Hybrid Privacy Scheme

Yavor Litchev, Abigail Thomas

Abstract

Local Differential Privacy (LDP) is a security algorithm that allows a central server to compute on data submitted by multiple users while maintaining the privacy of each user (Bebensee 2019). LDP is an efficient approach to privacy because it runs very quickly; however, as privacy increases, the accuracy of the computations decreases. Multi-Party Computation (MPC) is a process by which multiple parties work together to compute the output of a function without revealing their individual information (Goldreich 1998). MPC is highly secure and accurate for such computations, but it is very computationally expensive and slow. The proposed hybrid privacy model harnesses the benefits of both LDP and MPC to create a secure, accurate, and fast algorithm for machine learning.

Keywords: privacy, local differential privacy, multi-party computation

Introduction

People are becoming more and more aware of the importance of data privacy, as secret data breaches could negatively impact subjects. Instances such as identity fraud, stalking, and surveillance directly infringe on the rights and liberties of individuals. Moreover, leaked critical data could result in discrimination, economic losses, or even physical injury to individuals. For example, protecting medical records of individuals is crucial as it could result in future job discrimination, harassment, etc. Another example that illustrates the importance of privacy is the protection of voter identities as the leakage of such can make voters feel insecure, and as a result, not vote in their best interest, thus compromising the effectiveness of the political system. As shown by the examples, the protection of data privacy, while computations are being done on it, is crucial.

There are many existing algorithms designed to protect data privacy. Some of them achieve this by changing the underlying mechanism. One example is Federated Learning (Yang et al. 2019). It is a type of distributed machine learning algorithm where a number of machines, each with a piece of data stored locally, train a model jointly. It is assumed to protect data privacy because the machines do not have to exchange the raw data with each other. However, even if they do not transmit the raw data, properties of the private data can still be inferred from the gradient information they shared with each other (Yang et al. 2019). Therefore, information about the data is still leaking. In this project, we would like to propose a scheme that takes information leakage other than the raw data into consideration.

Besides designing a new scheme, there are also general efforts to protect data privacy. Data encryption converts the plaintext data into ciphertexts. Without a decryption key, the ciphertexts look like random bits. However, this only serves store-only purposes. Because of the security, no meaningful operations can be done on the ciphertexts. Homomorphic encryption schemes allow a party to perform computations directly on the ciphertexts while hiding the plaintexts (Gentry and Boneh 2009). Using homomorphic encryption could trivially eliminate privacy leakage, but as of today, it is too computationally expensive to be deployed in the real world. Differential privacy schemes add random noise to the data to hide the plain contents (Bebensee 2019). Differential privacy schemes are usually efficient because they only incur an overhead of sampling the random noise and later aggregating them. However, this leads to inaccurate results. Secure Multi-Party Computation (MPC) is a type of cryptographic protocol that allows multiple parties to compute a function regarding their private inputs without leaking any computationally reasonable information regarding these private inputs (Goldreich 1998; Beaver and Goldwasser 1989; Damgård et al. 2009; Canetti et al. 1996). MPC produces accurate computation results. However, similar to homomorphic encryption schemes, MPCs are usually computationally expensive.

In this paper, we present a hybrid approach to combine the advantages of differential privacy and MPC to protect data privacy with reasonable efficiency and accuracy. We also define a new notion of privacy considering both the loss implied by the final model (or the forwarding result computed by this model) and the loss during the training process. We also formally prove that our approach is secure. Our scheme provides a parameterized tradeoff between accuracy and privacy. We also present a theoretical analysis of the scheme.

We implemented this hybrid approach on a support vector machine in order to train a model using the MNIST dataset. The model (when a MASCOT MPC protocol is used in the

Hybrid Model) achieves an accuracy of 91% compared to the LDP baseline accuracy of 89%, and the model takes roughly 19 hours to run compared to the projected MPC baseline runtime of approximately 190 hours, showing that our hybrid model is better than each of the individual algorithms. These results are still seen when different MPC protocols are used: with a semi-honest Shamir secret sharing MPC protocol, the Hybrid Model achieves an accuracy of 96% as opposed to 89%, and takes about 19 minutes to run as opposed to 3 hours. Finally, when malicious Shamir secret sharing MPC protocol is used, the Hybrid Model reaches an accuracy of 96% as opposed to 89%, and takes approximately half an hour to run as opposed to over 5 hours.

This paper is organized as follows: Related Works outlines existing security algorithms, the Hybrid Privacy for Federated Learning section describes in more detail each of the algorithms, the Hybrid Model section outlines the specifics of our Hybrid Model as well as a short proof on why it is secure, the Evaluation presents and discusses the results from our model, and the Conclusion summarizes the significance of our findings.

Related Works

Federated Learning (Kairouz et al. 2019) is a machine learning scheme that allows a number of parties to collaboratively train a model with their data kept in local storage. It mitigates the data privacy concern by not exchanging the raw data over the network. It has many applications, especially in scenarios where data privacy is critical, such as medical records (Ju et al. 2020). However, even though the algorithm does not send training data, some aggregated information about the training data could leak by the communication data (e.g. the gradients) (McMahan et al. 2017; Bonawitz et al. 2017).

