High-Precision Privacy-Preserving Real-Valued Function Evaluation

Christina Boura[2,1], Ilaria Chillotti[2], Nicolas Gama[1,2], Dimitar Jetchev[1,3], Stanislav Peceny[1], Alexander Petric[1]

[1] Inpher, Lausanne, Switzerland and New-York, USA
https://inpher.io
[2] Laboratoire de Mathématiques de Versailles, UVSQ, CNRS, Université Paris-Saclay, 78035 Versailles, France
[3] École Polytechnique Fédérale de Lausanne, EPFL SB MATHGEOM GR-JET, Lausanne, Switzerland

Abstract. We propose a novel multi-party computation protocol for evaluating continuous real-valued functions with high numerical precision. Our method is based on approximations with Fourier series and uses at most two rounds of communication during the online phase. For the offline phase, we propose a trusted-dealer and honest-but-curious aided solution, respectively. We apply our algorithm to train a logistic regression classifier via a variant of Newton's method (known as IRLS) to compute unbalanced classification problems that detect rare events and cannot be solved using previously proposed privacy-preserving optimization algorithms (e.g., based on piecewise-linear approximations of the sigmoid function). Our protocol is efficient as it can be implemented using standard quadruple-precision floating point arithmetic. We report multiple experiments and provide a demo application that implements our algorithm for training a logistic regression model.

1 Introduction

Privacy-preserving computing allows multiple parties to evaluate a function while keeping the inputs private and revealing only the output of the function and nothing else. Recent advances in multi-party computation (MPC), homomorphic encryption, and differential privacy made these models practical. An example of such computations, with applications in medicine and finance, among others, is the training of supervised models where the input data comes from distinct secret data sources [17], [23], [25], [26] and the evaluation of predictions using these models.

In machine learning classification problems, one trains a model on a given dataset to predict new inputs, by mapping them into discrete categories. The classical *logistic regression* model predicts a class by providing a probability associated with the prediction. The quality of the model can be measured in several ways, the most common one being the *accuracy* that indicates the percentage of correctly predicted answers.

It appears that for a majority of the datasets (e.g., the MNIST database of handwritten digits [15] or the ARCENE dataset [14]), the classification achieves very good accuracy after only a few iterations of the gradient descent using a piecewise-linear approximation of the sigmoid function $\texttt{sigmo} : \mathbb{R} \to [0, 1]$ defined as

$$\texttt{sigmo}(x) = \frac{1}{1 + e^{-x}},$$

although the current cost function is still far from the minimum value [25]. Other approximation methods of the sigmoid function have also been proposed in the past. In [29], an approximation with low degree polynomials resulted in a more efficient but less accurate algorithm. Conversely, a higher-degree polynomial approximation applied to deep learning algorithms in [24] yielded more accurate, but less efficient algorithms (and thus, less suitable for privacy-preserving computing). In parallel, approximation solutions for privacy-preserving methods based on homomorphic encryption [2], [27], [18], [22] and differential privacy [1], [10] have been proposed in the context of both classification algorithms and deep learning.

Nevertheless, accuracy itself is not always a sufficient measure for the quality of the model, especially if, as mentioned in [19, p.423], our goal is to detect a rare event such as a rare disease or a fraudulent financial transaction. If, for example, one out of every one thousand transactions is fraudulent, a naïve model that classifies all transactions as honest achieves 99.9% accuracy; yet this model has no predictive capability. In such cases, measures such as *precision*, *recall* and *F1-score* allow for better estimating the quality of the model. They bound the rates of false positives or negatives relative to only the positive events rather than the whole dataset.

The techniques cited above achieve excellent accuracy for most balanced datasets, but since they rely on a rough approximation of the sigmoid function, they do not converge to the same model and thus, they provide poor scores on datasets with a very low acceptance rate. In this paper, we show how to regain this numerical precision in MPC, and to reach the same score as the plaintext regression. Our MPC approach is mostly based on additive secret shares with precomputed multiplication triplets [4]. This means that the computation is divided in two phases: an *offline* phase that can be executed before the data is shared between the players, and an *online phase* that computes the actual result. For the offline phase, we propose a first solution based on a *trusted dealer*, and then discuss a protocol where the dealer is *honest-but-curious*.

1.1 Our contributions

Fourier approximation of the sigmoid function. Evaluation of real-valued functions has been widely used in privacy-preserving computations. For instance, in order to train linear and logistic regression models, one is required to compute real-valued functions such as the square root, the exponential, the logarithm, the sigmoid or the softmax function and use them to solve non-linear optimization problems. In order to train a logistic regression model, one needs to minimize

a cost function which is expressed in terms of logarithms of the continuous sigmoid function. This minimum is typically computed via iterative methods such as the gradient descent. For datasets with low acceptance rate, it is important to get much closer to the exact minimum in order to obtain a sufficiently precise model. We thus need to significantly increase the number of iterations (naïve or stochastic gradient descent) or use faster-converging methods (e.g., IRLS [5, §4.3]). The latter require a numerical approximation of the sigmoid that is much better than what was previously achieved in an MPC context, especially when the input data is not normalized or feature-scaled. Different approaches have been considered previously such as approximation by Taylor series around a point (yielding only good approximation locally at that point), or polynomial approximation (by e.g., estimating least squares). Although better than the first one, this method is numerically unstable due to the variation of the size of the coefficients. An alternative method based on approximation by piecewise-linear functions has been considered as well. In MPC, this method performs well when used with garbled circuits instead of secret sharing and masking, but does not provide enough accuracy.

In our case, we approximate the sigmoid using Fourier series, an approach applied for the first time in this context. This method works well as it provides a better uniform approximation assuming that the function is sufficiently smooth (as is the case with the sigmoid). In particular, we virtually re-scale and extend the sigmoid to a periodic function that we approximate with a trigonometric polynomial which we then evaluate in a stable privacy-preserving manner. To approximate a generic function with trigonometric polynomials that can be evaluated in MPC, one either use the Fourier series of a smooth periodic extension or finds directly the closest trigonometric polynomial by the method of least squares for the distance on the half-period. The first approach yields a super-algebraic convergence at best, whereas the second converges exponentially fast. On the other hand, the first one is numerically stable whereas the second one is not (under the standard Fourier basis). In the case of the sigmoid, we show that one can achieve both properties at the same time.

Floating-point representation and masking. A typical approach to multi-party computation protocols with masking is to embed fixed-point values into finite groups and use uniform masking and secret sharing. Arithmetic circuits can then be evaluated using, e.g., precomputed multiplication triplets and following Beaver's method [4]. This idea has been successfully used in [13] and [12]. Whereas the method works well on low multiplicative depth circuits like correlations or linear regression [17], in general, the required group size increases exponentially with the multiplicative depth. In [25], this exponential growth is mitigated by a two-party rounding solution, but the technique does not extend to three or more players where an overflow in the most significant bits can occur. In this work, we introduce an alternative sharing scheme, where fixed-point values are shared directly using (possibly multibit) floating points, and present a technique to reduce the share sizes after each multiplication. This technique easily extends to an arbitrary number of players.

Significant reduction in communication time. In this paper, we follow the same approach as in [25] and define dedicated triplets for high-level instructions, such as large matrix multiplications, a system resolution, or an oblivious evaluation of the sigmoid. This approach is less generic than masking low-level instructions as in SPDZ, but it allows to reduce the communication and memory requirements by large factors. Masks and operations are aware of the type of vector or matrix dimensions and benefit from the vectorial nature of the high-level operations. For example, multiplying two matrices requires a single round of communication instead of up to $O(n^3)$ for coefficient-wise approaches, depending on the batching quality of the compiler. Furthermore, masking is defined per immutable variable rather than per elementary operation, so a constant matrix is masked only once during the whole algorithm. Combined with non-trivial local operations, these triplets can be used to achieve much more than just ring additions or multiplications. In a nutshell, the amount of communications is reduced as a consequence of reusing the same masks, and the number of communication rounds is reduced as a consequence of masking directly matrices and other large structures. Therefore, the total communication time becomes negligible compared to the computing cost.

New protocol for the honest but curious offline phase extendable to n players. We introduce a new protocol for executing the offline phase in the honest-but-curious model that is easily extendable to a generic number n of players while remaining efficient. To achieve this, we use a broadcast channel instead of peer-to-peer communication which avoids a quadratic explosion in the number of communications. This is an important contribution, as none of the previous protocols for $n > 3$ players in this model are efficient. In [17], for instance, the authors propose a very efficient algorithm in the trusted dealer model; yet the execution time of the oblivious transfer protocol is quite slow.

2 Notation and Preliminaries

Assume that $P_1, \ldots, P_n$ are distinct computing parties (players). We recall some basic concepts from multi-party computation that will be needed for this paper.

2.1 Secret sharing and masking

Let $(G, \bullet)$ be a group and let $x \in G$ be a group element.

