A probabilistic design for practical homomorphic majority voting with intrinsic differential privacy

Arnaud Grivet Sébert
arnaud.grivetsebert@gmail.com
Université Paris-Saclay, CEA, List
Palaiseau, FR

Martin Zuber
martin.zuber@cea.fr
Université Paris-Saclay, CEA, List
Palaiseau, FR

Oana Stan
oana.stan@cea.fr
Université Paris-Saclay, CEA, List
Palaiseau, FR

Renaud Sirdey
renaud.sirdey@cea.fr
Université Paris-Saclay, CEA, List
Palaiseau, FR

Cédric Gouy-Pailler
cedric.gouy-pailler@cea.fr
Université Paris-Saclay, CEA, List
Palaiseau, FR

ABSTRACT

As machine learning (ML) has become pervasive throughout various fields (industry, healthcare, social networks), privacy concerns regarding the data used for its training have gained a critical importance. In settings where several parties wish to collaboratively train a common model without jeopardizing their sensitive data, the need for a private training protocol is particularly stringent and implies to protect the data against both the model's end-users and the other actors of the training phase. In this context of secure collaborative learning, Differential Privacy (DP) and Fully Homomorphic Encryption (FHE) are two complementary countermeasures of growing interest to thwart privacy attacks in ML systems. Central to many collaborative training protocols, in the line of PATE, is majority voting aggregation. Thus, in this paper, we design SHIELD, a probabilistic approximate majority voting operator which is faster when homomorphically executed than existing approaches based on exact argmax computation over an histogram of votes. As an additional benefit, the inaccuracy of SHIELD is used as a feature to provably enable DP guarantees. Although SHIELD may have other applications, we focus here on one setting and seamlessly integrate it in the SPEED collaborative training framework from [20] to improve its computational efficiency. After thoroughly describing the FHE implementation of our algorithm and its DP analysis, we present experimental results. To the best of our knowledge, it is the first work in which relaxing the accuracy of an algorithm is constructively usable as a degree of freedom to achieve better FHE performances.

CCS CONCEPTS

• **Security and privacy** → **Information-theoretic techniques**; *Privacy-preserving protocols*;

KEYWORDS

collaborative machine learning; differential privacy; homomorphic encryption

ACM Reference format:
Arnaud Grivet Sébert, Martin Zuber, Oana Stan, Renaud Sirdey, and Cédric Gouy-Pailler. 2018. A probabilistic design for practical homomorphic majority voting with intrinsic differential privacy. In *Proceedings of 11th Workshop on Encrypted Computing & Applied Homomorphic Cryptography, Copenhagen, Denmark, November 26, 2023 (WAHC 2023),* 12 pages.
https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

Most works combining Differential Privacy (DP) and Fully Homomorphic Encryption (FHE) falls into two categories, either an exact FHE is used and, independently, the plaintexts are adequatly noised to achieve DP [21, 24, 37] or the post-decryption noise of an approximate FHE such as CKKS is tuned to provide such guarantees [32, 36]. The latter approach is presently limited to simple aggregation rules as no efficient techniques for e.g. argmin/max computations are presently known for CKKS (to the best of our knowledge). In the present paper, we focus on the first of these two approaches and aim at avoiding explicit DP noise addition over the plaintexts by designing an homomorphic majority voting[1] operator (that may be viewed as a stochastic argmax), called SHIELD (Secure and Homomorphic Imperfect Election via Lightweight Design), whose structural stochasticity leads to both lighter FHE computational cost and DP protection. In fact, we prove that the inaccuracies due to the approximate behavior of our algorithm translate into consistent DP guarantees, and therefore that explicit noise addition becomes unnecessary for DP. In doing so, and by means of a carefully crafted FHE implementation of the algorithm, we are able to achieve a reduction of more than 25% in the homomorphic computation time compared to the state of the art on majority voting based on *exact* argmax computations over an histogram [9, 23]. To the best of our knowledge, it is the first work in which relaxing the accuracy of an homomorphic calculation *at the algorithmic level*[2] is constructively usable as a degree of freedom to achieve better FHE performances.

A well-suited use-case to SHIELD is the PATE protocol [33] and especially its extension SPEED [20]. In a nutshell, the PATE protocol labels a subset of a public dataset and uses this partially labeled dataset to train a student model in a semi-supervised way. The labelization is achieved by aggregating (usually by means of a majority voting) the labels - considered as votes - provided by a set of teacher models trained on private datasets. Since the teachers' labels would leak information on their training data, the PATE protocol makes use of differential privacy (DP). To get a reasonable privacy-utility trade-off, the aggregation of votes is performed on a trusted independent server. The SPEED approach [20] builds

[1] By majority voting, which is sometimes called plurality voting, we mean the selection of the candidate with the most votes.
[2] By opposition to the CKKS-based approaches which relax the precision of the homomorphic operations.

upon the work from [33] and uses Fully Homomorphic Encryption (FHE) to blind the server by having it perform the aggregation directly over encrypted votes, therefore protecting the data against an honest-but-curious server. Still, FHE being computationally intensive, this comes at significant communication and computation costs on the server (6.5 minutes to compute the homomorphic argmax for 100 queries). We here propose that the server performs the vote aggregation using SHIELD, which is then seamlessly integrated in SPEED. Our experiments show that SHIELD reduces the server's computational burden while providing DP guarantees comparable to the ones in [20] with respect to the teachers (albeit to the exception of the server which should not be revealed the final model). Overall, in our experiments, the correct majority vote is output with a probability of more than 90% and, more importantly, SHIELD only incurs a model accuracy loss of 0.7% compared to an exact argmax-based aggregation with no DP.

The paper is organized as follows. First of all, we explore the related work in Section 2 and give some preliminaries about HE and DP in Section 3. Then, we introduce and describe our operator SHIELD in Section 4 and more specifically its FHE implementation in Section 5, before presenting the SPEED application case in Section 6. Section 7 develops an analysis of SHIELD from the points of view of DP and HE. Finally, our experimental results are presented in Section 8.

2 RELATED WORK

In [41], the authors survey recent works in which DP and cryptographic primitives take advantage of each other, either

- cryptography for DP: cryptographic primitives allow to get the privacy-utility trade-off of a standard DP mechanism but without the need of a trusted server [2, 10, 16, 19]. This is an improvement compared to *local DP* which, by making the data owners noise their data before outsourcing them, does not need a trusted server either but gives a poorer privacy-utility trade-off [26, 38],
- or DP for cryptography: design "leaky" cryptographic primitives that ensure DP and are more efficient than traditional primitives [5, 39, 40].

The approaches from [5, 39, 40] are tailored to specific applications, respectively SQL queries, anonymous communication systems and oblivious RAM. Our work uses exact FHE but with an approximate algorithm, in the context of election.

In [42], the authors propose an algorithm with a close goal, namely heavy-hitters (most frequent items) detection, which is inherently differentially private thanks to random sampling. Nevertheless, the goal of this inherent probabilistic behavior is not computational efficiency since the method is not articulated with cryptographic primitives. Moreover, this algorithm works on sequential data. Even if it does not restrict its generality since any data can be seen as sequential, the utility does depend on the sequential representation of the data, which may not be optimal if there is no semantic value to this representation. Finally, the algorithm is iterative and thus requires a lot of communication with the users.

Some recent works propose emerging approaches to combine DP and approximate FHE like CKKS. Among them, [29] uses a post-processing noise ensuring DP to turn an IND-CPA approximate cryptosystem into an IND-CPA^D one. Indeed, whereas IND-CPA^D is equivalent to IND-CPA for exact cryptosystems, it is not the case for approximate ones. In [32], the author analyzes the noise native to CKKS encryption and proves that it can provide DP without additional noise, yet their analysis holds at most quadratic polynomial evaluations. The work from [36] also leverages the error induced by encryption to derive DP guarantees, but in a federated learning context. The aggregation protocol is based on the security of LWE problem and on the Multi-Party Computation protocol of Packed Shamir secret sharing scheme [18]. Nevertheless, LWE is not used to directly encrypt the values of interest but rather to generate one-time pads of the same dimension of these values while only needing to communicate much smaller vectors to the server. These one-time pads allow a secure aggregation and DP guarantees are ensured by the error induced by LWE encryption.

In contrast with these latter works, we use an exact HE cryptosystem, namely BFV, and craft a majority voting algorithm which is approximate by design. It is this approximate behavior of the algorithm that we leverage to achieve DP, and not the noise native to encryption like in the aforementioned CKKS-based approaches.

Some works in the literature try to increase the speed of an argmax operation by making it stochastic but *without* this stochasticity leading to provable DP guarantees. Fair comparison with such works is tricky. Among the very few works in this line, [22] have a very efficient CKKS-based approximate argmax operator, but it is difficult to use outside of the biometric identification protocol for which it is intended and does not provide DP guarantees. For this reason we only compare our work with state-of-the-art on majority voting based on exact homomorphic argmax operators over an histogram [9, 23].

