2019 IEEE Symposium on Security and Privacy

Helen: Maliciously Secure Coopetitive Learning for Linear Models

Wenting Zheng, Raluca Ada Popa, Joseph E. Gonzalez, and Ion Stoica
UC Berkeley

***Abstract*—Many organizations wish to collaboratively train machine learning models on their combined datasets for a common benefit (e.g., better medical research, or fraud detection). However, they often cannot share their plaintext datasets due to privacy concerns and/or business competition. In this paper, we design and build Helen, a system that allows multiple parties to train a linear model without revealing their data, a setting we call *coopetitive learning*. Compared to prior secure training systems, Helen protects against a much stronger adversary who is *malicious* and can compromise $m-1$ out of m parties. Our evaluation shows that Helen can achieve up to five orders of magnitude of performance improvement when compared to training using an existing state-of-the-art secure multi-party computation framework.**

I. Introduction

Today, many organizations are interested in training machine learning models over their aggregate sensitive data. The parties also agree to release the model to every participant so that everyone can benefit from the training process. In many existing applications, collaboration is advantageous because training on more data tends to yield higher quality models [40]. Even more exciting is the potential of enabling new applications that are not possible to compute using a single party's data because they require training on complementary data from multiple parties (e.g., geographically diverse data). However, the challenge is that these organizations cannot share their sensitive data in plaintext due to privacy policies and regulations [3] or due to business competition [67]. We denote this setting using the term *coopetitive learning*[1], where the word "coopetition" [30] is a portmanteau of "cooperative" and "competitive". To illustrate coopetitive learning's potential impact as well as its challenges, we summarize two concrete use cases.

A banking use case. The first use case was shared with us by two large banks in North America. Many banks want to use machine learning to detect money laundering more effectively. Since criminals often hide their traces by moving assets across different financial institutions, an accurate model would require training on data from different banks. Even though such a model would benefit all participating banks, these banks cannot share their customers' data in plaintext because of privacy regulations and business competition.

A medical use case. The second use case was shared with us by a major healthcare provider who needs to distribute vaccines during the annual flu cycle. In order to launch an effective vaccination campaign (i.e., sending vans to vaccinate people in remote areas), this organization would like to identify areas that have high probabilities of flu outbreaks using machine learning. More specifically, this organization wants to train a linear model over data from seven geographically diverse medical organizations. Unfortunately, such training is impossible at this moment because the seven organizations cannot share their patient data with each other due to privacy regulations.

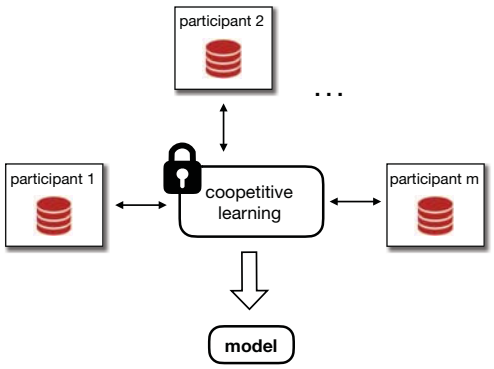


Fig. 1: The setting of coopetitive learning.

The general setup of coopetitive learning fits within the cryptographic framework of secure multi-party computation (MPC) [8, 37, 70]. Unfortunately, implementing training using generic MPC frameworks is extremely inefficient, so recent training systems [56, 41, 54, 34, 20, 35, 5] opt for tailored protocols instead. However, many of these systems rely on outsourcing to non-colluding servers, and all assume a passive attacker who never deviates from the protocol. In practice, these assumptions are often not realistic because they essentially require an organization to base the confidentiality of its data on the correct behavior of other organizations. In fact, the banks from the aforementioned use case informed us that they are not comfortable with trusting the behavior of their competitors when it comes to sensitive business data.

Hence, we need a much stronger security guarantee: each organization should *only trust itself.* This goal calls for maliciously secure MPC in the setting where $m-1$ out of m parties can fully misbehave.

In this paper, we design and build Helen, a platform for maliciously secure coopetitive learning. Helen supports a significant slice of machine learning and statistics problems: regularized linear models. This family of models includes ordinary least squares regression, ridge regression, and LASSO. Because these models are statistically robust and easily interpretable, they are widely used in cancer research [48], genomics [28, 59],

[1]We note that Google uses the term *federated learning* [67] for a different but related setting: a semi-trusted cloud trains a model over the data of millions of user devices, which are intermittently online.

DOI 10.1109/SP.2019.00045

financial risk analysis [63, 17], and are the foundation of basis pursuit techniques in signals processing.

The setup we envision for Helen is similar to the use cases above: a few organizations (usually less than 10) have large amounts of data (on the order of hundreds of thousands or millions of records) with a smaller number of features (on the order of tens or hundreds).

While it is possible to build such a system by implementing a standard training algorithm like Stochastic Gradient Descent (SGD) [61] using a generic maliciously secure MPC protocol, the result is very inefficient. To evaluate the practical performance difference, we implemented SGD using SPDZ, a maliciously secure MPC library [1]. For a configuration of 4 parties, and a real dataset of 100K data points per party and 90 features, such a baseline can take an estimated time of 3 months to train a linear regression model. Using a series of techniques explained in the next section, Helen can train the same model in less than 3 hours.

A. Overview of techniques

To solve such a challenging problem, Helen combines insights from cryptography, systems, and machine learning. This synergy enables an efficient and scalable solution under a strong threat model. One recurring theme in our techniques is that, while the overall training process needs to scale linearly with the total number of training samples, the more expensive *cryptographic* computation can be reformulated to be *independent* of the number of samples.

Our first insight is to leverage a classic but under-utilized technique in distributed convex optimization called Alternating Direction Method of Multipliers (ADMM) [15]. The standard algorithm for training models today is SGD, which optimizes an objective function by iterating over the input dataset. With SGD, the number of iterations scales at least linearly with the number of data samples. Therefore, naïvely implementing SGD using a generic MPC framework would require an expensive MPC synchronization protocol for every iteration. Even though ADMM is less popular for training on plaintext data, we show that it is much more efficient for cryptographic training than SGD. One advantage of ADMM is that it converges in very few iterations (e.g., a few tens) because each party repeatedly solves local optimization problems. Therefore, utilizing ADMM allows us to dramatically reduce the number of MPC synchronization operations. Moreover, ADMM is very efficient in the context of linear models because the local optimization problems can be solved by closed form solutions. These solutions are also easily expressible in cryptographic computation and are especially efficient because they operate on small summaries of the input data that only scale with the dimension of the dataset.

However, merely expressing ADMM in MPC does not solve an inherent scalability problem. As mentioned before, Helen addresses a strong threat model in which an attacker can deviate from the protocol. This malicious setting requires the protocol to ensure that the users' behavior is correct. To do so, the parties need to commit to their input datasets and prove that they are consistently using the same datasets throughout the computation. A naïve way of solving this problem is to have each party commit to the entire input dataset and calculate the summaries using MPC. This is problematic because 1) the cryptographic computation will scale linearly in the number of samples, and 2) calculating the summaries would also require Helen to calculate complex matrix inversions within MPC (similar to [57]). Instead, we make a second observation that each party can use singular value decomposition (SVD) [38] to decompose its input summaries into small matrices that scale only in the number of features. Each party commits to these decomposed matrices and proves their properties using matrix multiplication to avoid explicit matrix inversions.

Finally, one important aspect of ADMM is that it enables decentralized computation. Each optimization iteration consists of two phases: *local optimization* and *coordination*. The local optimization phase requires each party to solve a local sub-problem. The coordination phase requires all parties to synchronize their local results into a single set of global weights. Expressing both phases in MPC would encode local optimization into a computation that is done by every party, thus losing the decentralization aspect of the original protocol. Instead, we observe that the local operations are all linear matrix operations between the committed summaries and the global weights. Each party knows the encrypted global weights, as well as its own committed summaries in plaintext. Therefore, Helen uses partially homomorphic encryption to encrypt the global weights so that each party can solve the local problems in a decentralized manner, and enables each party to efficiently prove in zero-knowledge that it computed the local optimization problem correctly.

II. Background

A. Preliminaries

In this section, we describe the notation we use for the rest of the paper. Let $P_1, ..., P_m$ denote the m parties. Let $\mathbb{Z}_N$ denote the set of integers modulo N, and $\mathbb{Z}_p$ denote the set of integers modulo a prime p. Similarly, we use $\mathbb{Z}_N^*$ to denote the multiplicative group modulo N.

We use z to denote a scalar, $\mathbf{z}$ to denote a vector, and $\mathbf{Z}$ to denote a matrix. We use $\texttt{Enc}_{\texttt{PK}}(x)$ to denote an encryption of x under a public key PK. Similarly, $\texttt{Dec}_{\texttt{SK}}(y)$ denotes a decryption of y under the secret key SK.

Each party P_i has a feature matrix $\mathbf{X}_i \in \mathbb{R}^{n \times d}$, where n is the number of samples per party and d is the feature dimension. $\mathbf{y}_i \in \mathbb{R}^{n \times 1}$ is the labels vector. The machine learning datasets use floating point representation, while our cryptographic primitives use groups and fields. Therefore, we represent the dataset using fixed point integer representation.

B. Cryptographic building blocks

In this section, we provide a brief overview of the cryptographic primitives used in Helen.

1) Threshold partially homomorphic encryption: A partially homomorphic encryption scheme is a public key encryption scheme that allows limited computation over the ciphertexts.

For example, Paillier [58] is an additive homomorphic encryption scheme: multiplying two ciphertexts together (in a certain group) generates a new ciphertext such that its decryption yields the sum of the two original plaintexts. Anyone with the public key can encrypt and manipulate the ciphertexts based on their homomorphic property. This encryption scheme also acts as a perfectly binding and computationally hiding homomorphic commitment scheme [39], another property we use in Helen.

A *threshold* variant of such a scheme has some additional properties. While the public key is known to everyone, the secret key is split across a set of parties such that a subset of them must participate together to decrypt a ciphertext. If not enough members participate, the ciphertext cannot be decrypted. The threshold structure can be altered based on the adversarial assumption. In Helen, we use a threshold structure where *all* parties must participate in order to decrypt a ciphertext.

2) Zero knowledge proofs: Informally, zero knowledge proofs are proofs that prove that a certain statement is true without revealing the prover's secret for this statement. For example, a prover can prove that there is a solution to a Sudoku puzzle without revealing the actual solution. Zero knowledge *proofs of knowledge* additionally prove that the prover indeed knows the secret. Helen uses modified Σ-protocols [25] to prove properties of a party's local computation. The main building blocks we use are ciphertext proof of plaintext knowledge, plaintext-ciphertext multiplication, and ciphertext interval proof of plaintext knowledge [23, 14], as we further explain in Section IV. Note that Σ-protocols are honest verifier zero knowledge, but can be transformed into full zero-knowledge using existing techniques [24, 32, 33]. In our paper, we present our protocol using the Σ-protocol notation.

3) Malicious MPC: We utilize SPDZ [27], a state-of-the-art malicious MPC protocol, for both Helen and the secure baseline we evaluate against. Another recent malicious MPC protocol is authenticated garbled circuits [69], which supports boolean circuits. We decided to use SPDZ for our baseline because the majority of the computation in SGD is spent doing matrix operations, which is not efficiently represented in boolean circuits. For the rest of this section we give an overview of the properties of SPDZ.

An input $a \in \mathbb{F}_{p^k}$ to SPDZ is represented as $\langle a \rangle = (\delta, (a_1, \ldots, a_n), (\gamma(a)_1, \ldots, \gamma(a)_n))$, where a_i is a share of a and $\gamma(a)_i$ is the MAC share authenticating a under a SPDZ global key α. Player i holds $a_i, \gamma(a)_i$, and δ is public. During a correct SPDZ execution, the following property must hold: $a = \sum_i a_i$ and $\alpha(a + \delta) = \sum_i \gamma(a)_i$. The global key α is not revealed until the end of the protocol; otherwise the malicious parties can use α to construct new MACs.

