Aggregated Private Information Retrieval

A First Practical Implementation to Support Large-Scale Disease Analytics

Lukas Helminger[1,2], Daniel Kales[1], Christian Rechberger[1], and Roman Walch[1,2]

[1] Graz University of Technology, Graz, Austria
[2] Know-Center GmbH, Graz, Austria

Abstract. With the outbreak of the coronavirus, governments rely more and more on location data shared by European mobile network operators to monitor the advancements of the disease. In order to comply with often strict privacy requirements, this location data, however, has to be anonymized, limiting its usefulness for making statements about a filtered part of the population, like already infected people.
In this research, we aim to assist with the disease tracking efforts by designing a protocol to detect coronavirus hotspots from mobile data while still maintaining compliance with privacy expectations. We use various state-of-the-art privacy-preserving cryptographic primitives to design a protocol that can best be described as aggregated private information retrieval (APIR). Our protocol is based on homomorphic encryption, with additional measures to protect against malicious requests from clients.
We have implemented our APIR protocol in the SEAL library and tested it for parameters suitable to create a coronavirus hotspot map for entire nationstates. This demonstrates that it is feasible to apply our APIR protocol to support nationwide disease analysis while still preserve the privacy of infected people.

Keywords: FHE, APIR, Sars-CoV-2, Corona virus, BFV

1 Introduction

Due to the ongoing global threat of the SARS-CoV-2 virus, a number of technological approaches are currently developed to help reduce its spread and impact. A lot of focus is on automatic contact tracing, challenges include privacy-friendliness, scalability and utility, efforts include [22,7,15,6,10,16,50,1,31,45,51].

These approaches crucially rely on sizable parts of the population using smartphones, enabling Bluetooth, and installing a new App on their phones. In contrast, our proposal does not help with contact tracing, but gives potentially useful epidemiological information to health authorities without requiring people to carry around smartphones, as any mobile phone will be sufficient, and Bluetooth is also not needed.

Mobile network operators increasingly share location data to manage the coronavirus. Example: Governments in Italy, Germany, and Austria are relying on this metadata to monitor how people are complying with stay-at-home orders[3].

In addition, the EU Commission urged Europe's biggest mobile network operators to hand over location data in the coronavirus fight[4]. In order to respect Europe's strict privacy requirements, mobile network operators only share anonymous, aggregated information that they have already gathered. Other types of such aggregated data sharing are e.g. provided by Google[5].

Although this information is useful, it has its limitations. After the process of anonymization and aggregation, it is only possible to make statements about the whole population. For example, the shared data can not be used to question whether there is a difference in behavior between people who got infected with Sars-Cov-2 and those who did not. Consequently, it is impossible with this data to detect public places where there is a significantly higher risk of getting infected with Sars-CoV-2 (hotspots). This, in turn, could be valuable knowledge in the further containment of Sars-CoV-2. The question is if there is a way to find Sars-CoV-2 hotspots without undermining privacy requirements. Landau in [39] independently also motivates increased efforts in aggregated location tracking as contact tracing may not work as well as hoped.

This research aims to construct a protocol for detecting coronavirus hotspots from mobile data in accordance with strict privacy requirements. To accomplish this, we design a specialized private information retrieval (PIR) protocol.

PIR protocols, as introduced by Chor et al. [20], aim to realize efficient methods for a client to retrieve elements from a database managed by one or many untrusted servers. Thereby, none of the involved servers should learn which element was retrieved by the client. In this paper, we are interested in the single-server variants, namely computational PIR (CPIR), which rely on cryptographic hardness assumptions to hide the query from the server. Recent work [43,14,26,38,40,41,52,3] has heavily improved on the original ideas of Chor et al. Many CPIR implementations use homomorphic encryption (HE) to hide the queries from the server while still allowing him to perform operations on the query.

HE is a cryptographic primitive that allows performing computations on encrypted data without knowing the secret decryption key. The first fully homomorphic encryption (FHE) scheme, which in theory allows computing all possible functions on encrypted data, was introduced by Gentry [29] in 2009 and has been drastically improved upon by followup publications [12,11,28,18,19]. However, the limitation of FHE schemes is that they add a substantial perfor-

[3] https://www.reuters.com/article/us-health-coronavirus-europe-telecoms/european-mobile-operators-share-data-for-coronavirus-fight-idUSKBN2152C2

[4] https://www.sciencemag.org/news/2020/03/cellphone-tracking-could-help-stem-spread-coronavirus-privacy-price

[5] https://www.google.com/covid19/mobility/

mance penalty of several orders of magnitude compared to performing equivalent computations in plain. Nevertheless, recent publications have shown, that highly optimized protocols can leverage homomorphic encryption to increase privacy of complex tasks, like machine learning [23,35,21,9,8,5,37,36].

However, PIR protocols, by definition, do not hide individual database entries from clients. In this paper, we want to go one step further by using HE to additionally aggregate multiple database entries to keep the server data private as well.

1.1 Aggregated Private Information Retrieval

We present a protocol for aggregated private information retrieval (APIR). In the use-case we will describe later, APIR would enable officials to get anonymous, aggregated information about movement records of Sars-Cov-2 patients. In the following, we first outline the main idea of our APIR protocol.

We assume without loss of generality that the first column of the database of the server consists of unique identifiers. The client's input to the protocol is a list of such ids, and the output is a vector representing the aggregated columns of the server's database with respect to the client's identifiers.

The threat model of the APIR protocol is similar to PIR protocols, i.e., the server should not know which elements were retrieved by the client. Additionally, in an APIR protocol, the client should also not be able to retrieve individual entries of the database, only receiving aggregated values is allowed. Note that the identifiers of the server are not considered to be private.

To achieve the privacy goals outlined above, we make heavy use of homomorphic encryption. In particular, the client homomorphically encrypts its identifiers before sending it to the server. Due to the nature of homomorphic encryption, the server can still perform the data aggregation. To prevent the client from learning individual entries, we make sure that the client's list of identifiers has a guaranteed minimum cardinality and that each identifier is unique. In addition, we apply differential privacy before releasing the output.

1.2 Benchmark results

We report concrete benchmarks of our protocol for parameters suitable to create a Sars-Cov-2 hotspot map for small nations with millions of subscribers and tens of 1000s of base stations. In our concrete example this corresponds to an APIR protocol applied on a database consisting of $\approx 2^{38}$ 32-bit entries. We report multithreaded runtimes of 30 minutes for the standard APIR protocol, and 1 hour when extra steps are applied to ensure the input vector is not malicious. Our benchmarks show that it is feasible to use our APIR protocol on relatively large databases to support disease analysis of small nation states or large cities while still maintaining the privacy of infected people. We also show that costs for doing the same for larger nation states can still be expected to be very modest.

1.3 Roadmap

After introducing preliminaries in Section 2, we describe our APIR protocol more formally in Section 3. In Section 4, we illustrate an application of our APIR protocol, which can have a significant impact on the understanding where Sars-Cov-2 infection hotspots can be found. Afterward, in Section 5, we will have a look at the details of our implementation as well as the benchmarks.

2 Preliminaries

In this section, we cover the preliminaries required for the rest of the paper. We will first introduce the notations we use in the rest of the paper, before we describe homomorphic encryption, private information retrieval and differential privacy.

2.1 Notation

We follow the widespread convention to write vectors in bold lower case letters and matrices in upper case letters. We use x_i to access the i-th element of vector $\boldsymbol{x}$. For $m \in \mathbb{N}$ and $x \in \mathbb{Z}$, let $\boldsymbol{x}^m$ be defined as the vector of powers of x: $\boldsymbol{x}^m = (x, x^1, ..., x^m)$. We denote by $\boldsymbol{c} \circ \boldsymbol{d}$ the element-wise multiplication (Hadamard product) of the vectors $\boldsymbol{c}$ and $\boldsymbol{d}$. For a positive integer p, we identify $\mathbb{Z}_p = \mathbb{Z} \cap [-p/2, p/2)$.

2.2 Homomorphic Encryption

The concept of homomorphic encryption (HE) has often been considered to be the holy grail in cryptography since it allows us to work on encrypted data without requiring the secret decryption key. It was first introduced by Rivest et al. [47] and partially HE schemes, i.e. schemes which allow performing a limited set of operations on encrypted data, have been known for years: The RSA [48] encryption scheme is homomorphic for multiplication and Paillier's cryptosystem [44] is homomorphic for addition. However, it was not until Gentry's groundbreaking work from 2009 [29] that we were able to construct the first fully homomorphic encryption (FHE) scheme, a scheme which in theory can evaluate an arbitrary circuit on encrypted data. His construction is based on ideal lattices and is deemed to be too impractical ever to be used, but it led the way to construct more efficient schemes in many following publications [12,11,28,18,19].

Modern HE schemes are based on the learning with errors (LWE) [46] hardness assumption, and its variant over polynomial rings, the ring learning with error (RLWE) [42] hardness assumption. In the following, we recall one variant of the RLWE hardness assumption, the Decision-RLWE hardness assumption.

Definition 1 (Decision-RLWE). *For security parameter λ, let $f(x)$ be a cyclotomic polynomial $\Phi_m(x)$ with $deg(f) = \varphi(m)$ depending on λ and set $\mathcal{R} =$*

$\mathbb{Z}[x]/(f(x))$. Let $q = q(\lambda) \geq 2$ be an integer. For a random element $s \in \mathcal{R}_q$ and a distribution $\chi = \chi(\lambda)$ over $\mathcal{R}$, denote with $A^{(q)}_{s,\chi}$ the distribution obtained by choosing a uniformly random element $a \leftarrow \mathcal{R}_q$ and a noise term $e \leftarrow \chi$ and outputting $(a, [a \cdot s + e]_q)$. The Decision-RLWE$_d, q, \chi$ problem is to distinguish between the distribution $A^{(q)}_{s,\chi}$ and the uniform distribution $U(\mathcal{R}_q^2)$. The Decision-RLWE$_d, q, \chi$ assumption is that the Decision-RLWE$_d, q, \chi$ problem is infeasible.