3 PRELIMINARIES

3.1 Homomorphic encryption

Fully homomorphic encryption (FHE) schemes allow to perform arbitrary computations directly over encrypted data. That is, with a fully homomorphic encryption scheme Enc, we can compute an addition $\mathsf{Enc}(\mathsf{m}_1 + \mathsf{m}_2)$ and a multiplication $\mathsf{Enc}(\mathsf{m}_1 \times \mathsf{m}_2)$ from encrypted messages $\mathsf{Enc}(\mathsf{m}_1)$ and $\mathsf{Enc}(\mathsf{m}_2)$.

In this section we recall the general principles of the **BFV** homomorphic cryptosystem [17]. Since we know in advance the function to be evaluated homomorphically, we can stick to the somewhat homomorphic encryption (SHE) version described below. Let $R = \mathbb{Z}[x]/\Phi_m(x)$ denote the polynomial ring modulo the m-cyclotomic polynomial with $n = \varphi(m)$. The ciphertexts in the scheme are elements of the polynomial ring R_q, where R_q is the set of polynomials in R with coefficients in $\mathbb{Z}_q$. The plaintexts are polynomials belonging to the ring $R_t = R/tR$ for $t << q$. For $a \in R$, we denote by $[a]_q$ the element in R obtained by applying modulo q to all its coefficients.

As with all SHE schemes, BFV allows us to evaluate additions and multiplications and comes with a relinearization function that helps to manage the size of the ciphertexts. As homomorphic multiplication increases the size of the ciphertexts, relinearization reduces the size back to a smaller, more manageable form. This process is crucial for maintaining the efficiency of computations over encrypted data. Using Single Instruction, Multiple Data (SIMD) packing (or

batching), one can significantly increase the time performance of BFV-based computation. SIMD is a technique used to perform computations on multiple values simultaneously. It allows to pack multiple plaintexts into a single ciphertext and perform operations on them in parallel. This method was originally described in [7, 35]. For further details we refer the reader to the original paper [17].

3.2 Differential privacy

Differential privacy [14] is a gold standard concept in privacy-preserving data analysis. It provides a guarantee that under a reasonable privacy budget (ϵ, δ), two adjacent databases produce statistically indistinguishable results. In the context of SPEED, we consider, as in [20], that a database is the concatenation of the data owners' (teachers) datasets and that two databases are adjacent if they differ by one teacher.

In the whole paper, $\mathbb{P}(A)$ denotes the probability of event A.

Definition 3.1. A randomized mechanism $\mathcal{M}$ with output range $\mathcal{R}$ satisfies (ϵ, δ)-*DP* if for any two adjacent databases d and d' and for any subset of outputs $S \subset \mathcal{R}$ one has

$$\mathbb{P}\left[\mathcal{M}(d) \in S\right] \leq e^{\epsilon}\mathbb{P}\left[\mathcal{M}(d') \in S\right] + \delta.$$

(ϵ, δ) constitutes the *privacy cost* of the mechanism - the lower ϵ and δ, the more private the mechanism. Definition 3.1 implies that a differentially private mechanism is necessarily probabilistic. Most often, a deterministic mechanism is turned differentially private by adding random noise at a certain moment in the computation.

Since the privacy cost increases with the number of queries to the mechanism, we determine the privacy cost of our protocol via a two-fold approach. First of all, we determine the privacy cost per query and then we compose the privacy costs of each query to get the overall cost. The classical composition theorem (see e.g. [15]) states that the guarantees ϵ of sequential queries add up. Nevertheless, training a deep neural network, even with a collaborative framework as presented in this paper, requires a large amount of calls to the databases, precluding the use of this classical composition. Therefore, to obtain reasonable DP guarantees, we need to keep track of the privacy cost with a more refined tool called the moments accountant, introduced in [1] and closely related to Rényi DP [31]. This technique allows for far better composition of the privacy costs along the queries.

Finally, an important property of DP, widely used in DP analysis, is that it is immune to post-processing. This means that, quite intuitively, applying a function to the output of a differentially private mechanism does not disclose more information to the adversary.

Proposition 3.2 ([15]). *Let $\mathcal{M}$ be a probabilistic mechanism, with output range $\mathcal{R}$, that is (ϵ, δ)-differentially private, with $(\epsilon, \delta) \in (\mathbb{R}_+)^2$. Let $\phi : \mathcal{R} \to \mathcal{R}'$ be an arbitrary probabilistic mapping. Then $\phi \circ \mathcal{M}$ is (ϵ, δ)-differentially private.*

4 SHIELD: SECURE AND HOMOMORPHIC IMPERFECT ELECTION VIA LIGHTWEIGHT DESIGN

4.1 Principle of SHIELD

We propose a novel operator that can be viewed as a stochastic majority voting rule, or (more or less) as a probabilistic argmax. Previous works address this matter by homomorphically building an histogram of the votes and then applying an homomorphic argmax operator. In contrast, SHIELD (Secure and Homomorphic Imperfect Election via Lightweight Design) directly selects one of the votes stochastically, therefore bypassing the complex and heavy argmax operation. What we lose in accuracy, we gain in DP guarantees and computational time.

SHIELD is an iterative algorithm that takes as input a list of votes for candidates. Informally, at each iteration, also called a try, some of the votes are randomly drawn and the algorithm checks if they are all equal. If yes, it outputs the common vote[3], otherwise, the algorithm performs another try. The output of SHIELD can be any of the candidates for whom there is at least one vote but the more votes a candidate got, the more likely this candidate will be the output. By tuning the parameters (especially the number of tries and the number of selected votes at each try), we can increase the probability of getting the candidate with the most votes.

Let us now formally describe SHIELD. First of all, SHIELD is meant to be computed in the homomorphic domain. Here are some notations we will use to describe its homomorphic behavior. Enc and Dec respectively denote the encryption and decryption functions of some homomorphic encryption system defined on $\mathbb{Z}_2$. $\oplus$ and $\otimes$ respectively represent the homomorphic addition and multiplication. When these operators are applied on vectors, they denote the element-wise corresponding operations. Note that the negation of $x \in \mathbb{Z}_2$ is homomorphically performed via $\mathsf{Enc}(1) \oplus \mathsf{Enc}(x)$ and the homomorphic *or* operator, denoted $\heartsuit$, between $x \in \mathbb{Z}_2$ and $y \in \mathbb{Z}_2$ is performed via $[\mathsf{Enc}(x) \oplus \mathsf{Enc}(y)] \oplus [\mathsf{Enc}(x) \otimes \mathsf{Enc}(y)]$ and will be written $\mathsf{Enc}(x) \heartsuit \mathsf{Enc}(y)$ in the following.

In this paper, for any $m \in \mathbb{N}$, $[m]$ denotes the set $\{1, ..., m\}$ (which is, by convention, the empty set if $m = 0$). Let K be the number of classes of the classification problem. Let n be the number of voters (or teachers for SPEED) and, given $k \in [K]$, let n_k be the number of voters who voted for class k.

Definition 4.1. Let $K \in \mathbb{N}^*$. A vector $z \in (\mathbb{Z}_2)^K$ is said to be a *one-hot encoding* (vector) if there exists $k_0 \in [K]$ such that $z_{k_0} = 1$ and, for all $k \in [K] \setminus \{k_0\}$, $z_k = 0$. In this case, we say that z *codes* for the class k_0 or that z is the one-hot encoding of the class k_0.

Let $(p, a) \in (\mathbb{N}^*)^2$. Let $\mathcal{S}_{p,a}$ denote SHIELD operator with parameters p and a, that we define in the following.

Let $(n, K) \in (\mathbb{N}^*)^2$ that we consider fixed in the remainder of this section. Let $Z = (\mathsf{Enc}(z^{(i)}))_{i \in [n]}$ be a list of n encrypted one-hot encoding K-dimensional vectors, some of these vectors being possibly equal (it is necessarily the case for some vectors when $K < n$). Z models the list of the votes, each vote being represented as a one-hot encoding. Then $\mathcal{S}_{p,a}(Z)$ is an encryption of one of the $z^{(i)}$, and with high probability (see Section 8 for quantitative results) $\mathcal{S}_{p,a}(Z)$ is an encryption of the most frequent of the one-hot encoding vectors of Z. $\mathcal{S}_{p,a}$ is formally defined in Algorithm 1 where, for the sake of clarity, we do not explicitly write the encryption function (e.g. $res = z^{(i_0)}$ instead of $res = \mathsf{Enc}(z^{(i_0)})$). $\mathcal{S}_{p,a}$ performs a iterations (or tries) and, at each iteration, it draws p vectors of Z *with replacement* in a uniformly random manner and multiply

[3]The algorithm should then terminate but, in the secret domain, it has to perform a fixed number of tries so the output is saved until the end.

them. The resulting vector π is an encryption of the one-hot encoding of the class k_0, with $k_0 \in [K]$, if all the p drawn encrypted vectors code for the same class k_0. Otherwise, π is the null vector of $(\mathbb{Z}_2)^K$. If a non-null vector was already found in a previous iteration, the current π is ignored (since the bit $found\_not\_null$ has been set to 1). Of course, since the algorithm is computed in the encrypted domain, it has to run until the end of the *for* loop (all the a iterations) but everything works as if the algorithm repeated this operation until it gets a non-null vector and then ignored the remaining product vectors. This first non-null vector is the output of $\mathcal{S}_{p,a}$. If no non-null vector was produced after a iterations, a null vector is output and we say that $\mathcal{S}_{p,a}$ *failed*.