A *secret share* of x, denoted by $[\![x]\!]_\bullet$ (by a slight abuse of notation), is a tuple $(x_1, \ldots, x_n) \in G^n$ such that $x = x_1 \bullet \cdots \bullet x_n$. If $(G, +)$ is abelian, we call the secret shares $x_1, \ldots, x_n$ *additive secret shares*. A secret sharing scheme is computationally secure if for any two elements $x, y \in G$, strict sub-tuples of shares $[\![x]\!]_\bullet$ or $[\![y]\!]_\bullet$ are indistinguishable. If G admits a uniform distribution, an information-theoretic secure secret sharing scheme consists of drawing $x_1, \ldots, x_{n-1}$ uniformly at random and choosing $x_n = x_{n-1}^{-1} \bullet \cdots \bullet x_1^{-1} \bullet x$. When G is not compact, the condition can be relaxed to statistical or computational indistinguishability.

A closely related notion is the one of *group masking*. Given a subset $\mathcal{X}$ of G, the goal of masking $\mathcal{X}$ is to find a distribution $\mathcal{D}$ over G such that the distributions of $x \bullet \mathcal{D}$ for $x \in \mathcal{X}$ are all indistinguishable. Indeed, such distribution can be used to create a secret share: one can sample $\lambda \leftarrow \mathcal{D}$, and give λ^{-1} to a player and $x \bullet \lambda$ to the other. Masking can also be used to evaluate non-linear operations in clear over masked data, as soon as the result can be privately unmasked via homomorphisms (as in, e.g., the Beaver's triplet multiplication technique [4]).

2.2 Arithmetic with secret shares via masking

Computing secret shares for a sum $x + y$ (or a linear combination if $(G, +)$ has a module structure) can be done non-interactively by each player by adding the corresponding shares of x and y. Computing secret shares for a product is more challenging. One way to do that is to use an idea of Beaver based on precomputed and secret shared multiplicative triplets. From a general point of view, let $(G_1, +)$, $(G_2, +)$ and $(G_3, +)$ be three abelian groups and let $\pi \colon G_1 \times G_2 \to G_3$ be a bilinear map.

Given additive secret shares $[\![x]\!]_+$ and $[\![y]\!]_+$ for two elements $x \in G_1$ and $y \in G_2$, we would like to compute secret shares for the element $\pi(x, y) \in G_3$. With Beaver's method, the players must employ precomputed single-use random triplets $([\![\lambda]\!]_+, [\![\mu]\!]_+, [\![\pi(\lambda, \mu)]\!]_+)$ for $\lambda \in G_1$ and $\mu \in G_2$, and then use them to mask and reveal $a = x + \lambda$ and $b = y + \mu$. The players then compute secret shares for $\pi(x, y)$ as follows:

- Player 1 computes $z_1 = \pi(a, b) - \pi(a, \mu_1) - \pi(\lambda_1, b) + (\pi(\lambda, \mu))_1$;
- Player i (for $i = 2, \ldots, n$) computes $z_i = -\pi(a, \mu_i) - \pi(\lambda_i, b) + (\pi(\lambda, \mu))_i$.

The computed $z_1, \ldots, z_n$ are the additive shares of $\pi(x, y)$. A given λ can be used to mask only one variable, so one triplet must be precomputed for each multiplication during the *offline phase* (i.e. before the data is made available to the players). Instantiated with the appropriate groups, this abstract scheme allows to evaluate a product in a ring, but also a vectors dot product, a matrix-vector product, or a matrix-matrix product.

2.3 MPC evaluation of real-valued continuous functions

For various applications (e.g., logistic regression in Section 6), we need to compute continuous real-valued functions over secret shared data. For non-linear functions (e.g. exponential, log, power, cos, sin, sigmoid, etc.), different methods are proposed in the literature.

A straightforward approach consists of implementing a full floating point arithmetic framework [6, 12], and to compile a data-oblivious algorithm that evaluates the function over floats. This is for instance what Sharemind and SPDZ use. However, these two generic methods lead to prohibitive running times if the floating point function has to be evaluated millions of times.

The second approach is to replace the function with an approximation that is easier to compute: for instance, [25] uses garbled circuits to evaluate fixed point comparisons and absolute values; it then replaces the sigmoid function in the logistic regression with a piecewise-linear function. Otherwise, [24] approximates the sigmoid with a polynomial of fixed degree and evaluates that polynomial with the Horner method, thus requiring a number of rounds of communications proportional to the degree.

Another method that is close to how SPDZ [13] computes inverses in a finite field, is based on polynomial evaluation via multiplicative masking: using a precomputed triplet of the form $(\llbracket\lambda\rrbracket_+, \llbracket\lambda^{-1}\rrbracket_+, \ldots, \llbracket\lambda^{-p}\rrbracket_+)$, players can evaluate $P(x) = \sum_{i=0}^{p} a_p x^p$ by revealing $u = x\lambda$ and outputting the linear combination $\sum_{i=0}^{p} a_i u^i \llbracket\lambda^{-i}\rrbracket_+$.

Multiplicative masking, however, involves some leakage: in finite fields, it reveals whether x is null. The situation gets even worse in finite rings where the multiplicative orbit of x is disclosed (for instance, the rank would be revealed in a ring of matrices), and over $\mathbb{R}$, the order of magnitude of x would be revealed.

For real-valued polynomials, the leakage could be mitigated by translating and rescaling the variable x so that it falls in the range $[1, 2)$. Yet, in general, the coefficients of the polynomials that approximate the translated function explode, thus causing serious numerical issues.

2.4 Full threshold honest-but-curious protocol

Since our goal is to emphasize new functionalities, such as efficient evaluation of real-valued continuous functions and good quality logistic regression, we often consider a scenario where all players follow the protocol without introducing any errors. The players may, however, record the whole transaction history and try to learn illegitimate information about the data. During the online phase, the security model imposes that any collusion of at most $n-1$ players out of n cannot distinguish any semantic property of the data beyond the aggregated result that is legitimately and explicitly revealed. To achieve this, Beaver triplets (used to mask player's secret shares) can be generated and distributed by a single entity called the *trusted dealer*. In this case, no coalition of at most $n-1$ players should get any computational advantage on the plaintext triplet information. However, the dealer himself knows the plaintext triplets, and hence the whole data, which only makes sense on some computation outsourcing use-cases. In Section 5, we give an alternative *honest-but-curious* (or semi-honest) protocol to generate the same triplets, involving this time bi-directional communications with the dealer. In this case, the dealer and the players collaborate during the offline phase in order to generate the precomputed material, but none of them have access to the whole plaintext triplets. This makes sense as long as the dealer does not collude with any player, and at least one player does not collude with the other players. We leave the design of actively secure protocols for future work.

3 Statistical Masking and Secret Share Reduction

In this section, we present in this section our masking technique for fixed-point arithmetic and provide an algorithm for the MPC evaluation of real-valued continuous functions. In particular, we show that to achieve p bits of numerical precision in MPC, it suffices to have $p + 2\tau$-bit floating points where τ is a fixed security parameter.

The secret shares we consider are real numbers. We would like to mask these shares using floating point numbers. Yet, as there is no uniform distribution on $\mathbb{R}$, no additive masking distribution over reals can perfectly hide the arbitrary inputs. In the case when the secret shares belong to some known range of numerical precision, it is possible to carefully choose a masking distribution, depending on the precision range, so that the masked value computationally leaks no information about the input. A distribution with sufficiently large standard deviation could do the job: for the rest of the paper, we refer to this type of masking as "statistical masking". In practice, we choose a normal distribution with standard deviation $\sigma = 2^{40}$.

On the other hand, by using such masking, we observe that the sizes of the secret shares increase every time we evaluate the multiplication via Beaver's technique (Section 2.2). In Section 3.3, we address this problem by introducing a technique that allows to reduce the secret share sizes by discarding the most significant bits of each secret share (using the fact that the sum of the secret shares is still much smaller than their size).

3.1 Floating point, fixed point and interval precision

Suppose that B is an integer and that p is a non-negative integer (the number of bits). The class of fixed-point numbers of exponent B and numerical precision p is:

$$\mathcal{C}(B, p) = \{x \in 2^{B-p} \cdot \mathbb{Z}, |x| \leq 2^B\}.$$

Each class $\mathcal{C}(B, p)$ is finite, and contains $2^{p+1}+1$ numbers. They could be rescaled and stored as $(p + 2)$-bit integers. Alternatively, the number $x \in \mathcal{C}(B, p)$ can also be represented by the floating point value x, provided that the floating point representation has at least p bits of mantissa. In this case, addition and multiplication of numbers across classes of the same numerical precision are natively mapped to floating-point arithmetic. The main arithmetic operations on these classes are:

- **Lossless Addition:** $\mathcal{C}(B_1, p_1) \times \mathcal{C}(B_2, p_2) \to \mathcal{C}(B, p)$ where $B = max(B_1, B_2) + 1$ and $p = B - \min(B_1 - p_1, B_2 - p_2)$;
- **Lossless Multiplication:** $\mathcal{C}(B_1, p_1) \times \mathcal{C}(B_2, p_2) \to \mathcal{C}(B, p)$ where $B = B_1 + B_2$ and $p = p_1 + p_2$;
- **Rounding:** $\mathcal{C}(B_1, p_1) \to \mathcal{C}(B, p)$, that maps x to its nearest element in $2^{B-p}\mathbb{Z}$.