During the encryption of a plaintext in RLWE based schemes, random noise is introduced into the ciphertext. This noise grows with the evaluation of homomorphic operations, negligible for addition, but significantly for homomorphic multiplication. Once this noise becomes too large and exceeds a threshold, the ciphertext cannot be decrypted correctly anymore. We call such a scheme a somewhat homomorphic encryption scheme (SHE), a scheme that allows evaluating an arbitrary circuit over encrypted data up to a certain depth. The specific depth depends on the choice of encryption parameters, and choosing parameters for larger depths comes, in general, with a considerable performance penalty.

In his work [29], Gentry introduced the novel bootstrapping technique, a procedure that reduces the noise in a ciphertext and can turn a (bootstrappable) SHE scheme into an FHE scheme. However, this bootstrapping operation comes with high computational complexity. In many practical applications it is, therefore, faster to omit bootstrapping and choose a SHE scheme with large enough parameters to evaluate the desired circuit. In this work, we use the BFV [11,28] SHE scheme to homomorphically encrypt the inputs of our protocol.

We rely on HE instead of other privacy-preserving protocols, such as secure multi-party computation (MPC), due to several considerations:

- Homomorphic ciphertext-ciphertext multiplications are very costly in HE schemes, however, in our protocol we mainly rely on the cheaper plaintext-ciphertext multiplications. Therefore, all the operations involved in our protocol can be expressed relatively cheap using HE.
- HE has the advantage of outsourcing computations. After the client sends the encrypted data to the server, the server can do the computations without further data exchange with the client. MPC protocols, in contrary, have a higher number of communication rounds and all parties have to participate in the computations.
- MPC protocols usually have a higher communicational complexity, i.e., the amount of exchanged data is much higher. Especially over WAN networks, the communication alone can introduce significant delays.

2.3 Private Information Retrieval

In 1995, Chor et al. [20] introduced the notion of private information retrieval (PIR), server-client protocols which allow clients to hide queries to a database managed by one or many server, but still receive the correct elements. Thereby, the goal is to be more efficient than the straight-forward solution of letting the client download the whole database. In the literature we distinguish between two different types of PIR: information theoretic PIR (IT-PIR) [20,4,24,25,30] protocols which rely on multiple, non-colluding servers to ensure privacy; and computational PIR (CPIR) [43,14,26,38,40,41,52,3] where a single server manages the database and encryption is used hide the query.

Our approach is similar to the single-server CPIR variant, in which best-performing CPIR protocols [43,3] use homomorphic encryption to ensure that the server is still able to compute on the query to provide the final result. One major issue with CPIR protocols is that the server must process all entries of the database to answer a single query. If this would not be the case, the server must somehow have learned, which entries the client is not interested in, and thus, leaked information about the query. The computation cost of a server is, therefore, linear in the database size, which limits the practicability of CPIR to small databases.

CPIR using homomorphic encryption boils down to essentially evaluate a vast vector-matrix multiplication on a one-hot encoded input vector. State-of-the-art CPIR implementation employs many optimizations to make retrieving single elements very feasible for limited database sizes. In this work, however, we are not interested in retrieving only one item; in fact, we want to ensure that single database entries are kept hidden from the client as well. We, therefore, employ homomorphic encryption to retrieve multiple aggregations of many database entries at once and invalidate responses for malicious queries. Furthermore, we want to implement use cases involving a huge database consisting of $\approx 2^{38}$ 32-bit entries.

2.4 Differential Privacy

Another thing to consider when designing a privacy-preserving data analytics protocol is that the result which the client obtains in plain can still leak information about the underlying dataset. In our case, an aggregated PIR protocol, the data aggregation can leak information about single database entries. We can use the well-established notion of differential privacy to help protect against such kind of information leakage. Let us recall the definition of ϵ-differential privacy [27]:

Lemma 1 (ϵ-Differential Privacy). *A randomized mechanism $\mathcal{A}$ gives ϵ-differential privacy if for any neighboring datasets D and D', and any $S \in Range(\mathcal{A})$: $Pr[\mathcal{A}(D) = S] \leq e^{\epsilon} Pr[\mathcal{A}(D') = S]$.*

Since D and D' are interchangeable, Lemma 1 implies:

$$e^{-\epsilon} \leq \frac{Pr[\mathcal{A}(D) = S]}{Pr[\mathcal{A}(D') = S]} \leq e^{\epsilon} \tag{1}$$

An established technique to achieve ϵ-differential privacy is the Laplace mechanism, i.e., to add Laplacian noise to the final result of the computation. The noise is, thereby, calibrated with the privacy budget ϵ and the global sensitivity Δq of the computation q: $\Delta q = \max_{D,D'} ||q(D) - q(D')||$ for all neighboring D and

D'. In other words, the global sensitivity represents the maximum possible value of each element in the dataset. The Laplace distribution for a scale factor b is given as $Lap(x|b) = \frac{1}{2b} e^{-\frac{|x|}{b}}$, in the Laplace mechanism a scale factor of $b = \frac{\Delta q}{\epsilon}$ is used.

We can apply differential privacy to our APIR protocol by adding Laplacian noise to each element of the final result of the matrix multiplication $\boldsymbol{v}^T \cdot Z$:

$$\boldsymbol{h}^T = \boldsymbol{v}^T \cdot Z + \boldsymbol{\ell}^T \tag{2}$$

with $\ell_i \xleftarrow{\$} Lap(\frac{\Delta q_i}{\epsilon})$ for $0 \leq i < k$.

We note that in general, each column of the matrix multiplication can represent a different dataset with a different sensitivity Δq_i. However, in the use case we describe in our work, each Δq_i is equal.

3 APIR

In Section 3.1, we give a high-level description of our protocol below. A formal definition of the protocol is provided in Figure 1. In addition, we illustrate our protocol in Figure 2 including an interactive proof to protect against malicious queries described later in this section. Then we discuss in Section 3.5 the concrete privacy and a generalization of our APIR protocol in Section 3.6.

3.1 Protocol

We use a HE scheme $\mathsf{HE} = (\mathsf{HE.KGen}, \mathsf{HE.Enc}, \mathsf{HE.Dec}, \mathsf{HE.Eval})$. Before executing the protocol, the client runs KGen to obtain an evaluation key evk and a private key sk. We assume that the server knows evk before running the protocol.

First, observe that according to our assumptions, the server is allowed to share the identifiers with the client. We can exploit this fact by server not only sending the identifiers but also the indices of the identifiers in the database. In this case, the client not only gets to know which identifiers it has in common with the server but also their exact location in the database of server. This information is stored in the vector $\boldsymbol{v} = \begin{bmatrix} v_1 \cdots v_N \end{bmatrix}$. Sending $\boldsymbol{v}$ to the server would stand in contradiction to our threat model because the server would directly learn the identifiers of the client. Therefore, the client homomorphically encrypts the vector v before sending it to the server. The server can not learn anything from the encrypted vector provided that the HE scheme is semantically secure.

Remark. In some applications the list of identifiers in the server database might also be considered private information. In such cases, one can augment this step by using a variant of Private Set Intersection (PSI) with associated data transfer. In particular, the labeled PSI protocol in [17] provides the required features while conveniently being based on Fully Homomorphic Encryption.

Public inputs: Client and server agree on parameters $N, n, W, s \in \mathbb{N}, \epsilon \in \mathbb{R}_{>0}$.
Private inputs: The client has input $(x_1, \ldots, x_\nu) \in \{0,1\}^{n \times \nu}$ with $\nu \in \mathbb{N}$, and keys $(\mathsf{evk}, \mathsf{sk})$ for a homomorphic encryption scheme. The server has input $(y_1, \ldots, y_N) \in \{0,1\}^{n \times N}$, $Z = (z_{ij}) \in \mathbb{Z}^{N \times k}$ with $k \in \mathbb{N}$ and knows evk.

Protocol:

1. Server sends $\{y_i, i\}_{i \in [N]}$ to the client.
2. Client sets $v_i = 1$ if $\{x_1, \ldots, x_\nu\} \cap y_i \neq \emptyset$ and 0 otherwise, for all $i \in [N]$.
3. Client computes $c_i \leftarrow \mathsf{HE.Enc}_{\mathsf{sk}}(v_i)$ for all $i \in [N]$, and sends $\{c_i\}_{i \in [N]}$ to server.
4. Server sets $\boldsymbol{d} \leftarrow \mathsf{HE.Eval}_{\mathsf{evk}}\left(\boldsymbol{c} - \mathbf{1}^N\right)$
   If *interactive* == TRUE:
      Server chooses $z_i, y_i, r_i, \rho_i \xleftarrow{\$} \mathbb{Z}_p$, computes
      $$\mu_i = \mathsf{HE.Eval}_{\mathsf{evk}}(z_i \cdot (\langle \boldsymbol{c}, \mathbf{1}^N \rangle - W) + \langle \boldsymbol{c}, \boldsymbol{d} \circ \boldsymbol{y_i}^N \rangle)) \cdot r_i + \rho_i,$$
      and sends $\boldsymbol{\mu} = \begin{bmatrix}\mu_1 \cdots \mu_s \; 0 \cdots 0\end{bmatrix}^T \in \mathbb{Z}_p^k$ to the client
   Else:
      Server chooses $\boldsymbol{r} \xleftarrow{\$} \mathbb{Z}_p^k$, and computes
      $$\boldsymbol{\mu} \leftarrow \mathsf{HE.Eval}_{\mathsf{evk}}\left((z \cdot (\langle \boldsymbol{c}, \mathbf{1}^N \rangle - W) + \langle \boldsymbol{c}, \boldsymbol{d} \circ \boldsymbol{y}^N \rangle) \cdot \boldsymbol{r}\right).$$
5. If *interactive* == TRUE:
      Client decrypts $\boldsymbol{\rho'} \leftarrow \mathsf{HE.Dec}_{\mathsf{sk}}(\boldsymbol{\mu})$, and sends $\boldsymbol{\rho'}$ to the server.
6. If *interactive* == TRUE:
      If $\boldsymbol{\rho'} \neq \boldsymbol{\rho} = \begin{bmatrix}\rho_1 \cdots \rho_s \; 0 \cdots 0\end{bmatrix}^T$ server aborts, otherwise server sets $\boldsymbol{\mu} \leftarrow \mathbf{0}$.
   Sever computes
   $$h_j^* \leftarrow \mathsf{HE.Eval}_{\mathsf{evk}}\left(\left\langle \begin{bmatrix} c_1 \\ \vdots \\ c_N \end{bmatrix}, \begin{bmatrix} z_{1j} \\ \vdots \\ z_{Nj} \end{bmatrix} \right\rangle + \mu_j\right).$$
7. Server chooses $\ell_j \xleftarrow{\$} Lap(\frac{\Delta q}{\epsilon})$ for $j \in [k]$, computes
   $$h_j^* \leftarrow \mathsf{HE.Eval}_{\mathsf{evk}}\left(h_j^* + \ell_j\right),$$
   and sends $\{h_j^*\}_{j \in [k]}$ to the client.
8. Client computes $h_j \leftarrow \mathsf{HE.Dec}_{\mathsf{sk}}(h_j^*)$ for $j \in [k]$ and outputs $\{h_j\}_{j \in [k]}$.

