Protecting Privacy of Users in Brain-Computer Interface Applications

Anisha Agarwal, Rafael Dowsley, Nicholas D. McKinney, Dongrui Wu, Chin-Teng Lin, Martine De Cock, and Anderson C. A. Nascimento

***Abstract*—Machine learning (ML) is revolutionizing research and industry. Many ML applications rely on the use of large amounts of personal data for training and inference. Among the most intimate exploited data sources is electroencephalogram (EEG) data, a kind of data that is so rich with information that application developers can easily gain knowledge beyond the professed scope from unprotected EEG signals, including passwords, ATM PINs, and other intimate data. The challenge we address is how to engage in meaningful ML with EEG data while protecting the privacy of users.**

Hence, we propose cryptographic protocols based on Secure Multiparty Computation (SMC) to perform linear regression over EEG signals from many users in a fully privacy-preserving (PP) fashion, i.e. such that each individual's EEG signals are not revealed to anyone else. To illustrate the potential of our secure framework, we show how it allows estimating the drowsiness of drivers from their EEG signals as would be possible in the unencrypted case, and at a very reasonable computational cost. Our solution is the first application of commodity-based SMC to EEG data, as well as the largest documented experiment of secret sharing based SMC in general, namely with 15 players involved in all the computations.

***Index Terms*—secure multiparty computation, cryptography, machine learning, linear regression, driver drowsiness estimation.**

I. Introduction

The application potential of Brain-Computer Interfaces (BCIs) is vast, going far beyond medicine and research into areas such as education, gaming, entertainment, wellness, and personalized marketing. The emergence of consumer-grade, low-cost BCIs and corresponding software development kits[1] is bringing the use of BCI within reach of application developers. They can capture neural signals, extract features from them, and subsequently use these extracted features to train and use machine learning (ML) models for all kinds of prediction and inference tasks. These include inferring emotions, sexual preferences and religious beliefs of individuals, detecting preferences of customers, measuring concentration, or estimating levels of drowsiness in drivers of cars [1]–[6].

While many BCI-applications can be, and are, developed with a benign intent of enriching and improving the quality of human life, giving access to a user's brain signals, or features extracted from them, can seriously harm the user's privacy. Brain spyware has for instance been used to infer users' 4-digit PINs, bank information, month of birth, location of residence, and whether they recognized a presented set of faces [7]. The impact of brain malware that can infer very intimate information about users, such as emotions, prejudices, religious and political beliefs, etc. and subsequently use that information to manipulate users, could be severe [8].

The awareness of the need for protecting the privacy of individuals and their data in ML applications has increased substantially over the last few years, as witnessed for instance in the National Privacy Research Strategy put forward by the National Science and Technology Council (Jun 2016)[2], the recommendations of the Commission on Evidence-Based Policy Making (Sep 2017)[3], and ACM's statement on preserving personal privacy (Mar 2018) [9]. Sensitive data includes user generated content on social media, patient healthcare records, genetic information, and – without a doubt – neural information such as recorded by EEG signals. There is plenty of evidence that anonymizing data does not offer sufficient protection [10]. In this paper we therefore focus on the use of cryptography, in particular Secure Multiparty Computation (SMC) [11], to ensure, in a mathematically provable way, that the EEG data of individuals used in ML applications is not revealed to anyone but themselves, while still being able to do meaningful computations over that data.

To this end, we propose cryptographic protocols for fully privacy-preserving linear regression (PPLR) with data from EEG signals, and their implementation in Lynx [12], a framework for SMC based on additive secret sharing. Our methods are applicable in any application that requires training an LR model from EEG data. In this paper, we demonstrate our

M. De Cock and A. Nascimento are with the School of Engineering and Technology, University of Washington Tacoma, WA 98402, USA, e-mail: {mdecock,andclay}@uw.edu.

R. Dowsley is with the Department of Computer Science, Bar-Ilan University, Israel. Email: rafael@dowsley.net. He is supported by the BIU Center for Research in Applied Cryptography and Cyber Security in conjunction with the Israel National Cyber Bureau in the Prime Minister's Office. This work was mostly done while he was with the Department of Computer Science, Aarhus University. He has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme under grant agreement No 669255 (MPCPRO).

A. Agarwal and N. McKinney were with the School of Engineering and Technology, University of Washington Tacoma, WA 98402, USA, e-mail: {anisha3,mckinnnd}@uw.edu.

D. Wu is with the Key Laboratory of the Ministry of Education for Image Processing and Intelligent Control, School of Automation, Huazhong University of Science and Technology, Wuhan, China, email: drwu09@gmail.com.

C.-T. Lin is with the School of Software, University of Technology, Sydney, Australia, email: Chin-Teng.Lin@uts.edu.au.

M. De Cock is a guest professor at the Department of Applied Mathematics, Computer Science, and Statistics, Ghent University, 9000 Ghent, Belgium, e-mail: martine.decock@ugent.be.

[1] E.g. https://www.emotiv.com/, http://neurosky.com/, https://myndplay.com/

[2] https://www.nitrd.gov/PUBS/NationalPrivacyResearchStrategy.pdf

[3] https://www.cep.gov/content/dam/cep/report/cep-final-report.pdf

protocols for estimating the drowsiness of drivers, which is the cause of 1000s of fatal crashes each year.[4] We consider two different scenarios. In the *first scenario*, a set of *source drivers* work together to train an LR model in a distributed fashion (many-party SMC). Throughout this process, none of the drivers can see the data from the other drivers in an unencrypted way at any point. At the end of the protocol, all source drivers hold encrypted shares of the trained model, and a *target driver* can obtain a prediction for his data by engaging in a cryptographic protocol with all of the source drivers (many-party SMC). In the *second scenario*, the target driver has calibration data that can be leveraged to train a personalized and more accurate model. The target driver engages in a separate cryptographic protocol with each of the source drivers (2-party SMC) to train LR models, namely as many models as there are source drivers. Each model is trained on data from one source driver, as well as on some of the calibration data from the target driver. As before, any individual's EEG data is not disclosed to anyone else. Furthermore, at the end of the training protocol, no single user knows any of the trained models. Instead, knowledge about the models is split into encrypted shares that are "owned" in a distributed fashion by the drivers. Finally, the target driver can obtain a prediction for his data, as an average of the predictions by all trained models, through engaging in a cryptographic protocol with all the source drivers.

Our secure framework allows to estimate the drowsiness of drivers as would be possible in the unencrypted case, and scales well with the number of drivers. It is the first application of commodity-based SMC to EEG data, as well as the largest documented experiment of secret sharing based SMC in general, with 15 players involved in all the computations.

In this paper, we focus on privacy-preserving machine learning (PPML) techniques based on SMC. There are alternative paradigms for obtaining such a goal. They differ – among other things – in how much information is leaked during training and deployment of the ML models. In this paper, we work in the most restrictive of these scenarios – where no leakage is allowed. We do not cover other approaches for PPML that leak some information, such as: differential privacy [13]; trusted enclaves [14]; federated learning [15], [16]. To the best of our knowledge, none of these approaches have previously been applied to training ML models based on EEG data either.

II. Related work

BCI technology is gradually becoming more ubiquitous. On the academic side, great progress was made in the development of technology for "mind reading" from fMRI activation patterns. Among others, this includes recent works by Wang et al. [17] who successfully trained a ridge regression model to identify complex thoughts, such as, "The witness shouted during the trial", and by Du et al. [18] who presented a method for identifying what a user is looking at, just by monitoring their brain activity. At the same time, a variety of neural engineering companies have already introduced inexpensive, consumer-grade BCI devices for measuring brain activity in the form of EEG signals, as well as so-called BCI App Stores to facilitate adoption of the BCI headsets [8], and efforts are underway to make the more informative magnetoencephalography (MEG) brain scanners wearable in practice [19].

The access that BCI applications have to neural signals rightly raises privacy concerns. A well known threat are subliminal attacks in which users are exposed to visual stimuli for a duration that is too short for cognitive perception yet long enough to learn private information about the users based on their neural reactions to the visual stimuli (e.g. brand logos) [1]. The data obtained in this way is valuable for example for phishing campaigns or ads. Neural signals have also been used to elicit information about a person's sexual orientation [3] or religious beliefs [2]. It is understood in the data science community that anonymization, i.e. removing personally identifiable information from data before release, is not sufficient to protect the privacy of individuals, since it still leaves the data vulnerable to linkage attacks [10]. True protection can come from cryptographic techniques that allow computations over encrypted data, such as Fully Homomorphic Encryption (FHE) or Secure Multiparty Computation (SMC) [11].