Algorithm 1: SHIELD

Input : number of vectors n, number of classes K, list of encrypted votes Z, number of multiplications p, number of terms a

Output : $res = \mathcal{S}_{p,a}(Z) = z^{(i_0)}$ where $i_0 \in [n]$

$res \leftarrow (0, \dots, 0) \in (\mathbb{Z}_2)^K$;
$found\_not\_null \leftarrow 0$;
for j *in* $[a]$ **do**
  $\pi \leftarrow (1, \dots, 1) \in (\mathbb{Z}_2)^K$;
  for l *in* $[p]$ **do**
    Draw a vector z of Z uniformly at random;
    $\pi \leftarrow \pi \otimes z$;
  end
  $res \leftarrow res \oplus (1 \oplus found\_not\_null) \otimes \pi$;
  $is\_not\_null \leftarrow \bigoplus_{k=1}^{K} \pi_k$;
  $found\_not\_null \leftarrow found\_not\_null \veebar is\_not\_null$;
end

The parameter a being fixed, the choice of p must consider the trade-off between, on one hand, the accuracy of the operator, i.e. the probability of getting the most frequent vector (see the considered accuracy metrics in Section 7.2), and, on the other hand, the probability of avoiding a failure and the computational complexity. Indeed, when p increases, the probability of getting a null vector (and then failing) increases, as well as the computational complexity, but the probability of getting the most frequent vector, knowing that the algorithm did not fail, increases too.

4.2 Multi-degree SHIELD

We can imagine a parameter p that decreases as the iterations run, as if it adapted to the vote distribution. Indeed, on one hand, a high p for the first iterations ensures (with high probability) that we get the most frequent vector if getting a non-null vector is easy (i.e. probable), which happens if a vast majority of the vectors code for the same class (i.e. a vast majority of voters agree on one candidate). On the other hand, if the first iterations failed, which suggests that getting a non-null vector is not so probable, the number p of multiplications decreases, making the production of a non-null vector easier. In this framework, our SHIELD operator can be represented by a polynomial $\sum_{p=1}^{D} a_p X^p$ with positive integer coefficients, where D is the highest value taken by p, $\sum_{p=1}^{D} a_p = a$ and some coefficients a_p which may be null. We call $\sum_{p=1}^{D} a_p X^p \in \mathbb{N}[X]$ the *polynomial parameterization* of SHIELD. There is indeed a bijection between the set of operators and $\mathbb{N}[X]$ since the order of the terms of different degrees is constrained to be the one of decreasing degrees. Nevertheless, the analogy seems to stop here since the algebraic structure of $\mathbb{N}[X]$ does not apply to the set of operators (think about a factorization like $X^2 \sum_{p=0}^{D} a_p X^{p-2}$, that would draw for once two vectors and use them for all the a terms/tries, whereas we here want to independently draw the vectors for each try).

Note that we can easily ensure that multi-degree SHIELD does not fail by imposing $a_1 = 1$. Indeed, when we draw only one one-hot encoding vector, without multiplying it with others, we cannot get a null vector. Besides, $a_1 > 1$ is useless since the first draw of a single vector will succeed.

It is easily seen that multi-degree SHIELD is a generalization of SHIELD and, as such, in the remainder of this article, multi-degree SHIELD will simply be referred to as SHIELD.

4.3 Offset parameter

The SHIELD operator as defined above cannot always provide finite DP guarantees. Let us consider two adjacent databases d and d' such that, in d, a class c was chosen by no voter and, in d', c was chosen by one voter. Then, with input d, SHIELD will never output c because it cannot pick a one-hot encoding for c, the probability of outputting c is then null. On the contrary, with input d', there is a non-null probability (even if it is small) of outputting c. Hence, the ratio of probabilities of outputting c is not bounded and we get an infinite privacy cost[4].

To avoid this problem, we force all the classes to have at least one vote by creating a dummy one-hot encoding for each class. More generally, ω dummy one-hot encodings can be created for each class, where ω is another parameter of SHIELD, called the *offset*.

Algorithm 2 gives the pseudocode of the multi-degree version of SHIELD with the offset parameter.

In our experiments, we fixed ω to 1, letting the optimization of this parameter for further work. It is nevertheless intuitive that the greater ω, the worse the accuracy because, when ω is large, the distribution of the votes is flattened and the probability of outputting the true argmax is lower.

4.4 Exponential argmax operator

As an inherently stochastic mechanism that does not resort to noise addition but rather outputs a value with a probability that is an increasing function of its utility (if we deem that the vote frequency of a class constitutes its utility), SHIELD can be compared to the exponential mechanism (introduced in [30]) which samples its output following the softmax distribution of the utility. However, the sampling in the encrypted domain constrains the shape of the probability distribution and introduces a dependency of the practically implementable distributions with the computational efficiency of the operator.

[4] At least, we get an infinite moments accountant, compelling us to use the classical composition theorem and then precluding reasonable DP guarantees.

Algorithm 2: Multi-degree SHIELD

Input : number of vectors n, number of classes K, list of encrypted votes Z, polynomial $(a_p)_{p\in[D]}$, offset ω

Output : $res = z^{(i_0)}$ where $i_0 \in [n]$

1 $Z \leftarrow Z$ augmented by ω encrypted one-hot encodings for each class;
2 $res \leftarrow (0,\dots,0) \in (\mathbb{Z}_2)^K$;
3 $found\_not\_null \leftarrow 0$;
4 **for** p *from* D *to* 1 **do**
5 **for** j *in* $[a_p]$ **do**
6 $\pi \leftarrow (1,\dots,1) \in (\mathbb{Z}_2)^K$;
7 **for** l *in* $[p]$ **do**
8 Draw a vector z of Z uniformly at random;
9 $\pi \leftarrow \pi \otimes z$;
10 **end**
11 $res \leftarrow res \oplus (1 \oplus found\_not\_null) \otimes \pi$;
12 $is\_not\_null \leftarrow \bigoplus_{k=1}^{K} \pi_k$;
13 $found\_not\_null \leftarrow found\_not\_null \veebar is\_not\_null$;
14 **end**
15 **end**

Note that softmax has been approximately implemented in FHE through polynomial approximation [28] but this requires a quite high multiplicative depth (with a polynomial of degree 12 for approximating the exponential function and even more for approximating the inverse function) and results in a significant computational overhead. Moreover, using such an implementation would still require additional homomorphic operations like comparisons to actually sample the output according to this distribution.

Rather, a method of sampling that follows the exponential distribution by construction, in the spirit of SHIELD as presented in this paper, would be more seducing. Sampling each vote independently with a fixed probability would actually yield an output distribution that exponentially depends on the vote frequencies but it seems that the probability of failing by not outputting any class would be quite high for practical parameters. We let further work on this question as a perspective.

5 FHE IMPLEMENTATION OF SHIELD

Algorithm 2 is a generic version of SHIELD that actually needs to be adapted for an implementation using an HE cryptosystem. We consider two kinds of encodings, depending on the encryption scheme that we use:

- With the BFV cryptosystem, we use batching i.e., encode a number of values together in a single polynomial which is then encrypted. A single operation on a ciphertext thus leads to the same operation applied to all values encoded (slots) inside the ciphertext in a Single Instruction, Multiple Data (SIMD) fashion. On the downside, BFV performances are very sensitive to multiplicative depth and inter-slot operations are costly.
- With the TFHE cryptosystem [12], we use a single ciphertext to encrypt a single value i.e. proceed in a Single Instruction, Single Data (SISD) fashion. This is a priori less efficient than using SIMD but the performances are decoupled from the multiplicative depth of Algorithm 2 and costly operations such as slot rotations are avoided.

To sum up, we implement SHIELD with two separate methods: one uses the BFV cryptosystem with SIMD operations (with results provided in Section 5.1); the other uses the TFHE cryptosystem with SISD operations (with results provided in Section 8.3).

5.1 Implementing SIMD-SHIELD

Although using BFV allows us to speed up SHIELD considerably by batching different samples together in the same ciphertext, some constraints require adapting parts of Algorithm 2 for them to work.

a. Multiplicative depth. As it is the case for other similar HE schemes, we need to set the parameters of BFV according to the multiplicative depth of the computation. The higher the multiplicative depth, the larger the parameters, and the less efficient the overall computation. For this reason, some parts of the algorithm, like Line 9, need to be changed. We can store all of the values for $\mathsf{Enc}(z)$ over the loop and multiply them in a classic tree-based approach (instead of multiplying them sequentially) which reduces the multiplicative depth of the computation from p to $\log_2(p)$.