Fig. 1: Full description of our APIR protocol.

As mentioned before, we want to ensure that the client does not get the values of individual rows. Therefore, the server checks that the hamming weight of $\boldsymbol{c}$ is not too low. Depending on the specific use of the protocol, the threshold W can vary. This precaution alone is not enough against a malicious client who could also send an arbitrary non-binary vector. As an example, by sending a vector of all zeros except one coordinate equal to the minimum hamming weight W, the client would still learn the values of this particular row. To prevent such abuse, the server also checks that the vector $\boldsymbol{c}$ is binary. Only if both conditions are satisfied, the following data aggregation is valid. Technically, we developed two variants of how the server can check the correctness of the query vector $\boldsymbol{v}$.

One can detect a malicious client before doing the costly matrix multiplication by introducing one additional round of communication, the other invalidates the result of our APIR protocol for malicious queries. For details see Section 3.3 and Section 3.4.

The data aggregation is done by computing a large matrix multiplication. The vector $\boldsymbol{c}$ contains the information, which rows should part of the sum. Since $\boldsymbol{c}$ is encrypted, according to the security guarantees of the HE scheme it does not reveal any information to the server, however, by using Eval and evk, the server can still perform the matrix multiplication. We add a small random noise to the result of the homomorphic matrix multiplication for privacy reasons (see Section 2.4). By decrypting h^* with its private key sk, the client gets the final output of the protocol.

3.2 Proving that the Query Vector is Binary

One assumption of the protocol is that it requires the input vector to be binary. If this is not the case, the client can arbitrarily modify the contribution of a single person to the overall aggregated result, which can leak private information. Since the server only receives an encryption of the input vector, simply checking for binary values is not an option. However, we can use similar techniques to the ones used in Bulletproofs [13] to provide assurance that the query vector $\boldsymbol{c}$ is made up of binary elements, i.e., $\forall i : c_i \in \{0, 1\}$.

Lemma 2. *Let $\boldsymbol{c} \in \mathbb{Z}^N$ be the query vector and let $\boldsymbol{d} = \boldsymbol{c} - \mathbf{1}$. If the following condition holds, $\boldsymbol{c}$ is a binary vector $\boldsymbol{c} \in \mathbb{Z}_2^N$:*

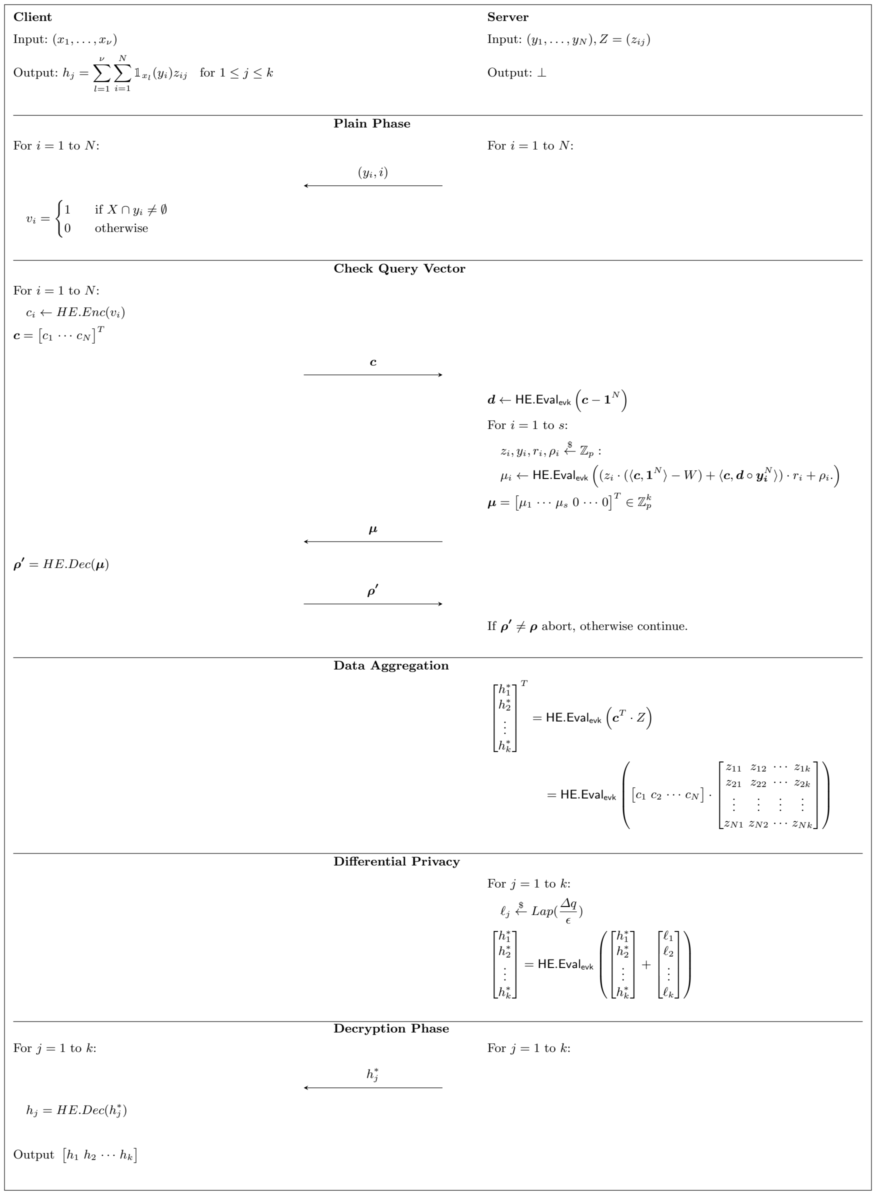
$$\boldsymbol{c} \circ \boldsymbol{d} = \mathbf{0}^N .$$

Note in our scenario, $\boldsymbol{d}$ can be computed by the server. However, the server needs to compare a homomorphically encrypted vector to zero to check Lemma 2, which is not trivially possible. Therefore, we want to encode this comparison as a mask $\boldsymbol{\mu}_{\mathtt{bin}} \in \mathbb{Z}_p^k$, which will either be zero if $\boldsymbol{c} \in \mathbb{Z}_2^N$, random otherwise. Adding $\boldsymbol{\mu}_{\mathtt{bin}}$ to the response of the server will invalidate the result for malicious input vectors, but do not affect correct queries.

The result of $\boldsymbol{c} \circ \boldsymbol{d}$ can be aggregated into a single value by calculating the inner product $\langle \boldsymbol{c}, \boldsymbol{d} \rangle$, which will again be zero if $\boldsymbol{c} \in \mathbb{Z}_2^N$. The server then adds a random value $y \xleftarrow{\$} \mathbb{Z}_p$ to reduce the probability for the client to cheat by letting several entries of $\boldsymbol{c}$ cancel each other out during the inner product. A final randomization with a vector $\boldsymbol{r} \xleftarrow{\$} \mathbb{Z}_p^k$, produces the final mask:

$$\boldsymbol{\mu}_{\mathtt{bin}} = \langle \boldsymbol{c}, \boldsymbol{d} \circ \boldsymbol{y}^N \rangle \cdot \boldsymbol{r} . \tag{3}$$

For the generic case of a vector $\boldsymbol{b}$ and a randomly chosen y, $\langle \boldsymbol{b}, \boldsymbol{y}^N \rangle = 0$ will hold for a $\boldsymbol{b} \neq 0$ only with probability N/p [13]. Using a 60 bit prime, i.e. $p \approx 2^{60}$, for $N = 2^{23}$, this probability is $Pr \approx \frac{2^{23}}{2^{60}} = 2^{-37}$, i.e. a soundness error of 37 bits.



Client

Input: $(x_1, \dots, x_\nu)$

Output: $h_j = \sum_{l=1}^{\nu} \sum_{i=1}^{N} \mathbb{1}_{x_l}(y_i) z_{ij}$ for $1 \le j \le k$

Server

Input: $(y_1, \dots, y_N), Z = (z_{ij})$

Output: $\perp$

Plain Phase

Client: For $i = 1$ to N:

Server: For $i = 1$ to N:

$\xleftarrow{(y_i, i)}$

$$v_i = \begin{cases} 1 & \text{if } X \cap y_i \neq \emptyset \\ 0 & \text{otherwise} \end{cases}$$

Check Query Vector

Client:

For $i = 1$ to N:

$\quad c_i \leftarrow HE.Enc(v_i)$

$\boldsymbol{c} = \begin{bmatrix} c_1 \cdots c_N \end{bmatrix}^T$

$\xrightarrow{\boldsymbol{c}}$

Server:

$\boldsymbol{d} \leftarrow \mathsf{HE.Eval}_{\mathsf{evk}} \left(\boldsymbol{c} - \mathbf{1}^N \right)$

For $i = 1$ to s:

$\quad z_i, y_i, r_i, \rho_i \overset{\$}{\leftarrow} \mathbb{Z}_p$:

$\quad \mu_i \leftarrow \mathsf{HE.Eval}_{\mathsf{evk}} \left((z_i \cdot (\langle \boldsymbol{c}, \mathbf{1}^N \rangle - W) + \langle \boldsymbol{c}, \boldsymbol{d} \circ \boldsymbol{y}_{\boldsymbol{i}}^N \rangle) \cdot r_i + \rho_i . \right)$

$\boldsymbol{\mu} = \begin{bmatrix} \mu_1 \cdots \mu_s \ 0 \cdots 0 \end{bmatrix}^T \in \mathbb{Z}_p^k$

$\xleftarrow{\boldsymbol{\mu}}$

Client: $\boldsymbol{\rho}' = HE.Dec(\boldsymbol{\mu})$

$\xrightarrow{\boldsymbol{\rho}'}$

Server: If $\boldsymbol{\rho}' \neq \boldsymbol{\rho}$ abort, otherwise continue.