Multiple approaches for secure LR have been proposed in the literature. Some are not based on SMC [20]–[25], and some use SMC like we do [26]–[30]. Several existing approaches assume that the data is vertically partitioned [24], [25], [29], hence it can not be used for the application that we study in this paper, in which each user has the information about his own EEG signals (i.e., horizontally partitioned data).

Homomorphic encryption (HE) based approaches. Hall et al. [21] achieve security in a two-party LR scenario, using HE on datasets over a finite field. The truncation protocol used in [21] to scale down the finite field has a small problem which is documented in [28]. The HE based method of Aono et al. [22] outsources the computations to a server. The entire LR model is present at the server, and the client evaluates its data securely. Our approach differs from the above in various ways. Our method enables training and inference in a fully distributed fashion, i.e. such that the coefficients of the trained LR model never have to be brought together in one place. Furthermore, our method allows an arbitrary number of parties, and computations are fast (less than 6 min for training an LR model with over 16,000 training examples distributed over 14 parties, and seconds for inference). Nikolaenko proposed a hybrid model for secure LR using HE and garbled circuits [23]. While their approach does handle multiple parties, they upload encrypted data to a third party responsible for evaluating the model with the help of a semi-honest Crypto Service Provider. We eliminate the need of a third party to actively participate in the protocol while achieving better runtimes.

SMC based approaches. Du et al. [27] proposed an early approach for SMC based simple LR, i.e. when there is only a single scalar predictor variable. The method we propose in this paper works for multivariable LR, which is far more common in practice. Karr et al. [26] provided a sketch for secure LR on horizontally partitioned data. They did not address important challenges that would need to be solved when implementing it in practice, such as how to perform matrix inversion in a secure manner and how to handle datasets with real numbers.

[4]https://www.cdc.gov/features/dsdrowsydriving/index.html

Finally, Du et al.'s secure two-party approach [30] is different from ours in goal: they explore the trade-off between security requirements and efficiency based on the assumption that a dishonest party might be able to learn some information about the other party's private data. We are the first to implement cryptographic protocols for performing secret sharing based LR in which any number of parties can participate. To this end, we extend the PPLR technique by De Cock et al. [28] to m parties. While the general protocols for LR in [28] are similar to ours, the version presented in [28] was not implemented for several parties. The implementation in [28] was a simulation where all the parties were running within the same machine and did not include the delay due to the network connecting all the parties. To the best of our knowledge, our work is the largest documented experiment of privacy-preserving machine learning in terms of the number of parties. Moreover, our work includes private scoring, which was not present in [28]. This paper extends our prior work to a real application with a realistic deployment, code that is re-usable and publicly available. For other recent work on the use of SMC for PPML (other than LR) we refer to [31]–[34] and references therein.

The implementation of new SMC protocols is facilitated by libraries that provide a general framework with a built-in implementation of cryptographic protocols for basic operations – such as multiplication and comparison – that one can use to implement more complex protocols in a modular fashion. In this paper we use the SMC library Lynx [12], which is based on additive secret sharing. Other existing frameworks for SMC are Sharemind [35], FairPlay [36], and Chameleon [37]. Both Sharemind and Chameleon are secret sharing based frameworks (like Lynx) developed in C++ whereas Fairplay uses both secret sharing and garbled circuits. Sharemind uses a fixed modulus of 2^{32} and, as it stands, is limited to computations on integers for three parties. Chameleon, a two-party framework with protocols similar to that of Sharemind, supports computations on floating point numbers in addition to integers. Like Lynx, Chameleon uses a trusted third party to generate correlated randomness. Fairplay, relies on a custom function definition language to define the boolean circuits. The need to learn a custom language makes it less user friendly. Chameleon and Sharemind are limited to 2 and 3 parties respectively. Fairplay can handle more than 3 parties but doing so comes at a substantial computational cost. The Lynx library that we use in this paper (see Section V) allows participation of an arbitrary number of parties. Lynx is designed to scale well with an increasing number of parties, among other things due to the use of a bulletin board functionality that enables efficient communication among many parties who are simultaneously involved in computations. To the best of our knowledge, ours is the first documented application of secret sharing based SMC for ML with computations done by more than 3 players.

We illustrate the power of our solution by applying it to the problem of privately estimating driver's drowsiness based on EEG data. The U.S. Department of Transportation reports that drowsy driving, i.e. driving while experiencing sleepiness or fatigue, claimed 846 lives in 2014.[5] According to the Centers for Disease Control and Prevention, up to 6,000 fatal crashes each year may be caused by drowsy drivers.[6] The company *Panasonic* has announced the release of an in-car system for driver drowsiness detection, through a combination of a camera and sensors which constantly measure blinking features, facial expressions, heat loss from the body, and illuminance [38]. Depending on the detected level of tiredness, either the temperature in the car is changed (for moderate drowsiness), or an alarm is sounded (for severe drowsiness). Wu et al. [6] have successfully trained linear regression models for inferring the level of drowsiness of drivers from their EEG signals, both in a setting where a model trained with data from m source drivers is used to infer the drowsiness of a target driver, as well as in transfer learning settings where calibration data from the target driver is leveraged to personalize the predictive models, leading to more accurate drowsiness estimates. In this paper we show how regression models like those from Wu et al. [6] can be trained and used in a fully privacy-preserving (PP) way, without any loss of accuracy, and at a very reasonable computational cost. A high level sketch of our work appeared previously [39].

III. Preliminaries

In this section we introduce the notation for LR that we will adhere to in the paper, and we recall preliminaries about performing secure computations with additive secret sharings.

Throughout this paper we use capital letters such as X to denote matrices, bold face letters such as $\mathbf{y}$ to denote vectors, and regular letters such as y to denote scalar values. Let X be an $n \times k$ matrix and $\mathbf{y}$ a vector of length n as follows:

$$X = \begin{pmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \\ \dots \\ \mathbf{x}_n \end{pmatrix} \text{ and } \mathbf{y} = \begin{pmatrix} y_1 \\ y_2 \\ \dots \\ y_n \end{pmatrix} \tag{1}$$

Performing LR with X and $\mathbf{y}$ means finding a coefficient vector $\boldsymbol{\beta} = (\beta_0 \quad \beta_1 \quad \dots \quad \beta_k)$ that minimizes

$$\frac{1}{n}\sum_{i=1}^{n}((\beta_1(\mathbf{x}_i)_1 + \beta_2(\mathbf{x}_i)_2 + \ldots + \beta_k(\mathbf{x}_i)_k + \beta_0) - y_i)^2 \tag{2}$$

In a supervised ML application, X and $\mathbf{y}$ contain information about training examples, where $\mathbf{x}_i$ is the input feature vector for the ith example and y_i is the associated output. The goal is to leverage these training examples to predict the unknown outcome for a previously unseen input as accurately as possible by learning a linear function defined by the coefficient vector $\boldsymbol{\beta}$. The coefficients that minimize the mean squared error over the training examples (2) can be computed as

$$\boldsymbol{\beta} = (X^T X)^{-1} X^T \mathbf{y} \tag{3}$$

In the scenarios that we are interested in, the data needed to train the LR model is not owned by a single party but is instead distributed across multiple parties who are not willing to disclose it. In other words, each of the parties has some of the entries of the matrix X and the vector $\mathbf{y}$, and the parties

[5]https://www.nhtsa.gov/risky-driving/drowsy-driving

[6]https://www.cdc.gov/features/dsdrowsydriving/index.html

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TNSRE.2019.2926965, IEEE Transactions on Neural Systems and Rehabilitation Engineering

are unwilling or unable to send their entries to each other or to a trusted third party to perform LR over the combined data.

To efficiently train LR models over distributed data in a PP way, we work in the commodity-based model [40]. In this approach, there is a setup assumption about the existence of a Trusted Initializer (TI) that pre-distributes correlated random numbers during an initialization phase (which can happen far before the ML models are trained, even before knowing the training data) to the parties participating in the protocol. The TI is not involved in any other part of the execution and does not learn any input from the parties. The main advantage of the commodity-based approach is that it enables very efficient solutions with unconditional security. It has been used in the context of PPML [28], [34], [41], [42], as well as in other applications [43]–[49].