SPDZ has two phases: an offline phase and an online phase. The offline phase is independent of the function and generates precomputed values that can be used during the online phase, while the online phase executes the designated function.

C. Learning and Convex Optimization

Much of contemporary machine learning can be framed in the context of minimizing the *cumulative error* (or loss) of a model over the training data. While there is considerable excitement around deep neural networks, the vast majority of real-world machine learning applications still rely on robust linear models because they are well understood and can be efficiently and reliably learned using established convex optimization procedures.

In this work, we focus on linear models with squared error and various forms of regularization resulting in the following set of multi-party optimization problems:

$$\hat{\mathbf{w}} = \arg\min_{w} \frac{1}{2} \sum_{i=1}^{m} \|\mathbf{X}_i \mathbf{w} - \mathbf{y}_i\|_2^2 + \lambda \mathbf{R}(\mathbf{w}), \qquad (1)$$

where $\mathbf{X}_i \in \mathbb{R}^{n \times d}$ and $\mathbf{y}_i \in \mathbb{R}^n$ are the training data (features and labels) from party i. The regularization function $\mathbf{R}$ and regularization tuning parameter λ are used to improve prediction accuracy on high-dimensional data. Typically, the regularization function takes one of the following forms:

$$\mathbf{R}_{L^1}(\mathbf{w}) = \sum_{j=1}^{d} |\mathbf{w}_j|, \qquad \mathbf{R}_{L^2}(\mathbf{w}) = \frac{1}{2} \sum_{j=1}^{d} \mathbf{w}_j^2$$

corresponding to Lasso (L^1) and Ridge (L^2) regression respectively. The estimated model $\hat{\mathbf{w}} \in \mathbb{R}^d$ can then be used to render a new prediction $\hat{y}_* = \hat{\mathbf{w}}^T \mathbf{x}_*$ at a query point $\mathbf{x}_*$. It is worth noting that in some applications of LASSO (e.g., genomics [28]) the dimension d can be larger than n. However, in this work we focus on settings where d is smaller than n, and the real datasets and scenarios we use in our evaluation satisfy this property.

ADMM. Alternating Direction Method of Multipliers (ADMM) [15] is an established technique for distributed convex optimization. To use ADMM, we first reformulate Eq. 1 by introducing *additional* variables and constraints:

$$\begin{aligned} \underset{\{\mathbf{w}_i\}_{i=1}^m, \mathbf{z}}{\textbf{minimize:}} \quad & \frac{1}{2} \sum_{i=1}^{m} \|\mathbf{X}_i \mathbf{w}_i - \mathbf{y}_i\|_2^2 + \lambda \mathbf{R}(\mathbf{z}), \\ \textbf{such that:} \quad & \mathbf{w}_i = \mathbf{z} \text{ for all } i \in \{1, \ldots, p\} \end{aligned} \qquad (2)$$

This equivalent formulation splits $\mathbf{w}$ into $\mathbf{w}_i$ for each party i, but still requires that $\mathbf{w}_i$ be equal to a global model $\mathbf{z}$. To solve this constrained formulation, we construct an *augmented Lagrangian*:

$$\begin{aligned} L\left(\{\mathbf{w}_i\}_{i=1}^m, \mathbf{z}, \mathbf{u}\right) = \frac{1}{2} \sum_{i=1}^{m} \|\mathbf{X}_i \mathbf{w}_i - \mathbf{y}_i\|_2^2 + \lambda \mathbf{R}(\mathbf{z}) + \\ \rho \sum_{i=1}^{m} \mathbf{u}_i^T \left(\mathbf{w}_i - \mathbf{z}\right) + \frac{\rho}{2} \sum_{i=1}^{m} ||\mathbf{w}_i - \mathbf{z}||_2^2, \end{aligned} \qquad (3)$$

where the dual variables $\mathbf{u}_i \in \mathbb{R}^d$ capture the mismatch between the model estimated by party i and the global model $\mathbf{z}$ and the augmenting term $\frac{\rho}{2} \sum_{i=1}^{m} ||\mathbf{w}_i - \mathbf{z}||_2^2$ adds an additional penalty (scaled by the constant ρ) for deviating from $\mathbf{z}$.

The ADMM algorithm is a simple iterative dual ascent on the augmented Lagrangian of Eq. (2). On the k^{th} iteration, each party locally solves this closed-form expression:

$$\mathbf{w}_i^{k+1} \leftarrow \left(\mathbf{X}_i^T \mathbf{X}_i + \rho \mathbf{I}\right)^{-1} \left(\mathbf{X}_i^T \mathbf{y}_i + \rho \left(\mathbf{z}^k - \mathbf{u}_i^k\right)\right) \qquad (4)$$

and then shares its local model $\mathbf{w}_i^{k+1}$ and Lagrange multipliers $\mathbf{u}_i^k$ to solve for the new global weights:

$$\mathbf{z}^{k+1} \leftarrow \arg\min_{\mathbf{z}} \lambda \mathbf{R}(\mathbf{z}) + \frac{\rho}{2} \sum_{i=1}^{m} ||\mathbf{w}_i^{k+1} - \mathbf{z} + \mathbf{u}_i^k||_2^2. \quad (5)$$

Finally, each party uses the new global weights $\mathbf{z}^{k+1}$ to update its local Lagrange multipliers

$$\mathbf{u}_i^{k+1} \leftarrow \mathbf{u}_i^k + \mathbf{w}_i^{k+1} - \mathbf{z}^{k+1}. \quad (6)$$

The update equations (4), (5), and (6) are executed iteratively until all updates reach a fixed point. In practice, a fixed number of iterations may be used as a stopping condition, and that is what we do in Helen.

LASSO. We use LASSO as a running example for the rest of the paper in order to illustrate how our secure training protocol works. LASSO is a popular regularized linear regression model that uses the L^1 norm as the regularization function. The LASSO formulation is given by the optimization objective $\arg\min_{\mathbf{w}} \|\mathbf{X}\mathbf{w}-\mathbf{y}\|_2^2+\lambda\|\mathbf{w}\|$. The boxed section below shows the ADMM training procedure for LASSO. Here, the quantities in color are quantities that are intermediate values in the computation and need to be protected from every party, whereas the quantities in black are private values known to one party.

The coopetitive learning task for LASSO

Input of party P_i: $\mathbf{X}_i, \mathbf{y}_i$
1) $\mathbf{A}_i \leftarrow \left(\mathbf{X}_i^T\mathbf{X}_i + \rho\mathbf{I}\right)^{-1}$
2) $\mathbf{b}_i \leftarrow \mathbf{X}_i^T\mathbf{y}_i$
3) $\mathbf{u^0}, \mathbf{z^0}, \mathbf{w^0} \leftarrow \mathbf{0}$
4) For $k = 0$, ADMMIterations-1:
   a) $\mathbf{w}_i^{k+1} \leftarrow \mathbf{A}_i(\mathbf{b}_i + \rho\left(\mathbf{z}^k - \mathbf{u}_i^k\right))$
   b) $\mathbf{z}^{k+1} \leftarrow S_{\lambda/m\rho}\left(\frac{1}{m}\sum_{i=1}^{m}\left(\mathbf{w}_i^{k+1} + \mathbf{u}_i^k\right)\right)$
   c) $\mathbf{u}_i^{k+1} \leftarrow \mathbf{u}_i^k + \mathbf{w}_i^{k+1} - \mathbf{z}^{k+1}$

$S_{\lambda/m\rho}$ is the soft the soft thresholding operator, where

$$S_\kappa(a) = \begin{cases} a-\kappa & a > \kappa \\ 0 & |a| \le \kappa \\ a+\kappa & a < -\kappa \end{cases} \quad (7)$$

The parameters λ and ρ are public and fixed.

III. System overview

Figure 2 shows the system setup in Helen. A group of m participants (also called parties) wants to jointly train a model on their data without sharing the plaintext data. As mentioned in Section I, the use cases we envision for our system consist of a few large organizations (around 10 organizations), where each organization has a lot of data (n is on the order of hundreds of thousands or millions). The number of features/columns in the dataset d is on the order of tens or hundreds. Hence $d \ll n$.

We assume that the parties have agreed to publicly release the final model. As part of Helen, they will engage in an interactive protocol during which they share encrypted data, and only at the end will they obtain the model in decrypted form. Helen supports regularized linear models including least squares linear regression, ridge regression, LASSO, and elastic net. In the rest of the paper, we focus on explaining Helen via LASSO, but we also provide update equations for ridge regression in Section VII.

A. Threat model

We assume that all parties have agreed upon a single functionality to compute and have also consented to releasing the final result of the function to every party.

We consider a strong threat model in which all but one party can be compromised by a malicious attacker. This means that the compromised parties can deviate arbitrarily from the protocol, such as supplying inconsistent inputs, substituting their input with another party's input, or executing different computation than expected. In the flu prediction example, six divisions could collude together to learn information about one of the medical divisions. However, as long as the victim medical division follows our protocol correctly, the other divisions will not be able to learn anything about the victim division other than the final result of the function. We now state the security theorem.

Theorem 6. *Helen securely evaluates an ideal functionality f_{ADMM} in the (f_{crs}, f_{SPDZ})-hybrid model under standard cryptographic assumptions, against a malicious adversary who can statically corrupt up to $m-1$ out of m parties.*

We formalize the security of Helen in the standalone MPC model. f_{crs} and f_{SPDZ} are ideal functionalities that we use in our proofs, where f_{crs} is the ideal functionality representing the creation of a common reference string, and f_{SPDZ} is the ideal functionality that makes a call to SPDZ. Due to space constraints, we present the formal definitions as well as the security proofs in the full version of this paper.

Out of scope attacks/complementary directions. Helen does not prevent a malicious party from choosing a bad dataset for the coopetitive computation (e.g., in an attempt to alter the computation result). In particular, Helen does not prevent poisoning attacks [44, 18]. MPC protocols generally do not protect against bad inputs because there is no way to ensure that a party provides true data. Nevertheless, Helen will ensure that once a party supplies its input into the computation, the party is bound to using the same input consistently throughout the entire computation; in particular, this prevents a party from providing different inputs at different stages of the computation, or mix-and-matching inputs from other parties. Further, some additional constraints can also be placed in pre-processing, training, and post-processing to mitigate such attacks, as we elaborate in Section IX-B.

Helen also does not protect against attacks launched on the public model, for example, attacks that attempt to recover the training data from the model itself [65, 16]. The parties are responsible for deciding if they are willing to share with each other the model. Our goal is only to conduct this computation securely: to ensure that the parties do not share their raw

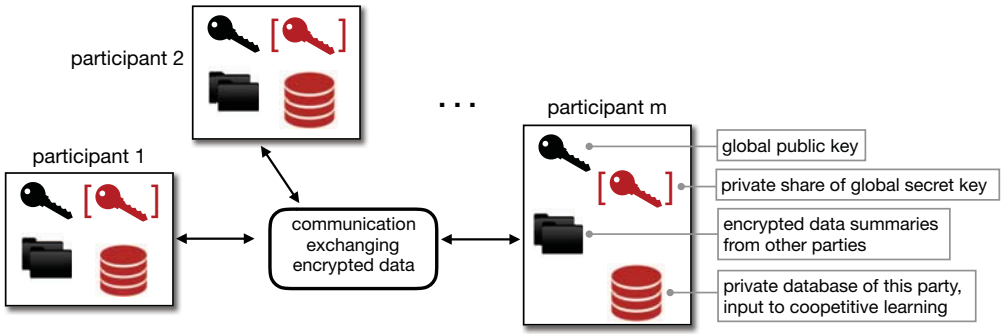


Fig. 2: Architecture overview of Helen. Every red shape indicates secret information known only to the indicated party, and black indicates public information visible to everyone (which could be private information in encrypted form). For participant m, we annotate the meaning of each quantity.

plaintext datasets with each other, that they do not learn more information than the resulting model, and that only the specified computation is executed. Investigating techniques for ensuring that the model does not leak too much about the data is a complementary direction to Helen, and we expect that many of these techniques could be plugged into a system like Helen. For example, Helen can be easily combined with some differential privacy tools that add noise before model release to ensure that the model does not leak too much about an individual record in the training data. We further discuss possible approaches in Section IX-B.