The same change is applied everywhere it is needed, that is to say at Lines 11 and 13 of Algorithm 2.

b. Selecting the teacher. Selecting the voter, also called teacher because of SPEED application case (see Section 6), at lines 8 and 9 of Algorithm 2 is easy enough when SHIELD algorithm is called for a single sample at once. However, in order to speed the algorithm up and fully make use of the SIMD property of the BFV cryptosystem, we actually run SHIELD algorithm for a number of samples at a time.

For instance, if $\pi^{(i)}$ is the π vector of K values for sample i, then the actual vector encoded in the ciphertext for the packed algorithm would be

$$\pi = \left(\pi_1^{(1)},\dots,\pi_K^{(1)},\pi_1^{(2)},\dots,\pi_K^{(2)},\dots\right) \tag{1}$$

This allows us to use the full size of the polynomials we encrypt. These polynomials have degrees in the order of $\approx 2^{15}$ while K is usually less than 1000.

Therefore the teacher selection step has to be modified. The new encoding of teacher t's vote for sample i is:

$$\left(0,\dots,0|0,\dots,0|\dots|z_t^{(i)}|\dots|0,\dots,0|\right) \in \left(\mathbb{B}^K\right)^N$$

which is a vector with N slots of K binary values where $z_t^{(i)}$ is teacher t's original one-hot encoded vote for sample i. It is located at the i^{th} slot of the encoding. From now on we'll call $z_t^{(i)}$ this new encoding of the teacher's vote. Algorithm 3 presents the process for teacher selection and creation of the π vector using this new encoding. At step 9 a mask m_t is updated but no detail is given for clarity.

Algorithm 3: Teacher selection. With n the total number of teachers, this algorithm describes the actual steps for selecting the teachers that get to vote in the SIMD encoding paradigm.

```
1  for j in [a_p] do
2      for l in [p] do
3          for t in [n] do
4              z_t ← (0, ..., 0);
5          end
6          for i in [N] do
7              Draw a vector z_t^(i) of Z uniformly at random;
8              z_t ← z_t ⊕ z_t^(i) ;
9              update m_t ;
10         end
11         π ← (1, ..., 1);
12         for t in [n] do
13             z_t ← z_t ⊕ m_t ;
14             π ← π ⊗ z_t ;
15         end
16     end
17 end
```

For every teacher t, the mask m_t is a plaintext vector that contains 0s in the place of samples for which the teacher votes and 1s in the place of samples for which the teacher does not vote. As an example, for $K = 2$ and $N = 4$, if teacher t votes for samples 1 and 3, then $m_t = (0, 0|1, 1|0, 0|1, 1)$.

This mask is then added to z_t before the multiplication to the π vector so that all the samples that are not voted on do not impact the result: their slots are filled by ones. If the mask is not used, then all non-selected slots will be filled with 0s and therefore would set everything to 0 after the multiplication.

For this multiplication, as mentioned before, we opt to store all of the z_t vectors and create a multiplication tree to reduce the multiplicative depth.

c. Rotations. One other constraint that schemes such as BFV suffer from, is that it is very hard and costly to extract certain values from the ciphertext to apply an operation *only* to them. Such is the case when trying to implement Line 12 in Algorithm 2. The individual π_k values cannot be extracted and summed together in a straight-forward manner. One thing we can do however, at a relatively low cost (both in terms of performance and noise inside the ciphertext), is to rotate the vector encoded in the ciphertext. This leads to an implementation of Line 12 that we present using the example $\pi^{(i)} = (0, 0, 0, 1, 0, 0, 0, 0, 0, 0)$.

$$
\begin{aligned}
&(0, 0, 0, 1, 0, 0, 0, 0, 0, 0) \\
+\,&(0, 0, 0, 0, 0, ?, ?, ?, ?, ?) && \leftarrow \text{rotate by } 5 \\
=\,&(0, 0, 0, 1, 0, ?, ?, ?, ?, ?) \\
+\,&(0, 1, 0, ?, ?, ?, ?, ?, ?, ?) && \leftarrow \text{rotate by } 2 \\
=\,&\ldots
\end{aligned}
$$

One can see how, using $\log_2(K)$ rotations and sums, we can obtain $\sum_j \pi_j^{(i)}$ in the *first* coordinate of the $\pi^{(i)}$ vector. The question marks ? represent values that are rotated over from the next slot, (recall the complete form of π in Equation 1).

Therefore, we cannot control the values in the rest of the coordinates. And this is not enough. For Line 13 to work, we need to have a vector where *all* coordinates $\pi_j^{(i)}$ are filled with $\sum_j \pi_j^{(i)}$, not just the first one. To obtain this, we have to multiply by a plaintext with values $(1, 0, 0, \ldots)$ to select only for the first coordinate of π and then re-populate the rest of the coordinates using rotations and sums exactly in the opposite way as used for the computation of the sum of the $\pi_j^{(i)}$ values.

d. Packing the polynomial rounds together. Up until now, for clarity, we presented a version of our algorithm that packed all or some of the N samples together in a single ciphertext. In practice, to speed up the computation further, we also pack the polynomial rounds together. What we mean by "polynomial rounds" is the two *for* loops at Lines 1 and 2 in Algorithm 3. We can remove these *for* loops and compute them in parallel in a single ciphertext.

6 AN APPLICATION CASE: SPEED

6.1 PATE workflow

Our SHIELD operator is very well adapted to a learning protocol called SPEED, from [20], itself inspired from PATE [33]. SPEED method is illustrated by Figure 1, inspired from [20]. Assuming the existence of a public unlabeled database Δ (we will keep this notation throughout the paper), PATE enables several data-owners, called *teachers*, to collaboratively train a classification model without outsourcing their data that are considered private. The idea is to label Δ and use it to train the final classification model, called the *student* model or simply the student. To do so, each teacher is asked to train a model beforehand for the same task as the student's target task with its own data only. For each sample of Δ to label, every teacher infers a label through its model and sends this label to an aggregation server. The server then counts the number of labels received for each class, also seen as votes, and outputs the class with the most votes which is sent to the student for training.

As it was described, the baseline PATE protocol does not protect the data from the actors of the process, namely the teachers, the server, the student and the end-users. All of them are considered honest-but-curious, which means that they execute their task correctly but may use the data they have access to to retrieve sensitive information about the teachers' data. The teachers do not actually get any information other than their own data unless they are also end-users of the student model, which may be the case in many real-life scenarios.

6.2 Data protection in SPEED

To prevent the student and *a fortiori* the end-users (by post-processing) from discovering sensitive information by attacks such as e.g. model inversion or membership inference, we apply DP. The teachers noise their votes before sending them to the server.

One could argue that the noise added by the teachers would also blur the sensitive information to the server. Nevertheless, the

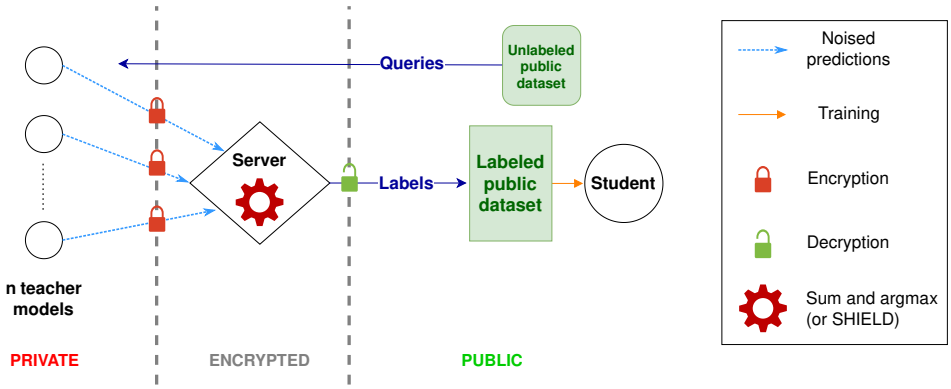

Figure 1: SPEED learning protocol (PATE uses the same protocol but in the plaintext domain)

added noise is precisely scaled so that it protects the output of the aggregation, i.e. the class with the most votes, without harming the student accuracy too much. If the individual votes sent to the server were to be protected by DP *before* aggregation, thus achieving what is called *local DP* ([13, 25, 26]), this would require much more noise, too much noise to ensure a reasonable accuracy for the student model. As a consequence, the votes need to be protected from the server another way. This is where homomorphic encryption makes its entrance. After noising their votes, the teachers encrypt them. The server then receives the encrypted votes and perform their aggregation (sum and argmax) in the homomorphic layer. Finally, the output of the aggregation is sent to the student that owns the decryption key and is therefore able to decrypt it.

A real-life scenario could involve hospitals that own patients' medical data and aim at training a global model that would help the early diagnosis of a specific disease. In this case, the end-users would be the hospitals themselves.