To protect the data privacy further, a number of approaches are explored in recent literature. Encrypting the data and operating directly on ciphertexts (Popa et al. 2011) could provide computationally secure privacy because without a decryption key, the ciphertexts look similar to random bits. A number of schemes are proposed to train a model over encrypted training data (Gilad-Bachrach et al. 2016; Chou et al. 2018; Hesamifard, Takabi, and Ghasemi 2017; Bourse et al. 2018; Zhang et al. 2020). But the resulting overhead from techniques like homomorphic encryption (Gentry and Boneh 2009) prohibits heavy tasks like training a large scale machine learning model from being used in practice. Another paradigm that can mitigate the privacy issue is *differential privacy* (DP) (Dwork 2008). Differentially private schemes proceed by adding controlled noise to the input data such that distinguishing two datasets that differ minimally is statistically difficult. DP has already been used in improving machine learning privacy (Mohassel and Zhang 2017; McMahan et al. 2017; Gong et al. 2020). However, differential privacy by its definition sacrifices the accuracy of the results, namely, the model from a differentially private training algorithm would always be away from optimality. Moreover, the more privacy the scheme guarantees, the less accuracy the users enjoy. Secure Multi-Party computation (MPC) (Goldreich 1998; Beaver and Goldwasser 1989; Damgård et al. 2009; Canetti et al. 1996) is a family of cryptographic protocols that allow a number of parties to collaboratively compute a function $y = f(x_1, x_2, \ldots, x_n)$ where x_i is a private input from the i-th party, such that except any information can be inferred from y (which is inevitable as y will be made public), no other information about x_i is revealed. The privacy guarantee from MPC is strong. However, similar to homomorphic encryption, MPC schemes usually incur a large computational overhead compared to computing the target function f natively. There have been efforts in applying MPC in machine learning (Juvekar, Vaikuntanathan, and Chandrakasan 2018; Riazi et al. 2018; Rouhani, Riazi, and Koushanfar 2018).

Hybrid Privacy for Federated Learning

Now that we have outlined the intuition behind some other privacy schemes, let us formally introduce federated learning, LDP, and MPC. The following section will also qualify what information is revealed during such schemes. This will be important later on when qualifying the level of privacy guaranteed by the hybrid scheme.

Definition of Federated Learning

Federated Learning is formally defined as follows:

Definition 1. *Federated Learning Define N data owners $\mathcal{F}_1, ...\mathcal{F}_N$, all of whom wish to train a machine learning model by consolidating their respective data $\mathcal{D}_1, ...\mathcal{D}_N$. A conventional method is to put all data together and use $\mathcal{D} = \mathcal{D}_1 \cup ... \cup \mathcal{D}_N$ to train a model $\mathcal{M}_{SUM}$. A federated learning system is a learning process in which the data owners collaboratively train a model $\mathcal{M}_{FED}$, in which process any data owner $\mathcal{F}_i$ does not expose its data $\mathcal{D}_i$ directly to others. Often, a central server that is computationally powerful is involved (Yang et al. 2019).*

More generally, we formalize the federated learning process as a multi-round query-answer protocol, where the central server, which does the computation in each round r, sends a query $q_i^{(r)}$ to the data owner $\mathcal{F}_i$ iteratively, and upon receiving the query $\mathcal{F}_i$, computes answers $a_i^{(r)}$ based on their local data $\mathcal{D}_i$. With these answers, the central server updates the weights of the model. Finally, the trained model is revealed.

In the process of federated learning, the local data is not directly exposed, but since an aggregated result related to the data is transmitted to the central server, we argue that important statistical properties about the local data are still leaked. Such leakage in some situations could harm privacy. For example, multiple healthcare institutes with private patient information participate in a collaborative machine learning task. An average of the geographical information could directly reveal the community identity of the cohort. In another scenario, the average income of citizens within a certain area can be used to accurately estimate the salaries of people in that area, thus revealing personal financial information.

Definition of Local Differential Privacy

Local differential privacy is an algorithm that protects individual inputs by locally adding random noise to the

individual user's inputs before performing computations on the data. This protects each user's privacy because the true values are masked by a margin of uncertainty due to the noise, which gives users plausible deniability as to what their actual inputs are. However, even with the added noise, one can still analyze general trends and perform computations on the data, albeit less accurately due to the noise. This provides a powerful and simple privacy mechanism.

Local Differential Privacy is formally defined as follows:

Definition 2 (LDP Security)**.** *We say that an algorithm π satisfies ε-Local Differential Privacy where $\varepsilon > 0$ if and only if for any input v and v' that are adjacent*

$$\forall y \in Range(\pi) : \frac{\Pr[\pi(v) = y]}{\Pr[\pi(v') = y]} \leq e^{\varepsilon}$$

where Range(π) denotes every possible output of the algorithm π (Bebensee 2019).

In terms of a query-answer model, one way to make the protocol differentially private is to add noise to the answers generated by data owners:

$$\widetilde{a_i^{(r)}} = a_i^{(r)} + e,$$

where e denotes random noise. Since each party submits $\widetilde{a_i^{(r)}}$ to the protocol, the protocol leaks the users' noisy data when active. Furthermore, by publicizing the final result of the protocol, the aggregate information of the users' noisy data is also leaked. However, this information is not enough to leak individual user's data; therefore, LDP is acceptably secure in ensuring privacy to the users.

Definition of Multi-Party Computation

An MPC protocol is an algorithm where parties jointly compute a function given private user inputs. What makes MPC so powerful is that no information that could be used to reveal the users' inputs is leaked, aside from information that the final function value reveals. Specifically, the intermediary values leaked while jointly computing the function value cannot be used to infer the individual inputs from the parties.

In terms of our query-answer model, we have N parties $\mathcal{F}_1, \dots \mathcal{F}_N$, each with data $\mathcal{D}_1, \dots \mathcal{D}_N$. They seek to compute $f(\mathcal{D}_1, \dots \mathcal{D}_N)$. The parties follow a specific protocol where $\mathcal{F}_i$ sends a query $q_{i,j,k}^{(r)}$ to party $\mathcal{F}_j$, where k represents the kth interaction between the two parties and r represents round r. Party $\mathcal{F}_j$ sends party $\mathcal{F}_i$ answer $a_{i,j,k}^{(r)}$, where party $\mathcal{F}_j$ may now perform computations on $a_{i,j,k}^{(r)}$ and perform additional queries/receive additional answers. The specific queries and answers depend on the MPC protocol, however through these queries the parties compute $f(\mathcal{D}_1, \dots \mathcal{D}_N)$ without revealing $\mathcal{D}_1, \dots \mathcal{D}_N$.