Finally, Helen does not protect against denial of service – all parties must participate in order to produce a model.

B. Protocol phases

We now explain the protocol phases at a high level. The first phase requires all parties to agree to perform the coopetitive computation, which happens before initializing Helen. The other phases are run using Helen.

Agreement phase. In this phase, the m parties come together and agree that they are willing to run a certain learning algorithm (in Helen's case, ADMM for linear models) over their joint data. The parties should also agree to release the computed model among themselves.

The following discrete phases are run by Helen. We summarize their purposes here and provide the technical design for each in the following sections.

Initialization phase. During initialization, the m parties compute the encryption keys using a generic MPC protocol. The public output of this protocol is a public key PK that is known to everyone. Each party also receives a piece (called a *share*) of the corresponding secret key SK: party P_i receives the i-th share of the key denoted as $[\text{SK}]_i$. A value encrypted under PK can only be decrypted via all shares of the SK, so every party needs to agree to decrypt this value. Fig. 2 shows these keys. This phase only needs to run once for the entire training process, and does not need to be re-run as long as the parties' configuration does not change.

Input preparation phase. In this phase, each party prepares its data for the coopetitive computation. Each party P_i precomputes summaries of its data and commits to them by broadcasting encrypted summaries to all other parties. The parties also need to prove that they know the values inside these encryptions using zero-knowledge proofs of knowledge. From this moment on, party P_i will not be able to use different inputs for the rest of the computation.

By default, each party stores the encrypted summaries from other parties. This is a viable solution since these summaries are much smaller than the data itself. It is possible to also store all m summaries in a public cloud by having each party produce an integrity MAC of the summary from each other party and checking the MAC upon retrieval which protects against a compromised cloud.

Model compute phase. This phase follows the iterative ADMM algorithm, in which parties successively compute locally on encrypted data, followed by a coordination step with other parties using a generic MPC protocol.

Throughout this protocol, each party receives only encrypted intermediate data. No party learns the intermediate data because, by definition, an MPC protocol should not reveal any data beyond the final result. Moreover, each party proves in zero knowledge to the other parties that it performed the local computation correctly using data that is consistent with the private data that was committed in the input preparation phase. If any one party misbehaves, the other parties will be able to detect the cheating with overwhelming probability.

Model release phase. At the end of the model compute phase, all parties obtain an encrypted model. All parties jointly decrypt the weights and release the final model. However, it is possible for a set of parties to not receive the final model at the end of training if other parties misbehave (it has been proven that it is impossible to achieve fairness for generic MPC in the malicious majority setting [19]). Nevertheless, this kind of malicious behavior is easily detectable in Helen and can be enforced using legal methods.

IV. Cryptographic Gadgets

Helen's design combines several different cryptographic primitives. In order to explain the design clearly, we split Helen into modular gadgets. In this section and the following sections, we discuss (1) how Helen implements these gadgets, and (2) how Helen composes them in the overall protocol.

For simplicity, we present our zero knowledge proofs as Σ-protocols, which require the verifier to generate random challenges. These protocols can be transformed into full zero knowledge with non-malleable guarantees with existing techniques [33, 32]. We present these transformations in the full version of the paper.

A. Plaintext-ciphertext matrix multiplication proof

Gadget 1. *A zero-knowledge proof for the statement: "Given public parameters: public key PK, encryptions $E_{\mathbf{X}}$, $E_{\mathbf{Y}}$ and $E_{\mathbf{Z}}$; private parameters: $\mathbf{X}$,*
- $\mathtt{Dec}_{SK}(E_{\mathbf{Z}}) = \mathtt{Dec}_{SK}(E_{\mathbf{X}}) \cdot \mathtt{Dec}_{SK}(E_{\mathbf{Y}})$*, and*
- *I know $\mathbf{X}$ such that $\mathtt{Dec}_{SK}(E_{\mathbf{X}}) = \mathbf{X}$."*

Gadget usage. We first explain how Gadget 1 is used in Helen. A party P_i in Helen knows a plaintext $\mathbf{X}$ and commits to $\mathbf{X}$ by publishing its encryption, denoted by $\mathtt{Enc}_{\mathsf{PK}}(\mathbf{X})$. P_i also receives an encrypted matrix $\mathtt{Enc}_{\mathsf{PK}}(\mathbf{Y})$ and needs to compute $\mathtt{Enc}_{\mathsf{PK}}(\mathbf{Z}) = \mathtt{Enc}_{\mathsf{PK}}(\mathbf{XY})$ by leveraging the homomorphic properties of the encryption scheme. Since parties in Helen may be malicious, other parties cannot trust P_i to compute and output $\mathtt{Enc}_{\mathsf{PK}}(\mathbf{Z})$ correctly. Gadget 1 will help P_i prove in zero-knowledge that it executed the computation correctly. The proof needs to be zero-knowledge so that nothing is leaked about the value of $\mathbf{X}$. It also needs to be a proof of knowledge so that P_i proves that it knows the plaintext matrix $\mathbf{X}$.

Protocol. Using the Paillier ciphertext multiplication proofs [23], we can construct a naïve algorithm for proving matrix multiplication. For input matrices that are $\mathbb{R}^{l\times l}$, the naïve algorithm will incur a cost of l^3 since one has to prove each individual product. One way to reduce this cost is to have the prover prove that $\mathbf{tZ} = (\mathbf{tX})\mathbf{Y}$ for a randomly chosen $\mathbf{t}$ such that $\mathbf{t}_i = t^i \mod q$. For such a randomly chosen t, the chance that the prover can construct a $\mathbf{tZ}' = \mathbf{tXY}$ is exponentially small (an analysis is presented in our full paper).

As the first step, both the prover and the verifier apply the reduction to get the new statement $\mathtt{Enc}_{\mathsf{PK}}(\mathbf{tZ}) = \mathtt{Enc}_{\mathsf{PK}}(\mathbf{tX})\mathtt{Enc}_{\mathsf{PK}}(\mathbf{Y})$. To prove this reduced form, we apply the Paillier ciphertext multiplication proof in a straightforward way. This proof takes as input three ciphertexts: E_a, E_b, E_c. The prover proves that it knows the plaintext a^* such that $a^* = \mathtt{Dec}_{\mathsf{SK}}(E_a)$, and that $\mathtt{Dec}_{\mathsf{SK}}(E_c) = \mathtt{Dec}_{\mathsf{SK}}(E_a) \cdot \mathtt{Dec}_{\mathsf{SK}}(E_b)$. We apply this proof to every multiplication for each dot product in $(\mathbf{tX}) \cdot \mathbf{Y}$. The prover then releases the individual encrypted products along with the corresponding ciphertext multiplication proofs. The verifier needs to verify that $\mathtt{Enc}_{\mathsf{PK}}(\mathbf{tZ}) = \mathtt{Enc}_{\mathsf{PK}}(\mathbf{tXY})$. Since the encrypted ciphers from the previous step are encrypted using Paillier, the verifier can homomorphically add them appropriately to get the encrypted vector $\mathtt{Enc}_{\mathsf{PK}}(\mathbf{tXY})$.

Finally, the prover needs to prove that each element of $\mathbf{tZ}$ is equal to each element of $\mathbf{tXY}$. We can use the same ciphertext multiplication proof by setting $a^* = 1$.

B. Plaintext-plaintext matrix multiplication proof

Gadget 2. *A zero-knowledge proof for the statement: "Given public parameters: public key PK, encryptions $E_{\mathbf{X}}$, $E_{\mathbf{Y}}$, $E_{\mathbf{Z}}$; private parameters: $\mathbf{X}$ and $\mathbf{Y}$,*
- $\mathtt{Dec}_{SK}(E_{\mathbf{Z}}) = \mathtt{Dec}_{SK}(E_{\mathbf{X}}) \cdot \mathtt{Dec}_{SK}(E_{\mathbf{Y}})$*, and*
- *I know $\mathbf{X}$, $\mathbf{Y}$, and $\mathbf{Z}$ such that $\mathtt{Dec}_{SK}(E_{\mathbf{X}}) = \mathbf{X}$, $\mathtt{Dec}_{SK}(E_{\mathbf{Y}}) = \mathbf{Y}$, and $\mathtt{Dec}_{SK}(E_{\mathbf{Z}}) = \mathbf{Z}$."*

Gadget usage. This proof is used to prove matrix multiplication when the prover knows *both* input matrices (and thus the output matrix as well). The protocol is similar to the plaintext-ciphertext proofs, except that we have to do an additional proof of knowledge of $\mathbf{Y}$.

Protocol. The prover wishes to prove to a verifier that $\mathbf{Z} = \mathbf{XY}$ without revealing $\mathbf{X}$, $\mathbf{Y}$, or $\mathbf{Z}$. We follow the same protocol as Gadget 1. Additionally, we utilize a variant of the ciphertext multiplication proof that only contains the proof of knowledge component to show that the prover also knows $\mathbf{Y}$. The proof of knowledge for the matrix is simply a list of element-wise proofs for $\mathbf{Y}$. We do not explicitly prove the knowledge of $\mathbf{Z}$ because the matrix multiplication proof and the proof of knowledge for $\mathbf{Y}$ imply that the prover knows $\mathbf{Z}$ as well.

V. Input preparation phase

A. Overview

In this phase, each party prepares data for coopetitive training. In the beginning of the ADMM procedure, every party precomputes some summaries of its data and commits to them by broadcasting encrypted summaries to all the other parties. These summaries are then reused throughout the model compute phase. Some form of commitment is necessary in the malicious setting because an adversary can deviate from the protocol by altering its inputs. Therefore, we need a new gadget that allows us to efficiently commit to these summaries.

More specifically, the ADMM computation reuses two matrices during training: $\mathbf{A}_i = (\mathbf{X}_i^T\mathbf{X}_i + \rho\mathbf{I})^{-1}$ and $\mathbf{b}_i = \mathbf{X}_i^T\mathbf{y}_i$ from party i (see Section II-C for more details). These two matrices are of sizes $d \times d$ and $d \times 1$, respectively. In a semihonest setting, we would trust parties to compute $\mathbf{A}_i$ and $\mathbf{b}_i$ correctly. In a malicious setting, however, the parties can deviate from the protocol and choose $\mathbf{A}_i$ and $\mathbf{b}_i$ that are inconsistent with each other (e.g., they do not conform to the above formulations).

Helen does not have any control over what data each party contributes because the parties must be free to choose their own $\mathbf{X}_i$ and $\mathbf{y}_i$. However, Helen ensures that each party consistently uses the same $\mathbf{X}_i$ and $\mathbf{y}_i$ during the entire protocol. Otherwise, malicious parties could try to use different/inconsistent $\mathbf{X}_i$ and $\mathbf{y}_i$ at different stages of the protocol, and thus manipulate the

final outcome of the computation to contain the data of another party.