6.3 Faster SPEED with SHIELD

Our SHIELD operator can be used to replace the sum and argmax computations on the server side in SPEED (represented by the gear wheel in Figure 1). After receiving all the votes from the teachers *without noise*, the server randomly picks some vectors with replacement as described in Section 4.1. Note that, being honest-but-curious, the server is trusted to compute SHIELD without mistake. Interestingly, the rest of SPEED protocol remains unchanged, except the sending of dummy one-hot encodings by some teachers, according to the offset parameter (see Section 4.3).

7 ANALYSIS OF SHIELD

7.1 Computational complexity of SHIELD

Compared with previous argmax HE computation methods, SHIELD is unique in that its complexity depends only linearly on the number of classes for the chosen machine learning problem. Indeed, the main impact of an increase in the number of classes is that the encoding space increases by the same amount (and therefore the time overhead is linear). A secondary impact is the logarithmic increase in the number of rotations needed for the computation of $\sum_j \pi_j^{(i)}$ as seen in Section 5.1. All previous works use one (or a combination) of two methods to evaluate a homomorphic argmax over a number of values: a tournament method or a league method. We refer the reader to [9, 23] for specific implementation details. Here we focus on their complexity with respect to the number of classes.

- a league is a system of comparison where every value is compared with every other value. The winner is the value that was greater than every other one. Think of a football league like French first division league ("Ligue 1") for this kind of system. The use of a league method yields a quadratic complexity in the number of classes. This leads to very high performance overheads as the number of classes increases. However, contrary to the tournament method, increasing the number of classes does not affect the multiplicative depth of the circuit to be evaluated. This is what makes this method useful in the homomorphic domain in spite of its complexity.
- a tournament is a system where values are compared two-by-two and the losers are discarded at every round. Think of the FIFA World Cup for this kind of system. Using a tournament method has a - theoretical - linear complexity in the number of classes. In practice, this is not the case. As the number of classes increases, the comparison tree used for the evaluation increases in depth logarithmically. For leveled homomorphic schemes such as BFV or BGV (those we use in this article) used in [23], this means an increase in parameter size to match the multiplicative depth of the new tree. In turn, this impacts the performance of the overall scheme on top of the theoretical linear increase. After a given point, the increase in parameter size becomes prohibitive and one needs to resort to finishing the computation using a league method as they do in [23].

Compared to all other existing works therefore, ours scales much better with the number of classes and therefore fits particularly well with use-cases with high numbers of classes.

7.2 *A priori* accuracy metrics

The ultimate accuracy that we want to maximize in SPEED application case is obviously the testing accuracy of the student model. Nevertheless, it could be interesting to measure the accuracy of

the argmax operator itself, independently of the student training. Also, even if this depends on the teachers' votes and thus on the used dataset, this enables us to evaluate polynomial parameterizations without performing the student training, which is much faster and allows to test much more parameterizations. We call such an accuracy an *a priori* accuracy.

The most straightforward way to define the argmax accuracy is to consider the probability of getting the exact argmax. Nevertheless, this approach treats any mistake the same way. It could be argued that outputting, say, the class that received the second greatest number of votes is better than outputting the least preferred class. Taking such a concern into account in our metric would also give a better hint about the student accuracy. Indeed, while the most preferred class (i.e. the class with the most votes, or equivalently the exact argmax) is not always the actual class of the sample - called the *ground truth* class -, a class with a lot of votes is more likely to be the ground truth class.

We could then make the assumption that the frequency of votes for a class is proportional to the probability of this class being the ground truth class of the sample (which is not necessarily the most preferred class). This would correspond to an assumption of well-calibrated vote distributions. In this context, we introduce another a priori accuracy metric which is the *probability of outputting the ground truth class* of the sample. We call this metric the *ground truth accuracy*, since it does not focus on outputting the exact argmax but rather the ground truth class. For a given sample x, p_k being the probability of SHIELD outputting class k and $GT(x)$ denoting the ground truth class of x [5], the law of total probability gives the following expression for the ground truth accuracy for x, written $\mathrm{GTA}(x)$:

$$\mathrm{GTA}(x) = \sum_{k=1}^{K} \mathbb{P}[GT(x) = k]p_k = \sum_{k=1}^{K} \frac{n_k}{n} p_k.$$

Of course, both metrics must be averaged on all the samples sent to the teachers.

7.3 Differential privacy analysis

Since the student model training requires many requests to the teachers and, indirectly, to their private datasets, we use, as in [20], the moments accountant technique [1] to get a better privacy cost over composition.

We here consider that two databases d and d' are adjacent if they are the concatenations of the datasets from the same number of teachers and only one teacher differs from one database to the other. This implies that either all the n'_k, counts for database d', for $k \in [M]$, are equal to the n_k, counts for database d, in which case the corresponding moments accountant is null, or the n'_k differ from the n_k only for two values of k, say k_1 and k_2, such that $n'_{k_1} = n_{k_1} - 1$ and $n'_{k_2} = n_{k_2} + 1$ (i.e. the differing teacher votes for k_1 in d and k_2 in d').

Unlike most DP mechanisms, the stochastic behavior of our operator does not come from an additional random noise, since the operator is inherently probabilistic. This is this very property of our operator that we leverage to ensure DP. The moments accountant measures the discrepancy between the output distributions associated with two adjacent databases [6]. Computing the privacy cost of the training, as well as the *a priori* accuracy, requires thus knowing the probabilities of outputting each class.

7.4 Computing the probability distribution of the output

We compute the probability distribution of the output of the algorithm SHIELD with a given polynomial parameterization in a recursive manner.

For a sample x of Δ, let $\mathcal{M}_{P,x}$ be the mechanism that takes the whole database (concatenation of the teachers' datasets) as input and outputs the class sent to the student i.e. the output of SHIELD, with the polynomial parameterization $P \in \mathbb{N}[X]$.

Let d be the database composed of the teachers' data. Let k be a class of the problem.

If $P = X^p$, $p \in \mathbb{N}^*$, $\mathbb{P}[\mathcal{M}_{P,x}(d) = k] = \left(\frac{n_k}{n}\right)^p$.

If $P = X^p + Q(X)$, where $Q \in \mathbb{N}[\mathbb{X}]$ and $p \in \mathbb{N}^*$ is greater or equal than the degree of Q,

$$\mathbb{P}[\mathcal{M}_{P,x}(d) = k] = \left(\frac{n_k}{n}\right)^p + \left(1 - \sum_{j=1}^{K} \left(\frac{n_j}{n}\right)^p\right) \mathbb{P}[\mathcal{M}_{Q,x}(d) = k].$$

Using these expressions, we simply compute the moments accountant for each query by taking the maximum over all pairs (d, d') such that d is the database constituted by the concatenation of the teachers' database and d' is a database adjacent to d. We then derive the overall privacy cost using the moments accountant's composition properties.

Note that the obtained DP guarantees are *data-dependent* since we explored only the pairs of adjacent databases such that one of them is the actual database given by our application. The very values ϵ and δ of these guarantees then reveal some information about the training data. In a real-life scenario, these values should be sanitized before being published, as in [34] for instance, but this is beyond the scope of this work.

7.5 The differential privacy analysis does not apply to the server

When we compute the probabilities of outputting a class, we do not suppose anything about whose votes are drawn i.e. we do not condition the probabilities on some particular drawing event. This amounts to assume that the adversary only sees the output class, and does not know, in particular, which teachers were selected in the sampling. This assumption cannot apply to the server since it draws the one-hot encodings itself and knows which teachers they come from, for having receiving the encodings one by one from the teachers. Appendix A gives an insight of why this subtlety may be problematic.

This observation shows that we need to constrain the server not to see the student model once it is trained. Note that the information leakage induced by the server's knowledge may not jeopardize

[5] p_k and n_k obviously also depend on x but we omit to write this dependence for simplicity. p_k can be computed as showed in Section 7.4.

[6] The moments accountant involves the logarithm of the expected ratio of the probabilities of outputting a certain value for two adjacent databases but we do not detail it here since this is not the focus of this paper

much the data privacy in practice. We only argue here that our DP analysis does not allow us to derive DP guarantees from the point of view of the server, which might be possible with a more involved (and likely quite complex) analysis, although with probably worse guarantees.