Throughout this paper, we perform secure computations using additive secret sharings over a finite field $\mathbb{F}_q$. A value (number) $x \in \mathbb{F}_q$ is secret shared between parties $p_1, \ldots, p_m$ by picking $x_{p_1}, \ldots, x_{p_m} \in \mathbb{F}_q$ uniformly at random subject to the constraint that $x = \sum_{i=1}^{m} x_{p_i} \mod q$, and then revealing x_i to p_i. This secret sharing will be denoted by $[\![x]\!]_q$. Notice that from the point of view of any *proper subset* of parties, no information about x is revealed by the combination of their shares. A secret shared value can be revealed to one of the parties by sending him the shares of *all* the other parties.

Given secret sharings $[\![x]\!]_q$, $[\![y]\!]_q$ and a constant c, it is trivial for the parties to compute secret sharings corresponding to $z = x + y$, $z = x - y$, $z = cx$, or $z = x + c$ by performing addition, subtraction, etc, locally on the shares; the parties do not even need to communicate with each other to this end. These operations are respectively denoted by $[\![z]\!]_q \leftarrow [\![x]\!]_q + [\![y]\!]_q$, $[\![z]\!]_q \leftarrow [\![x]\!]_q - [\![y]\!]_q$, $[\![z]\!]_q \leftarrow c[\![x]\!]_q$, and $[\![z]\!]_q \leftarrow [\![x]\!]_q + c$. In the commodity-based model there is also a well-known protocol π_{DM} to multiply the values of two secret sharings [50], which has been generalized to a secure distributed matrix multiplication protocol π_{DMM}. At the start of the protocol π_{DMM}, the parties have element-wise secret sharings $[\![U]\!]_q$ and $[\![V]\!]_q$ of the matrices U and V. At the end of the protocol, the parties have a secret sharing $[\![UV]\!]_q$ of the product matrix UV. For a detailed description and a proof of security of this protocol, we refer to [28], [51].

For computing the coefficient vector using (3), besides matrix multiplication, we also need to compute the inverse of a covariance matrix. To do this in a PP fashion, we use a secure matrix inversion protocol that is based on a generalization of the Newton-Raphson division method to matrices [28]. At the start of the protocol, which we denote as Π_{MatInv}, the parties have shares of a covariance matrix A, and at the end of the protocol, they have shares of the inverted matrix A^{-1}. For details about the protocol Π_{MatInv}, we refer to [28].

The protocols described above, and their security proofs, assume all computations are done with numbers from the finite field $\mathbb{F}_q$. In real-life applications, such as the BCI application of estimating driver drowsiness that we consider later in this paper, the inputs are real numbers. We therefore need a way to approximate computations with real numbers by computations with numbers from $\mathbb{F}_q$. To this end, we adapt the method of Catrina and Saxena [52] for fixed-point representation of the numbers in the same way as was described in [28]. Similarly, when secure multiplications are performed, we use the slightly modified version Π_{Trunc} of the truncation protocol of Catrina and Saxena [52] that was presented in [28]. For the computation of the results in Section VI, numbers are represented with a $f = 64$ bit decimal precision and a $e = 64$ bit integer precision. For q, i.e. the dimension of the field, we use the first prime value larger than 2^{e+2f+1} to allow the truncation protocol to work correctly and not result in an overflow during intermediate computations.

IV. Cryptographic Protocols

We present a solution for PP training and inference with LR models in two different scenarios that are very relevant in practice, and both involve m source parties and a target party:

- **Target-independent LR.** In the target-independent LR scenario, one LR model is trained with data from m source parties, and used to make predictions about a target party. No data from the target party is used during the training phase. This scenario corresponds to "Baseline 1" in [6].
- **Target-calibrated LR.** In the target-calibrated LR scenario, m LR models are trained, each with data from one of the m source parties combined with some calibration data from the target party. Inferences for the target party are subsequently made by an ensemble of the trained LR models. This scenario corresponds to "DAMF" in [6].

Both approaches are valuable in practice, and even more sophisticated techniques to leverage calibration data exist [6]. Our goal in this paper is not to investigate which of these techniques can lead to the most accurate predictions. Instead, our aim is to show that the computations needed to train and use such regression models can be performed in a fully PP way, i.e. so that none of the parties involved has to disclose its data to anyone else in an unencrypted way. From the PP perspective, the two scenarios outlined above pose quite different challenges and require different protocols, which we describe in more detail below.

A. Training for target-independent LR

In the PP target-independent LR scenario, illustrated in Fig. 1, each of the m source parties $p_1, p_2, \ldots, p_m$ has its own rows of the matrix X and corresponding entries of the vector $\mathbf{y}$ from (1). Each party p_i can construct its own $n_{p_i} \times k$ matrix X_{p_i} and a vector $\mathbf{y}_{p_i}$ of length n_{p_i}, with n_{p_i} the number of training examples held by party p_i and k the number of features. We take advantage of the fact that the data is horizontally partitioned in this way, and propose a protocol for PP training of a LR model with the data from all parties that is more efficient in this situation than the more general protocol from De Cock et al. [28]. Our technique consists of the steps described below.

1) Offline phase: The TI distributes the necessary correlated random numbers to the parties. These random numbers will be needed for secure multiplications in the 4th and 5th step.
2) Local computation of $[\![X^T X]\!]_q$: Each source party p_i maps its fixed-point inputs to elements of a finite field $\mathbb{F}_q$ using

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TNSRE.2019.2926965, IEEE Transactions on Neural Systems and Rehabilitation Engineering

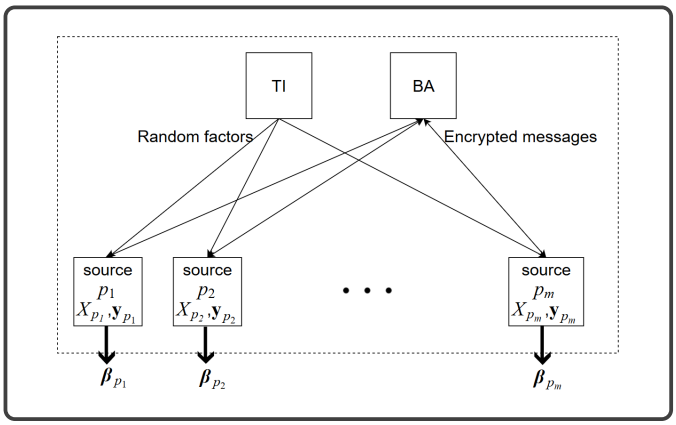

Fig. 1. Training phase of privacy-preserving target-independent LR.

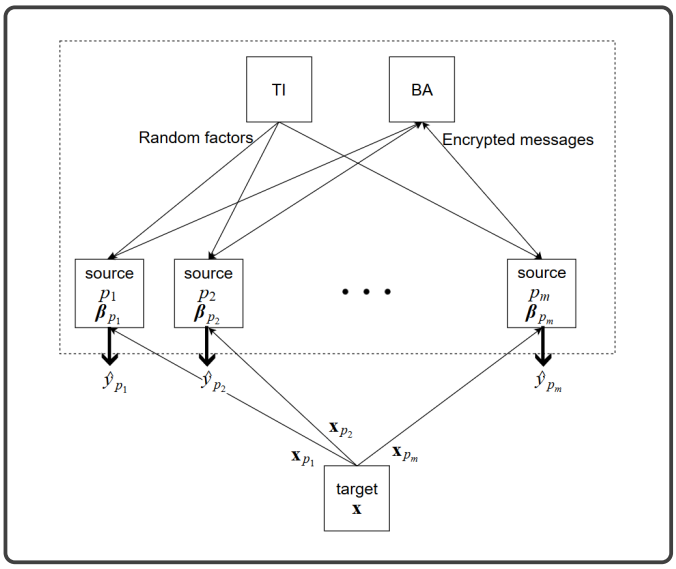

Fig. 2. Inference phase of privacy-preserving target-independent LR.

the method described in Section III, and creates an $n_{p_i} \times k$ matrix X_{p_i} and a vector $\mathbf{y}_{p_i}$ of length n_{p_i}, with n_{p_i} the number of training examples owned by party p_i and k the number of features. Next, each source party **locally** computes the $k \times k$ matrix $(X_{p_i})^T(X_{p_i})$. It holds that $X^TX = (X_{p_1})^T(X_{p_1}) + \ldots + (X_{p_m})^T(X_{p_m}) \mod q$. In other words, each party p_i now holds a secret share of the matrix X^TX from Equation (3).