One possibility to address this problem is for each party i to commit to its $\mathbf{X}_i$ in $\texttt{Enc}_{\text{PK}}(\mathbf{X}_i)$ and $\mathbf{y}_i$ in $\texttt{Enc}_{\text{PK}}(\mathbf{y}_i)$. To calculate $\mathbf{A}_i$, the party can calculate and prove $\mathbf{X}_i^T\mathbf{X}$ using Gadget 2, followed by computing a matrix inversion computation within SPDZ. The result $\mathbf{A}_i$ can be repeatedly used in the iterations. This is clearly inefficient because (1) the protocol scales linearly in n, which could be very large, and (2) the matrix inversion computation requires heavy compute.

Our idea is to prove using an alternate formulation via *singular value decomposition (SVD)* [38], which can be much more succinct: $\mathbf{A}_i$ and $\mathbf{b}_i$ can be decomposed using SVD to matrices that scale linearly in d. Proving the properties of $\mathbf{A}_i$ and $\mathbf{b}_i$ using the decomposed matrices is equivalent to proving using $\mathbf{X}_i$ and $\mathbf{y}_i$.

B. Protocol

1) Decomposition of reused matrices: We first derive an alternate formulation for $\mathbf{X}_i$ (denoted as $\mathbf{X}$ for the rest of this section). From fundamental linear algebra concepts we know that every matrix has a corresponding singular value decomposition [38]. More specifically, there exists unitary matrices $\mathbf{U}$ and $\mathbf{V}$, and a diagonal matrix $\mathbf{\Gamma}$ such that $\mathbf{X} = \mathbf{U}\mathbf{\Gamma}\mathbf{V}^T$, where $\mathbf{U} \in \mathbb{R}^{n\times n}$, $\mathbf{\Gamma} \in \mathbb{R}^{n\times d}$, and $\mathbf{V} \in \mathbb{R}^{d\times d}$. Since $\mathbf{X}$ and thus $\mathbf{U}$ are real matrices, the decomposition also guarantees that $\mathbf{U}$ and $\mathbf{V}$ are orthogonal, meaning that $\mathbf{U}^T\mathbf{U} = \mathbf{I}$ and $\mathbf{V}^T\mathbf{V} = \mathbf{I}$. If $\mathbf{X}$ is not a square matrix, then the top part of $\mathbf{\Gamma}$ is a diagonal matrix, which we will call $\mathbf{\Sigma} \in \mathbb{R}^{d\times d}$. $\mathbf{\Sigma}$'s diagonal is a list of singular values σ_i. The rest of the $\mathbf{\Gamma}$ matrix are 0's. If $\mathbf{X}$ is a square matrix, then $\mathbf{\Gamma}$ is simply $\mathbf{\Sigma}$. Finally, the matrices $\mathbf{U}$ and $\mathbf{V}$ are orthogonal matrices. Given an orthogonal matrix $\mathbf{Q}$, we have that $\mathbf{Q}\mathbf{Q}^T = \mathbf{Q}^T\mathbf{Q} = \mathbf{I}$.

It turns out that $\mathbf{X}^T\mathbf{X}$ has some interesting properties:

$$\begin{aligned}\mathbf{X}^T\mathbf{X} &= (\mathbf{U}\mathbf{\Gamma}\mathbf{V}^T)^T\mathbf{U}\mathbf{\Gamma}\mathbf{V}^T \\ &= \mathbf{V}\mathbf{\Gamma}^T\mathbf{U}^T\mathbf{U}\mathbf{\Gamma}\mathbf{V}^T \\ &= \mathbf{V}\mathbf{\Gamma}^T\mathbf{\Gamma}\mathbf{V}^T \\ &= \mathbf{V}\mathbf{\Sigma}^2\mathbf{V}^T.\end{aligned}$$

We now show that $(\mathbf{X}^T\mathbf{X} + \rho\mathbf{I})^{-1} = \mathbf{V}\mathbf{\Theta}\mathbf{V}^T$, where $\mathbf{\Theta}$ is the diagonal matrix with diagonal values $\dfrac{1}{\sigma_i^2 + \rho}$.

$$\begin{aligned}(\mathbf{X}^T\mathbf{X} + \rho\mathbf{I})\mathbf{V}\mathbf{\Theta}\mathbf{V}^T &= \mathbf{V}(\mathbf{\Sigma}^2 + \rho\mathbf{I})\mathbf{V}^T\mathbf{V}\mathbf{\Theta}\mathbf{V}^T \\ &= \mathbf{V}(\mathbf{\Sigma}^2 + \rho\mathbf{I})\mathbf{\Theta}\mathbf{V}^T \\ &= \mathbf{V}\mathbf{V}^T = \mathbf{I}.\end{aligned}$$

Using a similar reasoning, we can also derive that

$$\mathbf{X}^T\mathbf{y} = \mathbf{V}\mathbf{\Gamma}^T\mathbf{U}^T\mathbf{y}.$$

2) Properties after decomposition: The SVD decomposition formulation sets up an alternative way to commit to matrices $(\mathbf{X}_i^T\mathbf{X}_i + \rho\mathbf{I})^{-1}$ and $\mathbf{X}_i\mathbf{y}_i$. For the rest of this section, we describe the zero knowledge proofs that every party has to execute. For simplicity, we focus on one party and use $\mathbf{X}$ and $\mathbf{y}$ to represent its data, and $\mathbf{A}$ and $\mathbf{b}$ to represent its summaries.

During the ADMM computation, matrices $\mathbf{A} = (\mathbf{X}^T\mathbf{X} + \rho\mathbf{I})^{-1}$ and $\mathbf{b} = \mathbf{X}^T\mathbf{y}$ are repeatedly used to calculate the intermediate weights. Therefore, each party needs to commit to $\mathbf{A}$ and $\mathbf{b}$. With the alternative formulation, it is no longer necessary to commit to $\mathbf{X}$ and $\mathbf{y}$ individually. Instead, it suffices to prove that a party knows $\mathbf{V}$, $\mathbf{\Theta}$, $\mathbf{\Sigma}$ (all are in $\mathbb{R}^{d\times d}$) and a vector $\mathbf{y}^* = (\mathbf{U}^T\mathbf{y})_{[1:d]} \in \mathbb{R}^{d\times 1}$ such that:

1) $\mathbf{A} = \mathbf{V}\mathbf{\Theta}\mathbf{V}^T$,
2) $\mathbf{b} = \mathbf{V}\mathbf{\Sigma}^T\mathbf{y}^*$,
3) $\mathbf{V}$ is an orthogonal matrix, namely, $\mathbf{V}^T\mathbf{V} = \mathbf{I}$, and
4) $\mathbf{\Theta}$ is a diagonal matrix where the diagonal entries are $1/(\sigma_i^2 + \rho)$. σ_i are the values on the diagonal of $\mathbf{\Sigma}$ and ρ is a public value.

Note that $\mathbf{\Gamma}$ can be readily derived from $\mathbf{\Sigma}$ by adding rows of zeros. Moreover, both $\mathbf{\Theta}$ and $\mathbf{\Sigma}$ are diagonal matrices. Therefore, we only commit to the diagonal entries of $\mathbf{\Theta}$ and $\mathbf{\Sigma}$ since the rest of the entries are zeros.

The above four statements are sufficient to prove the properties of $\mathbf{A}$ and $\mathbf{b}$ in the new formulation. The first two statements simply prove that $\mathbf{A}$ and $\mathbf{b}$ are indeed decomposed into *some* matrices $\mathbf{V}$, $\mathbf{\Theta}$, $\mathbf{\Sigma}$, and $\mathbf{y}^*$. Statement 3) shows that $\mathbf{V}$ is an orthogonal matrix, since by definition an orthogonal matrix $\mathbf{Q}$ has to satisfy the equation $\mathbf{Q}^T\mathbf{Q} = \mathbf{I}$. However, we allow the prover to choose $\mathbf{V}$. As stated before, the prover would have been free to choose $\mathbf{X}$ and $\mathbf{y}$ anyway, so this freedom does not give more power to the prover.

Statement 4) proves that the matrix $\mathbf{\Theta}$ is a diagonal matrix such that the diagonal values satisfy the form above. This is sufficient to show that $\mathbf{\Theta}$ is correct according to *some* $\mathbf{\Sigma}$. Again, the prover is free to choose $\mathbf{\Sigma}$, which is the same as freely choosing its input $\mathbf{X}$.

Finally, we chose to commit to $\mathbf{y}^*$ instead of committing to $\mathbf{U}$ and $\mathbf{y}$ separately. Following our logic above, it seems that we also need to commit to $\mathbf{U}$ and prove that it is an orthogonal matrix, similar to what we did with $\mathbf{V}$. This is not necessary because of an important property of orthogonal matrices: $\mathbf{U}$'s columns span the vector space $\mathbb{R}^n$. Multiplying $\mathbf{U}\mathbf{y}$, the result is a linear combination of the columns of $\mathbf{U}$. Since we also allow the prover to pick its $\mathbf{y}$, $\mathbf{U}\mathbf{y}$ essentially can be any vector in $\mathbb{R}^n$. Thus, we only have to allow the prover to commit to the product of $\mathbf{U}$ and $\mathbf{y}$. As we can see from the derivation, $\mathbf{b} = \mathbf{V}\mathbf{\Gamma}^T\mathbf{U}\mathbf{y}$, but since $\mathbf{\Gamma}$ is simply $\mathbf{\Sigma}$ with rows of zeros, the actual decomposition only needs the first d elements of $\mathbf{U}\mathbf{y}$. Hence, this allows us to commit to $\mathbf{y}^*$, which is $d \times 1$.

Using our techniques, Helen commits only to matrices of sizes $d \times d$ or $d \times 1$, thus removing any scaling in n (the number of rows in the dataset) in the input preparation phase.

3) Proving the initial data summaries: First, each party broadcasts $\texttt{Enc}_{\text{PK}}(\mathbf{V})$, $\texttt{Enc}_{\text{PK}}(\mathbf{\Sigma})$, $\texttt{Enc}_{\text{PK}}(\mathbf{\Theta})$, $\texttt{Enc}_{\text{PK}}(\mathbf{y}^*)$, $\texttt{Enc}_{\text{PK}}(\mathbf{A})$, and $\texttt{Enc}_{\text{PK}}(\mathbf{b})$. To encrypt a matrix, the party simply individually encrypts each entry. The encryption scheme itself also acts as a commitment scheme [39], so we do not need an extra commitment scheme.

To prove these statements, we also need another primitive called an interval proof. Moreover, since these matrices act as inputs to the model compute phase, we also need to prove that $\mathbf{A}$ and $\mathbf{b}$ are within a certain range (this will be used by Gadget 4). The interval proof we use is from [14], which is an efficient way of proving that a committed number lies within a certain interval. However, what we want to prove is that an encrypted number lies within a certain interval. This can be solved by using techniques from [26], which appends the range proof with a commitment-ciphertext equality proof. This extra proof proves that, given a commitment and a Paillier ciphertext, both hide the same plaintext value.

To prove the first two statements, we invoke Gadget 1 and Gadget 2. This allows us to prove that the party knows all of the matrices in question and that they satisfy the relations laid out in those statements.

There are two steps to proving statement 3. The prover will compute $\texttt{Enc}_{\text{PK}}(\mathbf{V}^T\mathbf{V})$ and prove it computed it correctly using Gadget 1 as above. The result should be equal to the encryption of the identity matrix. However, since we are using fixed point representation for our data, the resulting matrix could be off from the expected values by some small error. $\mathbf{V}^T\mathbf{V}$ will only be close to $\mathbf{I}$, but not equal to $\mathbf{I}$. Therefore, we also utilize interval proofs to make sure that $\mathbf{V}^T\mathbf{V}$ is close to $\mathbf{I}$, without explicitly revealing the value of $\mathbf{V}^T\mathbf{V}$.