Now, let us define the terms used to define MPC security. Denote the set of corrupted parties C, who may collaborate and share additional values private to themselves besides those publicly available. Denote the fault tolerance of an MPC f. The fault tolerance is the maximum number of parties that may be corrupt in an MPC scheme while still preserving user privacy. Let x_i and y_i be inputs and outputs from party $\mathcal{F}_j$, and $view_j$ is $\mathcal{F}_j$'s view of the program during the simulation.

Definition 3 (MPC Security)**.** *An MPC protocol is called secure if there exists an efficient simulator S such that the simulated message transcript $S(\{x_j, y_j\}_{\mathcal{F}_j \in C}) := \{\overline{view_j}\}_{\mathcal{F}_j \in C}$ and the real leaked values $\{view_j\}_{\mathcal{F}_j \in C}$ have computationally indistinguishable distributions: for any transcript $\bar{t} \in \{\overline{view_j}\}_{\mathcal{F}_j \in C}$ and $t \in \{view_j\}_{\mathcal{F}_j \in C}$, for any p.p.t. adversary $\mathcal{A}$, the distribution of t and the distribution of $\bar{t}$ are computationally indistinguishable.*

The MPC protocol guarantees that a minimal amount of information is leaked regarding the users' inputs; however, it is still important to note this leakage. The final function output of the MPC may be used to infer information regarding the users. However, this leakage presents minimal risk for a party's privacy since the output function contains only aggregate information regarding the users' inputs. Since this protocol will be paired with LDP later on, it is also important to consider LDP's leakage. LDP leaks the parties' noisy data as well as the aggregate information of the users' data. Thus, when the MPC phase runs after the LDP phase, the difference in the aggregate information publicized after the LDP phase to that after the MPC and LDP phases can be used to compute the aggregate information of just the MPC phase. However, as described earlier, the risk posed by the leakage of this information is minimal. Furthermore, the leakage of information from the LDP phase can be mitigated by not revealing the aggregate information after the LDP phase.

Now that we have defined Federated Learning, LDP, and MPC, let us describe the hybrid model.

The Hybrid Model

In this subsection, we introduce our Hybrid Model. We first present a pseudocode algorithm, and then formally define the security for such a protocol, followed by a proof of security.

Pseudocode

The pseudocode of the hybrid privacy scheme is shown in Algorithm 1. We define the functions and variables used as follows:

$addNoise(x)$: adds sufficient noise to input value x such that ε-Local Differential Privacy is satisfied.

train$(x_1, x_2, \dots x_N, weights)$: performs a training algorithm such that the variable $weights$ is modified based on input values $x_1, x_2, \dots x_N$. The output of the function are the weights to a specific party (the weights are not necessarily publicized).

$cWeights$: Weights that are only known by the central server.

Algorithm 1: Hybrid privacy algorithm

Input: N parties each with answer $x_i = a_i^{(r)}$, where $a_i^{(r)}$ is computed from $\mathcal{D}_i$ for

$$i \in \{1, 2, \dots N\},$$

array of weights all equal to 0;
Result: Weights will be trained and optimized securely
Initialization:
`// Each Party computes` x_i' `locally`
for *i=1 to N* **do**
  $x_i' = addNoise(x_i)$;
end
cWeights = train$_{LDP}(x_1', x_2', \dots x_N', weights)$
`// LDP Portion`
`// Switch to MPC when model has reached peak performance during LDP`
weights = train$_{MPC}(x_1, x_2, \dots x_N, cWeights)$
`// MPC Portion`

train$_{LDP}(x_1, x_2, \dots x_N, weights)$: trains the weights in accordance with the LDP protocol. It is assumed that $x_1, x_2, \dots x_N$ are already noisy, so essentially train$_{LDP}() = train()$. The trained weights are only revealed to the centralized server.

train$_{MPC}(x_1, x_2, \dots x_N, cWeights)$: trains the weights using an MPC protocol, such that all operations present in $train()$ are securely performed to satisfy MPC privacy. Trained weights are made public after training.

Round r_s denotes the switch point. r_s is determined before the protocol commences such that rounds $1, 2, \dots, r_s - 1$ optimize the model to its peak performance, and then the switch point occurs. Rounds $1, 2, \dots, r_s - 1$ are the differential privacy phase. Rounds $r_s + 1, \dots, r_n$ are the secure multiparty computation phase. The query-answer protocol proceeds in two phases.

The Differential Privacy Phase: Users will first add noise to their data $\mathcal{D}_i$. The central server will start with an array of weights all equal to 0. For each round $r \in \{1, \dots r_s - 1\}$, the central server sends query $q_i^{(r)}$ to $\mathcal{F}_i$ for $i \in \{1, \dots N\}$. $\mathcal{F}_i$ sends $\widetilde{a_i^{(r)}}$ to the central server, where $\widetilde{a_i^{(r)}}$ is defined under the definition of Local Differential Privacy. The central server then trains the weights using the train$_{LDP}()$ algorithm.

The Switch Point On round r_s, the LDP training will stop. The weights that the central server has ($cWeights$) are not revealed and will instead be submitted directly to the MPC protocol as if the central server is a party $\mathcal{F}_{N+1}$ with data $\mathcal{D}_{N+1}$, where $\mathcal{D}_{N+1} = cWeights$.

The Secure Multiparty Computation Phase: For round $r \in \{r_s + 1, \dots r_n\}$, the parties seek to compute train$_{MPC}(x_1, x_2, \dots x_N, cWeights)$, where $x_1 = \mathcal{D}_1, x_2 = \mathcal{D}_2, \dots x_N = \mathcal{D}_N, weights = \mathcal{D}_{N+1} = cWeights$. train$_{MPC}()$ performs the exact same operations as train$_{LDP}()$, except that such operations are performed in a manner that is compliant with the security definition for MPC. For more information regarding the query-answer model for MPC and its security, please refer to the subsection that defines MPC. $cWeights$ will be made public after the MPC phase.