3) Local computation of $[\![X^T\mathbf{y}]\!]_q$: each source party p_i **locally** computes the $k \times 1$ matrix $(X_{p_i})^T(\mathbf{y}_{p_i})$. It holds that $X^T\mathbf{y} = (X_{p_1})^T(\mathbf{y}_{p_1}) + \ldots + (X_{p_m})^T(\mathbf{y}_{p_m}) \mod q$. In other words, each party p_i now holds a secret share of the matrix $X^T\mathbf{y}$ from Equation (3).
4) Joint computation of $[\![(X^TX)^{-1}]\!]_q$: the parties perform **joint** computations over their shares $[\![X^TX]\!]_q$ from step 2 to compute shares $[\![(X^TX)^{-1}]\!]_q$ of the inverse matrix $(X^TX)^{-1}$, using the protocol for covariance matrix inversion Π_{MatInv} mentioned in Section III, which in turn relies on the protocols for secure matrix multiplication π_{DMM} and truncation Π_{Trunc}.
5) Joint computation of $[\![\boldsymbol{\beta}]\!]_q$: the parties perform **joint** computations over their shares $[\![(X^TX)^{-1}]\!]_q$ from step 4 and $[\![X^T\mathbf{y}]\!]_q$ from step 3 to compute shares $[\![\boldsymbol{\beta}]\!]_q$ of the coefficient vector $\boldsymbol{\beta} = (X^TX)^{-1}X^T\mathbf{y}$. To this end, they perform distributed matrix multiplication between $(X^TX)^{-1}$ and $X^T\mathbf{y}$ by performing secure matrix multiplication π_{DMM} and secure truncation protocol Π_{Trunc} operations. In the end each party p_i has a share $\boldsymbol{\beta}_{p_i}$ of the estimated regression coefficient vector $\boldsymbol{\beta} = \boldsymbol{\beta}_{p_1} + \boldsymbol{\beta}_{p_2} + \ldots \boldsymbol{\beta}_{p_m} \mod q$. Note that each $\boldsymbol{\beta}_{p_i}$ is a vector itself, containing a share of each of the coefficients $\beta_0, \beta_1, \ldots, \beta_k$.

In step 4 and 5 above, all the m parties perform computations and exchange encrypted messages with each other. To facilitate the communication among all parties, and to limit the number of communication channels (sockets) that need to be opened during execution, we use a Broadcast Agent (BA), more details about which are provided in Section V.

The five steps outlined above allow m source parties to work together to train an LR model on all of their data. None of the source parties sees data from any of the other source parties in an unencrypted way, and none of the source parties can reconstruct the LR model by itself. Instead, at the end of the training protocol, shares of the coefficient vector are held in a distributed fashion by all m parties. This entails that, when making new predictions with the trained model, all m parties have to be involved, as we describe in Section IV-B.

B. Inference for target-independent LR

During the *inference phase* (Fig. 2), the target party obtains a prediction for its input data by sending shares of its input to all the m parties. The parties engage in an SMC protocol among themselves. At the end of the evaluation protocol, each of the m parties sends its share of the result back to the target party, which adds the shares up to obtain the prediction.

Concretely, inference in the target-independent LR scenario consists of the following four steps:

1) Offline phase: The TI distributes the necessary correlated random numbers to the parties. These random numbers will be needed for secure multiplications in the third step.
2) Distribution of $[\![\mathbf{x}]\!]_q$: The target party maps the numbers in its input vector $\mathbf{x}$ to elements of the finite field $\mathbb{F}_q$ using the method described in Section III, and sends a share $\mathbf{x}_{p_i}$ of the resulting vector $\mathbf{x}$ to each of the source parties, with $\mathbf{x} = \mathbf{x}_{p_1} + \mathbf{x}_{p_2} + \ldots + \mathbf{x}_{p_m} \mod q$.
3) Joint computation of $[\![\boldsymbol{\beta} \cdot \mathbf{x}^T]\!]_q$: the m source parties perform joint computations over their shares $[\![\boldsymbol{\beta}]\!]_q$ from step 5 in Section IV-A and their shares $[\![\mathbf{x}]\!]_q$ from step 2 above to obtain shares of $[\![\boldsymbol{\beta} \cdot \mathbf{x}^T]\!]_q$. To this end, they use the secure matrix multiplication π_{DMM} and secure truncation Π_{Trunc} protocols mentioned in Section III. Each source party sends it computed share, which we refer to as $\hat{y}_{p_i}$ below, back to the target party.
4) Local computation of $\hat{y}$: the target party adds the received shares $\hat{y}_{p_i}$ $(i = 1, \ldots, m)$ to learn the prediction $\hat{y} = \hat{y}_{p_1} + \hat{y}_{p_2} + \ldots + \hat{y}_{p_m} \mod q$.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TNSRE.2019.2926965, IEEE Transactions on Neural Systems and Rehabilitation Engineering

C. Training for target-calibrated LR

PP target-calibrated LR requires m cases of secure two-party computation during the *training phase* (Fig. 3), since each of the m LR models is trained with data from only two parties, namely a source party and the target party. That means that instead of just one matrix X as in Section IV-A, m such matrices are implicitly used, which we denote as $X^{(1)}, X^{(2)}, \ldots, X^{(m)}$. Each such matrix $X^{(i)}$ consists of rows with training examples from source party p_i and rows with calibration data from the target party. In practice, none of these matrices exist in one place. Instead, each matrix $X^{(i)}$ exists in a distributed fashion across source party p_i and the target party, who each have a share of it. In addition, for each matrix $X^{(i)}$ there is a corresponding response value vector $\mathbf{y}^{(i)}$ which is shared in a similar way between source party p_i and the target party.

At the end of the training protocol, the coefficients of the ith regression model ($i = 1, \ldots, m$) are shared between the target party and source party p_i. These coefficients are computed through the steps described below.

1) Offline phase: The TI distributes the necessary correlated random numbers to the parties. These random numbers will be needed for secure multiplications in step 4 and 5 below.
2) Local construction of $[\![X^{(i)}]\!]_q$ and $[\![\mathbf{y}^{(i)}]\!]_q$, $i = 1, \ldots, m$: the target party maps its calibration data to elements of a finite field $\mathbb{F}_q$ using the method described in Section III, and creates an $n_t \times k$ matrix X_t and a vector $\mathbf{y}_t$ of length n_t, with n_t the number of training examples in the calibration data. Likewise, each source party p_i maps its fixed-point inputs to elements of $\mathbb{F}_q$, and creates an $n_{p_i} \times k$ matrix X_{p_i} and a vector $\mathbf{y}_{p_i}$ of length n_{p_i}, with n_{p_i} the number of training examples owned by party p_i and k the number of features. At this point, the source parties and the target party are holding shares of the matrices $X^{(i)}$ and vectors $\mathbf{y}^{(i)}$ for $i = 1, \ldots, m$ in a distributed fashion such that $X^{(i)} = X_{p_i} + X_t \mod q$ and $\mathbf{y}^{(i)} = \mathbf{y}_{p_i} + \mathbf{y}_t \mod q$. Note that these matrices are never constructed in their entirety in practice.
3) Local computation of $[\![(X^{(i)})^T X^{(i)}]\!]_q$ and $[\![(X^{(i)})^T \mathbf{y}^{(i)}]\!]_q$, $i = 1, \ldots, m$: each source party **locally** computes the $k \times k$ matrix $(X_{p_i})^T (X_{p_i})$ and the $k \times 1$ matrix $(X_{p_i})^T (\mathbf{y}_{p_i})$. The target party **locally** computes the $k \times k$ matrix $(X_t)^T (X_t)$ and the $k \times 1$ matrix $X_t^T \mathbf{y}_t$.
4) Joint computation of $[\![((X^{(i)})^T X^{(i)})^{-1}]\!]_q$, $i = 1, \ldots, m$: the target party separately engages in **joint** computations with each source party p_i over their shares $[\![(X^{(i)})^T X^{(i)}]\!]_q$ from step 3 to compute shares $[\![((X^{(i)})^T X^{(i)})^{-1}]\!]_q$ using the protocol for covariance matrix inversion Π_{MatInv} mentioned in Section III.
5) Joint computation of $[\![\boldsymbol{\beta}^{(i)}]\!]_q$, $i = 1, \ldots, m$: the target party separately engages in **joint** computations with each source party p_i over their shares $[\![((X^{(i)})^T X^{(i)})^{-1}]\!]_q$ from step 4 and shares $[\![(X^{(i)})^T \mathbf{y}^{(i)}]\!]_q$ from step 3 to compute shares $[\![\boldsymbol{\beta}^{(i)}]\!]_q$ of the coefficient vector $\boldsymbol{\beta}^{(i)} = ((X^{(i)})^T X^{(i)})^{-1} (X^{(i)})^T \mathbf{y}^{(i)}$. To this end, they perform distributed matrix multiplication between $((X^{(i)})^T X^{(i)})^{-1}$ and $(X^{(i)})^T \mathbf{y}^{(i)}$ by performing secure matrix multiplication π_{DMM} and secure truncation Π_{Trunc} operations.