Finally, to prove statement 4, the prover does the following:

1) The prover computes and releases $\texttt{Enc}_{\text{PK}}(\mathbf{\Sigma}^2)$ because the prover knows $\mathbf{\Sigma}$ and proves using Gadget 1 that this computation is done correctly.
2) The prover computes $\texttt{Enc}_{\text{PK}}(\mathbf{\Sigma}^2 + \rho\mathbf{I})$, which anyone can compute because ρ and $\mathbf{I}$ are public. $\texttt{Enc}_{\text{PK}}(\mathbf{\Sigma}^2)$ and $\texttt{Enc}_{\text{PK}}(\rho\mathbf{I})$ can be multiplied together to get the summation of the plaintext matrices.
3) The prover now computes $\texttt{Enc}_{\text{PK}}(\mathbf{\Sigma}^2 + \rho\mathbf{I}) \times \texttt{Enc}_{\text{PK}}(\mathbf{\Theta})$ and proves this encryption was computed correctly using Gadget 1.
4) Similar to step 3), the prover ends this step by using interval proofs to prove that this encryption is close to encryption of the identity matrix.

VI. Model compute phase

A. Overview

In the model compute phase, all parties use the summaries computed in the input preparation phase and execute the iterative ADMM training protocol. An encrypted weight vector is generated at the end of this phase and distributed to all participants. The participants can jointly decrypt this weight vector to get the plaintext model parameters. This phase executes in three steps: initialization, training (local optimization and coordination), and model release.

B. Initialization

We initialize the weights $\mathbf{w}_i^0, \mathbf{z}^0$, and $\mathbf{u}_i^0$. There are two popular ways of initializing the weights. The first way is to set every entry to a random number. The second way is to initialize every entry to zero. In Helen, we use the second method because it is easy and works well in practice.

C. Local optimization

During ADMM's local optimization phase, each party takes the current weight vector and iteratively optimizes the weights based on its own dataset. For LASSO, the update equation is simply $\mathbf{w}_i^{k+1} \leftarrow \mathbf{A}_i(\mathbf{b}_i + \rho\left(\mathbf{z}^k - \mathbf{u}_i^k\right))$, where $\mathbf{A}_i$ is the matrix $(\mathbf{X}_i^T\mathbf{X}_i + \rho\mathbf{I})^{-1}$ and $\mathbf{b}_i$ is $\mathbf{X}_i^T\mathbf{y}_i$. As we saw from the input preparation phase description, each party holds encryptions of $\mathbf{A}_i$ and $\mathbf{b}_i$. Furthermore, given $\mathbf{z}^k$ and $\mathbf{u}_i^k$ (either initialized or received as results calculated from the previous round), each party can independently calculate $\mathbf{w}_i^{k+1}$ by doing plaintext scaling and plaintext-ciphertext matrix multiplication. Since this is done locally, each party also needs to generate a proof proving that the party calculated $\mathbf{w}_i^{k+1}$ correctly. We compute the proof for this step by invoking Gadget 1.

D. Coordination using MPC

After the local optimization step, each party holds encrypted weights $\mathbf{w}_i^{k+1}$. The next step in the ADMM iterative optimization is the coordination phase. Since this step contains non-linear functions, we evaluate it using generic MPC.

1) Conversion to MPC: First, the encrypted weights need to be converted into an MPC-compatible input. To do so, we formulate a gadget that converts ciphertext to arithmetic shares. The general idea behind the protocol is inspired by arithmetic sharing protocols [23, 27].

> **Gadget 3.** *For m parties, each party having the public key PK and a share of the secret key SK, given public ciphertext $\texttt{Enc}_{PK}(a)$, convert a into m shares $a_i \in \mathbb{Z}_p$ such that $a \equiv \sum a_i \mod p$. Each party P_i receives secret share a_i and does not learn the original secret value a.*

Gadget usage. Each party uses this gadget to convert $\texttt{Enc}_{\text{PK}}(\mathbf{w}_i)$ and $\texttt{Enc}_{\text{PK}}(\mathbf{u}_i)$ into input shares and compute the soft threshold function using MPC (in our case, SPDZ). We denote p as the public modulus used by SPDZ.

Protocol. The protocol proceeds as follows:

1) Each party P_i generates a random value $r_i \in [0, 2^{|p|+\kappa}]$ and encrypts it, where κ is a statistical security parameter. Each party should also generate an interval plaintext proof of knowledge of r_i, then publish $\texttt{Enc}_{\text{PK}}(r_i)$ along with the proofs.
2) Each party P_i takes as input the published $\{\texttt{Enc}_{\text{PK}}(r_j)\}_{j=1}^m$ and compute the product with $\texttt{Enc}_{\text{PK}}(a)$. The result is $c = \texttt{Enc}_{\text{PK}}(a + \sum_{j=1}^m r_j)$.
3) All parties jointly decrypt c to get plaintext b.
4) Party 0 sets $a_0 = b - r_0 \mod p$. Every other party sets $a_i \equiv -r_i \mod p$.
5) Each party publishes $\texttt{Enc}_{\text{PK}}(a_i)$ as well as an interval proof of plaintext knowledge.

2) Coordination: The ADMM coordination step takes in $\mathbf{w}_i^{k+1}$ and $\mathbf{u}_i^k$, and outputs $\mathbf{z}^{k+1}$. The $\mathbf{z}$ update requires computing the soft-threshold function (a non-linear function), so we express it in MPC. Additionally, since we are doing fixed point integer arithmetic as well as using a relatively small prime modulus for MPC (256 bits in our implementation), we need to reduce the scaling factors accumulated on $\mathbf{w}_i^{k+1}$ during plaintext-ciphertext matrix multiplication. We currently perform this operation inside MPC as well.

3) Conversion from MPC: After the MPC computation, each party receives shares of $\mathbf{z}$ and its MAC shares, as well as shares of $\mathbf{w}_i$ and its MAC shares. It is easy to convert these shares back into encrypted form simply by encrypting the shares, publishing them, and summing up the encrypted shares. We can also calculate $\mathbf{u}_i^{k+1}$ this way. Each party also publishes interval proofs of knowledge for each published encrypted cipher. Finally, in order to verify that they are indeed valid SPDZ shares (the specific protocol is explained in the next section), each party also publishes encryptions and interval proofs of all the MACs.

E. Model release

1) MPC conversion verification: Since we are combining two protocols (homomorphic encryption and MPC), an attacker can attempt to alter the inputs to either protocol by using different or inconsistent attacker-chosen inputs. Therefore, before releasing the model, the parties must prove that they correctly executed the ciphertext to MPC conversion (and vice versa). We use another gadget to achieve this.

> **Gadget 4.** *Given public parameters: encrypted value $\texttt{Enc}_{PK}(a)$, encrypted SPDZ input shares $\texttt{Enc}_{PK}(b_i)$, encrypted SPDZ MACs $\texttt{Enc}_{PK}(c_i)$, and interval proofs of plaintext knowledge, verify that*
> *1) $a \equiv \sum_i b_i \mod p$, and*
> *2) b_i are valid SPDZ shares and c_i's are valid MACs on b_i.*

Gadget usage. We apply Gadget 4 to all data that needs to be converted from encrypted ciphers to SPDZ or vice versa. More specifically, we need to prove that (1) the SPDZ input shares are consistent with $\texttt{Enc}_{\text{PK}}(\mathbf{w}_i^{k+1})$ that is published from each party, and (2) the SPDZ shares for $\mathbf{w}_i^{k+1}$ and $\mathbf{z}^k$ are authenticated by the MACs.

Protocol. The gadget construction proceeds as follows:

1) Each party verifies that $\texttt{Enc}_{\text{PK}}(a)$, $\texttt{Enc}_{\text{PK}}(b_i)$ and $\texttt{Enc}_{\text{PK}}(c_i)$ pass the interval proofs of knowledge. For example, b_i and c_i need to be within $[0, p]$.
2) Each party homomorphically computes $\texttt{Enc}_{\text{PK}}(\sum_i b_i)$, as well as $E_d = \texttt{Enc}_{\text{PK}}(a - \sum_i b_i)$.
3) Each party randomly chooses $r_i \in [0, 2^{|a|+|\kappa|}]$, where κ is again a statistical security parameter, and publishes $\texttt{Enc}_{\text{PK}}(r_i)$ as well as an interval proof of plaintext knowledge.
4) Each party calculates $E_f = E_d \prod_i \texttt{Enc}_{\text{PK}}(r_i)^p = \texttt{Enc}_{\text{PK}}((a - \sum_i b_i) + \sum_i (r_i \cdot p))$. Here we assume that $\log |m| + |p| + |a| + |\kappa| < |n|$.
5) All parties participate in a joint decryption protocol to decrypt E_f obtaining e_f.
6) Every party individually checks to see that e_f is a multiple of p. If this is not the case, abort the protocol.
7) The parties release the SPDZ global MAC key α.
8) Each party calculates $\texttt{Enc}_{\text{PK}}(\alpha(\sum b_i + \delta))$ and $\texttt{Enc}_{\text{PK}}(\sum c_i)$.
9) Use the same method in steps 2 – 6 to prove that $\alpha(\sum b_i + \delta) \equiv \sum c_i \mod p$.

The above protocol is a way for parties to verify two things. First, that the SPDZ shares indeed match with a previously published encrypted value (i.e., Gadget 3 was executed correctly). Second, that the shares are valid SPDZ shares. The second step is simply verifying the original SPDZ relation among value share, MAC shares, and the global key.

Note that we cannot verify these relations by simply releasing the plaintext data shares and their MACs since the data shares correspond to the intermediate weights. Furthermore, the shares need to be equivalent in modulo p, which is different from the Paillier parameter N. Therefore, we use an alternative protocol to test modulo equality between two ciphertexts, which is the procedure described above in steps 2 to 6.

Since the encrypted ciphers come with interval proofs of plaintext knowledge, we can assume that $a \in [0, l]$. If two ciphertexts encrypt plaintexts that are equivalent to each other, they must satisfy that $a \equiv b \mod p$ or $a = b + \eta p$. Thus, if we take the difference of the two ciphertexts, this difference must be ηp. We could then run the decryption protocol to test that the difference is indeed a multiple of p.

If $a \equiv \sum_i b_i \mod p$, simply releasing the difference could still reveal extra information about the value of a. Therefore, all parties must each add a random mask to a. In step 3, r_i's are generated independently by all parties, which means that there must be at least one honest party who is indeed generating a random number within the range. The resulting plaintext thus statistically hides the true value of $a - \sum_i b_i$ with the statistical parameter κ. If $a \not\equiv \sum_i b_i \mod p$, then the protocol reveals the difference between $a - \sum_i b_i \mod p$. This is safe because the only way to reveal $a - \sum_i b_i \mod p$ is when an adversary misbehaves and alters its inputs, and the result is independent from the honest party's behavior.

2) Weight vector decryption: Once all SPDZ values are verified, all parties jointly decrypt $\mathbf{z}$ and the final weights are released to everyone.

VII. Extensions to Other Models

Though we used LASSO as a running example, our techniques can be applied to other linear models like ordinary least-squares linear regression, ridge regression, and elastic net. Here we show the update rules for ridge regression, and leave its derivation to the readers.

