Fast Secure Comparison for Medium-Sized Integers and Its Application in Binarized Neural Networks

Mark Abspoel[1,2](✉), Niek J. Bouman[3](✉), Berry Schoenmakers[3], and Niels de Vreede[3]

[1] Centrum Wiskunde & Informatica (CWI), Amsterdam, The Netherlands
abspoel@cwi.nl
[2] Philips Research Eindhoven, Eindhoven, The Netherlands
[3] Technische Universiteit Eindhoven, Eindhoven, The Netherlands
n.j.bouman@tue.nl

Abstract. In 1994, Feige, Kilian, and Naor proposed a simple protocol for secure 3-way comparison of integers a and b from the range $[0, 2]$. Their observation is that for $p = 7$, the Legendre symbol $(x \mid p)$ coincides with the sign of x for $x = a - b \in [-2, 2]$, thus reducing secure comparison to secure evaluation of the Legendre symbol. More recently, in 2011, Yu generalized this idea to handle secure comparisons for integers from substantially larger ranges $[0, d]$, essentially by searching for primes for which the Legendre symbol coincides with the sign function on $[-d, d]$. In this paper, we present new comparison protocols based on the Legendre symbol that additionally employ some form of error correction. We relax the prime search by requiring that the Legendre symbol encodes the sign function in a noisy fashion only. Practically, we use the majority vote over a window of $2k+1$ adjacent Legendre symbols, for small positive integers k. Our technique significantly increases the comparison range: e.g., for a modulus of 60 bits, d increases by a factor of 2.8 (for $k = 1$) and 3.8 (for $k = 2$) respectively. We give a practical method to find primes with suitable noisy encodings.

We demonstrate the practical relevance of our comparison protocol by applying it in a secure neural network classifier for the MNIST dataset. Concretely, we discuss a secure multiparty computation based on the binarized multi-layer perceptron of Hubara et al., using our comparison for the second and third layers.

1 Introduction

Secure integer comparison has been a primitive of particular interest since the inception of multiparty computation (MPC). In 1982, even before general multiparty computation had been realized, Yao introduced the *Millionaires' Problem* [21], where two millionaires want to determine who of them has greater wealth without revealing any information beyond the outcome of this comparison to

M. Matsui (Ed.): CT-RSA 2019, LNCS 11405, pp. 453–472, 2019.
https://doi.org/10.1007/978-3-030-12612-4_23

each other or to any third party. Secure comparison has been investigated extensively since. A whole range of solutions is available with every solution aiming for a particular trade-off. Nonetheless, with respect to arithmetic-secret-sharing-based MPC, secure comparison remains among the most expensive basic operations in terms of round complexity. Hence, for applications that require many comparisons, achieving high throughput (important for privacy-preserving data processing applications) or low latency (crucial for certain applications, like blind auctions for real-time advertisement sales) can be challenging.

1.1 Related Work

Whereas most secure comparison protocols work over finite fields of arbitrary order, Yu [22] presents a comparison protocol that only works for specifically chosen prime moduli. Although this clearly poses a restriction in terms of applicability, the main benefit is that the specifically chosen prime modulus p enables Yu to perform a comparison *in a single round of communication* in the online phase (the offline preprocessing phase requires three communication rounds), albeit in a range that is small compared to p (see Sect. 3.4 for explicit bounds). Namely, he chooses p such that the pattern of quadratic residues and non-residues modulo p coincides with the sign function on a given interval symmetric around zero, which is an idea that goes back to a protocol due to Feige, Kilian, and Naor [3], who use it to compute the sign of an element $x \in [-2, 2]$ in $\mathbb{F}_7$. Yu's comparison protocol for comparing arbitrary elements $a, b \in \mathbb{F}_p$ essentially works by breaking up the full-range comparison into several medium-range comparisons of the above type by performing a digit decomposition.

1.2 This Paper

In this paper, we pursue the line of work initiated by Yu [22]. Our main contribution is that we achieve an improvement in the comparison range while keeping the bit-length of the prime modulus fixed. Concretely, we propose a protocol that, for a fixed prime-length, achieves close to a *two-fold increase of the comparison range* (over Yu's results), while still enjoying a single-round online phase, at the cost of a constant amount of additional communication and some additional local computations. Also, we present a two-online-rounds protocol that achieves more than a *three-fold increase in the comparison range* when compared to Yu's approach. In other words, to compare two integers that lie in a given range (symmetric around zero), our methods require a smaller prime than the prime required for the protocol from [22]. Keeping the finite-field modulus as small as possible or within the machine's word size could be important, for example, in a setting where MPC protocols run on constrained hardware platforms. On such platforms, the complexity of prime-field arithmetic (which is directly related to the prime size) can have a significant impact on the runtime performance. Our protocols can be found in Sect. 5.