When all computations are finished, m LR models have been trained. The coefficient vector of the ith model ($i = 1, \ldots, m$) is secret shared between the ith source party on one hand and the target party on the other hand, i.e., $\boldsymbol{\beta}^{(i)} = \boldsymbol{\beta}_{p_i}^{(i)} + \boldsymbol{\beta}_t^{(i)} \mod q, \quad \text{for } i = 1, \ldots, m$

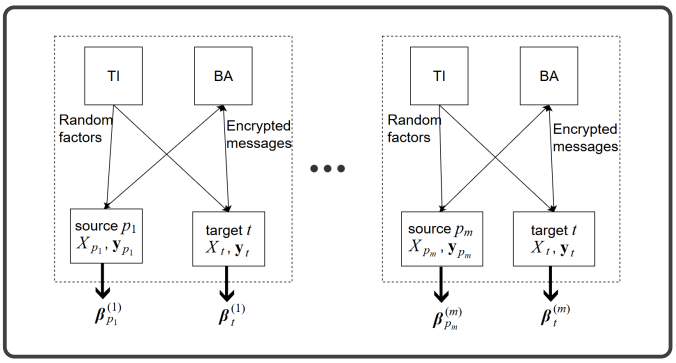

Fig. 3. Training phase of privacy-preserving target-calibrated LR.

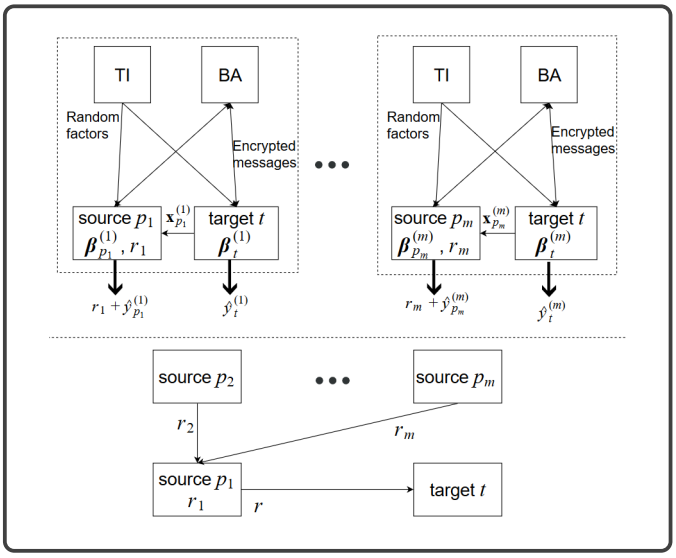

Fig. 4. Inference phase of privacy-preserving target-calibrated LR.

D. Inference for target-calibrated LR

The *inference phase* (Fig. 4) for target-calibrated LR requires the computation of the average of the outputs of the m regression models, which again involves secure two-party computations among the target party and each source party. As in Section IV-B, the target party has an input $\mathbf{x}$ for which it needs a prediction. To this end, the target party engages in a secure computation with each source party p_i to construct a secret sharing $[\![\hat{y}^{(i)}]\!]_q = [\![\boldsymbol{\beta}^{(i)} \cdot \mathbf{x}^T]\!]_q$. The final prediction is the average of all the $\hat{y}^{(i)}$ values, $i = 1, \ldots, m$. Instead of having each source party p_i send its share of $\hat{y}^{(i)}$ to the target party, which would reveal information that is not strictly necessary, each of the parties p_i first mask their prediction share by adding a random number r_i and open

the masked result to the target party. In addition, each party p_i sends its random mask r_i to one of the source parties (p_1 in Fig. 4), which adds them up, and sends the result r to the target party. Finally the target party locally adds up the masked shares of the $\hat{y}^{(i)}$ values (received directly from each of the source parties), subtracts the sum of the masks r (received from the designated source party that is responsible for constructing this sum), adds its own shares of the $\hat{y}^{(i)}$ values, and divides by the number of source parties to obtain the final prediction. Concretely, inference in the target-calibrated LR scenario consists of the steps described below.

1) Offline phase: The TI distributes the necessary correlated random numbers to the parties. These random numbers will be needed for secure multiplications in step 3.
2) Distribution of $[\![\mathbf{x}]\!]_q$: the target party maps the numbers in its input vector to elements of the finite field $\mathbb{F}_q$ using the method described in Section III. Next the target party secret shares the input vector $\mathbf{x}$ with each of the source parties, possibly in a different way, i.e. $\mathbf{x}^{(i)} = \mathbf{x}_{p_i}^{(i)} + \mathbf{x}_t^{(i)} \mod q$ for $i = 1, \ldots, m$.
3) Joint computation of $[\![\hat{y}^{(i)}]\!]_q$, $i = 1, \ldots, m$: the target party performs joint computations with each source party p_i over their shares $[\![\boldsymbol{\beta}^{(i)}]\!]_q$ from step 5 in Section IV-C and their shares $[\![\mathbf{x}]\!]_q$ from step 2 above to compute shares $[\![\hat{y}^{(i)}]\!]_q = [\![\boldsymbol{\beta}^{(i)} \cdot \mathbf{x}^T]\!]_q$. To this end, the parties use the secure matrix multiplication π_{DMM} and secure truncation Π_{Trunc} protocols mentioned in Section III. At the end of this step, shares of the predictions made by each of the m LR models have been constructed and are held, in a distributed fashion, by the target party and each of the source parties: $\hat{y}^{(i)} = \hat{y}_{p_i}^{(i)} + \hat{y}_t^{(i)} \mod q$ for $i = 1, \ldots, m$.
4) Local masking of shares: all the source parties mask their share $\hat{y}_{p_i}^{(i)}$ by adding a random value r_i, and subsequently send the masked prediction to the target party t.
5) Local computation of sum of masks: all the source parties send their random masks r_i, to one of the parties (chosen based on the asymmetric bit as mentioned in Section V). This designated party adds up all the random masks r_i, $i = 1, \ldots, m$, and sends the result to the target party.
6) Local computation of final prediction: the target party adds up all the masked prediction shares received in step 4, i.e. $\hat{y}_{p_i}^{(i)} + r_i$, $i = 1, \ldots, m$, subtracts the sum of the random masks received in step 5, adds its own shares $\hat{y}_t^{(i)}$, $i = 1, \ldots, m$ computed in step 3, and takes the average.

V. Implementation in Lynx

In this section we describe design decisions made when implementing the protocols from Section IV in Lynx [12], a framework that we developed for SMC. As explained in Section III, our protocols are developed for the commodity-based model, where the players running the distributed computations receive pre-distributed data from a trusted source (TI) during a setup phase. This data consist of correlated random numbers that help to mask information during the computations. The algorithms for secure LR described in Section IV rely on the cryptographic protocols for secure matrix multiplication, matrix inversion, and truncation that were mentioned in Section III. We implemented these protocols such that the performance scales with an increasing number of players involved in the computations. The ability to efficiently accomodate more than three parties to jointly perform the computations, sets our Lynx framework apart from existing SMC frameworks that are limited to two or three parties [35], [37] or that become computationally heavy with more than three parties [36].

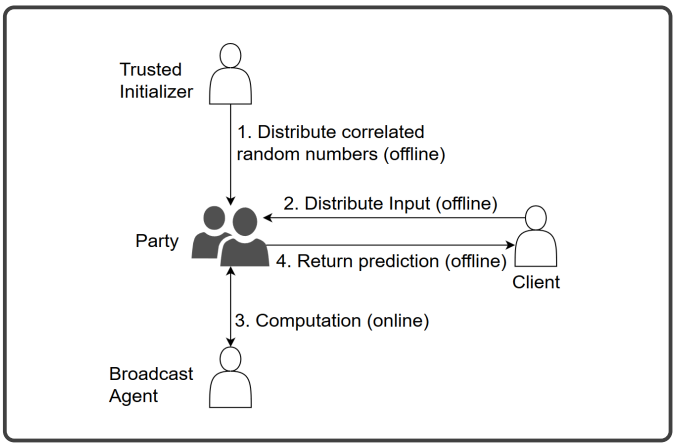

Fig. 5. Different roles in the SMC framework Lynx.