Ridge regression solves a similar problem as LASSO, except with L^2 regularization. Given a dataset $(\mathbf{X}, \mathbf{y})$ where $\mathbf{X}$

is the feature matrix and $\mathbf{y}$ is the prediction vector, ridge regression optimizes $\arg\min_{\mathbf{w}} \frac{1}{2}\|\mathbf{X}\mathbf{w} - \mathbf{y}\|_2^2 + \lambda\|\mathbf{w}\|_2$. The update equations for ridge regression are:

$$\begin{aligned}\mathbf{w}_i^{k+1} &= (\mathbf{X}_i^T\mathbf{X}_i + \rho I)^{-1}(\mathbf{X}_i^T\mathbf{y}_i + \rho(\mathbf{z}^k - \mathbf{u}_i^k)) \\ &\quad + (\rho/2)\|\mathbf{w}_i - \mathbf{z}^k + \mathbf{u}_i^k\|_2^2 \\ \mathbf{z}^{k+1} &= \frac{\rho}{2\lambda/m + \rho}(\bar{\mathbf{w}}^{k+1} + \bar{\mathbf{u}}^k) \\ \mathbf{u}_i^{k+1} &= \mathbf{u}_i^k + \mathbf{x}_i^{k+1} - \mathbf{z}^{k+1}\end{aligned}$$

The local update is similar to LASSO, while the coordination update is a linear operation instead of the soft threshold function. Elastic net, which combines L^1 and L^2 regularization, can therefore be implemented by combining the regularization terms from LASSO and ridge regression.

VIII. Evaluation

We implemented Helen in C++. We utilize the SPDZ library [1], a mature library for maliciously secure multi-party computation, for both the baseline and Helen. In our implementation, we apply the Fiat-Shamir heuristic to our zero-knowledge proofs [32]. This technique is commonly used in implementations because it makes the protocols non-interactive and thus more efficient, but assumes the random oracle model.

We compare Helen's performance to a maliciously secure baseline that trains using stochastic gradient descent, similar to SecureML [54]. Since SecureML only supports two parties in the semihonest setting, we implemented a similar baseline using SPDZ [27]. SecureML had a number of optimizations, but they were designed for the two-party setting. We did not extend those optimizations to the multi-party setting. We will refer to SGD implemented in SPDZ as the "secure baseline" (we explain more about the SGD training process in Section VIII-A). Finally, we do not benchmark Helen's Paillier key setup phase. This can be computed using SPDZ itself, and it is ran only once (as long as the party configuration does not change).

A. Experiment setup

We ran our experiments on EC2 using r4.8xlarge instances. Each machine has 32 cores and 244 GiB of memory. In order to simulate a wide area network setting, we created EC2 instances in Oregon and Northern Virginia. The instances are equally split across these two regions. To evaluate Helen's scalability, we used synthetic datasets that are constructed by drawing samples from a noisy normal distribution. For these datasets, we varied both the dimension and the number of parties. To evaluate Helen's performance against the secure baseline, we benchmarked both systems on two real world datasets from UCI [29].

Training assumptions. We do not tackle hyperparameter tuning in our work, and also assume that the data has been normalized before training. We also use a fixed number of rounds (10) for ADMM training, which we found experimentally using the real world datasets. We found that 10 rounds is often enough for the training process to converge to a reasonable error rate. Recall that ADMM converges in a small number of rounds because it iterates on a summary of the *entire dataset*. In contrast, SGD iteratively scans data from all parties at least once in order to get an accurate representation of the underlying distributions. This is especially important when certain features occur rarely in a dataset. Since the dataset is very large, even one pass already results in many rounds.

MPC configuration. As mentioned earlier, SPDZ has two phases of computation: an offline phase and an online phase. The offline phase can run independently of the secure function, but the precomputed values cannot be reused across multiple online phases. The SPDZ library provides several ways of benchmarking different offline phases, including MASCOT [46] and Overdrive [47]. We tested both schemes and found Overdrive to perform better over the wide area network. Since these are for benchmarking purposes only, we decided to estimate the SPDZ offline phase by dividing the number of triplets needed for a circuit by the benchmarked throughput. The rest of the evaluation section will use the estimated numbers for all SPDZ offline computation. Since Helen uses parallelism, we also utilized parallelism in the SPDZ offline generation by matching the number of threads on each machine to the number of cores available.

On the other hand, the SPDZ online implementation is not parallelized because the API was insufficient to effectively express parallelism. We note two points. First, while parallelizing the SPDZ library will result in a faster baseline, Helen also utilizes SPDZ, so any improvement to SPDZ also carries over to Helen. Second, as shown below, our evaluation shows that Helen still achieves significant performance gains over the baseline even if the online phase in the secure baseline is infinitely fast.

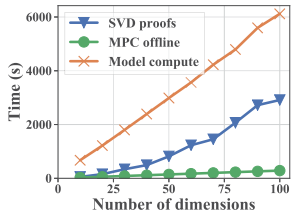
Finally, the parameters we use for Helen are: 128 bits for the secure baseline's SPDZ configuration, 256 bits for the Helen SPDZ configuration, and 4096 bits for Helen's Paillier ciphertext.

B. Theoretic performance

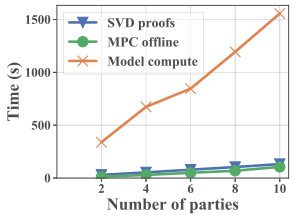
Baseline	Secure SGD	$C \cdot m^2 \cdot n \cdot d$
Helen	SVD decomposition	$c_1 \cdot n \cdot d^2$
	SVD proofs	$c_1 \cdot m \cdot d^2 + c_2 \cdot d^3$
	MPC offline	$c_1 \cdot m^2 \cdot d$
	Model compute	$c_1 \cdot m^2 \cdot d + c_2 \cdot d^2 + c_3 \cdot m \cdot d$

TABLE I: Theoretical scaling (complexity analysis) for SGD baseline and Helen. m is the number of parties, n is the number of samples per party, d is the dimension.

Table I shows the theoretic scaling behavior for SGD and Helen, where m is the number of parties, n is the number of samples per party, d is the dimension, and C and c_i are constants. Note that c_i's are not necessarily the same across the different rows in the table. We split Helen's input preparation phase into three sub-components: SVD (calculated in plaintext), SVD proofs, and MPC offline (since Helen uses SPDZ during the model compute phase, we also need to run the SPDZ offline phase).



(a) Helen's scaling as we increase the number of dimensions. The number of parties is fixed to be 4, and the number of samples per party is $100,000$.



(b) Helen's two phases as we increase the number of parties. The dimension is set to be 10, and the number of samples per party is $100,000$.

Fig. 3: Helen scalability measurements.

SGD scales linearly in n and d. If the number of samples per party is doubled, the number of iterations is also doubled. A similar argument goes for d. SGD scales quadratic in m because it first scales linearly in m due to the behavior of the MPC protocol. If we add more parties to the computation, the number of samples will also increase, which in turn increases the number of iterations needed to scan the entire dataset.

Helen, on the other hand, scales linearly in n only for the SVD computation. We emphasize that SVD is very fast because it is *executed on plaintext data*. The c_1 part of the SVD proofs formula scales linearly in m because each party needs to verify from every other party. It also scales linearly in d^2 because each proof verification requires d^2 work. The c_2 part of the formula has d^3 scaling because our matrices are $d \times d$), and to calculate a resulting encrypted matrix requires matrix multiplication on two $d \times d$ matrices.

The coordination phase from Helen's model compute phase, as well as the corresponding MPC offline compute phase, scale quadratic in m because we need to use MPC to re-scale weight vectors from each party. This cost corresponds to the c_1 part of the formula. The model compute phase's d^2 cost (c_2 part of the formula) reflects the matrix multiplication and the proofs. The rest of the MPC conversion proofs scale linearly in m and d (c_3 part of the formula).

C. Synthetic datasets

We want to answer two questions about Helen's scalability using synthetic datasets: how does Helen scale as we vary the number of features and how does it scale as we vary the number of parties? Note that we are not varying the number of input samples because that will be explored in Section VIII-D in comparison to the secure SGD baseline.

Fig. 3a shows a breakdown of Helen's cryptographic computation as we scale the number of dimensions. The plaintext SVD computation is not included in the graph. The SVD proofs phase is dominated by the matrix multiplication proofs, which scales in d^2. The MPC offline phase and the model compute phase are both dominated by the linear scaling in d, which corresponds to the MPC conversion proofs.

Fig. 3b shows the same three phases as we increase the number of parties. The SVD proofs phase scales linearly in the number of parties m. The MPC offline phase scales quadratic in m, but its effects are not very visible for a small number of parties. The model compute phase is dominated by the linear scaling in m because the quadratic scaling factor isn't very visible for a small number of parties.

Finally, we also ran a microbenchmark to understand Helen's network and compute costs. The experiment used 4 servers and a synthetic dataset with 50 features and 100K samples per party. We found that the network costs account for approximately 2% of the input preparation phase and 22% of Helen's model compute phase.

D. Real world datasets

We evaluate on two different real world datasets: gas sensor data [29] and the million song dataset [9, 29]. The gas sensor dataset records 16 sensor readings when mixing two types of gases. Since the two gases are mixed with random concentration levels, the two regression variables are independent and we can simply run two different regression problems (one for each gas type). For the purpose of benchmarking, we ran an experiment using the ethylene data in the first dataset. The million song dataset is used for predicting a song's published year using 90 features. Since regression problems produce real values, the year can be calculated by rounding the regressed value.

For SGD, we set the batch size to be the same size as the dimension of the dataset. The number of iterations is equal to the total number of sample points divided by the batch size. Unfortunately, we had to extrapolate the runtimes for a majority of the baseline online phases because the circuits were too big to compile on our EC2 instances.

Fig. 4 and Fig. 5 compare Helen to the baseline on the two datasets. Note that Helen's input preparation graph combines the three phases that are run during the offline: plaintext SVD computation, SVD proofs, and MPC offline generation. We can see that Helen's input preparation phase scales very slowly with the number of samples. The scaling actually comes from the plaintext SVD calculation because both the SVD proofs and the MPC offline generation do not scale with the number of samples. Helen's model compute phase also stays constant because we fixed the number of iterations to a conservative estimate. SGD, on the other hand, does scale linearly with the number of samples in both the offline and the online phases.

For the gas sensor dataset, Helen's total runtime (input preparation plus model compute) is able to achieve a 21.5x performance gain over the baseline's total runtime (offline plus online) when the number of samples is 1000. When the number of samples per party reaches 1 million, Helen is able to improve over the baseline by 20689x. For the song prediction dataset, Helen is able to have a 9.1x performance gain over the baseline when the number of samples is 1000. When the number of samples per party reaches 100K, Helen improves over the baseline by 911x. Even if we compare Helen to the baseline's offline phase only, we find that Helen still has close to constant scaling while the baseline scales linearly with the number of samples. The performance improvement compared

Samples per party	2000	4000	6000	8000	10K	40K	100K	200K	400K	800K	1M
sklearn L2 error	8937.01	8928.32	8933.64	8932.97	8929.10	8974.15	8981.24	8984.64	8982.88	8981.11	8980.35
Helen L2 error	8841.33	8839.96	8828.18	8839.56	8837.59	8844.31	8876.00	8901.84	8907.38	8904.11	8900.37
sklearn MAE	57.89	58.07	58.04	58.10	58.05	58.34	58.48	58.55	58.58	58.56	58.57
Helen MAE	57.23	57.44	57.46	57.44	57.47	57.63	58.25	58.38	58.36	58.37	58.40

TABLE II: Select errors for gas sensor (due to space), comparing Helen with a baseline that uses sklearn to train on all plaintext data. L2 error is the squared norm; MAE is the mean average error. Errors are calculated after post-processing.

Samples per party	1000	2000	4000	6000	8000	10K	20K	40K	60K	80K	100K
sklearn L2 error	92.43	91.67	90.98	90.9	90.76	90.72	90.63	90.57	90.55	90.56	90.55
Helen L2 error	93.68	91.8	91.01	90.91	90.72	90.73	90.67	90.57	90.54	90.57	90.55
sklearn MAE	6.86	6.81	6.77	6.78	6.79	6.81	6.80	6.79	6.79	6.80	6.80
Helen MAE	6.92	6.82	6.77	6.78	6.79	6.81	6.80	6.79	6.80	6.80	6.80

TABLE III: Errors for song prediction, comparing Helen with a baseline that uses sklearn to train on all plaintext data. L2 error is the squared norm; MAE is the mean average error. Errors are calculated after post-processing.