Data Aggregation

Server:

$$\begin{bmatrix} h_1^* \\ h_2^* \\ \vdots \\ h_k^* \end{bmatrix}^T = \mathsf{HE.Eval}_{\mathsf{evk}} \left(\boldsymbol{c}^T \cdot Z \right)$$

$$= \mathsf{HE.Eval}_{\mathsf{evk}} \left(\begin{bmatrix} c_1 \ c_2 \cdots c_N \end{bmatrix} \cdot \begin{bmatrix} z_{11} & z_{12} & \cdots & z_{1k} \\ z_{21} & z_{22} & \cdots & z_{2k} \\ \vdots & \vdots & \vdots & \vdots \\ z_{N1} & z_{N2} & \cdots & z_{Nk} \end{bmatrix} \right)$$

Differential Privacy

Server:

For $j = 1$ to k:

$\quad \ell_j \overset{\$}{\leftarrow} Lap(\frac{\Delta q}{\epsilon})$

$$\begin{bmatrix} h_1^* \\ h_2^* \\ \vdots \\ h_k^* \end{bmatrix} = \mathsf{HE.Eval}_{\mathsf{evk}} \left(\begin{bmatrix} h_1^* \\ h_2^* \\ \vdots \\ h_k^* \end{bmatrix} + \begin{bmatrix} \ell_1 \\ \ell_2 \\ \vdots \\ \ell_k \end{bmatrix} \right)$$

Decryption Phase

Client: For $j = 1$ to k:

Server: For $j = 1$ to k:

$\xleftarrow{h_j^*}$

Client: $\quad h_j = HE.Dec(h_j^*)$

Output $\begin{bmatrix} h_1 \ h_2 \cdots h_k \end{bmatrix}$

Fig. 2: Illustration of our APIR protocol with interactive proof for an honest query vector.

3.3 Proving the Hamming Weight of the Query Vector

Another problem of our protocol is that the client can target the values of single people by querying the server with an input vector of hamming weight $W =$

1. Again, since the query is encrypted, the server can not trivially check the hamming weight of the input vector. However, we can apply similar techniques as in the previous section to incorporate a hamming weight check into a masking value.

Again let $\boldsymbol{c}$ be the query vector and let W be its announced hamming weight. On the server-side, calculate $\langle \boldsymbol{c}, \mathbf{1}^N \rangle$, which is equal to the hamming weight of $\boldsymbol{c}$. Therefore, with a random vector $\boldsymbol{r} \xleftarrow{\$} \mathbb{Z}_p^k$, the following mask $\boldsymbol{\mu}_{\texttt{HW}} \in \mathbb{Z}_p^k$ is zero if $\boldsymbol{c}$ has the announced hamming weight W, random otherwise:

$$\boldsymbol{\mu}_{\texttt{HW}} = (\langle \boldsymbol{c}, \mathbf{1}^N \rangle - W) \cdot \boldsymbol{r} \,. \tag{4}$$

We then combine the two separate proofs in equation 3 and equation 4 into a combined masking value $\boldsymbol{\mu}$, adding a random value $z \xleftarrow{\$} \mathbb{Z}_p$ to avoid the separate masks canceling each other out:

$$\boldsymbol{\mu} = (z \cdot (\langle \boldsymbol{c}, \mathbf{1}^N \rangle - W) + \langle \boldsymbol{c}, \boldsymbol{d} \circ \boldsymbol{y}^N \rangle) \cdot \boldsymbol{r} \tag{5}$$

$\boldsymbol{\mu}$ is now equal to $\mathbf{0}^k$ if $\boldsymbol{c}$ is a binary vector with hamming weight W, random otherwise. A vector $\boldsymbol{c}$ not fulfilling these conditions will result in a masking value of $\mathbf{0}^k$ only with probability $\frac{N+1}{p}$.

3.4 Interaction vs. Masking

The masks calculated in the previous section is finally added to the result of the matrix multiplication to invalidate the result for malicious queries. However, one could also perform an additional round of interaction between the client and the server as follows: Calculate a challenge value like the previous mask, however, instead of using a random vector $\boldsymbol{r} \xleftarrow{\$} \mathbb{Z}_p^k$, we multiply the mask with a random scalar $r \xleftarrow{\$} \mathbb{Z}_p$ and also add a final random value $\rho \xleftarrow{\$} \mathbb{Z}_p$:

$$\mu_{\texttt{chal}} = (z \cdot (\langle \boldsymbol{c}, \mathbf{1}^N \rangle - W) + \langle \boldsymbol{c}, \boldsymbol{d} \circ \boldsymbol{y}^N \rangle) \cdot r + \rho \tag{6}$$

The server then sends this ciphertext as a challenge to the client, who decrypts it and sends back the result. Only if the sent value is equal to ρ, the matrix multiplication is carried out and the result is sent back to the client. This approach has multiple advantages: First, at the cost of one more round of interaction and the associated communication, the server can detect malicious input vectors before carrying out the expensive matrix multiplication. Furthermore, this approach can also be repeated in parallel to boost the soundness.

With a soundness error of $\frac{N+1}{p}$, repeating the challenge t times reduces the probability to cheat to $\left(\frac{N+1}{p}\right)^t$. Therefore, we require at least s challenges in parallel to achieve a soundness error of at least the security level λ (in bits), with s being:

$$s = \left\lceil \frac{-\lambda}{log_2(\frac{N+1}{p})} \right\rceil \tag{7}$$

3.5 Privacy

For the privacy of APIR, the selection of parameters is of outermost importance. By using the APIR protocol in an application, the parameters n, k, and N get fixed by the specific data sets, whereas the parameters ϵ and W can be freely chosen. The concrete privacy - in particular for the items in the client's database - is highly dependent on those two parameters. An anonymization expert should choose them with knowledge of the specific data sets and the actual application. The aim here is that the parameters are selected in a way that, from the output, no re-identification is possible but correctness and usefulness of the aggregated results is maintained.

3.6 Generalized View of APIR

The data aggregation of our APIR protocol is presented in a straightforward fashion. We use the simplest form of aggregations; namely, just adding all the information requested from the client, which corresponds to a single big matrix multiplication. Depending on the use-case, one could think of more involved methods. The computation on the server-side can consist of any combination of (matrix) multiplications, (matrix) additions as well as rotations of single rows or columns. In this light, APIR can be seen not only as a protocol but rather as a useful framework for privacy-preserving data analytics.

4 Application: National Scale Sars-CoV-2 Heat Map

Our APIR protocol can help to learn more about the behavioral patterns of people tested positive with Sars-CoV-2. Note that the whole objective of our approach is to gather this information without violating the privacy of patients or enabling the tracking of individuals. In this section, we describe how the generic protocol in Figure 1 can be used to create a private heat map of Sars-CoV-2 positive people at the time of their infection. Such a heat map could lead to a better understanding of Sars-CoV-2 hotspots, i.e., places where there is a significantly higher risk of getting infected with Sars-CoV-2. This, in turn, could be valuable knowledge in the further containment of Sars-CoV-2.

4.1 Heat Map Protocol

The parties involved in creating a Sars-CoV-2 heat map are a mobile network operator (M) and a health authority (H). An illustration of this application can be found in Figure 3. M takes the role of the server, where the identifiers are mobile phone numbers. Let N be the number of all mobile phone numbers registered by M and n the length to represent a mobile phone number. The matrix Z contains location data. More precisely, each row is the location data for one particular mobile phone number. For simplification, we assume that the location data is represented as follows. The value z_{ij} is the amount of time spent by person i in the cell site j. Therefore, k is the number of cell towers operated by M. On the other side, the role of the client is taken by H. The input of H is a list of ν mobile phone numbers of Sars-CoV-2 patients. The output of the protocol is a vector $\begin{bmatrix} h_1 \cdots h_k \end{bmatrix}$, where h_j is the total amount of time spent by all the Sars-CoV-2 patients in the cell site j.

Note that, as we have already mentioned in Section 3.6, our APIR protocol should be seen more as a framework. This means that with little adaptation, our APIR protocol can be used for many different analytical methods. In particular, we can exchange the data aggregation (matrix multiplication) by different computations to support a larger class of epidemiologically useful statistics. The heat map is only one concrete instantiation of this framework.

We now want to have a look at the privacy features of the application inherited from them generic protocol. Recall, the generic protocol has two checks to ensure that H does not learn the values of any row, i.e., H should not learn the movement pattern of any individual. More concretely, H is not allowed to query the location data for less than W different people. W has to be chosen in such a way that the data aggregation provides anonymity and its exact value will highly depend on the actual underlying data, which is the reason why we do not give a generic value in this paper. We urge users of our protocol to rely on an expert in the field of de-anonymization with sufficient knowledge of location data to choose a suitable value. To sum up, by using our protocol correctly, highly sensitive personal data is protected. In particular, M does not learn which people are infected by Sars-CoV-2. On the other side, H does not get the movement record of individual patients.

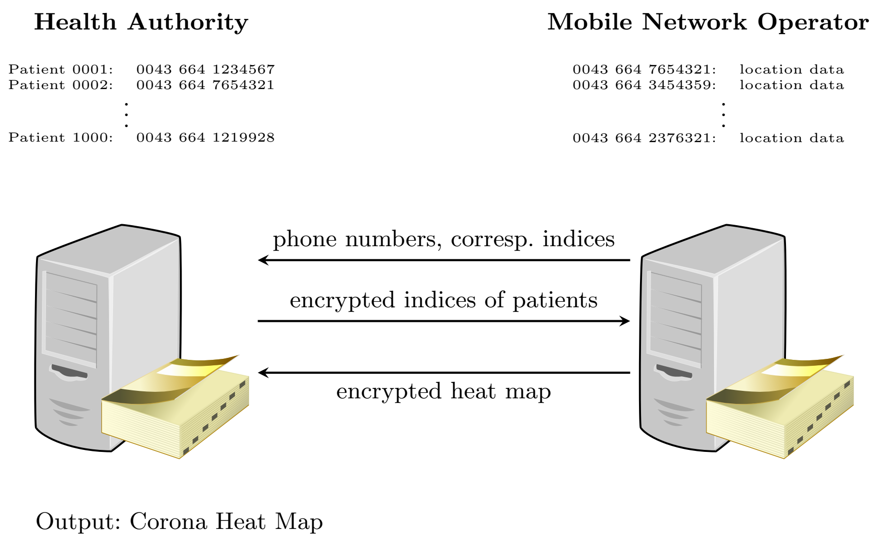

Health Authority
Patient 0001: 0043 664 1234567
Patient 0002: 0043 664 7654321
Patient 1000: 0043 664 1219928
Mobile Network Operator
0043 664 7654321: location data
0043 664 3454359: location data
0043 664 2376321: location data
phone numbers, corresp. indices
encrypted indices of patients
encrypted heat map
Output: Corona Heat Map

Fig. 3: Private Sars-CoV-2 Heat Map.