There are four significant roles that run at various stages for end-to-end model training and inference. They function as illustrated in Fig. 5. A deployed system consists of two or more Parties, one Broadcast Agent, one Trusted Initializer, and one or more Clients. The Parties communicate via the Broadcast Agent. The difference between a Party and a Client is that a Party engages in SMC computations, while a Client does not. The role of the latter is limited to distributing input data and receiving corresponding outputs that were computed by the Parties in a secure way. In the target-independent LR scenario for driver drowsiness prediction (see Section VI) for instance, the target driver is a Client, while in the target-calibrated LR scenario, the target driver is a Party as well as a Client.

1) Party: The Party is the core module responsible for model training and inference. The Parties take shares of the input, compute the result of a function over this input, and return the output shares. If Lynx is used for training, they produce shares of the trained model as an output, such as the shares of the coefficient vector of a LR model. When used for prediction, they produce the shares of the predicted result. At no point does any of the individual Parties know the data or the result held by any other Party in an unencrypted way.

2) Trusted Initializer: The TI runs as an offline program to generate the set of correlated random data required for the computations. It passes shares of this data to all the parties before the start of the computations and does not interfere with the computation any further from that point onward.

3) Broadcast Agent: One of the cornerstones of SMC is the pair-wise exchange of masked data between the Parties involved in the computations. While this works well in a 2-party scenario, the performance can get worse with an increase in the number of Parties, which is a plausible explanation for why most existing SMC implementations only

have few parties. In Lynx we have introduced a "bulletin board" functionality referred to as the Broadcast Agent. It is a dummy server which relays public messages to all Parties. The principal benefit of using the Broadcast Agent is to reduce the number of communication channels (sockets) that need to be opened, thereby greatly enhancing the efficiency of the communication. A traditional broadcast protocol would establish $O(m^2)$ sockets among the m parties, while only $O(m)$ sockets are necessary when a Broadcast Agent is used.

4) Client: The Client is the user that holds the private data and wants to get predictions from the model. The client secret shares the input among m Parties, such that none of the Parties knows the actual input, and receives back the shares of the predicted value. This way neither Party gets to know anything about the client's input or prediction. In the case of a target-calibrated LR, the target driver acts like a Party while training the model, and acts like a Client during the inference.

The Broadcast Agent and Trusted Initializer may exist on one or more servers. The Parties can run on a single or distributed network. Lynx uses two main architecture patterns: 1) Client-server architecture for all communications of the parties with the trusted initializer and the Broadcast Agent; 2) Microservices architecture to achieve modularity between all the SMC protocols. This allows to reuse the protocols and run them concurrently at different stages of computations. The Parties jointly compute an ML model (LR in this paper) by calling the different cryptographic protocols as microservices. Lynx is designed such that independent computations can happen in parallel, thus increasing throughput. Finally, we have created a number of utility protocols in Lynx which help in batch processing many cryptographic protocols to reduce communication overhead among parties.

VI. Experimental Results

A. Dataset and Hardware Specifications

We evaluated the implementation of our cryptographic protocols using the same data and scenarios for detecting driver drowsiness based on EEG signals as Wu et al. [6]. We used data from subjects who participated in a 60-90 minutes sustained-attention driving experiment in a real vehicle mounted on a motion platform immersed in a 360-degree virtual-reality scene. To induce drowsiness during driving, the virtual-reality scenes simulated monotonous driving at a fixed 100 km/h speed on a straight and empty highway. During the experiment, lane-departure events were randomly applied every 5-10 seconds, and participants were instructed to steer the vehicle to compensate for these perturbations as quickly as possible. 16 voluntary participants of age 24.2 ± 3.7 (10 males and 6 females) with normal or corrected-to-normal vision were recruited in this study. Data from one subject was not correctly recorded, so we used only 15 subjects.

We defined a function [53] to map the recorded response time τ to a drowsiness index $y \in [0, 1]$:

$$y = \max\left\{0,\ (1 - e^{-(\tau-\tau_0)})/(1 + e^{-(\tau-\tau_0)})\right\} \qquad (4)$$

$\tau_0 = 1$ was used in this paper, as in [6], [53]. The drowsiness indices were then smoothed using a 90-second square moving-average window to reduce variations. This does not reduce the sensitivity of the drowsiness index because the cycle lengths of drowsiness fluctuations are longer than 4 minutes [54].

During the experiment, the participants' scalp EEG signals were recorded using a 32-channel (30-channel EEGs plus 2-channel earlobes) 500 Hz Neuroscan NuAmps Express system (Compumedics Ltd., VIC, Australia). Afterwards, the EEG data was preprocessed and features were extracted, resulting in a sequence of 1200 epochs for each driver, in which each epoch is characterized by 30 numerical values extracted from the EEG signal. For each of the 15 drivers we therefore have a dataset consisting of 1200 rows, in chronological order, each consisting of 30 numerical input values and a response value (the level of drowsiness). For more details on the preprocessing of the data, we refer to Wu et al. [6].

The experiments documented below were run on a AWS c5.9xlarge machines with 36 vCPUs, 72.0 GiB Memory. Each of the Trusted Initializer, Broadcast Agent, and all Parties ran on separate machines. Each runtime experiment was repeated 3 times and average results are reported.

B. Results for Target-Independent LR

We train an LR model with data from m source drivers (Fig. 1) and apply it to make inferences about a new target subject (Fig. 2). Since each source driver has 1200 rows of data, the full matrix X from Equation (2) is a $(m \cdot 1200) \times 30$ matrix, while $\mathbf{y}$ is a $(m \cdot 1200) \times 1$ vector. At no point X and $\mathbf{y}$ are constructed in full. Each source party naturally has a share of X and $\mathbf{y}$ at the outset of the algorithm: a 1200×30 matrix X_{p_i} and a 1200×1 vector $\mathbf{y}_{p_i}$ with the data from driver i.

The first columns of Table I contain runtime results for training a target-independent LR model with data from m source drivers, as the number of source drivers increases from $m = 2$ up to $m = 14$. In the clear, i.e. without any encryption, training is very fast and completes within a fraction of a second. As expected, training in a PP fashion using SMC is computationally heavier. The runtime grows with the number of drivers, because there is more training data available that needs to be processed, and more parties that need to communicate and coordinate. Still, as is clear from Table I, an increase in the number of parties has a moderate impact on the runtime, demonstrating that the implementation in Lynx of the PP protocol for training a LR regression model is scalable.

Next we evaluate the predictive accuracy of the trained target-independent LR models. To this end, we treat driver 15, which was not used for training the models in Table I as the target driver. We use the trained models to predict the response value for each of the 1200 rows in the data of the target driver. In the target-independent scenario, the coefficient vector $\boldsymbol{\beta}$ of the trained LR model is kept in a distributed fashion with each of the m source parties involved in the training. Making PP predictions with the trained model is therefore an m-party SMC problem, the runtime of which grows with m, as shown in the "Inference" columns in Table I. The RMSE (Root Mean Square Error) for those predictions is reported in the last column of Table I. We obtained the same RMSE in the clear as when computing over encrypted data, highlighting that there is no accuracy loss when computing in a PP way.

TABLE I
RESULTS FOR TRAINING AND INFERENCE IN THE TARGET-INDEPENDENT LR SCENARIO WITH AN INCREASING NUMBER OF PARTIES (DRIVERS).

	Training		**Inference**		
	Runtime (sec)		**Runtime (sec)**		**RMSE**
# of parties	In the clear	SMC	In the clear	SMC	
2	0.10	48.51	0.004	2.82	0.051
3	0.15	77.55	0.004	3.25	0.050
4	0.22	106.91	0.004	3.81	0.043
5	0.28	132.24	0.004	4.43	0.087
6	0.35	153.90	0.004	4.98	0.129
7	0.42	171.87	0.004	5.73	0.106
8	0.46	201.26	0.004	6.46	0.090
9	0.49	225.58	0.004	7.03	0.082
10	0.51	245.74	0.004	7.91	0.074
11	0.52	280.96	0.004	8.42	0.071
12	0.53	299.11	0.004	9.12	0.071
13	0.45	328.67	0.004	10.03	0.055
14	0.43	350.39	0.004	10.08	0.048

TABLE II
RESULTS FOR THE TARGET-CALIBRATED LR SCENARIO. THE iTH ROW IN THE TABLE CONTAINS THE RESULTS ABOUT THE LR MODEL TRAINED WITH 1200 ROWS OF DATA FROM THE i-TH DRIVER ($i = 1, \ldots, 14$) COMBINED WITH 100 ROWS OF CALIBRATION DATA FROM DRIVER 15.