Lossless operations require p to increase exponentially in the multiplication depth, whereas fixed precision operations maintain p constant by applying a final rounding. Finally, note that the exponent B should be incremented to store the result of an addition, yet, B is a user-defined parameter in fixed point arithmetic. If the user forcibly chooses to keep B unchanged, any result $|x| > 2^B$ will not be representable in the output domain (we refer to this type of overflow as *plaintext overflow*).

3.2 Floating point representation

Given a security parameter τ, we say that a set $\mathcal{S}$ is a τ-secure *masking set* for a class $\mathcal{C}(B, p)$ if the following distinguishability game cannot be won with advantage $\geq 2^{-\tau}$: the adversary chooses two plaintexts m_0, m_1 in $\mathcal{C}(B, p)$, a challenger picks $b \in \{0, 1\}$ and $\alpha \in \mathcal{S}$ uniformly at random, and sends $c = m_b + \alpha$ to the adversary. The adversary has to guess b. Note that increasing such distinguishing advantage from $2^{-\tau}$ to $\approx 1/2$ would require to give at least 2^τ samples to the attacker, so $\tau = 40$ is sufficient in practice.

Proposition 1. *The class* $\mathcal{C}(B, p, \tau) = \{\alpha \in 2^{B-p}\mathbb{Z}, |\alpha| \leq 2^{B+\tau}\}$ *is a* τ*-secure masking set for* $\mathcal{C}(B, p)$

Proof. If $a, b \in \mathcal{C}(B, p)$ and $\mathcal{U}$ is the uniform distribution on $\mathcal{C}(B, p, \tau)$, the statistical distance between $a + \mathcal{U}$ and $b + \mathcal{U}$ is $(b - a) \cdot 2^{p-B} / \#\mathcal{C}(B, p, \tau) \leq 2^{-\tau}$. This distance upper-bounds any computational advantage. □

Again, the class $\mathcal{C}(B, p, \tau) = \mathcal{C}(B + \tau, p + \tau)$ fits in floating point numbers of $p + \tau$-bits of mantissa, so they can be used to securely mask fixed point numbers with numerical precision p. By extension, all additive shares for $\mathcal{C}(B, p)$ will be taken in $\mathcal{C}(B, p, \tau)$.

We now analyze what happens if we use Beaver's protocol to multiply two plaintexts $x \in \mathcal{C}(B_1, p)$ and $y \in \mathcal{C}(B_2, p)$. The masked values $x + \lambda$ and $y + \mu$ are bounded by $2^{B_1+\tau}$ and $2^{B_2+\tau}$ respectively. Since the mask λ is also bounded by $2^{B_1+\tau}$, and μ by $2^{B_2+\tau}$, the computed secret shares of $x \cdot y$ will be bounded by $2^{B_1+B_2+2\tau}$, So the lossless multiplication sends $\mathcal{C}(B_1, p, \tau) \times \mathcal{C}(B_2, p, \tau) \to \mathcal{C}(B, 2p, 2\tau)$ where $B = B_1 + B_2$ instead of $\mathcal{C}(B, p, \tau)$. Reducing p is just a matter of rounding, and it is done automatically by the floating point representation. However, we still need a method to reduce τ, so that the output secret shares are bounded by $2^{B+\tau}$.

3.3 Secret share reduction algorithm

The algorithm we propose depends on two auxiliary parameters: the *cutoff*, defined as $\eta = B + \tau$ so that 2^η is the desired bound in absolute value, and an auxiliary parameter $M = 2^\kappa$ larger than the number of players.

The main idea is that the initial share contains large components $z_1, \ldots, z_n$ that sum up to the small secret shared value z. Additionally, the most significant

bits of the share beyond the cutoff position (say $\mathrm{MSB}(z_i) = \lfloor z_i/2^\eta \rceil$) do not contain any information on the data, and are all safe to reveal. We also know that the MSB of the sum of the shares (i.e. MSB of the data) is null, so the sum of the MSB of the shares is very small. The share reduction algorithm simply computes this sum, and redistributes it evenly among the players. Since the sum is guaranteed to be small, the computation is done modulo M rather than on large integers. More precisely, using the cutoff parameter η, for $i = 1, \ldots, n$, player i writes his secret share z_i of z as $z_i = u_i + 2^\eta v_i$, with $v_i \in \mathbb{Z}$ and $u_i \in [-2^{\eta-1}, 2^{\eta-1})$. Then, he broadcasts $v_i \bmod M$, so that each player computes the sum. The individual shares can optionally be re-randomized using a precomputed share $[\![\nu]\!]_+$, with $\nu = 0 \mod M$. Since $w = \sum v_i$'s is guaranteed to be between $-M/2$ and $M/2$, it can be recovered from its representation mod M. Thus, each player locally updates its share as $u_i + 2^\eta w/n$, which have by construction the same sum as the original shares, but are bounded by 2^η. This construction is summarized in Algorithm 3 in Appendix B.

4 Fourier Approximation

Fourier theory allows us to approximate certain periodic functions with trigonometric polynomials. The goal of this section is two-fold: to show how to evaluate trigonometric polynomials in MPC and, at the same time, to review and show extensions of some approximation results to non-periodic functions.

4.1 Evaluation of trigonometric polynomials

Recall that a complex trigonometric polynomial is a finite sum of the form $t(x) = \sum_{m=-P}^{P} c_m e^{imx}$, where $c_m \in \mathbb{C}$ is equal to $a_m + ib_m$, with $a_m, b_m \in \mathbb{R}$. Each trigonometric polynomial is a periodic function with period 2π. If $c_{-m} = \overline{c_m}$ for all $m \in \mathbb{Z}$, then t is real-valued, and corresponds to the more familiar cosine decomposition $t(x) = a_0 + \sum_{m=1}^{N} a_m \cos(mx) + b_m \sin(mx)$. Here, we describe how to evaluate trigonometric polynomials in an MPC context, and explain why it is better than regular polynomials.

We suppose that, for all m, the coefficients a_m and b_m of t are publicly accessible and they are $0 \leq a_m, b_m \leq 1$. As t is 2π periodic, we can evaluate it on inputs modulo 2π. Remark that as $\mathbb{R} \mod 2\pi$ admits a uniform distribution, we can use a uniform masking: this method completely fixes the leakage issues that were related to the evaluation of classical polynomials via multiplicative masking. On the other hand, the output of the evaluation is still in $\mathbb{R}$: in this case we continue using the statistical masking described in previous sections. The inputs are secretly shared and additively masked: for sake of clarity, to distinguish the classical addition over reals from the addition modulo 2π, we temporarily denote this latter by $\oplus$. In the same way, we denote the additive secret shares with respect to the addition modulo 2π by $[\![\cdot]\!]_\oplus$. Then, the transition from $[\![\cdot]\!]_+$ to $[\![\cdot]\!]_\oplus$ can be achieved by trivially reducing the shares modulo 2π.

Then, a way to evaluate t on a secret shared input $[\![x]\!]_+ = (x_1, \ldots, x_n)$ is to convert $[\![x]\!]_+$ to $[\![x]\!]_\oplus$ and additively mask it with a shared masking $[\![\lambda]\!]_\oplus$, then reveal $x \oplus \lambda$ and rewrite our target $[\![e^{imx}]\!]_+$ as $e^{im(x\oplus\lambda)} \cdot [\![e^{im(-\lambda)}]\!]_+$. Indeed, since $x \oplus \lambda$ is revealed, the coefficient $e^{im(x\oplus\lambda)}$ can be computed in clear. Overall, the whole trigonometric polynomial t can be evaluated in a single round of communication, given a precomputed triplet such as $([\![\lambda]\!]_\oplus, [\![e^{-i\lambda}]\!]_+, \ldots, [\![e^{-i\lambda P}]\!]_+)$ and thanks to the fact that $x \oplus \lambda$ has been revealed.

Also, we notice that to work with complex numbers of absolute value 1 makes the method numerically stable, compared to power functions in regular polynomials. It is for this reason that the evaluation of trigonometric polynomials is a better solution in our context.

4.2 Approximating non-periodic functions

If one is interested in uniformly approximating (with trigonometric polynomials on a given interval, e.g. $[-\pi/2, \pi/2]$) a non-periodic function f, one cannot simply use the Fourier coefficients. Indeed, even if the function is analytic, its Fourier series need not converge uniformly near the end-points due to Gibbs phenomenon.