Security of the Hybrid Model

Now that the protocol of the Hybrid Model is established, the next section explains privacy is preserved during the execution of the protocol. This is done by first defining the security of a Hybrid Protocol and then proving the model satisfies such security. Hybrid Security is informally defined through the security of the MPC-phase of the protocol and states that the Hybrid Model is secure as long as the MPC-protocol remains secure despite LDP leakage. More formally:

Definition 4 (Hybrid Security)**.** *We say a Hybrid Model is secure if there exists an efficient simulator S such that, given the answers in the DP phase, the simulated message transcript $S(\{a_i^{(r)}\}|_{r=1}^{r_s}, \{x_j, y_j\}_{\mathcal{F}_j \in C}) := \{\overline{view_j}\}_{\mathcal{F}_j \in C}$ and the real leaked values $\{view_j\}_{\mathcal{F}_j \in C}$ are close: for any transcript $\bar{t} \in \{\overline{view_j}\}_{\mathcal{F}_j \in C}$ and $t \in \{view_j\}_{\mathcal{F}_j \in C}$, for any p.p.t. adversary $\mathcal{A}$*

$$\Pr[\mathcal{A}(t) = 1] \leq e^{\varepsilon} \Pr[\mathcal{A}(\bar{t}) = 1]$$

for a constant ε. We say such a scheme satisfies ε Hybrid Model privacy.

We now seek to prove the security of our Hybrid Model. This will be accomplished by first constructing a simulator for the Hybrid Model. We will then show that if there exists an adversary that can break the Hybrid Model, then there must also exist another adversary that can break the LDP security using the simulator and the first adversary. This will break an earlier assumption that LDP is secure, which will form a contradiction. Thus, we will create a sketch of a proof by contradiction.

Theorem 1. *Assume the MPC protocol is secure, and the LDP is ε differentially private, then the hybrid protocol is secure.*

We sketch the proof of the above theorem using a proof by contradiction.

Proof: Assume the MPC protocol is secure, and denote its simulator S_{MPC}. Let us define a simulator S as follows:

1. The simulator virtually runs N parties and the central server, exactly following the hybrid protocol.
2. $train_{LDP}()$ is performed using $\mathcal{D}_i \in C$ where C is the set of corrupted parties, and uses the leaked values $\{a_i^{(r)}\}|_{r=1}^{r_s}$ as the rest of the inputs. Although the simulator does not know the private inputs $\mathcal{D}_i$ of the honest parties, the simulator answers the servers queries such that whenever the central server queries $q_i^{(r)}$ to $\mathcal{F}_i$ in round r, the simulator will produce answer $a_i^{(r)}$ using leaked values given in the input.

3. After the LDP protocol finishes, everything needed for the MPC protocol is available except for the private inputs $\mathcal{D}_i$ where $\mathcal{F}_i \notin C$.
4. The simulator S_{MPC} commences using $\{x_j, y_j\}_{\mathcal{F}_j \in C}$.
5. Return the transcript $\bar{t}$ that is produced by the simulator S_{MPC}.

We proceed through a proof by contradiction. Assume there exists an adversary $\mathcal{A}_H$ that may compromise the security of the Hybrid Protocol. We show that with $\mathcal{A}_H$, we can construct an adversary $\mathcal{A}_{LDP}$ to break the security of the LDP scheme. $\mathcal{A}_{LDP}$ performs the following steps:

1. Given an LDP instance, $\mathcal{A}_{LDP}$ seeks to compromise its security. $\mathcal{A}_{LDP}$ participates in the scheme per usual with other honest parties.
2. Using S, $\mathcal{A}_{LDP}$ simulates a transcript $\bar{t}$ for the MPC portion of the hybrid protocol.
3. $\mathcal{A}_{LDP}$ then sends the combined LDP and MPC transcript to $\mathcal{A}_H$.
4. By assumption, $\mathcal{A}_H$ breaks the security of the Hybrid Protocol.
5. $\mathcal{A}_H$ gives the output to $\mathcal{A}_{LDP}$. $\mathcal{A}_{LDP}$ may now break the LDP scheme. Since the adversary is a surjective mapping, the fact that the distribution of the output from the adversary challenges the ε-local differential privacy indicates the input of the adversary also challenges the ε-local differential privacy.
6. This contradicts our earlier assumption of LDP being (ε) differentially private. Thus, our Hybrid Protocol must satisfy ε Hybrid Privacy.

The security of the hybrid model is slightly weaker than LDP or MPC alone if the parameters are the same, due to the additional parameters. However, the Hybrid Model provides a better accuracy (compared to LDP) and better runtime (compared to MPC), so such a trade-off is beneficial in some cases. The parameter r_s plays a role in this trade-off. For example, if $r_s = 0$, then the protocol degrades to a pure MPC algorithm. If $r_s = r_n$, then the protocol degrades to a pure LDP algorithm.

Instantiation

Our project applies LDP and MPC on a Support Vector Machine (SVM) to train on the MNIST dataset, which will output a set of weights that accurately predict what digit an image is while maintaining the privacy of the users who submit images. The model consists of 3 main parts: LDP, MPC, and the point where the algorithm switches from LDP to MPC. As described earlier, using just LDP or MPC to compute a function (in our case to train an SVM) has both costs and benefits, and we wish to reap the benefits of both approaches. This is where the switch point comes in. The idea is that the SVM is trained through LDP for some time using MNIST images with added noise. After the loss reported by the SVM reaches a plateau, the weights transfer to an MPC framework where they are further modified based on unaltered images of the MNIST data set. This way, the SVM can be trained quickly with LDP for most of the way until it approaches the global minimum of the loss function. After the loss function plateaus, the SVM can transfer to an MPC framework, which will allow the SVM to get much closer to the global minimum because it is now training on the actual pixel values of the images. Since the SVM has been trained most of the way, it will spend less time on the much slower MPC calculations. As a result, we get a fast, secure, and accurate algorithm.