4.2 Practical Considerations

Preprocessing. A preprocessing phase on the side of the mobile network provider could increase the value of the heat map as well as provide even more privacy. First, M could try to find the place of residence by assuming people sleep at home. Therefore, they can look at the location of people at night and remove those places for the heat map creation. Once M has completed this step, the heat map will additionally be more informative in the sense of H getting a better picture, which public places might bear a high risk of getting infected with Sars-CoV-2. Secondly, to further enhance the privacy of patients, M should filter out isolated profiles. Such filtering of the data obviously does not affect the meaningfulness of the heat map, since crowded public places would not be affected. Note that both preprocessing steps are done on the side of the mobile network operator. That means those steps can be done on non-encrypted data and therefore are negligible when it comes to the runtime of our protocol.

Applying Differential Privacy. An attack vector for H in our protocol can be to abuse the heatmap to track individual people by querying for isolated profiles. For example, they want to query for a person living in the westmost area of the country. To overcome the hamming weight check on the server-side, they additionally query for $W - 1$ people living on the east side of the country. The location data of the targeted person is clearly shown on the resulting heatmap; H just has to look at the western part of the result.

Applying differential privacy with suitable parameters will protect against such an attack since the overall goal of differential privacy is to decrease the statistical dependence of the final result to a single database entry as much as possible. In our use-case, therefore, differential privacy achieves that it is infeasible to distinguish between heatmaps, in which we include a single person in the accumulation and heatmaps in which we do not.

A general issue of differential privacy is that the privacy guarantees get weaker the more queries the client is allowed to do. This, however, is no issue in our use case, since the location data changes every couple of days, and we can limit H to query a heatmap to, e.g., once a day.

Choosing proper parameters, however, highly depends on the underlying dataset. On the one hand, the chosen ϵ should be small enough to satisfy privacy concerns; on the other hand, it should be big enough not to overflow the result with noise. In our protocol, accumulations of a sufficient amount of people should not be affected by the noise, i.e., the noise on its own should not be able to create hotspots. Again, we urge users of our protocol to rely on an expert in the field of de-anonymization with sufficient knowledge of location data to choose suitable values for the application of differential privacy.

5 Implementation and Performance

We implemented our protocol using the BFV [11,28] homomorphic encryption scheme, more specifically its implementation in the SEAL v3.4 [49] library. SEAL is an actively developed open-source library maintained by Microsoft Research compatible with all major operating systems, including Windows, Linux, and OS X.

The computationally most expensive phase in the protocol is the Data Aggregation phase, in which the server multiplies a huge matrix to a homomorphically encrypted input vector. Therefore, the main objective of our implementation is to perform this huge matrix multiplication as efficiently as possible.

5.1 Packing

Modern HE schemes allow for packing a vector of n plaintexts into only one ciphertext. Performing an operation on this ciphertext then is implicitly applied to each slot of the encrypted vector, similar to single-instruction-multiple-data (SIMD) instructions on modern CPU's (AVX, SSE2, etc.). However, the size of the ciphertext does not depend on the exact number ($\leq n$) of plaintexts encoded. The HE schemes support a variety of SIMD operations, including slot-wise addition, subtraction and multiplication, and slot-rotation. However, one can not directly access a specific slot of the encoded vector. We can use the SIMD encoding to speed up the matrix multiplication of our protocol significantly.

In the BFV scheme (and its implementation in the SEAL library), the number of available SIMD slots is equal to the degree of the cyclotomic reduction polynomial ($x^n + 1$); thus, it is always a power of two. In the ciphertexts, the n slots are arranged as matrix of dimensions ($2 \times n/2$). A ciphertext rotation affects either all rows, or all columns of the matrix simultaneously. Therefore, we can think of the inner matrix as two rotatable vectors, which can be swapped. Figure 4 highlights the inner structure of the ciphertext and the rotation operations.

5.2 Baby-Step Giant-Step Matrix Multiplication

The SIMD encoding can be used to efficiently speed up matrix multiplication by using the diagonal method introduced by Halevi and Shoup in [32]. They have shown that a matrix-vector multiplication of a matrix $Z \in \mathbb{Z}^{m \times m}$ and vector $\boldsymbol{v} \in \mathbb{Z}^m$ can be expressed by m elementwise vector-vector multiplications, $m-1$ rotations, and $m-1$ additions, operations that can easily be evaluated in an HE scheme:

$$Z \cdot \boldsymbol{v} = \sum_{i=0}^{m-1} \texttt{diag}(M, i) \circ \texttt{rot}(\boldsymbol{v}, i) \tag{8}$$

$\texttt{diag}(Z, i)$ in equation 8 expresses the i-th diagonal of matrix Z in a vector of size m and $\texttt{rot}(\boldsymbol{v}, i)$ rotates the vector $\boldsymbol{v}$ by index i to the left.

However, rotations are very expensive in terms of computational effort in the BFV encryption scheme. Luckily, the diagonal method can further be improved

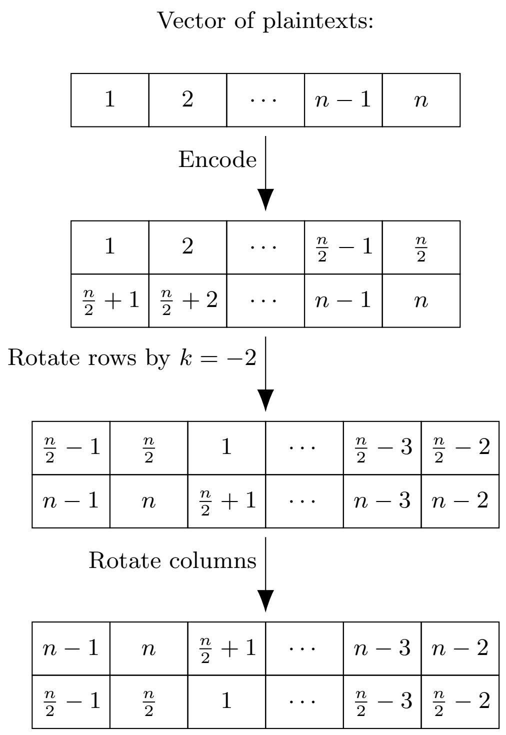

Vector of plaintexts:
1 | 2 | ⋯ | n − 1 | n
Encode
1 | 2 | ⋯ | n/2 − 1 | n/2
n/2 + 1 | n/2 + 2 | ⋯ | n − 1 | n
Rotate rows by k = −2
n/2 − 1 | n/2 | 1 | ⋯ | n/2 − 3 | n/2 − 2
n − 1 | n | n/2 + 1 | ⋯ | n − 3 | n − 2
Rotate columns
n − 1 | n | n/2 + 1 | ⋯ | n − 3 | n − 2
n/2 − 1 | n/2 | 1 | ⋯ | n/2 − 3 | n/2 − 2

Fig. 4: Effect of rotations to the inner structure of a BFV ciphertext with n slots.

by applying the baby-step giant-step algorithm [33,34]:

$$\begin{aligned} Z \cdot \boldsymbol{v} &= \sum_{i=0}^{m-1} \mathtt{diag}(Z, i) \circ \mathtt{rot}(\boldsymbol{v}, i) \\ &= \sum_{k=0}^{m_2-1} \sum_{j=0}^{m_1-1} \mathtt{diag}(Z, km_1 + j) \circ \mathtt{rot}(\boldsymbol{v}, km_1 + j) \\ &= \sum_{k=0}^{m_2-1} \mathtt{rot}\left(\sum_{j=0}^{m_1-1} \mathtt{diag}'(Z, km_1 + j) \circ \mathtt{rot}(\boldsymbol{v}, j), km_1 \right) \end{aligned} \tag{9}$$

where $m = m_1 \cdot m_2$ and $\mathtt{diag}'(Z, i) = \mathtt{rot}\left(\mathtt{diag}(Z, i), -\lfloor i/m_1 \rfloor \cdot m_1\right)$.[6] Note, that $\mathtt{rot}(\boldsymbol{v}, j)$ only has to be computed once for each $j < m_1$, therefore, equation 9 only requires $m_1 + m_2 - 2$ rotations of the vector $\boldsymbol{v}$ in total.

[6] In equation 9, $\lfloor i/m_1 \rfloor$ is equal to k.

Trivially, we can use the following equation to implement a $\boldsymbol{v}^T \cdot Z$ multiplication, like we use in our protocol:

$$\begin{aligned}(\boldsymbol{v}^T \cdot Z)^T &= Z^T \cdot \boldsymbol{v} \\ &= \sum_{k=0}^{m_2-1} \texttt{rot}\left(\sum_{j=0}^{m_1-1} \texttt{diag}'(Z^T, km_1 + j) \circ \texttt{rot}(\boldsymbol{v}, j), km_1\right)\end{aligned} \tag{10}$$

5.3 Homomorphic $N \times k$ Matrix Multiplication

In our protocol we want to homomorphically evaluate $\boldsymbol{v}^T \cdot Z$, where $\boldsymbol{v} \in \{0,1\}^N$ and $Z \in \mathbb{Z}_p^{N\times k}$, for big parameters N and k. As described in Section 5.1, the inner structure of the BFV ciphertext consists of two vectors of size $n/2$ each, and it does not allow a cyclic rotation over the whole input vector of size n. However, a rotation over the whole input vector is required by the baby-step giant-step algorithm. Therefore, we only can perform a baby-step giant-step multiplication with a $(n/2 \times n/2)$ matrix using this packing. Fortunately, we can use the remaining $n/2$ slots (i.e., the second vector in the inner structure of the BFV ciphertext) to perform a second $(n/2 \times n/2)$ matrix multiplication simultaneously. Therefore, after a homomorphic baby-step giant-step matrix multiplication, the result is a ciphertext, where each of the two inner vectors encodes the result of a $(1 \times n/2) \times (n/2 \times n/2)$ vector-matrix multiplication. The sum of those two vectors can easily be obtained by rotating the columns of the ciphertext and adding it to the first result:

$$c_{sum} = c + \texttt{rot}_{\texttt{col}}(c) \tag{11}$$

Thus, we can use one $(n/2 \times n/2)$ baby-step giant-step matrix multiplication and equation 11 to implement a homomorphic $(1 \times n) \times (n \times n/2) = (1 \times n/2)$ vector-matrix multiplication.