Approximations via C^∞-extensions. One way to remedy this problem is to look for a periodic extension of the function to a larger interval and look at the convergence properties of the Fourier series for that extension. To obtain exponential convergence, the extension needs to be analytic too, a condition that can rarely be guaranteed. In other words, the classical Whitney extension theorem [28] will rarely yield an analytic extension that is periodic at the same time. A constructive approach for extending differentiable functions is given by Hestenes [20] and Fefferman [16] in a greater generality. The best one can hope for is to extend the function to a C^∞-function (which is not analytic). As explained in [8], [9], such an extension yields a super-algebraic approximation at best that is not exponential.

Least-square approximations. An alternative approach for approximating a non-periodic function with a trigonometric functions is to search for these functions on a larger interval (say $[-\pi, \pi]$), such that the restriction (to the original interval) of the L^2-distance between the original function and the approximation is minimized. This method was first proposed by [7], but it was observed that the coefficients with respect to the standard Fourier basis were numerically unstable in the sense that they diverge (for the optimal solution) as one increases the number of basis functions. The method of [21] allows to remedy this problem by using a different orthonormal basis of certain half-range Chebyshev polynomials of first and second kind for which the coefficients of the optimal solution become numerically stable. In addition, one is able to calculate numerically these coefficients using a Gaussian quadrature rule. More details are given in Appendix C.

Approximating the sigmoid function. We now restrict to the case of the sigmoid function over the interval $[-B/2, B/2]$ for some $B > 0$. We can rescale the variable to approximate $g(x) = \texttt{sigmo}(Bx/\pi)$ over $[-\pi/2, \pi/2]$. If we extend g by anti-periodicity (odd-even) to the interval $[\pi/2, 3\pi/2]$ with the mirror condition $g(x) = g(\pi - x)$, we obtain a continuous 2π-periodic piecewise C^1 function. By Dirichlet's global theorem, the Fourier serie of g converges uniformly over $\mathbb{R}$, so for all $\varepsilon > 0$, there exists a degree N and a trigonometric polynomial g_N such that $\|g_N - g\|_\infty \leq \epsilon$. To compute $\texttt{sigmo}(t)$ over secret shared t, we first apply the affine change of variable (which is easy to evaluate in MPC), to get the corresponding $x \in [-\pi/2, \pi/2]$, and then we evaluate the trigonometric polynomial $g_N(x)$ using a Fourier triplet. This method is sufficient to get 24 bits of precision with a polynomial of only 10 terms, however asymptotically, the convergence rate is only in $\Theta(n^{-2})$ due to discontinuities in the derivative of g. In other words, approximating g with λ bits of precision requires to evaluate a trigonometric polynomial of degree $2^{\lambda/2}$. Luckily, in the special case of the sigmoid function, we can make this degree polynomial by explicitly constructing a 2π-periodic analytic function that is exponentially close to the rescaled sigmoid on the whole interval $[-\pi, \pi]$ (not the half interval). Besides, the geometric decay of the coefficients of the trigonometric polynomial ensures perfect numerical stability. The following theorem, whose proof can be found in Appendix D summarizes this construction.

Theorem 1. *Let $h_\alpha(x) = 1/(1+e^{-\alpha x}) - x/2\pi$ for $x \in (-\pi, \pi)$. For every $\varepsilon > 0$, there exists $\alpha = O(\log(1/\varepsilon))$ such that h_α is at uniform distance $\varepsilon/2$ from a 2π-periodic analytic function g. There exists $N = O(\log^2(1/\varepsilon))$ such that the Nth term of the Fourier series of g is at distance $\varepsilon/2$ of g, and thus, at distance $\leq \varepsilon$ from h_α.*

5 Honest but Curious Model

In the previous sections, we defined the shares of multiplication, power and Fourier triplets, but did not explain how to generate them. Of course, a single *trusted dealer* approved by all players (TD model) could generate and distribute all the necessary shares to the players. Since the trusted dealer knows all the masks, and thus all the data, the TD model is only legitimate for few computation outsourcing scenarios.

We now explain how to generate the same triplets efficiently in the more traditional *honest-but-curious* (HBC) model. To do so, we keep an external entity, called again the dealer, who participates in an interactive protocol to generate the triplets, but sees only masked information. Since the triplets in both the HBC and TD models are similar, the online phase is unchanged. Notice that in this HBC model, even if the dealer does not have access to the secret shares, he still has more power than the players. In fact, if one of the players wants to gain information on the secret data, he has to collude with all other players, whereas the dealer would need to collaborate with just one of them.

In what follows, we suppose that, during the offline phase, a private channel exists between each player and the dealer. In the case of an HBC dealer, we also

Algorithm 1 Honest but curious triplets generation for a trigonometric poly

Output: Shares $([\![\lambda]\!], [\![e^{im_1\lambda}]\!]_+, \ldots, [\![e^{im_N\lambda}]\!]_+)$.

1: Each player P_i generates λ_i, a_i (uniformly modulo 2π)
2: Each player P_i broadcasts a_i to all other players.
3: Each player computes $a = a_1 + \cdots + a_n \mod 2\pi$.
4: Each player P_i sends to the dealer $\lambda_i + a_i \mod 2\pi$.
5: The dealer computes $\lambda + a \mod 2\pi$ and $w^{(1)} = e^{im_1(\lambda+a)}, \ldots, w^{(N)} = e^{im_N(\lambda+a)}$
6: The dealer creates $[\![w^{(1)}]\!]_+, \ldots, [\![w^{(N)}]\!]_+$ and sends $w_i^{(1)}, \ldots, w_i^{(N)}$ to player P_i.
7: Each player P_i multiplies each $w_i^{(j)}$ by $e^{-im_j a}$ to get $(e^{im_j\lambda})_i$, for all $j \in [1, N]$.

assume that an additional private broadcast channel (a channel to which the dealer has no access) exists between all the players (Figure 5 in Appendix E). Afterwards, the online phase only requires a public broadcast channel between the players (Figure 6 in Appendix E). In practice, because of the underlying encryption, private channels (e.g., SSL connections) have a lower throughput (generally $\approx$ 20MB/s) than public channels (plain TCP connections, generally from 100 to 1000MB/s between cloud instances).

The majority of HBC protocols proposed in the literature present a scenario with only 2 players. In [11] and [3], the authors describe efficient HBC protocols that can be used to perform a fast MPC multiplication in a model with three players. The two schemes assume that the parties follow correctly the protocol and that two players do not collude. The scheme proposed in [11] is very complex to scale for more than three parties, while the protocol in [3] can be extended to a generic number of players, but requires a quadratic number of private channels (one for every pair of players). We propose a different protocol for generating the multiplicative triplets in the HBC scenario, that is efficient for any arbitrary number n of players. In our scheme, the dealer evaluates the non-linear parts in the triplet generation, over the masked data produced by the players, then he distributes the masked shares. The mask is common to all players, and it is produced thanks to the private broadcast channel that they share. Finally, each player produces his triplet by unmasking the precomputed data received from the dealer.

In Appendix E, we present in detail two algorithms: one for the generation of multiplicative Beaver's triplets (Algorithm 4) and the other for the generation of the triplets used in the computation of the power function (Algorithm 5), both in the honest-but-curious scenario. Following the same ideas, Algorithm 1 describes our triplets generation for the evaluation of a trigonometric polynomial in the HBC scenario.

6 Application to Logistic Regression

In a classification problem one is given a data set, also called a training set, that we will represent here by a matrix $X \in \mathcal{M}_{N,k}(\mathbb{R})$, and a training vector $\mathbf{y} \in \{0,1\}^N$. The data set consists of N input vectors of k features each, and the

coordinate y_i of the vector $\mathbf{y}$ corresponds to the class (0 or 1) to which the i-th element of the data set belongs to. Formally, the goal is to determine a function $h_\theta : \mathbb{R}^k \to \{0, 1\}$ that takes as input a vector $\mathbf{x}$, containing k features, and which outputs $h_\theta(\mathbf{x})$ predicting reasonably well y, the corresponding output value.