Now, let us describe the specifics of our SVM training program. First, the weights are initialized to 0. Although the number of parties is flexible, we have tested our program on five parties. Only four of the parties will submit image data to train the model, and the final party serves as the centralized server to train the model. Our protocol implements LDP via a random response scheme that adds noise to each image submitted by the user (these images are all part of the MNIST dataset). It works in the following way: consider a probability value of l. Each pixel within an image has a $\frac{1}{2}l$ chance of assigning 0 to the pixel, a $\frac{1}{2}l$ chance of assigning 1 to the pixel value, and a $1 - l$ chance of assigning the true value to the pixel. Then, the images are given to the SVM to train on them via gradient descent. This grants the parties privacy as it allows each image to have plausible deniability to its actual value; thus, anyone who intercepts the altered image cannot confirm the true digit of the image. However, the totality of all the MNIST images will overcome this noise while the SVM trains on it, and as a result, a semi-accurate model can be achieved. The calculations section describes specifically how our program satisfies ε-Local Differential Privacy based on the epsilon value that is passed into our program, as it will then compute the necessary value of l needed to give each image enough plausible deniability.

After the LDP phase, the weights transfer to the second phase of the protocol, where MPC is used to securely perform the operations and functions of an SVM. Specifically, we took an MP-SPDZ framework that has many MPC algorithms (Keller 2020). In particular, we used a MASCOT protocol which does efficient arithmetic operations due to it using arithmetic circuits (Keller, Orsini, and Scholl 2016). It also provides malicious, dishonest majority computation, and therefore it has a fault tolerance of $f = N - 1$, where N is the number of parties. Additionally, the MASCOT protocol uses oblivious transfer in order to achieve fast multiplication. This is useful as gradient and loss function computations use lots of multiplication; therefore, our MPC portion is as fast as possible. We have also used a semi-honest Shamir secret sharing protocol ($f = \lfloor \frac{N}{2} \rfloor$) as well as a malicious Shamir secret sharing protocol ($f = N - 1$) from the same framework and achieved similar results to that of MASCOT. Using one of these protocols, four users each submit an image, and one user (the same user who served as a central server for the LDP phase) submits the weights obtained from the LDP portion to the MPC. The weights are then further optimized using the MPC. After a new set of weights have been created, the centralized server submits the resulting weights to the MPC again, and each of the four users also submits a new image. This process repeats five times, such that the SVM trains on 20 images during the

MPC phase. Note that the weights may be revealed after all 20 images have been used to train the SVM; the intermediary weights do not have to be revealed.

As a result, we have a secure hybrid protocol, and our research demonstrates the ability to merge LDP and MPC. When the two approaches are combined, the resulting algorithm is more accurate than LDP alone and faster than MPC alone.

Calculations

Using the definition of LDP, described in the Definition of Differential Privacy Section, we can solve for epsilon to implement it in our local differential privacy scheme (Bebensee 2019). We implement epsilon-pixel-level DP, which guarantees security for each pixel of the input images. Though this is a weaker notion of privacy, it is necessary to obtain usable results without using an extremely large value of epsilon. A low epsilon value means the images would be extremely modified, to the point where the SVM would not be able to accurately classify the images, making the results unusable.

Assuming we discretize the input into 0 and 1 by rounding. To get

$$\Pr[\pi(v) = y] \leq \Pr[\pi(v') = y] \cdot e^{\epsilon}$$

Consider a random response scheme per pixel $b_{i,j}$ (the bit on the ith row and jth column). The party outputs

$$b'_{i,j} = \begin{cases} 0 & \frac{1}{2}l \\ 1 & \frac{1}{2}l \\ b_{i,j} & 1-l \end{cases}$$

where l is the probability of the pixel value being reassigned. Then for a single pixel, we have

$$\forall_{i,j},\ \Pr[b'_{i,j} = 1 | b_{i,j} = 1] = \frac{1}{2}l + 1 - l = 1 - \frac{1}{2}l$$

$$\Pr[b'_{i,j} = 1 | b_{i,j} = 0] = \frac{1}{2}l$$

so for a single pixel,

$$1 - \frac{1}{2}l \leq \frac{1}{2}l \cdot e^{\epsilon}$$
$$\implies \epsilon \geq \ln \frac{2-l}{l}$$

Now that ϵ is expressed in terms of l, we can use this to find l in terms of ϵ to make this parameter adjustable.

$$\epsilon \geq \ln \frac{2-l}{l}$$
$$l \geq \frac{2}{e^{\epsilon}+1}$$

The program takes in ϵ as a parameter when running and calculates l to be used in the random response scheme. The experiments run in this paper use an epsilon value of $\ln 3$, which translates to an l value of $l = \frac{1}{2}$.

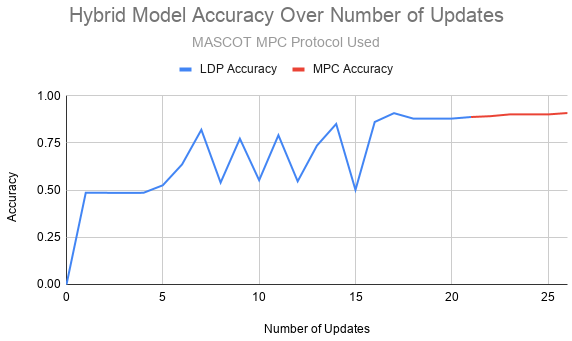

Figure 1: Hybrid Model Accuracy Over Number of Updates. MASCOT MPC Protocol was Used

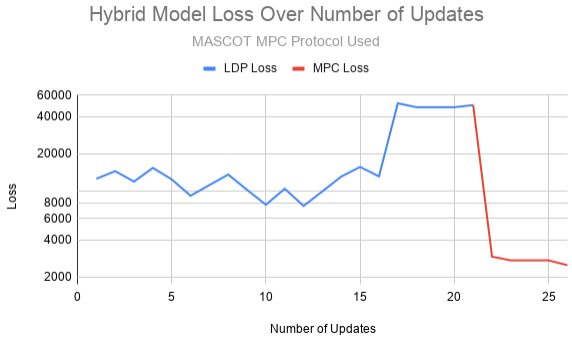

Figure 2: Hybrid Model Loss Over Number of Updates. MASCOT MPC Protocol was Used

Evaluation

After running our hybrid model using the MASCOT MPC protocol on the MNIST dataset[1], we then graphed the accuracy and loss[2] of the LDP and MPC portions of the algorithm respectively over the number of program updates that have elapsed[3].