Taking this into account, we split the huge $(N \times k)$ matrix into $n_v \cdot n_o$ submatrices of size $(n \times n/2)$, with $n_v = \lceil \frac{N}{n} \rceil$ and $n_o = \lceil \frac{2k}{n} \rceil$, padding the submatrices with zeros if necessary. We split the input vector $\boldsymbol{v}$ into n_v vectors of size n (padding the last vector with zeros if necessary) and encrypt each of these vectors to get n_v ciphertexts c_i. The final result of the $\boldsymbol{v}^T \cdot Z$ matrix multiplication can be computed with the following equation:

$$\tilde{c}_i = \sum_{j=0}^{n_v-1} \texttt{MatMul}(\texttt{SubMat}(Z, j, i)^T, c_j) \;\; \forall 0 \le i < n_o \tag{12}$$

where, $\texttt{SubMat}(M, j, i)$ returns the submatrix of Z with size $(n \times n/2)$, starting at row $n \cdot j$ and column $\frac{n}{2} \cdot i$, and $\texttt{MatMul}(Z, c)$ performs the homomorphic baby-step giant-step matrix multiplication $Z \cdot c$ followed by equation 11.

Equation 12 produces n_o ciphertexts $\tilde{c}_i$, with the final results being located in the first $n/2$ slots of the ciphertexts. Overall, our algorithm to homomorphically calculate $\boldsymbol{v}^T \cdot Z$ requires $n_v \cdot n_o$ baby-step giant-step matrix multiplications and the total multiplicative depth is 1 plaintext-ciphertext multiplication.

5.4 Homomorphic Evaluation of the Masking Value

To calculate the masking value (equation 5), or the challenge (equation 6), we need to calculate the inner product of two homomorphically encrypted ciphertexts c and d. After an initial multiplication $c \cdot d$, the inner product requires $\log_2(n/2)$ rotations and addition, followed by equation 11 to produce a ciphertext, where the result is encoded in each of the n slots.

Our implementation uses rejection sampling and the SHAKE128 algorithm to cryptographically secure sample all the required random values in $\mathbb{Z}_p$. The total multiplicative depth to homomorphically evaluate the final mask (equation 5 or equation 6) is 1 ciphertext-ciphertext multiplication and 2 plaintext-ciphertext multiplications.

If we want to boost soundness in the interactive version of the challenge mask, we can encode multiple values of different $\mu_{\texttt{chal}}$ into the SIMD slots of one ciphertext, filling the remaining slots with zeros:

$$\boldsymbol{\mu} = (\mu_{\texttt{chal},0}, \ldots, \mu_{\texttt{chal},s-1}, 0, \ldots, 0) \tag{13}$$

We can evaluate equation 13 by multiplying a vector $\boldsymbol{r} = (r_i, 0, \ldots, 0)$ instead of a scalar r in equation 6 to get $\boldsymbol{\mu}_{\texttt{chal},i}$ and add rotations of those individual masks:

$$\boldsymbol{\mu} = \sum_{i=0}^{s-1} \texttt{rot}(\boldsymbol{\mu}_{\texttt{chal},i}, -i) \tag{14}$$

5.5 BFV Parameters

In BFV, one can choose three different parameters which greatly impact the runtime and the available noise budget (i.e. how much further noise can be introduced until decryption will fail):

– Plaintext modulus p: p defines the Ring $\mathbb{Z}_p$ to which the homomorphic operations correspond to. Every result encoded in the ciphertext vector will be an element of $\mathbb{Z}_p$. Therefore, one has to make sure that p is big enough, such that no computation overflows. On the other hand, a big p has a bad impact on the ciphertext noise, where the noise cost of homomorphic operations is higher for bigger p. Additionally, the size of p will also affect the runtime of homomorphic operations. In general, SEAL allows arbitrary plaintext moduli $t \geq 2 \in \mathbb{Z}$; however, if we want to enable SIMD-packing (Section 5.1), then the plaintext modulus has to be a prime p and congruent to 1 (mod $2n$).

– Ciphertext modulus q: q defines the available noise budget. Therefore, a bigger q allows for a bigger depth in homomorphic operations. However, bigger q's have an adverse effect on the security of the encryption scheme. Additionally, q also influences the runtime of homomorphic operations; more specifically, the number of primes q is composed of. The more primes, the longer the computation times.

– Degree n of the reduction polynomial: In BFV in SEAL n is always a power of two and has a direct impact on the runtime of the scheme. A bigger n drastically increases the time a homomorphic operation needs for evaluation. On the other hand, a bigger n also increases the security of the scheme and, therefore, allows for a bigger ciphertext modulus q to increase the noise budget.

We test our implementation for a security level of $\lambda = 128\,\text{bit}$ and $\lambda = 80\,\text{bit}$. We use the LWE estimator [2] by Albrecht et al. to find suitable BFV parameters which provide 80 bit security against known attacks; for 128 bit security SEAL already provides parameters for different reduction polynomial degrees n. See Appendix A for more details on the used parameters.

5.6 Benchmarks

Multithreading. Since in our use cases N is much bigger than k, we implemented multithreading, such that the threads split the number of rows in the matrix (more specifically, the number of submatrices in the rows n_v) equally amongst all available threads. Therefore, each thread has to perform at most $\left\lceil \frac{n_v}{\#\text{threads}} \right\rceil \cdot n_o$ `MatMul` evaluations, which will be combined at the end by summing up the intermediate results. In case we want to add the mask to the result, an extra thread will perform the mask-evaluation in parallel to the matrix multiplication. In the other case, the case of the interactive challenge round, we leverage multithreading, such that we distribute the computation of the masks $\mu_{\texttt{bin}}$ equally amongst all available threads, i.e., at most $\left\lceil \frac{s}{\#\text{threads}} \right\rceil$ evaluations of $\mu_{\texttt{bin}}$ per thread. In parallel, an additional thread performs the evaluation of $\mu_{\texttt{HW}}$ before the results are combined with new random values z and r for each slot of the final mask.

Benchmark Platform. Our implementation[7] is compatible to Linux and Windows; however, we ran our benchmarks on a Linux cluster with two Intel Xeon E5-2699 v4 CPU's (total of 44 cores @ 2.2 GHz, 88 threads) and 512 GB RAM available.

Runtime. The runtime of our protocol is $\mathcal{O}(n_v n_o)$, i.e., it scales linearly in the number of `MatMul` evaluations. This can be seen in Table 1 in which we summarize the runtime of the homomorphic matrix multiplication for different matrix dimensions using only one thread. For better comparability, we evaluate the different sizes with the same BFV parameter set. For real-world matrix dimensions, some added runtime has to be expected due to thread synchronization and the accumulation of the intermediate thread results.

[7] The source code is available at https://github.com/IAIK/CoronaHeatMap.

Table 1: Runtime for the Data Aggregation Phase for different matrix dimensions using only one thread.

	BFV				Matrix		#`MatMul`	Runtime
Nr.	$\log_2(p)$	$\log_2(q)$	n	λ	N	k		sec
1	33	218	8192	128	8192	4096	1	17.4
2	33	218	8192	128	40960	4096	5	87.1
3	33	218	8192	128	81920	4096	10	173.1
4	33	218	8192	128	163840	4096	20	359.5
5	33	218	8192	128	8192	20480	5	88.3
6	33	218	8192	128	8192	40960	10	172.5
7	33	218	8192	128	8192	81920	20	349.4
8	33	218	8192	128	40960	20480	25	434.7
9	33	218	8192	128	98304	32768	96	1778.9

Real World Matrix Dimension. At the time of writing, approximately 8.9 million people live in Austria[8] and the total number of cell cites is 18389[9] (June 2019). In our benchmarks, we want to capture the Austrian use case and set the matrix dimensions to $N = 2^{23}$ and $k = 2^{15}$. In Table 2 we list the runtime for a homomorphic $(1 \times 2^{23}) \times (2^{23} \times 2^{15})$ matrix multiplication, for different BFV parameters, using 88 threads. We also provide the total number of `MatMul` evaluations and the (maximum) number of evaluations per thread. Additionally, we give performance results for parameters also capable of evaluating the masking value.

Table 2: Runtime for the Data Aggregation Phase for different parameters using 88 threads. The column Masking indicates whether this parameter set is only able to evaluate the matrix multiplication (✗), or is additionally able to evaluate the calculation of the masking values (✓).

	BFV				Matrix		#`MatMul`	Masking	Runtime
Nr.	$\log_2(p)$	$\log_2(q)$	n	λ	N	k	total / per thread		min
1	33	218	8192	128	2^{23}	2^{15}	8192 / 96	✓	59.36
2	60	218	8192	128	2^{23}	2^{15}	8192 / 96	✗	89.87
3	60	438	16384	128	2^{23}	2^{15}	2048 / 24	✓	267.19
4	33	162	4096	80	2^{23}	2^{15}	32768 / 384	✗	33.55
5	33	329	8192	80	2^{23}	2^{15}	8192 / 96	✓	89.32
6	60	329	8192	80	2^{23}	2^{15}	8192 / 96	✓	140.82

[8] https://de.statista.com/statistik/daten/studie/19292/umfrage/gesamtbevoelkerung-in-oesterreich/

[9] https://www.senderkataster.at/

As Table 2 shows, a matrix multiplication takes approximately 1 hour for a 33 bit plaintext modulus with 128 bit security and 1.5 hour for the bigger 60 bit modulus. The noise budget for $n = 8192$ and a 60 bit modulus, however, is not sufficient to evaluate the masking value, which has a bigger multiplicative depth than the matrix multiplication. Increasing n leads to a performance drop, more specifically, the evaluation with a 60 bit plaintext modulus takes 4.5 hours.