In logistic regression typically one uses hypothesis functions $h_\theta : \mathbb{R}^{k+1} \to [0, 1]$ of the form $h_\theta(\mathbf{x}) = \texttt{sigmo}(\theta^T \mathbf{x})$, where $\theta^T \mathbf{x} = \sum_{i=0}^{k} \theta_i x_i \in \mathbb{R}$ and $x_0 = 1$. The vector θ, also called *model*, is the parameter that needs to be determined. For this, a convex *cost function* $C_{\mathbf{x},y}(\theta)$ measuring the quality of the model at a data point $(\mathbf{x}, y)$ is defined as

$$C_{\mathbf{x},y}(\theta) = -y \log h_\theta(\mathbf{x}) - (1 - y) \log(1 - h_\theta(\mathbf{x})).$$

The cost for the whole dataset is thus computed as $\sum_{i=1}^{N} C_{\mathbf{x_i},y_i}(\theta)$. The overall goal is to determine a model θ whose cost function is as close to 0 as possible. A common method to achieve this is the so called *gradient descent* which consists of constantly updating the model θ as

$$\theta := \theta - \alpha \nabla C_{\mathbf{x},y}(\theta),$$

where $\nabla C_{\mathbf{x},y}(\theta)$ is the gradient of the cost function and $\alpha > 0$ is a constant called the *learning rate*. Choosing the optimal α depends largely on the quality of the dataset: if α is too large, the method may diverge, and if α is too small, a very large number of iterations are needed to reach the minimum. Unfortunately, tuning this parameter requires either to reveal information on the data, or to have access to a public fake training set, which is not always feasible in private MPC computations. This step is often silently ignored in the literature. Similarly, preprocessing techniques such as feature scaling, or orthogonalization techniques can improve the dataset, and allow to increase the learning rate significantly. But again, these techniques cannot easily be implemented when the input data is shared, and when correlation information should remain private. In this work, we choose to implement the IRLS method [5, §4.3], which does not require feature scaling, works with learning rate 1, and converges in much less iterations, provided that we have enough floating point precision. In this case, the model is updated as:

$$\theta := \theta - H(\theta)^{-1} \cdot \nabla C_{\mathbf{x},y}(\theta),$$

where $H(\theta)$ is the Hessian matrix.

6.1 Implementation and Experimental Results

We implemented an MPC proof-of-concept of the logistic regression algorithm in C++. We represented numbers in $\mathcal{C}(B, p)$ classes with 128-bit floating point numbers, and set the masking security parameter to $\tau = 40$ bits. Since a 128-bit number has 113 bits of precision, and the multiplication algorithm needs $2\tau = 80$ bits of masking, we still have 33 bits of precision that we can freely use throughout the computation. Since our benchmarks are performed on a regular x86_64

CPU, 128-bit floating point arithmetic is emulated using GCC's quadmath library, however additional speed-ups could be achieved on more recent hardware that natively supports these operations (eg. IBM's next POWER9 processor). In our proof of concept, our main focus was to improve the running time, the floating point precision, and the communication complexity of the online phase, so we implemented the offline phase only for the trusted dealer scenario, leaving the honest but curious dealer variant as a future work.

Algorithm 2 Model training: $\mathsf{Train}(X, \mathbf{y})$

Input: A dataset $X \in \mathcal{M}_{N,k}(\mathbb{R})$ and a training vector $\mathbf{y} \in \{0,1\}^N$
Output: The model $\theta \in \mathbb{R}^k$ that minimizes $\mathsf{Cost}_{X,\mathbf{y}}(\theta)$

1: Precompute $\text{Prods}_i = X_i^T X_i$ for $i \in [0, N-1]$
2: $\theta \leftarrow [0, \ldots, 0] \in \mathbb{R}^k$
3: **for** iter $= 1$ to IRLS_ITERS **do** ▷ In practice IRLS_ITERS $= 8$
4: $\quad \mathbf{a} \leftarrow X \cdot \theta$
5: $\quad \mathbf{p} \leftarrow [\texttt{sigmo}(a_0), \ldots, \texttt{sigmo}(a_{N-1})]$
6: $\quad \text{pmp} \leftarrow [p_0(1-p_0), \ldots, p_{N-1}(1-p_{N-1})]$
7: $\quad \text{grad} \leftarrow X^T(\mathbf{p} - \mathbf{y})$
8: $\quad H \leftarrow \text{pmp} \cdot \text{Prods}$
9: $\quad \theta = \theta - H^{-1} \cdot \text{grad}$
10: **end for**
11: return θ

Model-training algorithm with the IRLS method. The algorithm is explained over the plaintext. In the MPC instantiation, each player gets a secret share for each variables. Every product is evaluated using the bilinear formula of Section 2, and the sigmoid using the Fourier method of Section 4.

We implemented the logistic regression model training described in Algorithm 2. Each iteration of the main loop evaluates the gradient (grad) and the Hessian (H) of the cost function at the current position θ, and solves the Hessian system (line 7) to find the next position. Most of the computation steps are bilinear on large matrices or vectors, and each of them is evaluated via a Beaver triplet in a single round of communication. In step 5 the sigmoid functions are approximated (in parallel) by an odd trigonometric polynomial of degree 23, which provides 20 bits of precision on the whole interval. We therefore use a vector of Fourier triplets, as described in Section 4. The Hessian system (step 9) is masked by two (uniformly random) orthonormal matrices on the left and the right, and revealed, so the resolution can be done in plaintext. Although this method reveals the norm of the gradient (which is predictable anyway), it hides its direction entirely, which is enough to ensure that the final model remains private. Finally, since the input data is not necessarily feature-scaled, it is essential to start from the zero position (step 2) and not a random position, because the first one is guaranteed to be in the IRLS convergence domain.

To build the MPC evaluation of Algorithm 2, we wrote a small compiler to preprocess this high level listing, unroll all for loops, and turn it into a sequence of

instructions on immutable variables (which are read-only once they are affected). More importantly, the compiler associates a single additive mask λ_U to each of these immutable variables U. This solves two important problems that we saw in the previous sections: first, the masking information for huge matrices that are re-used throughout the algorithm are transmitted only once during the whole protocol (this optimization already appears in [25], and in our case, it has a huge impact for the constant input matrix, and their precomputed products, which are re-used in all IRLS iterations). It also mitigates the attack that would retrieve information by averaging its masked distribution, because an attacker never gets two samples of the same distribution. This justifies the choice of 40 bits of security for masking.

During the offline phase, the trusted dealer generates one random mask value for each immutable variable, and secret shares these masks. For all matrix-vector or matrix-matrix products between any two immutable variables U and V (coming from lines 1, 4, 6, 7 and 8 of Alg.2), the trusted dealer also generates a specific multiplication triplet using the masks λ_U of U and λ_V of V. More precisely, he generates and distributes additive shares for $\lambda_U \cdot \lambda_V$ as well as integer vectors/matrices of the same dimensions as the product for the share-reduction phase. These integer coefficients are taken modulo 256 for efficiency reasons.

6.2 Results

We implemented all the described algorithms and we tested our code for two and three parties, using cloud instances on both the AWS and the Azure platforms, having Xeon E5-2666 v3 processors. In our application each instance communicates via its public IP address. Furthermore, we use the zeroMQ library to handle low-level communications between the players (peer-to-peer, broadcast, central nodes etc...).

In the results that are provided in Table 1 in Appendix A, we fixed the number of IRLS iterations to 8, which is enough to reach a perfect convergence for most datasets, and we experimentally verified that the MPC computation outputs the same model as the one with plaintext iterations. We see that for the datasets of 150000 points, the total running time of the online phase ranges from 1 to 5 minutes. This running time is mostly due to the use of emulated quad-float arithmetic, and this MPC computation is no more than 20 times slower than the plaintext logistic regression on the same datasets, if we implement it using the same 128-bit floats (yet, of course, the native double-precision version is much faster). More interestingly, we see that the overall size of the totality of the triplets and the amount of online communications are small: for instance, a logistic regression on 150000 points with 8 features requires only 756MB of triplets per player, and out of it, only 205MB of data are broadcasted during the online phase per player. This is due to the fact that a Fourier triplet is much larger than the value that is masked and exchanged. Because of this, the communication time is insignificant compared to the whole running time, even with regular WAN bandwidth.

Finally, when the input data is guaranteed to be feature-scaled, we can improve the whole time, memory and communication complexities by about 30% by performing 3 classical gradient descent iterations followed by 5 IRLS iterations instead of 8 IRLS iterations. We tested this optimization for both the plaintext and the MPC version and in Appendix A, we show the evolution of the cost function, during the logistic regression, and of the F-score (Figure 1 and 2), depending on the method used.

We have tested our platform on datasets that were provided by the banking industry. For privacy reasons, these datasets cannot be revealed. However, the behaviour described in this paper can be reproduced by generating random data sets, for instance, with Gaussian distribution, setting the acceptance threshold to 0.5%, and adding some noise by randomly swapping a few labels. Readers interested in testing the platform should contact the authors.

Open problems. A first important open question is the indistinguishability of the distributions after our noise reduction algorithm. On a more fundamental level, one would like to find a method of masking using the basis of half-range Chebyshev polynomials defined in the appendix as opposed to the standard Fourier basis. Such a method, together with the exponential approximation, would allow us to evaluate (in MPC) any function in $L^2([-1,1])$.

Acknowledgements

We thank Hunter Brooks, Daniel Kressner and Marco Picasso for useful conversations on data-independent iterative optimization algorithms. We are grateful to Jordan Brandt, Alexandre Duc and Morten Dahl for various useful discussions regarding multi-party computations and privacy-preserving machine learning.