The results are presented in Figure 1 and Figure 2. The x-axis displays the number of updates. In the LDP phase, each update consists of one input image. In the MPC phase, each update consists of four input images. In the first figure, the y-axis graphs the accuracy when the model is evaluated on the test dataset, whereas the second figure graphs the loss of the test dataset. Even though there are many more updates that occurred during LDP training than MPC training, the LDP portion of the program trained very quickly, taking 1.063 seconds to reach a final accuracy of 89%. However, after some initial volatility in the accuracy metric for the LDP, the accuracy increase for the LDP stagnated. This is

[1]We modified the MNIST dataset such that the digit 0 appears approximately 50% of the time

[2]Loss Formula: $J(w) = \frac{1}{2}||w||^2 + C[\frac{1}{n}\sum_{i=0}^{n} max(0, 1 - y_i * (w \cdot x_i + b))]$, where w is the weights, x_i and y_i are the input images and their labels, n is the number of images, C is a constant, and $b = 0$

[3]Data retrieved from training on computer with 8th gen core i7, 16GB RAM, no GPU

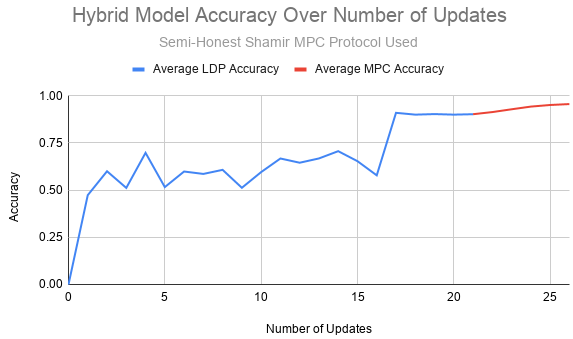

Figure 3: Hybrid Model Accuracy Over Number of Updates. Semi-Honest Shamir MPC Protocol was Used

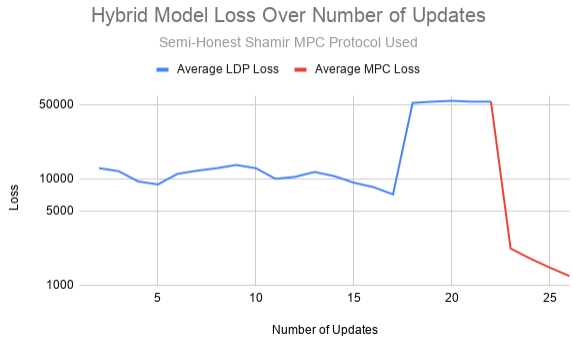

Figure 4: Hybrid Model Loss Over Number of Updates. Semi-Honest Shamir MPC Protocol was Used

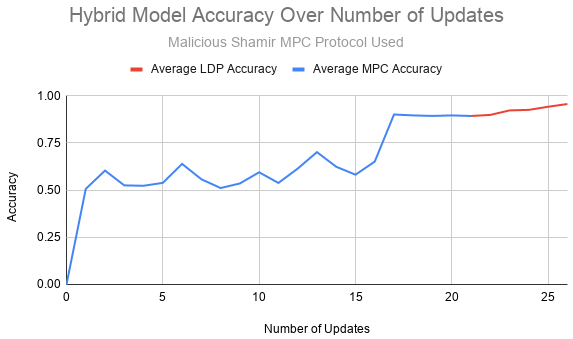

Figure 5: Hybrid Model Accuracy Over Number of Updates. Malicious Shamir MPC Protocol was Used

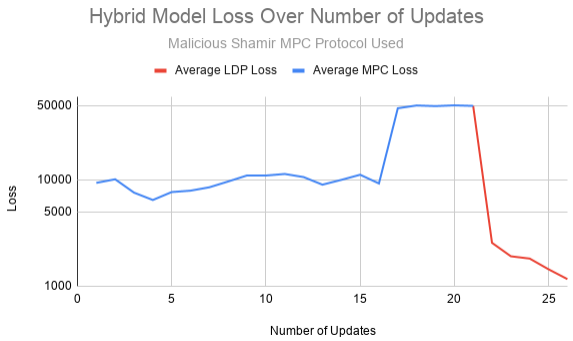

Figure 6: Hybrid Model Loss Over Number of Updates. Malicious Shamir MPC was Used

seen in the small blue plateau on the graph shortly before the 21st update has elapsed (Figure 1). After the switch-point occurs, the MPC portion of the program starts to train the model, and the accuracy is optimized to reach approximately 91%. However, this process is excruciatingly slow, as the MPC portion ran for 19 hours. This shows that although the LDP portion quickly optimized the SVM model, it quickly stagnated in its progression of optimizing the model. The MPC portion that ensued was very slow, but it fine-tuned the weights even further.

Towards the middle of the LDP portion of the graph, one may observe volatility in the accuracy of the model. We hypothesize that this is caused by the random noise that is added to the images during the LDP phase, which at first modifies the model in a somewhat-random fashion. This would lead to rapid changing of the model's accuracy due to the model being changed quickly, and we believe that this leads to the volatility of the graph. However, as more images are inputted to the model, the model is already partially trained and therefore images with random noise do not modify the model by a significant margin.