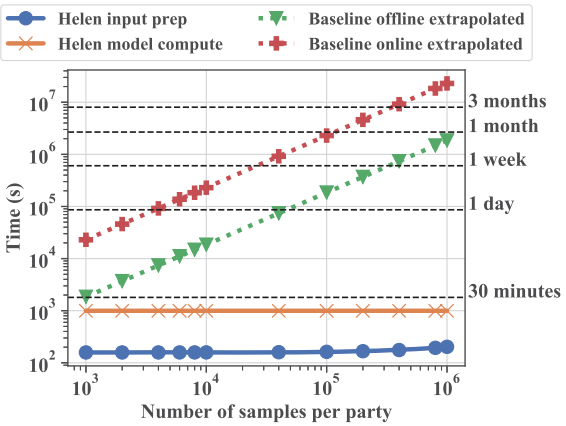


Fig. 4: Helen and baseline performance on the gas sensor data. The gas sensor data contained over 4 million data points; we partitioned into 4 partitions with varying number of sample points per partition to simulate the varying number of samples per party. The number of parties is 4, and the number of dimensions is 16.

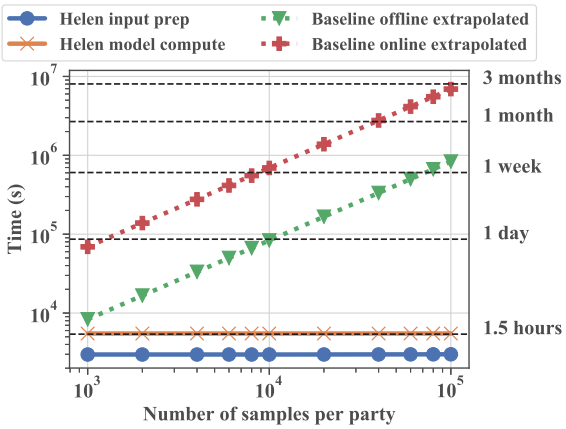


Fig. 5: Helen and baseline performance on the song prediction data, as we vary the number of samples per party. The number of parties is 4, and the number of dimensions is 90.

Fig. 6: Helen comparison with SGD

to the baseline offline phase is up to 1540x for the gas sensor dataset and up to 98x for the song prediction dataset.

In Table II and Table III, we evaluate Helen's test errors on the two datasets. We compare the L2 and mean average error for Helen to the errors obtained from a model trained using sklearn (a standard Python library for machine learning) on the plaintext data. We did not directly use the SGD baseline because its online phase does not compile for larger instances, and using sklearn on the plaintext data is a conservative estimate. We can see that Helen achieves similar errors compared to the sklearn baseline.

IX. Related work

We organize the related work section into related coopetitive systems and attacks.

A. Coopetitive systems

Coopetitive training systems. In Fig. 7, we compare Helen to prior coopetitive training systems [56, 41, 34, 20, 35, 5, 54, 66]. The main takeaway is that, excluding *generic* maliciously secure MPC, prior training systems do not provide malicious security. Furthermore, most of them also assume that the training process requires outsourcing to two non-colluding servers. At the same time, and as a result of choosing a weaker security model, some of these systems provide richer functionality than Helen, such as support for neural networks. As part of our future work, we are exploring how to apply Helen's techniques to logistic regression and neural networks.

Other coopetitive systems. Other than coopetitive training systems, there are prior works on building coopetitive systems for applications like machine learning prediction and SQL analytics. Coopetitive prediction systems [13, 62, 60, 51, 36, 45] typically consist of two parties, where one party holds a model and the other party holds an input. The two parties jointly

Work	Functionality	n-party?	Maliciously secure?	Practical?
Nikolaenko et al. [56]	ridge regression	no	no	–
Hall et al. [41]	linear regression	**yes**	no	–
Gascon et al. [34]	linear regression	no	no	–
Cock et al. [20]	linear regression	no	no	–
Giacomelli et al. [35]	ridge regression	no	no	–
Alexandru et al. [5]	quadratic opt.	no	no	–
SecureML [54]	linear, logistic, deep learning	no	no	–
Shokri&Shmatikov [66]	deep learning	not MPC (heuristic)	no	–
Semi-honest MPC [7]	any function	**yes**	no	–
Malicious MPC [27, 37, 11, 2]	any function	**yes**	**yes**	no
Our proposal, Helen: regularized linear models		**yes**	**yes**	**yes**

Fig. 7: **Insufficiency of existing cryptographic approaches.** "n-party" refers to whether the $n(>2)$ organizations can perform the computation with *equal trust* (thus not including the two non-colluding servers model). We answer the practicality question only for maliciously-secure systems. We note that a few works that we marked as not coopetitive and not maliciously secure discuss at a high level how one might extend their work to such a setting, but they did not flesh out designs or evaluate their proposals.

compute a prediction without revealing the input or the model to the other party. Coopetitive analytics systems [6, 55, 12, 21, 10] allow multiple parties to run SQL queries over all parties' data. These computation frameworks do not directly translate to Helen's training workloads. Most of these works also do not address the malicious setting.

Trusted hardware based systems. The related work presented in the previous two sections all utilize purely software based solutions. Another possible approach is to use trusted hardware [53, 22], and there are various secure distributed systems that could be extended to the coopetitive setting [64, 42, 71]. However, these hardware mechanisms require additional trust and are prone to side-channel leakages [49, 68, 50].

B. Attacks on machine learning

Machine learning attacks can be categorized into data poisoning, model leakage, parameter stealing, and adversarial learning. As mentioned in §III-A, Helen tackles the problem of cryptographically running the training algorithm without sharing datasets amongst the parties involved, while defenses against these attacks are *orthogonal* and complementary to our goal in this paper. Often, these machine learning attacks can be separately addressed outside of Helen. We briefly discuss two relevant attacks related to the training stage and some methods for mitigating them.

Poisoning. Data poisoning allows an attacker to inject poisoned inputs into a dataset before training [44, 18]. Generally, malicious MPC does not prevent an attacker from choosing incorrect initial inputs because there is no way to enforce this requirement. Nevertheless, there are some ways of mitigating arbitrary poisoning of data that would complement Helen's training approach. Before training, one can check that the inputs are confined within certain intervals. The training process itself can also execute *cross validation*, a process that can identify parties that do not contribute useful data. After training, it is possible to further post process the model via techniques like fine tuning and parameter pruning [52].

Model leakage. Model leakage [65, 16] is an attack launched by an adversary who tries to infer information about the training data from the model itself. Again, malicious MPC does not prevent an attacker from learning the final result. In our coopetitive model, we also assume that all parties want to cooperate and have agreed to release the final model to everyone. One way to alleviate model leakage is through the use of differential privacy [43, 4, 31]. For example, a simple technique that is complementary to Helen is adding carefully chosen noise directly to the output model [43].

X. Conclusion

In this paper, we propose Helen, a coopetitive system for training linear models. Compared to prior work, Helen assumes a stronger threat model by defending against *malicious* participants. This means that each party only needs to trust itself. Compared to a baseline implemented with a state-of-the-art malicious framework, Helen is able to achieve up to five orders of magnitude of performance improvement. Given the lack of efficient maliciously secure training protocols, we hope that our work on Helen will lead to further work on efficient systems with such strong security guarantees.

XI. Acknowledgment

We thank the anonymous reviewers for their valuable reviews, as well as Shivaram Venkataraman, Stephen Tu, and Akshayaram Srinivasan for their feedback and discussions. This research was supported by NSF CISE Expeditions Award CCF-1730628, as well as gifts from the Sloan Foundation, Hellman Fellows Fund, Alibaba, Amazon Web Services, Ant Financial, Arm, Capital One, Ericsson, Facebook, Google, Huawei, Intel, Microsoft, Scotiabank, Splunk and VMware.

REFERENCES

[1] bristolcrypto/spdz-2: Multiparty computation with SPDZ, MASCOT, and Overdrive offline phases. https://github.com/bristolcrypto/SPDZ-2. Accessed: 2018-10-31.

[2] VIFF, the Virtual Ideal Functionality Framework. http://viff.dk/, 2015.

[3] Health insurance portability and accountability act, April 2000.

[4] ABADI, M., CHU, A., GOODFELLOW, I., MCMAHAN, H. B., MIRONOV, I., TALWAR, K., AND ZHANG, L. Deep learning with differential privacy. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security* (2016), ACM, pp. 308–318.

[5] ALEXANDRU, A. B., GATSIS, K., SHOUKRY, Y., SESHIA, S. A., TABUADA, P., AND PAPPAS, G. J. Cloud-based quadratic optimization with partially homomorphic encryption. *arXiv preprint arXiv:1809.02267* (2018).

[6] BATER, J., ELLIOTT, G., EGGEN, C., GOEL, S., KHO, A., AND ROGERS, J. Smcql: secure querying for federated databases. *Proceedings of the VLDB Endowment 10*, 6 (2017), 673–684.

[7] BEN-DAVID, A., NISAN, N., AND PINKAS, B. Fairplaymp: a system for secure multi-party computation. www.cs.huji.ac.il/project/Fairplay/FairplayMP.html, 2008.

[8] BEN-OR, M., GOLDWASSER, S., AND WIGDERSON, A. Completeness theorems for non-cryptographic fault-tolerant distributed computation. In *Proceedings of the twentieth annual ACM symposium on Theory of computing* (1988), ACM, pp. 1–10.

[9] BERTIN-MAHIEUX, T., ELLIS, D. P., WHITMAN, B., AND LAMERE, P. The million song dataset. In *Ismir* (2011), vol. 2, p. 10.

[10] BITTAU, A., ERLINGSSON, U., MANIATIS, P., MIRONOV, I., RAGHUNATHAN, A., LIE, D., RUDOMINER, M., KODE, U., TINNES, J., AND SEEFELD, B. Prochlo: Strong privacy for analytics in the crowd. In *Proceedings of the 26th Symposium on Operating Systems Principles* (2017), ACM, pp. 441–459.

[11] BOGDANOV, D., LAUR, S., AND WILLEMSON, J. *Sharemind: A Framework for Fast Privacy-Preserving Computations*. 2008.

[12] BONAWITZ, K., IVANOV, V., KREUTER, B., MARCEDONE, A., MCMAHAN, H. B., PATEL, S., RAMAGE, D., SEGAL, A., AND SETH, K. Practical secure aggregation for privacy-preserving machine learning. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security* (2017), CCS '17.

[13] BOST, R., POPA, R. A., TU, S., AND GOLDWASSER, S. Machine learning classification over encrypted data. In *Network and Distributed System Security Symposium (NDSS)* (2015).

[14] BOUDOT, F. Efficient proofs that a committed number lies in an interval. In *International Conference on the Theory and Applications of Cryptographic Techniques* (2000), Springer, pp. 431–444.

[15] BOYD, S., PARIKH, N., CHU, E., PELEATO, B., AND ECKSTEIN, J. Distributed optimization and statistical learning via the alternating direction method of multipliers. In *Foundations and Trends in Machine Learning, Vol. 3, No. 1* (2010).

[16] CARLINI, N., LIU, C., KOS, J., ERLINGSSON, Ú., AND SONG, D. The secret sharer: Measuring unintended neural network memorization & extracting secrets. *arXiv preprint arXiv:1802.08232* (2018).

[17] CHEN, H., AND XIANG, Y. The study of credit scoring model based on group lasso. *Procedia Computer Science 122* (2017), 677 – 684. 5th International Conference on Information Technology and Quantitative Management, ITQM 2017.