References

1. M. Abadi, A. Chu, I. Goodfellow, H. Brendan McMahan, I. Mironov, K. Talwar, and L. Zhang. Deep learning with differential privacy. *CoRR*, abs/1607.00133, 2016.
2. Y. Aono, T. Hayashi, L. Trieu Phong, and L. Wang. Privacy-preserving logistic regression with distributed data sources via homomorphic encryption. *IEICE Transactions*, 99-D(8):2079–2089, 2016.
3. T. Araki, J. Furukawa, Y. Lindell, A. Nof, and K. Ohara. High-throughput semi-honest secure three-party computation with an honest majority. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October 24-28, 2016*, pages 805–817, 2016.
4. D. Beaver. Efficient Multiparty Protocols Using Circuit Randomization. In *CRYPTO '91*, volume 576 of *Lecture Notes in Computer Science*, pages 420–432. Springer, 1992.
5. A. Björck. *Numerical Methods for Least Squares Problems.* Siam Philadelphia, 1996.

6. D. Bogdanov, S. Laur, and J. Willemson. Sharemind: A framework for fast privacy-preserving computations. In *ESORICS 2008*, pages 192–206. Springer, 2008.
7. J. Boyd. A comparison of numerical algorithms for Fourier extension of the first, second, and third kinds. *J. Comput. Phys.*, 178(1):118–160, May 2002.
8. J. Boyd. Fourier embedded domain methods: Extending a function defined on an irregular region to a rectangle so that the extension is spatially periodic and c^∞. *Appl. Math. Comput.*, 161(2):591–597, February 2005.
9. J. Boyd. Asymptotic fourier coefficients for a C infinity bell (smoothed-"top-hat") & the fourier extension problem. *J. Sci. Comput.*, 29(1):1–24, 2006.
10. K. Chaudhuri and C. Monteleoni. Privacy-preserving logistic regression. In Daphne Koller, Dale Schuurmans, Yoshua Bengio, and Léon Bottou, editors, *Advances in Neural Information Processing Systems 21, Proceedings of the Twenty-Second Annual Conference on Neural Information Processing Systems, Vancouver, British Columbia, Canada, December 8-11, 2008*, pages 289–296. Curran Associates, Inc., 2008.
11. R. Cramer, I. Damgård, and J. B. Nielsen. *Secure Multiparty Computation and Secret Sharing*. Cambridge University Press, 2015.
12. I. Damgård, V. Pastro, N. Smart, and S. Zakarias. Multiparty computation from somewhat homomorphic encryption. In Reihaneh Safavi-Naini and Ran Canetti, editors, *Advances in Cryptology - CRYPTO 2012 - 32nd Annual Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2012. Proceedings*, volume 7417 of *Lecture Notes in Computer Science*, pages 643–662. Springer, 2012.
13. I. Damgård, V. Pastro, N. P. Smart, and S. Zakarias. SPDZ Software. `https://www.cs.bris.ac.uk/Research/CryptographySecurity/SPDZ/`.
14. Dataset. Arcene Data Set. `https://archive.ics.uci.edu/ml/datasets/Arcene`.
15. Dataset. MNIST Database. `http://yann.lecun.com/exdb/mnist/`.
16. C. Fefferman. Interpolation and extrapolation of smooth functions by linear operators. *Rev. Mat. Iberoamericana*, 21(1):313–348, 2005.
17. 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*, 4:248–267, 2017.
18. R. Gilad-Bachrach, N. Dowlin, K. Laine, K. E. Lauter, M. Naehrig, and J. Wernsing. Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy. In *Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016*, pages 201–210, 2016.
19. I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.
20. M. R. Hestenes. Extension of the range of a differentiable function. *Duke Math. J.*, 8:183–192, 1941.
21. D. Huybrechs. On the fourier extension of nonperiodic functions. *SIAM J. Numerical Analysis*, 47(6):4326–4355, 2010.
22. A. Jäschke and F. Armknecht. Accelerating homomorphic computations on rational numbers. In *ACNS 2016*, volume 9696 of *LNCS*, pages 405–423. Springer, 2016.
23. Y. Lindell and B. Pinkas. Privacy preserving data mining. In *Advances in Cryptology - CRYPTO 2000, 20th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24, 2000, Proceedings*, pages 36–54, 2000.
24. R. Livni, S. Shalev-Shwartz, and O. Shamir. On the computational efficiency of training neural networks. In Zoubin Ghahramani, Max Welling, Corinna Cortes,

Neil D. Lawrence, and Kilian Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 2014, December 8-13 2014, Montreal, Quebec, Canada*, pages 855–863, 2014.

25. P. Mohassel and Y. Zhang. SecureML: A system for scalable privacy-preserving machine learning. In *2017 IEEE Symposium on Security and Privacy, SP 2017, San Jose, CA, USA, May 22-26, 2017*, pages 19–38. IEEE Computer Society, 2017.
26. V. Nikolaenko, U. Weinsberg, S. Ioannidis, M. Joye, D. Boneh, and N. Taft. Privacy-preserving ridge regression on hundreds of millions of records. In *2013 IEEE Symposium on Security and Privacy, SP 2013, Berkeley, CA, USA, May 19-22, 2013*, pages 334–348. IEEE Computer Society, 2013.
27. L. Trieu Phong, Y. Aono, T. Hayashi, L. Wang, and S. Moriai. Privacy-preserving deep learning: Revisited and enhanced. In Lynn Batten, Dong Seong Kim, Xuyun Zhang, and Gang Li, editors, *Applications and Techniques in Information Security - 8th International Conference, ATIS 2017, Auckland, New Zealand, July 6-7, 2017, Proceedings*, volume 719 of *Communications in Computer and Information Science*, pages 100–110. Springer, 2017.
28. H. Whitney. Analytic extensions of differentiable functions defined in closed sets. *Trans. Amer. Math. Soc.*, 36(1):63–89, 1934.
29. S. Wu, T. Teruya, J. Kawamoto, J. Sakuma, and H. Kikuchi. Privacy-preservation for stochastic gradient descent application to secure logistic regression. *The 27th Annual Conference of the Japanese Society for Artificial Intelligence*, 27:1–4, 2013.

A Timings for $n = 3$ players

We present in this section a table (Table 1) summarizing the different measures we obtained during our experiments for $n = 3$ players. For this we considered datasets containing from 10000 to 1500000 points having 8, 12 or 20 features each.

Dataset size N	# features k	communication size (MB)	Precomputed data size (MB)	Time (sec) offline phase	Time (sec) online phase
10000	8	13.75	50.55	20.07	6.51
10000	12	21.88	66.18	26.6	9.81
10000	20	45.97	113.1	46.26	19.83
25000	8	34.2	126.23	51.59	19.24
25000	12	54.52	165.14	68.14	24.7
25000	20	114.5	281.98	115.56	48.8
50000	8	68.53	252.35	103.41	32.89
50000	12	108.93	330.1	135.07	49.99
50000	20	228.7	563.46	229.17	103.3
100000	8	137	504.6	205.53	67.11
100000	12	217.75	659.96	269.04	99.99
100000	20	457.1	1126.41	457.33	205.28
150000	8	205.48	756.84	308.14	101.36
150000	12	326.56	989.83	343.86	152.41
150000	20	685.51	1689.36	685.74	314.4

Table 1. Summary of the different measures (time, amount of exchanged data and amount of precomputed data) for $n = 3$ players.

B Mask reduction algorithm

Algorithm 3 details our method, described in Section 3.3, for reducing the size of the secret shares. This procedure is used inside the classical MPC multiplication involving floating points.

Algorithm 3 Mask reduction

Input: $[\![z]\!]_+$ and one triplet $[\![\nu]\!]_+$, with $\nu = 0 \mod M$.

Output: Secret shares for the same value z with smaller absolute values of the shares.

1: Each player P_i computes $u_i \in [-2^{\eta-1}, 2^{\eta-1})$ and $v_i \in \mathbb{Z}$, such that $z_i = u_i + 2^\eta v_i$.
2: Each player P_i broadcasts $v_i + \nu_i \mod M$ to other players.
3: The players compute $w = \frac{1}{n}(\sum_{i=1}^{n}(v_i + \nu_i) \mod M)$.
4: Each player P_i computes the new share of z as $z_i' = u_i + 2^\eta w$

Fig. 1. Evolution of the cost function depending on the method

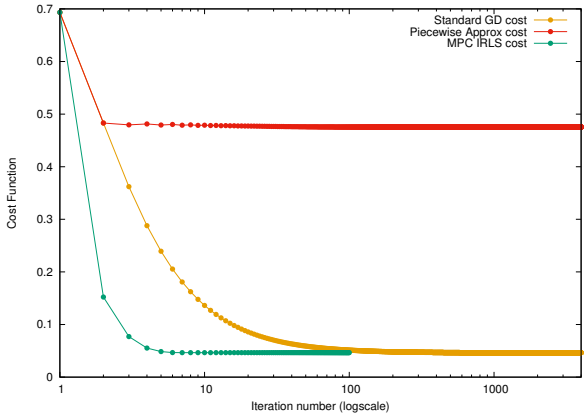


Figure 1 shows the evolution of the cost function during the logistic regression as a function of the number of iterations, on a test dataset of 150000 samples, with 8 features and an acceptance rate of 0.5%. In yellow is the standard gradient descent with optimal learning rate, in red, the gradient descent using the piecewise linear approximation of the sigmoid function (as in [25]), and in green, our MPC model (based on the IRLS method). The MPC IRLS method (as well as the plaintext IRLS) method converge in less than 8 iterations, against 500 iterations for the standard gradient method. As expected, the approx method does not reach the minimal cost.