	Training		**Inference**		
	Runtime (sec)		**Runtime (sec)**		**RMSE**
Source party id	In the clear	SMC	In the clear	SMC	
1	0.06	51.23	0.004	2.61	0.114
2	0.06	51.95	0.003	2.68	0.045
3	0.06	51.95	0.003	2.67	0.145
4	0.06	51.88	0.004	2.71	0.055
5	0.06	51.62	0.003	2.71	0.086
6	0.06	51.41	0.003	2.64	0.097
7	0.06	51.57	0.004	2.65	0.062
8	0.06	51.49	0.003	2.64	0.045
9	0.07	52.06	0.004	2.62	0.066
10	0.06	51.92	0.003	2.64	0.078
11	0.06	51.83	0.003	2.66	0.057
12	0.06	52.31	0.003	2.68	0.194
13	0.06	51.45	0.003	2.61	0.186
14	0.06	51.50	0.003	2.65	0.053
All	0.07	52.00	0.008	2.89	0.048

C. Results for Target-Calibrated LR

In the target-calibrated LR scenario, m LR models are trained (Fig. 3). For each LR model, the matrix $X^{(i)}$ consists of the 1200 rows from the ith source driver, followed by the first 100 rows of the target driver, which we use as calibration data. This means that each $X^{(i)}$ is a 1300×30 matrix, for $i = 1, \ldots, m$. Similarly, each $\mathbf{y^{(i)}}$ is a 1300×1 vector.

Table II present the runtimes for training target-calibrated regression models with calibration data from a target driver (in this case, driver 15) combined with data from one of the source drivers. Since training each regression model only involves two parties (the target and one of the source drivers), this is a 2-party computation. As shown in Table I, the average runtime for training a regression model with two parties is around 51.73 sec. As all 14 models can be trained in parallel, the training time to learn the entire target-calibrated model is approximately 52 sec. We evaluate the predictive accuracy of the trained target-calibrated LR models when predicting the response value for each of the remaining 1100 rows in the data of the target driver, i.e. the rows that were not used as calibration data. The RMSE for those predictions is reported in the last column of Table II, along with the time needed to make those predictions. The final prediction is computed as the average of the predictions of all $m = 14$ LR models. In the SMC based approach, an additional time of 0.24 sec is required for all parties to mask their prediction shares, to send the masked prediction shares to the target party, to send the mask to one of the parties, and to allow the target party to compute the final result (cfr. step 4 to 6 in Section IV-D). The time on average to make a prediction for the 1100 rows of a target driver is 0.008 sec when done in the clear, i.e. without encryption, and $2.65+0.24$ i.e 2.89 sec when done in a PP way using SMC. The RMSE is the same whether the predictions are made with full exposure of the EEG data or in private.

VII. CONCLUSION

This work presented the first application of commodity-based SMC for privacy-preserving processing of EEG data, as well as the largest documented experiment of secret sharing based SMC in general, with 15 players involved in all the computations. We proposed algorithms for PPLR in a target-independent as well as a target-calibrated scenario. We have implemented these algorithms in Lynx, a new SMC framework that we created to enable efficient SMC among many parties. The runtime results of our experiments for predicting driver drowsiness show that our LR protocols and their implementation scale very nicely with an increasing number of drivers involved in the computations, and that the privately trained LR models are as accurate as those trained in the clear, i.e. without any encryption. Our work shows that additive secret sharing based SMC is a viable mechanism for protecting the privacy of users in future brain-computer interface applications. However, our running times were obtained using powerful machines and much work is needed to make these protocols practical in constrained computing devices. Interesting future research directions include: (i) to design protocols that work for more restrictive adversarial models (such as fully malicious or covert) and (ii) to improve communication, computational and round complexities for our protocols.

REFERENCES

[1] T. Bonaci, J. Herron, and H. Chizeck, "How susceptible is the brain to the side-channel private information extraction," *American Journal of Bioethics, Neuroscience*, vol. 6, no. 4, pp. 82–83, 2015.
[2] M. Inzlicht, I. McGregor, J. B. Hirsh, and K. Nash, "Neural markers of religious conviction," *Psychol. Sci.*, vol. 20, no. 3, pp. 385–392, 2009.
[3] M. Poel, C. Mühl, B. Reuderink, and A. Nijholt, "Guessing what's on your mind," in *Int. Conf. of Brain Informatics*, 2010, pp. 180–191.
[4] Z. G. Doborjeh, M. G. Doborjeh, and N. Kasabov, "Attentional bias pattern recognition in spiking neural networks from spatio-temporal EEG data," *Cognitive Computation*, vol. 10, no. 1, pp. 35–48, 2018.
[5] X. Qian, B. R. Y. Loo, F. X. Castellanos, S. Liu, H. L. Koh, X. W. W. Poh, R. Krishnan, D. Fung, M. W. Chee, C. Guan *et al.*, "Brain-computer-interface-based intervention re-normalizes brain functional network topology in children with attention deficit/hyperactivity disorder," *Translational Psychiatry*, vol. 8, no. 1, p. 149, 2018.
[6] D. Wu, V. J. Lawhern, S. Gordon, B. J. Lance, and C.-T. Lin, "Driver drowsiness estimation from EEG signals using online weighted adaptation regularization for regression (OwARR)," *IEEE Transactions on Fuzzy Systems*, vol. 25, no. 6, pp. 1522–1535, 2017.

[7] I. Martinovic, D. Davies, M. Frank, D. Perito, T. Ros, and D. Song, "On the feasibility of side-channel attacks with brain-computer interfaces," in *Proc. of the 21st USENIX Security Symposium*, 2012.

[8] T. Bonaci, R. Calo, and H. J. Chizeck, "App stores for the brain: Privacy & security in brain-computer interfaces," *IEEE Technology and Society Magazine*, vol. 34, no. 2, pp. 32–39, 2015.

[9] ACM, "USACM statement on the importance of preserving personal privacy," https://www.acm.org/articles/bulletins/2018/march/usacm-statement-on-data-privacy, 2018, accessed Mar 30, 2018.

[10] L. Sweeney, "Systems and methods for deidentifying entries in a data source," 2007, US Patent 7,269,578.

[11] R. Cramer, I. Damgård, and J. B. Nielsen, *Secure Multiparty Computation and Secret Sharing*. Cambridge University Press, 2015.

[12] A. Agarwal, K. Saminathan, and S. Bhagat, "Lynx: A framework for privacy preserving machine learning," https://bitbucket.org/uwtppml.

[13] M. Abadi, A. Chu, I. Goodfellow, H. B. McMahan, I. Mironov, K. Talwar, and L. Zhang, "Deep learning with differential privacy," in *23rd ACM SIGSAC Conf. on Comp. and Comm. Security*, 2016, pp. 308–318.

[14] S. Chandra, V. Karande, Z. Lin, L. Khan, M. Kantarcioglu, and B. Thuraisingham, "Securing data analytics on SGX with randomization," in *Eur. Symp. on Research in Comp. Sec.* Springer, 2017, pp. 352–369.

[15] K. Bonawitz, V. Ivanov, B. Kreuter, A. Marcedone, H. B. McMahan, S. Patel, D. Ramage, A. Segal, and K. Seth, "Practical secure aggregation for privacy-preserving machine learning," in *24th ACM SIGSAC Conf. on Comp. and Comm. Security*, 2017, pp. 1175–1191.

[16] R. Shokri and V. Shmatikov, "Privacy-preserving deep learning," in *CCS 2015*, 2015, pp. 1310–1321.

[17] J. Wang, V. L. Cherkassky, and M. A. Just, "Predicting the brain activation pattern associated with the propositional content of a sentence: Modeling neural representations of events and states," *Human Brain Mapping*, vol. 38, no. 10, pp. 4865–4881, 2017.

[18] C. Du, C. Du, and H. He, "Sharing deep generative representation for perceived image reconstruction from human brain activity," in *Proc. of International Joint Conf. on Neural Networks*, 2017, pp. 1049–1056.

[19] E. Boto, N. Holmes, J. Leggett, G. Roberts, V. Shah, S. S. Meyer, L. D. Muñoz, K. J. Mullinger, T. M. Tierney, S. Bestmann *et al.*, "Moving magnetoencephalography towards real-world applications with a wearable system," *Nature*, vol. 555, pp. 657–661, 2018.

