Listen

```
function Listen(encounterToken)f
  r^A_k = encounterToken
  param_enc = (getLocation(), getEncounterDuration(), getEncounterDistance())
  // store random value and encounter parameters from other participants A_k in the BLE range
  REC.add(r^A_k, param_enc)
g
```

Listing 18: PIR-PEM: Upload

```
function Upload(class_inf; param_user; param_sim; param_enc; encounterToken)f
  r^A_k = encounterToken
  m_k;j = getETI(class_inf; param_user; param_sim; param_enc)
  // 3a: blind message
  c_k;j = m_k;j + H(r^A_k jjj)
  return c_k;j
g
```

Listing 19: PIR-PEM: Fetch

```
function Fetch(encounterToken,param_enc)f
  if j==0: // first simulation nrightarrow generate & upload PIR requests
    list req = ;
    for r^B_i 2 SENT.encounterToken:
      req.add(HybridPIR.generateReqest(H(r^B_i)))
    // 3e
    sent(PIRServers, anonymCredentialsData, req, jSENTj)

  list ciphertexts, keyShares = ;
  for i 2 e:
    // 3f
    PIRResponse = receive(PIRservers)
    ct, ks = HybridPIR.reconstruct(PIRResponse)
    ciphertexts.add(ct)
    keyShares.add(ks)

  // combine using threshold secret sharing scheme (TSS)
  key_PIR = TSS.combine(keyShares)

  C_j = 0
  for i 2 1..ciphertexts.length:
    r^B_i = blindingValues[i]
    // decrypt and unblind message
    c = Dec_key_PIR(ciphertexts[i]) - H(r^B_i jjj)
    C_j += c

  return C_j
g
```

Listing 20: PIR-PEM: ExitNode (Shuffle)

```
Input: mixNet, participants
// simulation phase
J = getNumberOfSimulations(from = RI)
PIRQueries = ;
    // 3d: combine shares and create AGCT

for j 2 1...J:
  param_sim = getSimulationParameters(from = RI, j)
  S = getSimSteps(param_sim)
  for s 2 1...S:
    // 3d: combine shares and create AGCT
    sharesMessages, sharesHashes = receiveData(mixNet)
    messages, hashes = XORsharing.combine(sharesMessages, sharesHashes)
    AGCT = createAGCT(messages, hashes)

    for u 2 1..participants.length:
      if j == 1:
        // PIR queries are reused for all simulations
        PIRQueries[s][u] = getQueries(participants[u])

      key_PIR = KDF(s_PIR jj "PIR key" jj j jj s jj u)

      // create (2     u)-out-of-(AGCT.length) threshold-shares of key_PIR
      // where   u is the number of encounters of participant u
      keyShares = TSS.create(key_PIR, 2     u, AGCT.length)

      // create XOR sharing of zero
      r = XORSharing.create(0, 2     u)

      // precompute PIR responses
      PIRResponses = ;

      for l in 1..(2     u):
        PIRDB = ;

        for k 2 1..AGCT.length:
          PIRDB[k] = ((Enc_key_PIR(AGCT[k] + r[l]), keyShares[k]))
        PIRResponses[l] = HybridPIR.createResponses(PIRQueries[s][u][k], PIRDB)

      // 3f: send PIR responses
      send(participants[u], PIRResponses)
```