Fig. 2. Evolution of the F-score depending on the method

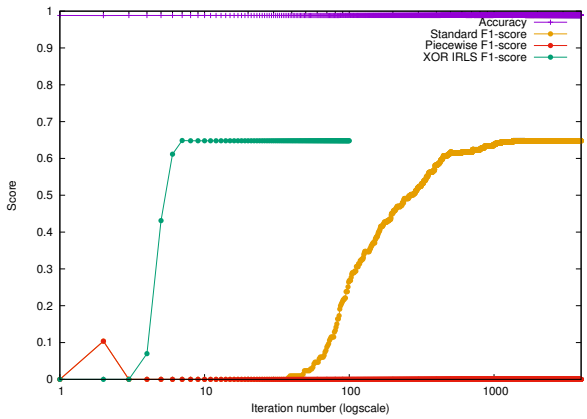


Figure 2 shows the evolution of the F-score during the same logistic regression as a function of the number of iterations. The standard gradient descent and our MPC produce the same model, with a limit F-score of 0.64. However, no positive samples are detected by the piecewise linear approximation, leading to a null F-score. However, in the three cases, the accuracy (purple) is nearly 100% from the first iteration.

C Approximation of functions by trigonometric polynomial over the half period

Let f be a square-integrable function on the interval $[-\pi/2, \pi/2]$ that is not necessarily smooth or periodic.

C.1 The approximation problem

Consider the set

$$G_n = \left\{ g(x) = \frac{a_0}{2} + \sum_{k=1}^{n} a_k \sin(kx) + \sum_{k=1}^{n} b_k \cos(kx) \right\}$$

of 2π-periodic functions and the problem

$$g_n(x) = \operatorname{argmin}_{g \in G_n} \|f - g\|_{L^2_{[-\pi/2, \pi/2]}}.$$

As it was observed in [7], if one uses the naïve basis to write the solutions, the Fourier coefficients of the functions g_n are unbounded, thus resulting in numerical instability. It was explained in [21] how to describe the solution in terms of two families of orthogonal polynomials closely related to the Chebyshev polynomials of the first and second kind. More importantly, it is proved that the solution converges to f exponentially rather than super-algebraically and it is shown how to numerically estimate the solution $g_n(x)$ in terms of these bases.

We will now summarize the method of [21]. Let

$$C_n = \frac{1}{\sqrt{2}} \cup \{\cos(kx) \colon k = 1, \ldots, n\}.$$

and let $\mathfrak{C}_n$ be the $\mathbb{R}$-vector space spanned by these functions (the subspace of even functions). Similarly, let

$$S_n = \{\sin(kx) \colon k = 1, \ldots, n\},$$

and let $\mathfrak{S}_n$ be the $\mathbb{R}$-span of S_n (the space of odd functions). Note that $C_n \cup S_n$ is a basis for G_n.

C.2 Chebyshev's polynomials of first and second kind

Let $T_k(y)$ for $y \in [-1, 1]$ be the kth Chebyshev polynomial of first kind, namely, the polynomial satisfying $T_k(\cos\theta) = \cos k\theta$ for all θ and normalized so that $T_k(1) = 1$ (T_k has degree k). As k varies, these polynomials are orthogonal with respect to the weight function $w_1(y) = 1/\sqrt{1-y^2}$. Similarly, let $U_k(y)$ for $y \in [-1, 1]$ be the kth Chebyshev polynomial of second kind, i.e., the polynomial satisfying $U_k(\cos\theta) = \sin((k+1)\theta)/\sin\theta$ and normalized so that $U_k(1) = k+1$. The polynomials $\{U_k(y)\}$ are orthogonal with respect to the weight function $w_2(y) = \sqrt{1-y^2}$.

It is explained in [21, Thm.3.3] how to define a sequence $\{T_k^h\}$ of half-range Chebyshev polynomials that form an orthonormal bases for the space of even functions. Similarly, [21, Thm.3.4] yields an orthonormal basis $\{U_k^h\}$ for the odd functions (the half-range Chebyshev polynomials of second kind). According to [21, Thm.3.7], the solution g_n to the above problem is given by

$$g_n(x) = \sum_{k=0}^{n} a_k T_k^h(\cos x) + \sum_{k=0}^{n-1} b_k U_k^h(\cos x) \sin x,$$

where

$$a_k = \frac{2}{\pi} \int_{-\frac{\pi}{2}}^{\frac{\pi}{2}} f(x) T_k^h(\cos x) dx,$$

and

$$b_k = \frac{2}{\pi} \int_{-\frac{\pi}{2}}^{\frac{\pi}{2}} f(x) U_k^h(\cos x) \sin x dx.$$

While it is numerically unstable to express the solution g_n in the standard Fourier basis, it is stable to express them in terms of the orthonormal basis $\{T_k^h\} \cup \{U_k^h\}$. In addition, it is shown in [21, Thm.3.14] that the convergence is exponential. To compute the coefficients a_k and b_k numerically, one uses Gaussian quadrature rules as explained in [21, §5].

D Proof of Theorem 1

We now prove Theorem 1, with the following methodology. We first bound the successive derivatives of the sigmoid function using a differential equation. Then, since the first derivative of the sigmoid decays exponentially fast, we can sum all its values for any x modulo 2π, and construct a C^∞ periodic function, which approximates tightly the original function over $[-\pi, \pi]$. Finally, the bounds on the successive derivatives directly prove the geometric decrease of the Fourier coefficients.

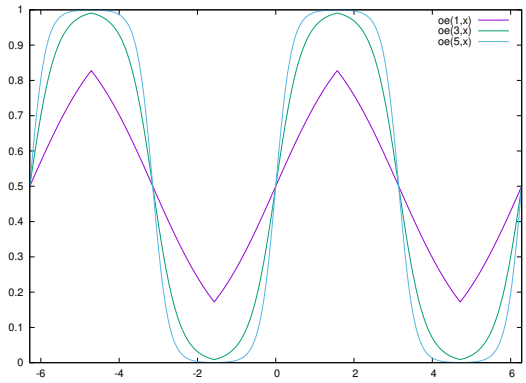
Proof. First, consider the $\sigma(x) = 1/(1 + e^{-x})$ the sigmoid function over $\mathbb{R}$. σ satisfies the differential equation $\sigma' = \sigma - \sigma^2$. By derivating n times, we have $\sigma^{(n+1)} = \sigma^{(n)} - \sum_{k=0}^{n} \binom{n}{k} \sigma^{(k)} \sigma^{(n-k)} = \sigma^{(n)}(1 - \sigma) - \sum_{k=1}^{n} \binom{n}{k} \sigma^{(k)} \sigma^{(n-k)}$. Dividing by $(n+1)!$, this yields

$$\left| \frac{\sigma^{(n+1)}}{(n+1)!} \right| \leq \frac{1}{n+1} \left(\left| \frac{\sigma^{(n)}}{n!} \right| + \sum_{k=1}^{n} \left| \frac{\sigma^{(k)}}{k!} \right| \left| \frac{\sigma^{(n-k)}}{(n-k)!} \right| \right)$$

From there, we deduce by induction that for all $n \geq 0$ and for all $x \in \mathbb{R}$, $\left| \frac{\sigma^{(n)}(x)}{n!} \right| \leq 1$ and it decreases with n, so for all $n \geq 1$,

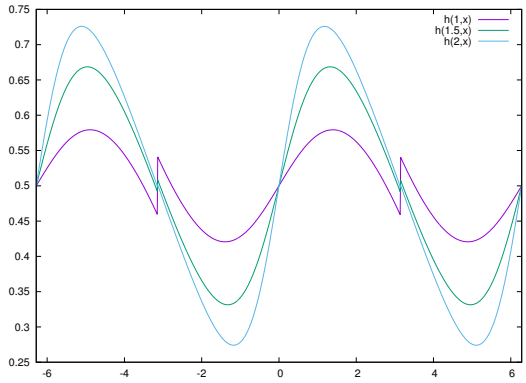
$$\left| \sigma^{(n)}(x) \right| \leq n! \, \sigma'(x) \leq n! \, e^{-|x|}$$

Fig. 3. Odd-even periodic extension of the rescaled sigmoid



The rescaled sigmoid function $g(\alpha x)$ is extended by anti-periodicity from $[-\frac{\pi}{2};\frac{\pi}{2}]$ to $[\frac{\pi}{2};\frac{3\pi}{2}]$. This graph shows the extended function for $\alpha = 1, 3, 5$. By symmetry, the Fourier serie of the output function has only odd sinus terms: $0.5+\sum_{n\in\mathbb{N}} a_{2n+1}\sin((2n+1)x)$. For $\alpha = 20/\pi$, the first Fourier form a rapidly decreasing sequence: [6.12e-1, 1.51e-1, 5.37e-2, 1.99e-2, 7.41e-3, 2.75e-3, 1.03e-3, 3.82e-4, 1.44e-4, 5.14e-5, 1.87e-5, ...], which rapidly achieves 24 bits of precision. However, the sequence asymptotically decreases in $O(n^{-2})$ due to the discontinuity in the derivative in $\frac{-\pi}{2}$, so this method is not suitable to get an exponentially good approximation.