8 EXPERIMENTAL RESULTS

8.1 Choice of the polynomial parameterization

We tested SPEED with SHIELD on MNIST dataset [27]. While the offset parameter has been set to 1, a key aspect of our experiments is the choice of a polynomial parameterization that realizes a good trade-off between model accuracy, DP guarantees and computational efficiency. Since the computational time overall depends on the sum of coefficients and the degree of the polynomial parameterization, we proceeded by constraining the maximum degree and the maximum value for the sum of coefficients of the polynomials. We fixed the maximum degree to 4 to achieve a reasonable balance between the multiplicative depth, the number of tries and the argmax accuracy. For several integer values (6, 12, 17, 32), we considered all the polynomials of degree at most 4 whose sum of coefficients is less than this value. We do not go beyond a sum of coefficients equal to 32 to keep the computational time low. We then computed the DP guarantee ϵ, with $\delta = 10^{-5}$ being fixed, for each polynomial, as well as its GTA score that acts as a proxy for the student model accuracy which could not be determined in reasonable time for so many polynomials. Finally, we focused on the polynomials belonging to the Pareto front for these two criteria - DP guarantee ϵ and GTA - and picked the ones that yielded among the best DP guarantees without harming the accuracy too much. In practice, as it can be seen on Figure 2 the DP guarantee guided more our choice because the GTA, besides being only a heuristic for the actual student model accuracy, did not vary much among the polynomials of the Pareto front. Note that the GTA of the exact argmax is 72.35%. The chosen polynomials are respectively $2X^3 + 3X^2 + X$, $2X^4 + 6X^3 + 3X^2 + X$, $6X^4 + 6X^3 + 4X^2 + X$ and $8X^4 + 6X^3 + 4X^2 + X$ for a sum of coefficients of at most 6, 12, 17, 32[7]. We did not display the Pareto front for a sum of coefficients of at most 6 because it only contains one polynomial.

Table 1 displays the GTA, the student model accuracy and the DP guarantee ϵ for the chosen polynomial parameterizations, $\delta = 10^{-5}$ being fixed. The GTA and DP guarantee are averaged on the whole set of 8000 samples used for semi-supervised training, the DP guarantee being remultiplied by 100, the number of actual queries to the teachers. The student model accuracy is averaged over ten runs, each of which used a different random subset of 100 samples as labeled samples. The table also displays the number of correctly labeled samples (comparing to the ground truth label) out of the 8000 samples. Note that, the observed non-mononotonous relationship between the model and argmax accuracy may deserve more investigations from an ML point of view. The variance of the model accuracy among the runs is quite important and may explain why the accuracy surprisingly does not increase when the

[7]The chosen polynomial among the ones with a sum of coefficients at most 32 has a sum of coefficients equal to 19 only. This is good news for computational complexity because it allows us to batch all samples into a single ciphertext and therefore optimize the computation.

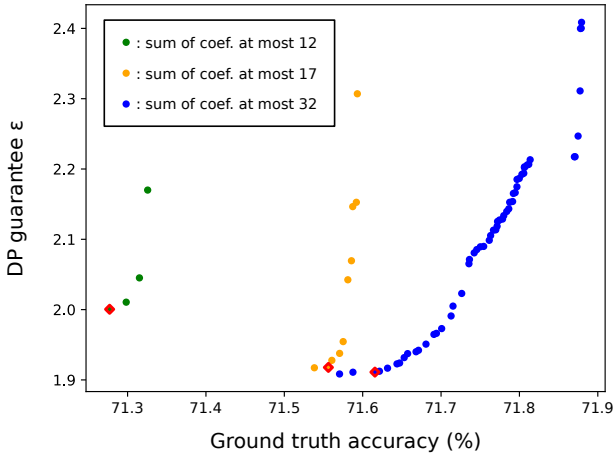

Figure 2: Pareto fronts of the polynomials for a fixed maximum sum of coefficients. The polynomials we chose for running the student model training are indicated by red-edged diamonds.

polynomial is better in terms of both GTA and number of correct labels. The polynomial $2X^4 + 6X^3 + 3X^2 + X$ being the one that yields the best model accuracy, we shall focus on this polynomial parameterization when comparing our computational efficiency with the state of the art (see Section 8.2).

polynomial	GTA	number of correct labels	model accuracy	ϵ
exact argmax	72.35%	7516 (93.95%)	95.36%	∞
$2X^3 + 3X^2 + X$	70.06%	7166 (89.58%)	90.91%	2.39
$\mathbf{2X^4 + 6X^3 + 3X^2 + X}$	71.26%	7327 (91.59%)	**94.66%**	2
$6X^4 + 6X^3 + 4X^2 + X$	71.56%	7358 (91.98%)	93.39%	1.92
$8X^4 + 6X^3 + 4X^2 + X$	71.62%	7367 (92.09%)	93.15%	1.91

Table 1: Accuracy and DP guarantee (with $\delta = 10^{-5}$) obtained with several polynomial parameterizations.

8.2 SIMD SHIELD with BFV

For our implementation of the SIMD SHIELD algorithm, we use the BFV cryptosystem in the openFHE library [4]. The parameters we choose are the following: $\log_2(q) = 540$; $p = 65537$; $m = 65536$; $N = 32768$. These parameters achieve a security level of $\lambda = 128$ bits with a standard deviation of 3.2. Our implementation was tested on a machine with an AMD Opteron(tm) Processor 6172 using a *single thread*.

We achieve performances presented in Table 2 for a set of different polynomial parameterizations. Although we tested using the MNIST data set, the performance of an HE algorithm does not depend on the underlying data *by construction*. Otherwise one could infer something on the data from seeing the computation happen in the encrypted domain. For our implementation, we need to run the SHIELD algorithm over 100 samples. In the table however, we also present computation times for the case whereby we optimize the batching space with a higher number of samples to

give an idea of what computation times could be achieved by optimizing parameters further. For now, these optimizations are not yet possible in keeping with the Homomorphic Encryption Security Standard [3] which recommends the use of power-of-two cyclotomic polynomials. A new standard is reported to be in the works which would open applications to the secure use of non-power-of-two cyclotomic polynomials. That would allow us to optimize our parameters further.

polynomial	samples	time (s)	time/sample (s)
$2X^3 + 3X^2 + X$	100	87.2	0.87
	341	112	0.33
$\mathbf{2X^4 + 6X^3 + 3X^2 + X}$	**100**	**123**	**1.23**
	143	**135**	**0.94**
$6X^4 + 6X^3 + 4X^2 + X$	100	138	1.38
$8X^4 + 6X^3 + 4X^2 + X$	100	144	1.44
paper	samples	time (s)	time/sample (s)
[20]	100	390	3.9
[20] + [9]	100	160	1.6
[23]	100	190	1.9

Table 2: Performance for the SIMD implementation of SHIELD (for 10 classes) for different polynomial parameterizations compared with previous work implementing exact argmax computations.

Table 2 also compares our method with previous existing methods for *non-stochastic argmax computations*. Among these methods, the one presented in [20] as well as its later improvement in [9] perform worse overall for all polynomial parameterizations that we tested. It is important to note that these methods do not (and cannot) use batching by construction. Therefore the time per sample is fixed and does not depend on the amount of samples processed. Times in Table 2 for [23] are taken from their Table 4 because it most closely matches our use-case. However important differences remain: we report their timings (amortized over 100 samples) extrapolated from 8 (1.52 s) to 10 (1.9 s) classes (this is conservative as the complexity of their method is at least linear in the number of items); additionally their timings are for a minimum computation, which is less time-consuming than an argmin computation, but no times are given for an argmin in [23]. On top of that, [23] makes heavy use of batching and also report additional amortized timings. However, by construction, they are constrained to batching sizes much higher than ours for security purposes, therefore the amortized time of 0.03s per sample reported in [23] cannot be obtained over 100 samples.

8.3 Bitwise SHIELD with Cingulata

To show the interest of the batching approach, we also implemented the basic version of SHIELD, as described in Algorithm 2, with Cingulata crypto-compiler and its TFHE backend.

Let us remind that Cingulata, formerly known as Armadillo [8], is a toolchain and run-time environment (RTE) for implementing applications running over homomorphic encryption. Cingulata provides high-level abstractions and tools to facilitate the implementation and the execution of privacy-preserving applications expressed as Boolean circuits, a representation which is natural for SHIELD.

Table 3 shows the execution times of SHIELD for different polynomial parameterizations when performed in a SISD fashion with TFHE and Cingulata. The experiments were performed with a single thread on an Intel Xeon processor with 16 GB of memory and Ubuntu 20.04 operating system. As shown in the table, the execution time of SHIELD increases with the degree of the polynomial and the sum of the polynomial coefficients. As expected, the overall performances are highly below the ones obtained when using BFV and its batching capabilities.

polynomial	samples	time (s)	time/sample (s)
$2X^3 + 3X^2 + X$	100	495,3	4,95
$2X^4 + 6X^3 + 3X^2 + X$	100	14287,8	14,29
$6X^4 + 6X^3 + 4X^2 + X$	100	20936,7	20,93

Table 3: Performance for the Cingulata with TFHE implementation of SHIELD

9 CONCLUSION AND PERSPECTIVES

We proposed SHIELD, a homomorphic stochastic operator whose design, necessary for fast homomorphic computations, yields DP as a natural "by-product". This work reconciliates two complementary but usually independent - or even mutually constraining - privacy tools in an all-in-one operator whose inaccuracy is a crucial feature.

We hope this work will encourage new works on the design of private algorithms where FHE (or other cryptographic primitives) and DP leverage the advantages of each other. For instance, developing algorithms that would be useful in other settings than an election and broaden the scope of machine learning applications seems promising. In this perspective, an argmax algorithm that takes an histogram of the votes as input rather than the "physical" votes represented as vectors would have a more general applicability.