Similar results are observed when one considers the loss of the model. The LDP portion of the training had a much higher loss when compared to the MPC portion of the program. Despite this, the LDP partially optimized the loss, and the MPC further refined the loss. On the LDP portion of the graph, one can see a spike in the loss values incurred at the at around the 17th update. We believe that this is a result of the noise having a greater effect on the loss function due to the weights being closer to the optimal when compared to the start of the program and that this is not indicative of faulty weights (as shown by the LDP accuracy reaching approximately 89% at the same time as the loss spike). Clearly, the LDP quickly produced a good yet somewhat inaccurate model, and the MPC refined the model further.

Semi-honest and malicious Shamir MPC protocols produce very similar results when compared to the MASCOT protocol, as is shown in Figure 3, Figure 4, Figure 5, and Figure 6. Each protocol was tested 3 times, and the results for each protocol were averaged and graphed. It should be noted that the number of updates performed during the LDP phase could differ for each trial, but since each trial produced at least 21 updates, and since all the trials were very close to 21 updates (no more than 2-3 updates), we only graphed 21 of the LDP updates (as subsequent updates did not improve the model by a large margin, as evidenced by the plateau of the LDP phase in Figure 3 and Figure 5). The LDP phase for the Hybrid Protocol that used semi-honest Shamir secret sharing had an LDP phase that lasted 1.066 seconds on average and achieved a final accuracy of 90%. The malicous Shamir had an LDP phase that lasted 1.119 seconds on average and achieved a final accuracy of 89%. The similarity between these results and the results of the Hybrid Model with MASCOT is expected, as the LDP phase of all three Hybrid

SVM Training	Baseline	Hybrid Model
LDP Portion		
Test Cost	55,718.1	49,839.3
Test Accuracy	0.8960	0.8878
Runtime	1.734 seconds	1.063 seconds
MPC Portion		
Cost	1007.91	2,449.83
Accuracy	0.9721	0.9089
Runtime	190.077 hours	19.049 hours

Table 1: Table of Hybrid Results (with MASCOT) vs Baseline

Models are the same. However, the MPC phase of the semi-honest Shamir protocol took 19.356 minutes to run, whereas the malicious Shamir protocol took 33.055 minutes to run. They achieved final accuracies of approximately 96%. Both protocols were faster than the MASCOT protocol, however all 3 MPC protocols were much slower when compared to the LDP phase. Furthermore, we estimate that if only MPC was used, the semi-honest Shamir protocol would take 3.226 hours to run, and the malicious Shamir protocol would take 5.509 hours. Thus, we can see through these trials that the LDP phase quickly improved the model, whereas the MPC phase fine-tunes the model in order to reach a higher accuracy, irrespective of the MPC protocol used.

Table 1 shows the data of the hybrid model that utilized the MASCOT protocol, and further exemplifies the benefits of the hybrid protocol. The hybrid model achieved a higher accuracy and a lower loss when compared to a pure LDP scheme. Furthermore, the hybrid model took much less time to run than a pure MPC scheme. The table shows that should the user desire a higher final accuracy for the hybrid model, the MPC phase of the hybrid protocol may be prolonged in order to achieve an accuracy and loss that approaches that of a pure MPC scheme. Thus, we can conclude that the LDP portion of our model quickly optimizes the weights, and the MPC portion of our model further calibrates the weights.

Conclusion

This project aimed to create a privacy scheme that combines existing privacy algorithms: Local Differential Privacy and Multi-Party Computation. We were able to accomplish this using our Hybrid Model, which harnesses the benefits of both algorithms and avoids the drawbacks, all while maintaining privacy. Looking at Table 1, the Hybrid Model that utilized MASCOT achieves an accuracy of 90.9% compared to the LDP baseline accuracy of 89.6%, and the model takes roughly 19 hours to run compared to the projected MPC baseline runtime of 190 hours, showing that our Hybrid Model is better than each of the individual algorithms. We achieved very similar results with hybrid models involving different MPC protocols: with the semi-honest Shamir MPC protocol, the Hybrid Model reaches an accuracy of 96% as opposed to the 89% LDP baseline accuracy, and took about 19 minutes to run as opposed to 3 hours. When the malicious Shamir MPC protocol is used, the Hybrid Model achieves an accuracy of 96%, compared to 89% LDP baseline accuracy, and takes approximately 30 minutes to run as opposed to over 5 hours. These results demonstrate that our Hybrid Model can be applied in the real world to securely perform complex computations, such as machine learning algorithms, on data. For example, many artificial intelligence models have been developed for disease prediction; however, the PHI they are run on must remain secure. By utilizing our Hybrid Scheme, these models can be run securely and efficiently.

Limitations

This study has some limitations due to the computational power of the machine used to run the model. The MPC phase of the model only trains using twenty images. Though these twenty images do significantly increase the accuracy and decrease the cost, these benefits could be amplified by training on more images. The computational power also limited what Machine Learning algorithms we could use. Additionally, due to the nature of Machine Learning, this Hybrid Model will never be able to achieve completely perfect accuracy or cost without over-fitting; however, a higher accuracy and lower cost could possibly be achieved by using a more advanced algorithm such as Convolutional Neural Networks.

Further Research

One area for future research is to implement the model on other machine learning problems. For example, the CIFAR dataset may also be run with such a model.

Another area for future research is looking into other MPC algorithms. The MASCOT, the semi-honest Shamir, and the malicious Shamir protocols used in this demonstration are very robust, but there exist other MPC protocols such as the ones based on Yao's Garbled circuits among others.

Finally, another area for development is to implement the Hybrid Model on more advanced architectures. The SVM may have difficulties with modeling more complex datasets, and thus the use of more complex architectures such as convolutional neural networks may be more practical in the real world.

Acknowledgements

We would like to thank our mentor Yu Xia for his continued support and guidance for this project, as well as the MIT PRIMES program for giving us this research opportunity.

References

Beaver, D.; and Goldwasser, S. 1989. Multiparty computation with faulty majority. In *Conference on the Theory and Application of Cryptology*, 589–590. Springer.