[18] CHEN, X., LIU, C., LI, B., LU, K., AND SONG, D. Targeted backdoor attacks on deep learning systems using data poisoning. *arXiv preprint arXiv:1712.05526* (2017).

[19] CLEVE, R. Limits on the security of coin flips when half the processors are faulty. In *Proceedings of the eighteenth annual ACM symposium on Theory of computing* (1986), ACM, pp. 364–369.

[20] COCK, M. D., DOWSLEY, R., NASCIMENTO, A. C., AND NEWMAN, S. C. Fast, privacy preserving linear regression over distributed datasets based on pre-distributed data. In *Proceedings of the 8th ACM Workshop on Artificial Intelligence and Security (AISec)* (2015).

[21] CORRIGAN-GIBBS, H., AND BONEH, D. Prio: Private, robust, and scalable computation of aggregate statistics. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)* (2017).

[22] COSTAN, V., AND DEVADAS, S. Intel sgx explained. *IACR Cryptology ePrint Archive 2016* (2016), 86.

[23] CRAMER, R., DAMGÅRD, I., AND NIELSEN, J. Multiparty computation from threshold homomorphic encryption. *EUROCRYPT 2001* (2001), 280–300.

[24] DAMGÅRD, I. Efficient concurrent zero-knowledge in the auxiliary string model. In *International Conference on the Theory and Applications of Cryptographic Techniques* (2000), Springer, pp. 418–430.

[25] DAMGÅRD, I. On σ-protocols. *Lecture Notes, University of Aarhus, Department for Computer Science* (2002).

[26] DAMGÅRD, I., AND JURIK, M. Client/server tradeoffs for online elections. In *International Workshop on Public Key Cryptography* (2002), Springer, pp. 125–140.

[27] DAMGÅRD, I., PASTRO, V., SMART, N., AND ZAKARIAS, S. Multiparty computation from somewhat homomorphic encryption. In *Advances in Cryptology–CRYPTO 2012*. Springer, 2012, pp. 643–662.

[28] D'ANGELO, G. M., RAO, D. C., AND GU, C. C. Combining least absolute shrinkage and selection operator (lasso) and principal-components analysis for detection of gene-gene interactions in genome-wide association studies. In *BMC proceedings* (2009).

[29] DHEERU, D., AND KARRA TANISKIDOU, E. UCI machine learning repository, 2017.

[30] DICTIONARIES, E. O. Coopetition.

[31] DUCHI, J. C., JORDAN, M. I., AND WAINWRIGHT, M. J. Local privacy, data processing inequalities, and statistical minimax rates. *arXiv preprint arXiv:1302.3203* (2013).

[32] FAUST, S., KOHLWEISS, M., MARSON, G. A., AND VENTURI, D. On the non-malleability of the fiat-shamir transform. In *International Conference on Cryptology in India* (2012), Springer, pp. 60–79.

[33] GARAY, J. A., MACKENZIE, P., AND YANG, K. Strengthening zero-knowledge protocols using signatures. In *Eurocrypt* (2003), vol. 2656, Springer, pp. 177–194.

[34] GASCN, A., SCHOPPMANN, P., BALLE, B., RAYKOVA, M., DOERNER, J., ZAHUR, S., AND EVANS, D. Privacy-preserving distributed linear regression on high-dimensional data. Cryptology ePrint Archive, Report 2016/892, 2016.

[35] GIACOMELLI, I., JHA, S., JOYE, M., PAGE, C. D., AND YOON, K. Privacy-preserving ridge regression with only linearly-homomorphic encryption. Cryptology ePrint Archive, Report 2017/979, 2017. https://eprint.iacr.org/2017/979.

[36] GILAD-BACHRACH, R., DOWLIN, N., LAINE, K., LAUTER, K., NAEHRIG, M., AND WERNSING, J. Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy. In *International Conference on Machine Learning* (2016), pp. 201–210.

[37] GOLDREICH, O., MICALI, S., AND WIGDERSON, A. How to play any mental game. In *Proceedings of the nineteenth annual ACM symposium on Theory of computing* (1987), ACM, pp. 218–229.

[38] GOLUB, G. H., AND VAN LOAN, C. F. *Matrix computations*, vol. 3. JHU Press, 2012.

[39] GROTH, J. Homomorphic trapdoor commitments to group elements. *IACR Cryptology ePrint Archive 2009* (2009), 7.

[40] HALEVY, A., NORVIG, P., AND PEREIRA, F. The unreasonable effectiveness of data. *IEEE Intelligent Systems 24*, 2 (Mar. 2009), 8–12.

[41] HALL, R., FIENBERG, S. E., AND NARDI, Y. Secure multiple linear regression based on homomorphic encryption. In *Journal of Official Statistics* (2011).

[42] HUNT, T., ZHU, Z., XU, Y., PETER, S., AND WITCHEL, E. Ryoan: A distributed sandbox for untrusted computation on secret data. In *OSDI* (2016), pp. 533–549.

[43] IYENGAR, R., NEAR, J. P., SONG, D., THAKKAR, O., THAKURTA, A., AND WANG, L. Towards practical differentially private convex optimization. In *2019 IEEE Symposium on Security and Privacy (SP)*, IEEE.

[44] JAGIELSKI, M., OPREA, A., BIGGIO, B., LIU, C., NITA-ROTARU, C., AND LI, B. Manipulating machine learning: Poisoning attacks and countermeasures for regression learning. *arXiv preprint arXiv:1804.00308* (2018).

[45] JUVEKAR, C., VAIKUNTANATHAN, V., AND CHANDRAKASAN, A. Gazelle: A low latency framework for secure neural network inference. *CoRR abs/1801.05507* (2018).

[46] KELLER, M., ORSINI, E., AND SCHOLL, P. Mascot: faster malicious arithmetic secure computation with oblivious transfer. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security* (2016), ACM, pp. 830–842.

[47] KELLER, M., PASTRO, V., AND ROTARU, D. Overdrive: making spdz great again. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques* (2018), Springer, pp. 158–189.

[48] KIDD, A. C., MCGETTRICK, M., TSIM, S., HALLIGAN, D. L., BYLESJO, M., AND BLYTH, K. G. Survival prediction in mesothelioma

using a scalable lasso regression model: instructions for use and initial performance using clinical predictors. *BMJ Open Respiratory Research 5*, 1 (2018).

[49] KOCHER, P., GENKIN, D., GRUSS, D., HAAS, W., HAMBURG, M., LIPP, M., MANGARD, S., PRESCHER, T., SCHWARZ, M., AND YAROM, Y. Spectre attacks: Exploiting speculative execution. *arXiv preprint arXiv:1801.01203* (2018).

[50] LEE, S., SHIH, M.-W., GERA, P., KIM, T., KIM, H., AND PEINADO, M. Inferring fine-grained control flow inside sgx enclaves with branch shadowing. In *26th USENIX Security Symposium, USENIX Security* (2017), pp. 16–18.

[51] LIU, J., JUUTI, M., LU, Y., AND ASOKAN, N. Oblivious neural network predictions via minionn transformations. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security* (2017), ACM, pp. 619–631.

[52] LIU, K., DOLAN-GAVITT, B., AND GARG, S. Fine-pruning: Defending against backdooring attacks on deep neural networks. *arXiv preprint arXiv:1805.12185* (2018).

[53] MCKEEN, F., ALEXANDROVICH, I., BERENZON, A., ROZAS, C. V., SHAFI, H., SHANBHOGUE, V., AND SAVAGAONKAR, U. R. Innovative instructions and software model for isolated execution. *HASP@ ISCA 10* (2013).

[54] MOHASSEL, P., AND ZHANG, Y. Secureml: A system for scalable privacy-preserving machine learning. *IACR Cryptology ePrint Archive 2017* (2017), 396.

[55] NARAYAN, A., AND HAEBERLEN, A. Djoin: Differentially private join queries over distributed databases. In *Proceedings of the 10th USENIX Conference on Operating Systems Design and Implementation* (2012), OSDI'12.

[56] NIKOLAENKO, V., WEINSBERG, U., IOANNIDIS, S., JOYE, M., BONEH, D., AND TAFT, N. Privacy-preserving ridge regression on hundreds of millions of records. In *Security and Privacy (SP), 2013 IEEE Symposium on* (2013), IEEE, pp. 334–348.

[57] NIKOLAENKO, V., WEINSBERG, U., IOANNIDIS, S., JOYE, M., BONEH, D., AND TAFT, N. Privacy-preserving ridge regression on hundreds of millions of records. In *Security and Privacy (SP), 2013 IEEE Symposium on* (2013), IEEE, pp. 334–348.

[69] WANG, X., RANELLUCCI, S., AND KATZ, J. Global-scale secure multiparty computation. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security* (2017), ACM, pp. 39–56.

[58] PAILLIER, P. Public-key cryptosystems based on composite degree residuosity classes. In *EUROCRYPT* (1999), pp. 223–238.

[59] PAPACHRISTOU, C., OBER, C., AND ABNEY, M. A lasso penalized regression approach for genome-wide association analyses using related individuals: application to the genetic analysis workshop 19 simulated data. *BMC Proceedings 10*, 7 (Oct 2016), 53.

[60] RIAZI, M. S., WEINERT, C., TKACHENKO, O., SONGHORI, E. M., SCHNEIDER, T., AND KOUSHANFAR, F. Chameleon: A hybrid secure computation framework for machine learning applications. Cryptology ePrint Archive, Report 2017/1164, 2017. https://eprint.iacr.org/2017/1164.

[61] ROBBINS, H., AND MONRO, S. A stochastic approximation method. In *Herbert Robbins Selected Papers*. Springer, 1985, pp. 102–109.

[62] ROUHANI, B. D., RIAZI, M. S., AND KOUSHANFAR, F. Deepsecure: Scalable provably-secure deep learning. *CoRR abs/1705.08963* (2017).

[63] ROY, S., MITTAL, D., BASU, A., AND ABRAHAM, A. Stock market forecasting using lasso linear regression model, 01 2015.

[64] SCHUSTER, F., COSTA, M., FOURNET, C., GKANTSIDIS, C., PEINADO, M., MAINAR-RUIZ, G., AND RUSSINOVICH, M. Vc3: Trustworthy data analytics in the cloud using sgx. In *Security and Privacy (SP), 2015 IEEE Symposium on* (2015), IEEE, pp. 38–54.

[65] SHMATIKOV, V., AND SONG, C. What are machine learning models hiding?

[66] SHOKRI, R., AND SHMATIKOV, V. Privacy-preserving deep learning. In *CCS* (2015).

[67] STOICA, I., SONG, D., POPA, R. A., PATTERSON, D., MAHONEY, M. W., KATZ, R., JOSEPH, A. D., JORDAN, M., HELLERSTEIN, J. M., GONZALEZ, J. E., ET AL. A berkeley view of systems challenges for ai. *arXiv preprint arXiv:1712.05855* (2017).

[68] VAN BULCK, J., MINKIN, M., WEISSE, O., GENKIN, D., KASIKCI, B., PIESSENS, F., SILBERSTEIN, M., WENISCH, T. F., YAROM, Y., AND STRACKX, R. Foreshadow: Extracting the keys to the intel sgx kingdom with transient out-of-order execution. In *Proceedings of the 27th USENIX Security Symposium. USENIX Association* (2018).

[70] YAO, A. C. Protocols for secure computations. In *Foundations of Computer Science, 1982. SFCS'08. 23rd Annual Symposium on* (1982), IEEE, pp. 160–164.

[71] ZHENG, W., DAVE, A., BEEKMAN, J. G., POPA, R. A., GONZALEZ, J. E., AND STOICA, I. Opaque: An oblivious and encrypted distributed analytics platform. In *USENIX Symposium of Networked Systems Design and Implementation (NDSI)* (2017), pp. 283–298.