Reducing the security level to $\lambda = 80$ bit allows us to use a smaller n for the evaluation of the mask with a 60 bit prime, and the matrix multiplication with a 33 bit plaintext modulus, splitting the respective runtimes in half. Unfortunately, n can not be reduced for the 33 bit prime with 80 bit security when masking is applied, increasing the runtime of the matrix multiplication compared to the 128 bit security case. This is due to the fact that in the 80 bit security case q is composed of more distinct primes q_i. We recommend, therefore, to always use 128 bit security parameters for the 33 bit prime when the masking value has to be evaluated.

Data Transmission. In Table 3, we list the sizes of all the data, which has to be transmitted between the server and the client. Each row corresponds to a different parameter set from Table 2. The sizes were obtained by storing each of the described elements on the file system on the cluster. The table lists the size of the ciphertexts (ct), Galois keys (gk), and relinearization keys (rk). Galois keys are required to perform homomorphic rotations, each rotation index requires one Galois key, plus an additional key for rotating the columns. When using the baby-step giant-step algorithm, we need a key for the index 1 to calculate $\texttt{rot}(\boldsymbol{v}, j)$, and a key for the indices $k \cdot m_1$, $\forall 0 < k < m_2$. Furthermore, when masking is applied, we need the keys for the power-of-2 indices to calculate the inner product of two ciphertexts. The relinearization key is required to linearize the result of a ciphertext-ciphertext multiplication. Since we only have to perform such a multiplication when we calculate the masking values, we can omit to send the relinearization key when the mask is not applied.

In addition to the values described in Table 3, the client has to announce the used BFV parameters and the hamming weight of the input vector. These values have a combined size of less than 300 bytes.

As Table 3 shows, client-to-server communication is significantly more extensive than the response of the server. The main parts of the communication are the initial ciphertexts; however, especially when masking has to be applied, the Galois keys have a significant size as well. The plaintext modulus p has little to no effect on the number of bytes, contrary to the reduction polynomial degree n, which influences the communication cost significantly. The response of the server is very small in comparison to the ciphertexts he receives from the client. One reason for that is the small parameter k compared to N. The other reason is, that our implementation performs a so-called modulus-switch to level 0 after the computation, reducing the ciphertext modulus q to only one of the moduli q_i it is composed of.

Table 3: Data transmission in MiB for the different parameters in Table 2. Values include keys for evaluating the masking value when applicable.

	Client				Server	Total
Nr.	ct	gk	rk	Total	ct	
1	256.2	87.6	1.3	345.1	1.0	346.1
2	256.2	81.4	-	337.6	2.0	339.6
3	512.1	639.2	9.0	1160.3	2.0	1162.3
4	128.3	6.2	-	134.5	1.0	135.5
5	384.2	183.9	2.6	570.7	1.0	571.7
6	384.2	183.9	2.6	570.7	2.0	572.7

Challenge Mask Compuation Time. In Table 4, we list the runtime of the evaluation of the interactive challenge mask for different BFV parameters with a soundness error of at least λ bits. As Table 4 shows, the evaluation of the challenge mask is significantly faster than performing the matrix multiplication, allowing the server to detect and filter malicious inputs before engaging in the costly matrix evaluation. As we already stated, calculating the masking value for a 33 bit prime is slower with parameters for 80 bit security than with parameters for 128 bit security (given multithreading with enough threads) since q is composed of more primes q_i and n stays the same for both parameter sets.

Table 4: Runtime for the challenge mask computation different parameters using $s + 1$ threads.

	BFV				Input	$\#\mu_{\texttt{bin}}$	Runtime
Nr.	$\log_2(p)$	$\log_2(q)$	n	λ	N	s	min
1	33	218	8192	128	2^{23}	13	3.79
2	60	438	16384	128	2^{23}	4	10.53
3	33	329	8192	80	2^{23}	9	6.11
4	60	329	8192	80	2^{23}	3	5.50

We note that we require additional s Galois keys for performing the challenge mask creation, increasing the communicated bytes. However, the additional keys require a comparably small amount of data, e.g., less than 25 MiB for entry nr. 3 in Table 3.

5.7 Price Estimation for Deployment in Germany

In this section, we want to give an estimate of the costs of deploying our system to create a corona heatmap for a larger country, more specifically, for Germany.

At the time of writing, about 83.1 million people live in Germany[10], and a total of 74280 cell sites are deployed[11]. With the BFV parameters of entry Nr. 1 in Table 2, i.e., $n = 8192$, this corresponds to a total number of of $n_v \cdot n_o = 192755$ `MatMul` evaluations. To get 96 `MatMul` evaluations per thread, we would have to acquire 21 CPU's capable of handling 96 threads each. To get a estimate for doing such computations used current market prizes[12] the cost of one CPU capable of handling 96 threads is $\sim 1.5\,\$$ per hour. Taking an additional overhead by handling so many threads and combining intermediate results, we estimate the cost of evaluating the homomorphic matrix multiplication for the German use case using AWS to $\sim 60\,\$$. While noting that a trivial outsourcing of such computations is not part of our proposal, this estimate still shows that it is likely very feasible to create a heatmap once a day to gain valuable insight into the spread of the disease, even for larger countries.

6 Concluding discussion

Our protocol shows that concrete privacy-preserving data analytics is possible even on a national scale.

For an interested audience with little security and cryptography background, we created a webpage[13] that describes our approach, and basically has the following message: Even in times of crisis where it is tempting to (temporarily) lower data protection standards for purposes of big data analytics, there are technical methods to keep data protection standards high. And those technical methods are practical and available.

An actual roll-out needs to consider a number of other aspects that are out of scope for this article. This includes legal aspects, e.g. making sure there is a consent in using collected telephone numbers for such a purpose. There are also parameters of our system that need to be chosen in view of a particular dataset, potentially in coordination with data-protection authorities such as fixing the minimum Hamming weight of the query vector $\boldsymbol{c}$ and differential privacy parameters.

Acknowledgments

This work was supported by EU's Horizon 2020 project Safe-DEED, grant agreement n°825225, and EU's Horizon 2020 project TRUSTS, grant agreement n°871481, and by the "DDAI" COMET Module within the COMET – Competence Centers for Excellent Technologies Programme, funded by the Austrian

[10] `https://www.destatis.de/DE/Themen/Gesellschaft-Umwelt/Bevoelkerung/Bevoelkerungsstand/_inhalt.html`

[11] `https://www.informationszentrum-mobilfunk.de/artikel/statistik-zur-zahl-der-funkanlagenstandorte-in-deutschland`

[12] `https://aws.amazon.com/ec2/spot/pricing/`

[13] `https://covid-heatmap.iaik.tugraz.at`

Federal Ministry for Transport, Innovation and Technology (bmvit), the Austrian Federal Ministry for Digital and Economic Affairs (bmdw), the Austrian Research Promotion Agency (FFG), the province of Styria (SFG) and partners from industry and academia. The COMET Programme is managed by FFG.

References

1. AISEC, F.: Pandemic contact tracing apps: Dp-3t, pepp-pt ntk, and robert from a privacy perspective. Cryptology ePrint Archive, Report 2020/489 (2020), https://eprint.iacr.org/2020/489
2. Albrecht, M.R., Player, R., Scott, S.: On the concrete hardness of learning with errors. J. Mathematical Cryptology **9**(3), 169–203 (2015)
3. Angel, S., Chen, H., Laine, K., Setty, S.T.V.: PIR with compressed queries and amortized query processing. In: IEEE Symposium on Security and Privacy. pp. 962–979. IEEE Computer Society (2018)
4. Beimel, A., Ishai, Y., Kushilevitz, E., Raymond, J.: Breaking the o(n1/(2k-1)) barrier for information-theoretic private information retrieval. In: FOCS. pp. 261–270. IEEE Computer Society (2002)
5. Bergamaschi, F., Halevi, S., Halevi, T.T., Hunt, H.: Homomorphic training of 30, 000 logistic regression models. In: ACNS. Lecture Notes in Computer Science, vol. 11464, pp. 592–611. Springer (2019)
6. Berke, A., Bakker, M., Vepakomma, P., Larson, K., Pentland, A.S.: Assessing disease exposure risk with location data: A proposal for cryptographic preservation of privacy (2020)
7. Beskorovajnov, W., Dörre, F., Hartung, G., Koch, A., Müller-Quade, J., Strufe, T.: Contra corona: Contact tracing against the coronavirus by bridging the centralized–decentralized divide for stronger privacy. Cryptology ePrint Archive, Report 2020/505 (2020), https://eprint.iacr.org/2020/505
8. Boemer, F., Costache, A., Cammarota, R., Wierzynski, C.: ngraph-he2: A high-throughput framework for neural network inference on encrypted data. In: WAHC@CCS. pp. 45–56. ACM (2019)
9. Bourse, F., Minelli, M., Minihold, M., Paillier, P.: Fast homomorphic evaluation of deep discretized neural networks. In: CRYPTO (3). Lecture Notes in Computer Science, vol. 10993, pp. 483–512. Springer (2018)
10. Brack, S., Reichert, L., Scheuermann, B.: Decentralized contact tracing using a dht and blind signatures. Cryptology ePrint Archive, Report 2020/398 (2020), https://eprint.iacr.org/2020/398
11. Brakerski, Z.: Fully homomorphic encryption without modulus switching from classical gapsvp. In: CRYPTO. Lecture Notes in Computer Science, vol. 7417, pp. 868–886. Springer (2012)
12. Brakerski, Z., Gentry, C., Vaikuntanathan, V.: (leveled) fully homomorphic encryption without bootstrapping. In: ITCS. pp. 309–325. ACM (2012)
13. Bünz, B., Bootle, J., Boneh, D., Poelstra, A., Wuille, P., Maxwell, G.: Bulletproofs: Short proofs for confidential transactions and more. In: IEEE Symposium on Security and Privacy. pp. 315–334. IEEE Computer Society (2018)
14. Cachin, C., Micali, S., Stadler, M.: Computationally private information retrieval with polylogarithmic communication. In: EUROCRYPT. Lecture Notes in Computer Science, vol. 1592, pp. 402–414. Springer (1999)