Testing SHIELD on more difficult datasets and especially datasets with numerous classes could reveal its full potential. Besides, a more thorough theoretical study to get results that may lead us through the choice of the parameters (polynomial, offset) is desirable. Other versions including sampling without replacement (Appendix A) or an exponential version of SHIELD (Section 4.4) would also deserve theoretical and experimental analyses. Studying SHIELD in terms of strategy-proofness and fairness could be interesting too and would extend the added value of SHIELD to the area of computational social choice and voting rules [11].

ACKNOWLEDGMENTS

We thank Victor Berger early discussions on the idea of ground truth accuracy as well as the anonymous referees for their comments which lead to improvements of this paper. This work was supported by the France 2030 ANR Project ANR-22-PECY-003 SecureCompute and the European Union's Horizon Europe Research and Innovation Programme under Grant Agreement No. 101070670 (ENCRYPT - A Scalable and Practical Privacy-preserving Framework).

REFERENCES

[1] Martin Abadi, Andy Chu, Ian Goodfellow, H Brendan McMahan, Ilya Mironov, Kunal Talwar, and Li Zhang. 2016. Deep learning with differential privacy. In *ACM SIGSAC*. 308–318.

[2] Naman Agarwal, Ananda Theertha Suresh, Felix Xinnan X Yu, Sanjiv Kumar, and Brendan McMahan. 2018. cpsgd: Communication-efficient and differentially-private distributed sgd. *NeurIPS* 31 (2018), 7564–7575.

[3] Martin Albrecht, Melissa Chase, Hao Chen, Jintai Ding, Shafi Goldwasser, Sergey Gorbunov, Shai Halevi, Jeffrey Hoffstein, Kim Laine, Kristin Lauter, Satya Lokam, Daniele Micciancio, Dustin Moody, Travis Morrison, Amit Sahai, and Vinod Vaikuntanathan. 2018. *Homomorphic Encryption Security Standard.* Technical Report. HomomorphicEncryption.org, Toronto, Canada.

[4] Ahmad Al Badawi, Jack Bates, Flavio Bergamaschi, David Bruce Cousins, Saroja Erabelli, Nicholas Genise, Shai Halevi, Hamish Hunt, Andrey Kim, Yongwoo Lee, Zeyu Liu, Daniele Micciancio, Ian Quah, Yuriy Polyakov, Saraswathy R.V., Kurt Rohloff, Jonathan Saylor, Dmitriy Suponitsky, Matthew Triplett, Vinod Vaikuntanathan, and Vincent Zucca. 2022. OpenFHE: Open-Source Fully Homomorphic Encryption Library. Cryptology ePrint Archive, Paper 2022/915. (2022).

[5] Johes Bater, Xi He, William Ehrich, Ashwin Machanavajjhala, and Jennie Rogers. 2018. Shrinkwrap: efficient sql query processing in differentially private data federations. *Proceedings of the VLDB Endowment* 12, 3 (2018).

[6] Andrea Bittau, Úlfar Erlingsson, Petros Maniatis, Ilya Mironov, Ananth Raghunathan, David Lie, Mitch Rudominer, Ushasree Kode, Julien Tinnes, and Bernhard Seefeld. 2017. Prochlo: Strong privacy for analytics in the crowd. In *Proceedings of the 26th symposium on operating systems principles*. 441–459.

[7] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. 2014. (Leveled) Fully Homomorphic Encryption without Bootstrapping. *ACM Trans. Comput. Theory* 6, 3, Article 13 (7 2014), 36 pages.

[8] Sergiu Carpov, Paul Dubrulle, and Renaud Sirdey. 2015. Armadillo: a compilation chain for privacy preserving applications. In *Proceedings of the 3rd International Workshop on Security in Cloud Computing*. 13–19.

[9] Olive Chakraborty and Martin Zuber. 2022. Efficient and Accurate Homomorphic Comparisons. In *Proceedings of the 10th Workshop on Encrypted Computing & Applied Homomorphic Cryptography (WAHC'22)*. Association for Computing Machinery, 35–46. https://doi.org/10.1145/3560827.3563375

[10] Albert Cheu, Adam Smith, Jonathan Ullman, David Zeber, and Maxim Zhilyaev. 2019. Distributed differential privacy via shuffling. In *Advances in Cryptology–EUROCRYPT 2019: 38th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Darmstadt, Germany, May 19–23, 2019, Proceedings, Part I 38*. Springer, 375–403.

[11] Yann Chevaleyre, Ulle Endriss, Jérôme Lang, and Nicolas Maudet. 2007. A short introduction to computational social choice. In *International conference on current trends in theory and practice of computer science*. Springer, 51–69.

[12] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. Faster Fully Homomorphic Encryption: Bootstrapping in Less Than 0.1 Seconds. In *Advances in Cryptology – ASIACRYPT 2016*, Jung Hee Cheon and Tsuyoshi Takagi (Eds.). Vol. 10031. Springer Berlin Heidelberg, 3–33. https://doi.org/10.1007/978-3-662-53887-6_1 Series Title: Lecture Notes in Computer Science.

[13] John C Duchi, Michael I Jordan, and Martin J Wainwright. 2013. Local privacy and statistical minimax rates. In *2013 IEEE 54th Annual Symposium on Foundations of Computer Science*. IEEE, 429–438.

[14] Cynthia Dwork, Krishnaram Kenthapadi, Frank McSherry, Ilya Mironov, and Moni Naor. 2006. Our data, ourselves: Privacy via distributed noise generation. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 486–503.

[15] Cynthia Dwork, Aaron Roth, et al. 2014. The algorithmic foundations of differential privacy. *Foundations and Trends® in Theoretical Computer Science* 9, 3–4 (2014), 211–407.

[16] Úlfar Erlingsson, Vitaly Feldman, Ilya Mironov, Ananth Raghunathan, Kunal Talwar, and Abhradeep Thakurta. 2019. Amplification by shuffling: From local to central differential privacy via anonymity. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms*. SIAM, 2468–2479.

[17] Junfeng Fan and Frederik Vercauteren. 2012. Somewhat Practical Fully Homomorphic Encryption. *IACR Cryptology ePrint Archive* 2012 (2012), 144.

[18] Matthew Franklin and Moti Yung. 1992. Communication Complexity of Secure Computation (Extended Abstract). In *Proceedings of the Twenty-Fourth Annual ACM Symposium on Theory of Computing (STOC '92)*. Association for Computing Machinery, New York, NY, USA, 699–710.

[19] Slawomir Goryczka and Li Xiong. 2015. A comprehensive comparison of multiparty secure additions with differential privacy. *IEEE transactions on dependable and secure computing* 14, 5 (2015), 463–477.

[20] Arnaud Grivet Sébert, Rafaël Pinot, Martin Zuber, Cedric Gouy-Pailler, and Renaud Sirdey. 2021. SPEED: secure, PrivatE, and efficient deep learning. *Machine Learning* 110, 4 (2021), 675–694.

[21] Meng Hao, Hongwei Li, Xizhao Luo, Guowen Xu, Haomiao Yang, and Sen Liu. 2019. Efficient and privacy-enhanced federated learning for industrial artificial intelligence. *IEEE Transactions on Industrial Informatics* 16, 10 (2019), 6532–6542.

[22] Alberto Ibarrondo, Hervé Chabanne, Vincent Despiegel, and Melek Önen. Grote: Group Testing for Privacy-Preserving Face Identification. In *Proceedings of the Thirteenth ACM Conference on Data and Application Security and Privacy* (2023-04-24). ACM, 117–128. https://doi.org/10.1145/3577923.3583656

[23] Ilia Iliashenko and Vincent Zucca. 2021. Faster homomorphic comparison operations for BGV and BFV. *Proceedings on Privacy Enhancing Technologies* 2021, 3 (2021), 246–264. https://doi.org/10.2478/popets-2021-0046 Publisher: De Gruyter Open.

[24] Bin Jia, Xiaosong Zhang, Jiewen Liu, Yang Zhang, Ke Huang, and Yongquan Liang. 2021. Blockchain-enabled federated learning data protection aggregation scheme with differential privacy and homomorphic encryption in IIoT. *IEEE Transactions on Industrial Informatics* 18, 6 (2021), 4049–4058.

[25] Peter Kairouz, Sewoong Oh, and Pramod Viswanath. 2016. Extremal mechanisms for local differential privacy. *The Journal of Machine Learning Research* 17, 1 (2016), 492–542.

[26] Shiva Prasad Kasiviswanathan, Homin K Lee, Kobbi Nissim, Sofya Raskhodnikova, and Adam Smith. 2011. What can we learn privately? *SIAM J. Comput.* 40, 3 (2011), 793–826.

[27] Yann LeCun, Corinna Cortes, and CJ Burges. 2010. MNIST handwritten digit database. 2010. *URL http://yann. lecun. com/exdb/mnist* 7 (2010), 23.