[20] F. Chen, T. Xiang, X. Lei, and J. Chen, "Highly efficient linear regression outsourcing to a cloud," *IEEE Transactions on Cloud Computing*, vol. 2, no. 4, pp. 499–508, 2014.

[21] R. Hall, S. E. Fienberg, and Y. Nardi, "Secure multiple linear regression based on homomorphic encryption," *Journal of Official Statistics*, vol. 27, no. 4, pp. 669–691, 2011.

[22] Y. Aono, T. Hayashi, L. T. Phong, and L. Wang, "Fast and secure linear regression and biometric authentication with security update." *IACR Cryptology ePrint Archive*, vol. 2015, p. 692, 2015.

[23] V. Nikolaenko, U. Weinsberg, S. Ioannidis, M. Joye, D. Boneh, and N. Taft, "Privacy-preserving ridge regression on hundreds of millions of records," in *IEEE Symp. on Security and Privacy*, 2013, pp. 334–348.

[24] A. F. Karr, X. Lin, A. P. Sanil, and J. P. Reiter, "Privacy-preserving analysis of vertically partitioned data using secure matrix products," *Journal of Official Statistics*, vol. 25, no. 1, p. 125, 2009.

[25] A. P. Sanil, A. F. Karr, X. Lin, and J. P. Reiter, "Privacy preserving regression modelling via distributed computation," in *10th ACM SIGKDD Int. Conf. on Knowledge Disc. and Data Mining*, 2004, pp. 677–682.

[26] A. F. Karr, X. Lin, A. P. Sanil, and J. P. Reiter, "Secure regression on distributed databases," *Journal of Computational and Graphical Statistics*, vol. 14, no. 2, pp. 263–279, 2005.

[27] W. Du and M. J. Atallah, "Privacy-preserving cooperative statistical analysis," in *17th Annual Comp. Sec. Appl. Conf.*, 2001, pp. 102–110.

[28] M. De Cock, R. Dowsley, A. C. A. Nascimento, and S. C. Newman, "Fast, privacy preserving linear regression over distributed datasets based on pre-distributed data," in *8th ACM Workshop on Artificial Intelligence and Security (AISec)*, 2015, pp. 3–14.

[29] A. Gascón, P. Schoppmann, B. Balle, M. Raykova, J. Doerner, S. Zahur, and D. Evans, "Privacy-preserving distributed linear regression on high-dimensional data," *Proceedings on Privacy Enhancing Technologies*, vol. 2017, no. 4, pp. 345 – 364, 2017.

[30] W. Du, Y. S. Han, and S. Chen, "Privacy-preserving multivariate statistical analysis: Linear regression and classification," in *4th SIAM International Conference on Data Mining*, 2004, pp. 222–233.

[31] C. Clifton, M. Kantarcioglu, J. Vaidya, X. Lin, and M. Y. Zhu, "Tools for privacy preserving distributed data mining," *SIGKDD Explor. Newsl.*, vol. 4, no. 2, pp. 28–34, 2002.

[32] C. C. Aggarwal and S. Y. Philip, "A general survey of privacy-preserving data mining models and algorithms," in *Privacy-Preserving Data Mining*. Springer, 2008, pp. 11–52.

[33] S. de Hoogh, B. Schoenmakers, P. Chen, and H. op den Akker, "Practical secure decision tree learning in a teletreatment application," in *International Conference on Financial Cryptography and Data Security*. Springer, 2014, pp. 179–194.

[34] K. Fritchman, K. Saminathan, R. Dowsley, T. Hughes, M. De Cock, A. Nascimento, and A. Teredesai, "Privacy-preserving scoring of tree ensembles: A novel framework for AI in healthcare," in *Proceedings of 2018 IEEE International Conference on Big Data*, 2018, pp. 2412–2421.

[35] D. Bogdanov, S. Laur, and J. Willemson, "Sharemind: A framework for fast privacy-preserving computations," in *13th Eur. Symp. on Research in Comp. Sec.*, 2008, pp. 192–206.

[36] A. Ben-David, N. Nisan, and B. Pinkas, "FairplayMP: A system for secure multi-party computation," in *Proc. of 15th ACM Conference on Computer and Communications Security*, 2008, pp. 257–266.

[37] M. Sadegh Riazi, C. Weinert, O. Tkachenko, E. M. Songhori, T. Schneider, and F. Koushanfar, "Chameleon: A Hybrid Secure Computation Framework for Machine Learning Applications," *ArXiv e-prints*, 2018.

[38] T. Sandle, "Artificial intelligence helps to keep tired drivers awake," http://www.digitaljournal.com/tech-and-science/technology/artificial-intelligence-helps-to-keep-tired-drivers-awake/article/499369.

[39] A. Agarwal, R. Dowsley, N. D. McKinney, D. Wu, C.-T. Lin, M. De Cock, and A. Nascimento, "Privacy-preserving linear regression for brain-computer interface applications," in *Proceedings of 2018 IEEE International Conference on Big Data*, 2018, pp. 5260–5261.

[40] D. Beaver, "One-time tables for two-party computation," in *Computing and Combinatorics*. Springer, 1998, pp. 361–370.

[41] B. David, R. Dowsley, R. Katti, and A. C. Nascimento, "Efficient unconditionally secure comparison and privacy preserving machine learning classification protocols," in *International Conference on Provable Security*. Springer, 2015, pp. 354–367.

[42] M. De Cock, R. Dowsley, C. Horst, R. Katti, A. Nascimento, W.-S. Poon, and S. Truex, "Efficient and private scoring of decision trees, support vector machines and logistic regression models based on pre-computation," *IEEE Transactions on Dependable and Secure Computing*, vol. 16, no. 2, pp. 217–230, March 2019.

[43] R. L. Rivest, "Unconditionally secure commitment and oblivious transfer schemes using private channels and a trusted initializer," 1999, preprint available at http://people.csail.mit.edu/rivest/Rivest- commitment.pdf.

[44] D. Beaver, "Precomputing oblivious transfer," in *Annual Int. Cryptology Conference*. Springer, 1995, pp. 97–109.

[45] R. Dowsley, J. Van De Graaf, D. Marques, and A. C. Nascimento, "A two-party protocol with trusted initializer for computing the inner product," in *International Workshop on Information Security Applications*. Springer, 2010, pp. 337–350.

[46] R. Dowsley, J. Müller-Quade, A. Otsuka, G. Hanaoka, H. Imai, and A. C. A. Nascimento, "Universally composable and statistically secure verifiable secret sharing scheme based on pre-distributed data," *IEICE Transactions*, vol. 94-A, no. 2, pp. 725–734, 2011.

[47] Y. Ishai, E. Kushilevitz, S. Meldgaard, C. Orlandi, and A. Paskin-Cherniavsky, "On the power of correlated randomness in secure computation," in *Theory of Cryptography*. Springer, 2013, pp. 600–620.

[48] R. Tonicelli, A. C. A. Nascimento, R. Dowsley, J. Müller-Quade, H. Imai, G. Hanaoka, and A. Otsuka, "Information-theoretically secure oblivious polynomial evaluation in the commodity-based model," *Int. Journal of Information Security*, vol. 14, no. 1, pp. 73–84, 2015.

[49] B. David, R. Dowsley, J. van de Graaf, D. Marques, A. C. A. Nascimento, and A. C. B. Pinto, "Unconditionally secure, universally composable privacy preserving linear algebra," *IEEE Transactions on Information Forensics and Security*, vol. 11, no. 1, pp. 59–73, 2016.

[50] D. Beaver, "Efficient multiparty protocols using circuit randomization," in *Annual Int. Cryptology Conf.* Springer, 1991, pp. 420–432.

[51] R. Dowsley, "Cryptography based on correlated data: Foundations and practice," Ph.D. dissertation, Karlsruhe Institute of Technology, Germany, 2016.

[52] O. Catrina and A. Saxena, "Secure computation with fixed-point numbers," in *Int. Conf. on Financial Cryptography and Data Security*. Springer, 2010, pp. 35–50.

[53] C.-S. Wei, Y.-P. Lin, Y.-T. Wang, T.-P. Jung, N. Bigdely-Shamlo, and C.-T. Lin, "Selective transfer learning for EEG-based drowsiness detection," in *IEEE SMC 2015*, 2015, pp. 3229–3232.

[54] S. Makeig and M. Inlow, "Lapse in alertness: coherence of fluctuations in performance and EEG spectrum," *Electroencephalography and Clinical Neurophysiology*, vol. 86, no. 1, pp. 23–35, 1993.