15. Canetti, R., Trachtenberg, A., Varia, M.: Anonymous collocation discovery: Harnessing privacy to tame the coronavirus (2020)
16. Chan, J., Foster, D., Gollakota, S., Horvitz, E., Jaeger, J., Kakade, S., Kohno, T., Langford, J., Larson, J., Singanamalla, S., Sunshine, J., Tessaro, S.: Pact: Privacy sensitive protocols and mechanisms for mobile contact tracing (2020)
17. Chen, H., Huang, Z., Laine, K., Rindal, P.: Labeled PSI from fully homomorphic encryption with malicious security. In: ACM Conference on Computer and Communications Security. pp. 1223–1237. ACM (2018)
18. Cheon, J.H., Kim, A., Kim, M., Song, Y.S.: Homomorphic encryption for arithmetic of approximate numbers. In: ASIACRYPT (1). Lecture Notes in Computer Science, vol. 10624, pp. 409–437. Springer (2017)
19. Chillotti, I., Gama, N., Georgieva, M., Izabachène, M.: Faster fully homomorphic encryption: Bootstrapping in less than 0.1 seconds. In: ASIACRYPT (1). Lecture Notes in Computer Science, vol. 10031, pp. 3–33 (2016)
20. Chor, B., Goldreich, O., Kushilevitz, E., Sudan, M.: Private information retrieval. In: FOCS. pp. 41–50. IEEE Computer Society (1995)
21. Chou, E., Beal, J., Levy, D., Yeung, S., Haque, A., Fei-Fei, L.: Faster cryptonets: Leveraging sparsity for real-world encrypted inference. CoRR **abs/1811.09953** (2018)
22. Dar, A.B., Lone, A.H., Zahoor, S., Khan, A.A., Naaz, R.: Applicability of mobile contact tracing in fighting pandemic (covid-19): Issues, challenges and solutions. Cryptology ePrint Archive, Report 2020/484 (2020), https://eprint.iacr.org/2020/484
23. Dathathri, R., Saarikivi, O., Chen, H., Laine, K., Lauter, K.E., Maleki, S., Musuvathi, M., Mytkowicz, T.: CHET: an optimizing compiler for fully-homomorphic neural-network inferencing. In: PLDI. pp. 142–156. ACM (2019)
24. Demmler, D., Herzberg, A., Schneider, T.: RAID-PIR: practical multi-server PIR. In: CCSW. pp. 45–56. ACM (2014)
25. Devet, C., Goldberg, I., Heninger, N.: Optimally robust private information retrieval. In: USENIX Security Symposium. pp. 269–283. USENIX Association (2012)
26. Dong, C., Chen, L.: A fast single server private information retrieval protocol with low communication cost. In: ESORICS (1). Lecture Notes in Computer Science, vol. 8712, pp. 380–399. Springer (2014)
27. Dwork, C.: Differential privacy. In: ICALP (2). Lecture Notes in Computer Science, vol. 4052, pp. 1–12. Springer (2006)
28. Fan, J., Vercauteren, F.: Somewhat practical fully homomorphic encryption. IACR Cryptology ePrint Archive **2012**, 144 (2012)
29. Gentry, C.: Fully homomorphic encryption using ideal lattices. In: STOC. pp. 169–178. ACM (2009)
30. Goldberg, I.: Improving the robustness of private information retrieval. In: IEEE Symposium on Security and Privacy. pp. 131–148. IEEE Computer Society (2007)
31. Google, Apple: Apple and google's exposure notification system. https://www.apple.com/covid19/contacttracing (2020)
32. Halevi, S., Shoup, V.: Algorithms in helib. In: CRYPTO (1). Lecture Notes in Computer Science, vol. 8616, pp. 554–571. Springer (2014)
33. Halevi, S., Shoup, V.: Bootstrapping for helib. In: EUROCRYPT (1). Lecture Notes in Computer Science, vol. 9056, pp. 641–670. Springer (2015)
34. Halevi, S., Shoup, V.: Faster homomorphic linear transformations in helib. In: CRYPTO (1). Lecture Notes in Computer Science, vol. 10991, pp. 93–120. Springer (2018)

35. Hesamifard, E., Takabi, H., Ghasemi, M.: Deep neural networks classification over encrypted data. In: CODASPY. pp. 97–108. ACM (2019)
36. Jiang, X., Kim, M., Lauter, K.E., Song, Y.: Secure outsourced matrix computation and application to neural networks. In: ACM Conference on Computer and Communications Security. pp. 1209–1222. ACM (2018)
37. Juvekar, C., Vaikuntanathan, V., Chandrakasan, A.: GAZELLE: A low latency framework for secure neural network inference. In: USENIX Security Symposium. pp. 1651–1669. USENIX Association (2018)
38. Kiayias, A., Leonardos, N., Lipmaa, H., Pavlyk, K., Tang, Q.: Optimal rate private information retrieval from homomorphic encryption. PoPETs **2015**(2), 222–243 (2015)
39. Landau, S.: Looking beyond contact tracing to stop the spread. https://www.lawfareblog.com/looking-beyond-contact-tracing-stop-spread (2020)
40. Lipmaa, H.: First CPIR protocol with data-dependent computation. In: ICISC. Lecture Notes in Computer Science, vol. 5984, pp. 193–210. Springer (2009)
41. Lipmaa, H., Pavlyk, K.: A simpler rate-optimal CPIR protocol. In: Financial Cryptography. Lecture Notes in Computer Science, vol. 10322, pp. 621–638. Springer (2017)
42. Lyubashevsky, V., Peikert, C., Regev, O.: On ideal lattices and learning with errors over rings. In: EUROCRYPT. Lecture Notes in Computer Science, vol. 6110, pp. 1–23. Springer (2010)
43. Melchor, C.A., Barrier, J., Fousse, L., Killijian, M.: XPIR : Private information retrieval for everyone. PoPETs **2016**(2), 155–174 (2016)
44. Paillier, P.: Public-key cryptosystems based on composite degree residuosity classes. In: EUROCRYPT. Lecture Notes in Computer Science, vol. 1592, pp. 223–238. Springer (1999)
45. Pinkas, B., Ronen, E.: Hashomer - a proposal for a privacy-preserving bluetooth based contact tracing scheme for hamagen. https://github.com/eyalr0/HashomerCryptoRef/blob/master/documents/hashomer.pdf (2020)
46. Regev, O.: On lattices, learning with errors, random linear codes, and cryptography. In: STOC. pp. 84–93. ACM (2005)
47. Rivest, R.L., Adleman, L., Dertouzos, M.L.: On data banks and privacy homomorphisms. Foundations of Secure Computation, Academia Press pp. 169–179 (1978)
48. Rivest, R.L., Shamir, A., Adleman, L.M.: A method for obtaining digital signatures and public-key cryptosystems. Commun. ACM **21**(2), 120–126 (1978)
49. Microsoft SEAL (release 3.4). https://github.com/Microsoft/SEAL (Oct 2019), microsoft Research, Redmond, WA.
50. Team, D.T.: Decentralized privacy-preserving proximity tracing. https://github.com/DP-3T (2020)
51. Trieu, N., Shehata, K., Saxena, P., Shokri, R., Song, D.: Epione: Lightweight contact tracing with strong privacy (2020)
52. Yi, X., Kaosar, M.G., Paulet, R., Bertino, E.: Single-database private information retrieval from fully homomorphic encryption. IEEE Trans. Knowl. Data Eng. **25**(5), 1125–1134 (2013)

A BFV parameters

In this section we list the BFV parameters used in our implementation.

A.1 Plaintext Moduli

In our benchmarks, we use two different plaintext moduli, one with a size of 33 bits, the other with a size of 60 bits. Table 5 lists the used moduli.

Table 5: Used plaintext moduli in hexadecimal notation and their size in bits.

Nr.	p	$\log_2(p)$
1	`0x1e21a0001`	33
2	`0xf4fc03ff53d0001`	60

A.2 Ciphertext Moduli

In this section we list all the ciphertext moduli used for different security levels λ and reduction polynomial degrees n. In SEAL the ciphertext modulus q is the product several primes q_i: $q = \prod_i q_i$.

$n = 4096$, $\lambda = 80$: The ciphertext modulus q is composed of 3 primes with a total size of 162 bit, which we list in Table 6.

Table 6: Primes composing the ciphertext modulus for $n = 4096$, $\lambda = 80$ in hexadecimal notation and their size in bits.

i	q_i	$\log_2(q_i)$
1	`0x3fffffffd6001`	54
2	`0x3fffffffd2001`	54
3	`0x3fffffffbe001`	54

$n = 8192$, $\lambda = 80$: The ciphertext modulus q is composed of 7 primes with a total size of 329 bit, which we list in Table 7.

$n = 8192$, $\lambda = 128$: The ciphertext modulus q is composed of 5 primes with a total size of 218 bit, which we list in Table 8.

$n = 16384$, $\lambda = 128$: The ciphertext modulus q is composed of 9 primes with a total size of 438 bit, which we list in Table 9.

Table 7: Primes composing the ciphertext modulus for $n = 8192$, $\lambda = 80$ in hexadecimal notation and their size in bits.

i	q_i	$\log_2(q_i)$
1	`0x7ffffffec001`	47
2	`0x7ffffffc8001`	47
3	`0x7ffffffb4001`	47
4	`0x7ffffff00001`	47
5	`0x7fffffefc001`	47
6	`0x7fffffecc001`	47
7	`0x7fffffe70001`	47

Table 8: Primes composing the ciphertext modulus for $n = 8192$, $\lambda = 128$ in hexadecimal notation and their size in bits.

i	q_i	$\log_2(q_i)$
1	`0x7ffffd8001`	43
2	`0x7ffffc8001`	43
3	`0xfffffffc001`	44
4	`0xffffff6c001`	44
5	`0xfffffebc001`	44

Table 9: Primes composing the ciphertext modulus for $n = 16384$, $\lambda = 128$ in hexadecimal notation and their size in bits.

i	q_i	$\log_2(q_i)$
1	`0xffffffd8001`	48
2	`0xffffffa0001`	48
3	`0xfffffff00001`	48
4	`0x1fffffff68001`	49
5	`0x1fffffff50001`	49
6	`0x1ffffffee8001`	49
7	`0x1ffffffea0001`	49
8	`0x1ffffffe88001`	49
9	`0x1ffffffe48001`	49