[28] Joon-Woo Lee, HyungChul Kang, Yongwoo Lee, Woosuk Choi, Jieun Eom, Maxim Deryabin, Eunsang Lee, Junghyun Lee, Donghoon Yoo, Young-Sik Kim, et al. 2022. Privacy-preserving machine learning with fully homomorphic encryption for deep neural network. *IEEE Access* 10 (2022), 30039–30054.

[29] Baiyu Li, Daniele Micciancio, Mark Schultz, and Jessica Sorrell. 2022. Securing approximate homomorphic encryption using differential privacy. In *Annual International Cryptology Conference*. Springer, 560–589.

[30] Frank McSherry and Kunal Talwar. 2007. Mechanism design via differential privacy. In *48th Annual IEEE Symposium on Foundations of Computer Science (FOCS'07)*. IEEE, 94–103.

[31] Ilya Mironov. 2017. Rényi differential privacy. In *2017 IEEE 30th Computer Security Foundations Symposium (CSF)*. IEEE, 263–275.

[32] Tabitha Ogilvie. 2023. Differential Privacy for Free? Harnessing the Noise in Approximate Homomorphic Encryption. Cryptology ePrint Archive, Paper 2023/701. (2023).

[33] Nicolas Papernot, Martín Abadi, Ulfar Erlingsson, Ian Goodfellow, and Kunal Talwar. 2016. Semi-supervised knowledge transfer for deep learning from private training data. *arXiv preprint arXiv:1610.05755* (2016).

[34] Nicolas Papernot, Shuang Song, Ilya Mironov, Ananth Raghunathan, Kunal Talwar, and Úlfar Erlingsson. 2018. Scalable private learning with pate. *arXiv preprint arXiv:1802.08908* (2018).

[35] N. P. Smart and F. Vercauteren. 2011. Fully Homomorphic SIMD Operations. Cryptology ePrint Archive, Paper 2011/133. (2011).

[36] Timothy Stevens, Christian Skalka, Christelle Vincent, John Ring, Samuel Clark, and Joseph Near. 2022. Efficient Differentially Private Secure Aggregation for Federated Learning via Hardness of Learning with Errors. In *31st USENIX Security Symposium (USENIX Security 22)*. 1379–1395.

[37] Stacey Truex, Nathalie Baracaldo, Ali Anwar, Thomas Steinke, Heiko Ludwig, Rui Zhang, and Yi Zhou. 2019. A hybrid approach to privacy-preserving federated learning. In *Proceedings of the 12th ACM workshop on artificial intelligence and security*. 1–11.

[38] Jonathan Ullman. 2018. Tight lower bounds for locally differentially private selection. *arXiv preprint arXiv:1802.02638* (2018).

[39] Jelle Van Den Hooff, David Lazar, Matei Zaharia, and Nickolai Zeldovich. 2015. Vuvuzela: Scalable private messaging resistant to traffic analysis. In *Proceedings of the 25th Symposium on Operating Systems Principles*. 137–152.

[40] Sameer Wagh, Paul Cuff, and Prateek Mittal. 2018. Differentially private oblivious ram. *Proceedings on Privacy Enhancing Technologies* 2018, 4 (2018), 64–84.

[41] Sameer Wagh, Xi He, Ashwin Machanavajjhala, and Prateek Mittal. 2021. DP-cryptography: marrying differential privacy and cryptography in emerging applications. *Commun. ACM* 64, 2 (2021), 84–93.

[42] Wennan Zhu, Peter Kairouz, Brendan McMahan, Haicheng Sun, and Wei Li. 2020. Federated heavy hitters discovery with differential privacy. In *International Conference on Artificial Intelligence and Statistics*. PMLR, 3837–3847.

A INFORMATION LEAKAGE VIA KNOWLEDGE OF THE SELECTED TEACHERS

Let us propose some concrete situations where the DP guarantees are obviously not protecting the vote of the server's victim, i.e. a teacher whose vote the server wants to know. With the polynomial parameterization $X^k + X$, $k \in \mathbb{N} * \backslash \{1\}$, if the server draws $k-1$ teachers and its victim for the term X^k and then its victim for the

term X, then the server will know that the class sent to the student is its victim's vote. Supposing that the server knows the votes of all the teachers except its victim's (classical assumption in DP), it will be able to recover its victim's votes in many cases. For instance, with the polynomial parameterization $X^k + X$, $k \in \mathbb{N}^* \setminus \{1\}$, if the server draws k teachers who do not all have the same vote for the term X^k and its victim for the term X, then the class sent to the student is its victim's vote.

To address this vulnerability, we could think of an additional entity that receives the votes from the teachers and shuffles them before sending them to the server. However, the server would know if a same vote was drawn several times (remind that the drawing is with replacement), which still constitutes some information we did not account for in our DP analysis. Suppose that the server knows that all the teachers except its victim voted for a class c. Moreover, suppose that the offset parameter is set to 1 and that there are $|C|$ classes in the problem. Then, there are $|C| - 1$ votes different than c and the victim's vote, which is unknown. Assume that the polynomial parameterization is $|C|X^2 + X$. If the $2|C| + 1$ votes that the server drew are all from different sources - teachers or dummy one-hot encodings - (remind that the server knows it) and the output class is not c, then the server knows with certainty that its victim did not vote for c (otherwise, there would have been $|C| + 1$ drawn votes for c and, among the $|C|$ pairs the server drew for the term in X^2, no pair would have been composed of two identical votes different from c and at least one pair would have been composed of two votes for c and then the output would have been c).

We could extend the threat model and assume that the server has access to the final model by designing a more complex algorithm for which the teachers would be homomorphically selected, yet with a significant additional computational cost. Another interesting idea mentioned above would be to make use of an intermediate entity that would shuffle the encrypted votes before the server receives them, with inspiration from the ESA (Encode, Shuffle, Analyze) method from [6]. Nevertheless, as we saw, the server would still know if it selected several times the same teacher, even without knowing which one it is, and this is still theoretically an information leakage that is not simple to analyze. A way to solve this issue and to actually leverage the anonymity provided by the shuffling would be to design an algorithm that uses sampling *without* replacement and to force the teachers to send a new encryption of their votes for each polynomial rounds, which would significantly increase the complexity of the protocol and its communication cost.

B ON COUNTER-PRODUCTIVE NOISE FOR DATA-DEPENDENT DIFFERENTIAL PRIVACY GUARANTEES

Null data-dependent privacy cost of the exact argmax: While doing experiments on a subset of MNIST with polynomial parameterizations that yield better and better accuracies (up to the probability of getting the true argmax being more than 99,99%) we remarked that the value epsilon of the privacy cost did not increase much and did not seem to approach infinity. This surprising result suggested that the exact argmax operator had a finite privacy cost. Actually, on the subset we were working on, for every sample, the dominant class had at least two more votes than the second dominant class. We will say in the following that the distribution has a *highly dominant* argmax. This implies that, any database which is adjacent (i.e. differs from at most one teacher) to the database d we were working on has the same dominant class as d for every sample. As a consequence, the output of the argmax does not leak *any* information about which of two adjacent databases was used as input. In other words, the privacy cost of the exact argmax operator is null in this case.

Counter-productivity of the noise regarding privacy: On the contrary, the so-called private argmax operator (noised by an additional random noise as in PATE [33, 34] and SPEED [20] or intrinsically stochastic as in SHIELD) may output any class and the probabilities of outputting a class depends on the frequencies of the votes for all classes. As a consequence, even changing only one teacher will change the probabilities of outputting a certain class, even if the effect is mild. Therefore, since the output of the DP argmax operator gives information on the probability of outputting a class, it also gives information on the frequencies of the classes in the votes. We end up in a (particular) situation where applying noise is counter-productive in the sense that it increases the privacy cost of revealing the output (by an infinite factor actually). Note, however, that this was not the case for the entire MNIST training set but only for a certain subset of it.

The case of data-independent DP guarantees: This consideration only applies to *data-dependent* DP guarantees. In the data-independent case, the privacy cost of the exact argmax would be infinite because we would consider the maximum over all the pairs of adjacent databases i.e. all the possible pairs of distributions of n votes among K classes that differ by one vote on two classes.

Example of the age's sign: The noise addition degrading privacy guarantees is very counter-intuitive and may surprise *a priori*. Let us take a simple example to understand how the noise affects privacy. Revealing the sign of the age of a person is infinitely private (epsilon and delta null) if we assume that the adversary already knows that a person must have a positive age (quite natural assumption!). Imagine now that we noise the age with a unimodal noise, whose mode is zero, say a Gaussian noise, before computing the sign. The lesser the unnoised age, the more likely the sign of the noised age will be negative. This implies that revealing the sign of the noised age does leak some information about the unnoised age. Clearly, the noise addition does harm the privacy guarantees in this case. Nevertheless, note that this does not contradict the post-processing immunity of DP. Indeed, the noise is not added at the end, over the infinitely private sign of the age, rather, it is added before the computation of the sign, inside the mechanism and not afterwards. Thus, the noise addition cannot be considered as a post-processing.