Bebensee, B. 2019. Local Differential Privacy: a tutorial.

Bonawitz, K.; Ivanov, V.; Kreuter, B.; Marcedone, A.; McMahan, H. B.; Patel, S.; Ramage, D.; Segal, A.; and Seth, K. 2017. Practical secure aggregation for privacy-preserving machine learning. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, 1175–1191.

Bourse, F.; Minelli, M.; Minihold, M.; and Paillier, P. 2018. Fast homomorphic evaluation of deep discretized neural networks. In *Annual International Cryptology Conference*, 483–512. Springer.

Canetti, R.; Feige, U.; Goldreich, O.; and Naor, M. 1996. Adaptively secure multi-party computation. In *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, 639–648.

Chou, E.; Beal, J.; Levy, D.; Yeung, S.; Haque, A.; and Fei-Fei, L. 2018. Faster cryptonets: Leveraging sparsity for real-world encrypted inference. *arXiv preprint arXiv:1811.09953* .

Damgård, I.; Geisler, M.; Krøigaard, M.; and Nielsen, J. B. 2009. Asynchronous multiparty computation: Theory and implementation. In *International workshop on public key cryptography*, 160–179. Springer.

Dwork, C. 2008. Differential privacy: A survey of results. In *International conference on theory and applications of models of computation*, 1–19. Springer.

Gentry, C.; and Boneh, D. 2009. *A fully homomorphic encryption scheme*, volume 20. Stanford university Stanford.

Gilad-Bachrach, R.; Dowlin, N.; Laine, K.; Lauter, K.; Naehrig, M.; and Wernsing, J. 2016. Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy. In *International Conference on Machine Learning*, 201–210.

Goldreich, O. 1998. Secure multi-party computation. *Manuscript Preliminary version* .

Gong, M.; Xie, Y.; Pan, K.; Feng, K.; and Qin, A. K. 2020. A Survey on Differentially Private Machine Learning. *IEEE Comput. Intell. Mag.* 15(2): 49–64.

Hesamifard, E.; Takabi, H.; and Ghasemi, M. 2017. Cryptodl: Deep neural networks over encrypted data. *arXiv preprint arXiv:1711.05189* .

Ju, C.; Zhao, R.; Sun, J.; Wei, X.; Zhao, B.; Liu, Y.; Li, H.; Chen, T.; Zhang, X.; Gao, D.; et al. 2020. Privacy-preserving technology to help millions of people: Federated prediction model for stroke prevention. *arXiv preprint arXiv:2006.10517* .

Juvekar, C.; Vaikuntanathan, V.; and Chandrakasan, A. 2018. {GAZELLE}: A low latency framework for secure neural network inference. In *27th {USENIX} Security Symposium ({USENIX} Security 18)*, 1651–1669.

Kairouz, P.; McMahan, H. B.; Avent, B.; Bellet, A.; Bennis, M.; Bhagoji, A. N.; Bonawitz, K.; Charles, Z.; Cormode, G.; Cummings, R.; D'Oliveira, R. G. L.; Rouayheb, S. E.; Evans, D.; Gardner, J.; Garrett, Z.; Gascón, A.; Ghazi, B.; Gibbons, P. B.; Gruteser, M.; Harchaoui, Z.; He, C.; He, L.; Huo, Z.; Hutchinson, B.; Hsu, J.; Jaggi, M.; Javidi, T.; Joshi, G.; Khodak, M.; Konečný, J.; Korolova, A.; Koushanfar, F.; Koyejo, S.; Lepoint, T.; Liu, Y.; Mittal, P.; Mohri, M.; Nock, R.; Özgür, A.; Pagh, R.; Raykova, M.; Qi, H.; Ramage, D.; Raskar, R.; Song, D.; Song, W.; Stich, S. U.; Sun, Z.; Suresh, A. T.; Tramèr, F.; Vepakomma, P.; Wang, J.; Xiong, L.; Xu, Z.; Yang, Q.; Yu, F. X.; Yu, H.; and Zhao, S. 2019. Advances and Open Problems in Federated Learning.

Keller, M. 2020. MP-SPDZ: A Versatile Framework for Multi-Party Computation. *IACR Cryptol. ePrint Arch.* 2020: 521.

Keller, M.; Orsini, E.; and Scholl, P. 2016. MASCOT: faster malicious arithmetic secure computation with oblivious transfer. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, 830–842.

McMahan, H. B.; Ramage, D.; Talwar, K.; and Zhang, L. 2017. Learning differentially private recurrent language models. *arXiv preprint arXiv:1710.06963* .

Mohassel, P.; and Zhang, Y. 2017. Secureml: A system for scalable privacy-preserving machine learning. In *2017 IEEE Symposium on Security and Privacy (SP)*, 19–38. IEEE.

Popa, R. A.; Redfield, C. M.; Zeldovich, N.; and Balakrishnan, H. 2011. CryptDB: protecting confidentiality with encrypted query processing. In *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*, 85–100.

Riazi, M. S.; Weinert, C.; Tkachenko, O.; Songhori, E. M.; Schneider, T.; and Koushanfar, F. 2018. Chameleon: A hybrid secure computation framework for machine learning applications. In *Proceedings of the 2018 on Asia Conference on Computer and Communications Security*, 707–721.

Rouhani, B. D.; Riazi, M. S.; and Koushanfar, F. 2018. Deepsecure: Scalable provably-secure deep learning. In *Proceedings of the 55th Annual Design Automation Conference*, 1–6.

Yang, Q.; Liu, Y.; Chen, T.; and Tong, Y. 2019. Federated Machine Learning: Concept and Applications.

Zhang, C.; Li, S.; Xia, J.; Wang, W.; Yan, F.; and Liu, Y. 2020. BatchCrypt: Efficient Homomorphic Encryption for Cross-Silo Federated Learning. In *2020 {USENIX} Annual Technical Conference ({USENIX} {ATC} 20)*, 493–506.