Fig. 4. Asymptotic approximation of the sigmoid via Theorem 1



As α grows, the discontinuity in the rescaled sigmoid function $g(\alpha x) - \frac{x}{2\pi}$ vanishes, and it gets exponentially close to an analytic periodic function, whose Fourier coefficients decrease geometrically fast. This method is numerically stable, and can evaluate the sigmoid with arbitrary precision in polynomial time.

We now construct a periodic function that should be very close to the derivative of h_α: consider $g_\alpha(x) = \sum_{k\in\mathbb{Z}} \frac{-\alpha}{(1+e^{-\alpha(x-2k\pi)})(1+e^{\alpha(x-2k\pi)})}$. By summation of geometric series, g_α is a well-defined infinitely derivable 2π-periodic function over $\mathbb{R}$. We can easily verify that for all $x \in (-\pi, \pi)$, the difference $\left|h'_\alpha(x) - \frac{1}{2\pi} - g_\alpha(x)\right|$ is bounded by $2\alpha \cdot \sum_{k=1}^{\infty} e^{\alpha(x-2k\pi)} \leq \frac{2\alpha e^{-\alpha\pi}}{1-e^{-2\pi\alpha}}$, so by choosing $\alpha = \Theta(\log(\frac{1}{\varepsilon}))$, this difference can be made smaller than $\frac{\varepsilon}{2}$.

We suppose now that α is fixed and we prove that g_α is analytic, i.e. its Fourier coefficients decrease exponentially fast. By definition, $g_\alpha(x) = \sum_{k\in\mathbb{Z}} \sigma(\alpha(x - 2k\pi))$, so for all $p \in \mathbb{N}$, $g_\alpha^{(p)}(x) = \alpha^{p+1} \sum_{k\in\mathbb{Z}} \sigma^{(p+1)}(\alpha x - 2\alpha k\pi)$, so $\left\|g_\alpha^{(p)}\right\|_\infty \leq 2\alpha^{p+1}(p+1)!$. This proves that the n-th Fourier coefficient $c_n(g_\alpha)$ is $\leq \min_{p\in\mathbb{N}} \frac{2\alpha^{p+1}(p+1)!}{n^p}$. This minimum is reached for $p + 1 \approx \frac{n}{\alpha}$, and yields $|c_n(g_\alpha)| = O(e^{-n/\alpha})$.

Finally, this proves that by choosing $N \approx \alpha^2 = \Theta(\log(1/\varepsilon)^2)$, the N-th term of the Fourier serie of g_α is at distance $\leq \frac{\varepsilon}{2}$ of g_α, and thus from $h'_\alpha - \frac{1}{2\pi}$. This bound is preserved by integrating the trigonometric polynomial (the g from the theorem is the primitive of g_α), which yields the desired approximation of the sigmoid over the whole interval $(-\pi, \pi)$. □

E Honest but curious model

E.1 Honest but curious communication channels

The figures presented in this section represent the communication channels between the players and the dealer in both the trusted dealer and the honest but curious models. Two types of communication channels are used: the private channels, that correspond in practice to SSL channels (generally $<$ 20MB/s), and the public channels, corresponding in practice to TCP connections (generally from 100MB to 1GB/s). In the figures, private channels are represented with dashed lines, while public channels are represented with plain lines.

Figure 5 illustrates the connections during the offline phase of the MPC protocols. In the TD model, the dealer is the only one generating all the precomputed data. He uses private channels to send to each player his share of the triplets (one-way arrows). In the HBC model, the players collaborate for the generation of the triplets. To do that, they need an additional private broadcast channel between them, that is not accessible to the dealer.

Figure 6 represents the communication channels between players during the online phase. The online phase is the same in both the TD and the HBC models and the dealer is not present.

E.2 Honest but curious algorithms

In this section we give two detailed algorithms in the honest but curious model, already described in Section 5. The first algorithm (Algorithm 4) describes the

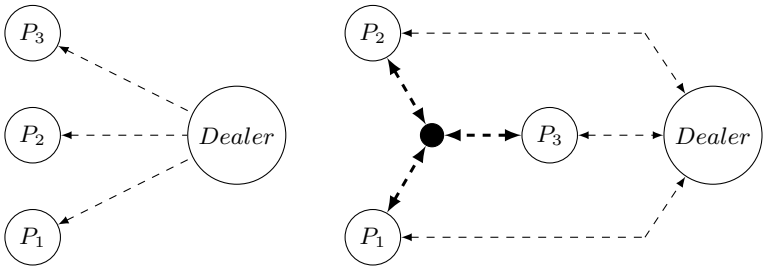


Fig. 5. Communication channels in the offline phase - The figure represents the communication channels in both the *trusted dealer model* (left) and in the *honest but curious model* (right) used during the offline phase. In the first model, the dealer sends the triplets to each player via a private channel. In the second model, the players have access to a private broadcast channel, shared between all of them and each player shares an additional private channel with the dealer. The private channels are denoted with dashed lines. The figure represents 3 players, but each model can be extended to an arbitrary number n of players.

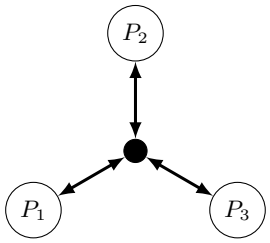


Fig. 6. Communication channels in the online phase - The figure represents the communication channels (the same type for both the honest but curious and the trusted dealer model) used during the online phase. The players send and receive masked values via a public broadcast channel (public channels are denoted with plain lines). Their number, limited to 3 in the example, can easily be extended to a generic number n of players.

generation of Beaver's multiplicative triplets, while the second algorithm (Algorithm 5) details the generation of the triplets used in the MPC computation of a power function.

In both algorithms, the dealer and the players collaborate for the generation of triplets and none of them is supposed to have access to the whole information. The general idea is that the players generate their secret shares (of λ and μ, in the first case, and of λ only, in the second case), that each one keeps secret. They also generate secret shares of a common mask, that they share between each other via the broadcast channel, but which remains secret to the dealer. The player then mask their secret shares with the common mask and sends them to the dealer, who evaluates the non-linear parts (product in Algorithm 4 and

power in Algorithm 5). The dealer generates new additive shares for the result and sends these values back to each player via the private channel. This way, the players don't know each other's shares. Finally, the players, who know the common mask, can independently unmask their secret shares, and obtain their final share of the triplet, which is therefore unknown to the dealer.

Algorithm 4 Honest but curious triplets generation

Output: Shares $([\![\lambda]\!], [\![\mu]\!], [\![z]\!])$ with $z = \lambda\mu$.

1: Each player P_i generates $a_i, b_i, \lambda_i, \mu_i$ (from the according distribution).
2: Each player P_i shares with all other players a_i, b_i.
3: Each player computes $a = a_1 + \cdots + a_n$ and $b = b_1 + \cdots + b_n$.
4: Each player P_i sends to the dealer $a_i + \lambda_i$ and $b_i + \mu_i$.
5: The dealer computes $a + \lambda$, $b + \mu$ and $w = (a + \lambda)(b + \mu)$.
6: The dealer creates $[\![w]\!]_+$ and sends w_i to player P_i, for $i = 1, \ldots, n$.
7: Player P_1 computes $z_1 = w_1 - ab - a\mu_1 - b\lambda_1$.
8: Player i for $i = 2, \ldots n$ computes $z_i = w_i - a\mu_i - b\lambda_i$.

Algorithm 5 Honest but curious triplets generation for the power function

Output: Shares $[\![\lambda]\!]$ and $[\![\lambda^{-\alpha}]\!]$.

1: Each player P_i generates λ_i, a_i (from the according distribution).
2: Each player P_i shares with all other players a_i.
3: Each player computes $a = a_1 + \cdots + a_n$.
4: Each player P_i generates z_i in a way that $\sum_{i=1}^{n} z_i = 0$.
5: Each player P_i sends to the dealer $z_i + a\lambda_i$.
6: The dealer computes $\mu\lambda$ and $w = (\mu\lambda)^{-\alpha}$.
7: The dealer creates $[\![w]\!]_+$ and sends w_i to player P_i, for $i = 1, \ldots, n$.
8: Each player P_i right-multiplies w_i with μ^{α} to obtain $(\lambda^{-\alpha})_i$.