The main idea is to somewhat relax the constraints on the prime modulus p: instead of requiring that the Legendre symbols of *all* elements in the interval

$[-d, d]$, for a given positive integer d, coincide with the sign function, we only require this coincidence for *most* elements (in a specific sense). Let us, for some fixed prime p, say that there is an *error* at position $x \in [-d, d]$ if $(x \mid p) \neq \mathrm{sgn}(x)$. Our improvement is based on exploiting a "local redundancy" property enjoyed by the sign function that lets us correct such errors as long as they are sufficiently "sparse", by means of inspecting also the Legendre symbols of some neighboring positions and then performing a majority vote.

This new approach raises the question of how to find primes that give rise to increased ranges. In Sect. 4, we present some results that considerably simplify this search, including tables of suitable primes for various bit lengths.

1.3 Application: Efficient Neural Network Evaluation in MPC

To demonstrate the practical value of our work, we apply our new comparison protocol to the problem of securely evaluating a neural network, in which the sign function is used as non-linearity. We use a binarized multi-layer perceptron (BMLP) for recognizing handwritten digits, as described in [7], which is trained (in the clear) on the well-known MNIST handwritten-digits data set. We consider an MPC scenario in which the input images are secret-shared between the parties, which then securely evaluate the BMLP to obtain the estimated digit in secret-shared form.

2 Preliminaries

Arithmetic Black Box. We suppose that we are given a secure arithmetic black-box (ABB) functionality that can securely evaluate multiplication and linear forms over the finite field $\mathbb{F}_p$. We write $[x]$ for the residue class $x \in \mathbb{F}_p$ encrypted under the ABB (e.g., x is secret-shared among a set of parties, or perhaps encrypted under some homomorphic encryption scheme). Abusing notation, for small $x \in \mathbb{F}_p$ we will also refer to x as an integer in $\mathbb{Z}$, given as the canonical lift of the residue class to the integers $[-\lfloor \frac{p}{2} \rfloor, \lfloor \frac{p}{2} \rfloor]$.

Sign vs. Binary Sign. The sign function and the binary sign function are respectively defined as

$$\mathrm{sgn}(z) = \begin{cases} 1 & \text{if } z > 0, \\ 0 & \text{if } z = 0, \\ -1 & \text{if } z < 0. \end{cases} \qquad \mathrm{bsgn}(z) = \begin{cases} 1 & \text{if } z \geq 0, \\ -1 & \text{if } z < 0. \end{cases}$$

Comparing two integers a and b is achieved by evaluating the sign (or bsgn) of their difference $a - b$. The sgn function gives rise to a three-way comparison, while the bsgn function corresponds to two-way comparison. In this paper, we will start our analysis in terms of the sgn function, but for reasons that will become clear later our protocols evaluate the bsgn function (i.e., achieve two-way comparison). We will sometimes be a bit sloppy and use the word "sign" also for the bsgn function; the precise meaning should nonetheless still be clear from its context.

The Legendre Symbol. Recall that for any odd prime p and any integer a, the Legendre symbol is defined as the integer

$$(a \mid p) = \begin{cases} 0 & \text{if } a \equiv 0 \pmod{p}, \\ 1 & \text{if } a \text{ is a quadratic residue modulo } p, \\ -1 & \text{otherwise.} \end{cases}$$

The Legendre symbol is a completely multiplicative function, which means that $(a \mid p)\,(b \mid p) = (ab \mid p)$ for all $a, b \in \mathbb{Z}$. The identity $(a \mid p) \equiv a^{\frac{p-1}{2}} \pmod{p}$ is known as *Euler's criterion.* The *law of quadratic reciprocity* asserts that for odd primes p and q,

$$(p \mid q)\,(q \mid p) = (-1)^{\frac{p-1}{2}\frac{q-1}{2}}.$$

Securely Evaluating Legendre Symbols. In principle, we can securely evaluate the Legendre symbol via Euler's criterion, which would require $O(\log p)$ secure multiplications. The complete multiplicativity of the Legendre symbol enables the following constant-rounds protocol for securely evaluating the Legendre symbol in the preprocessing model with an single-round online phase. In the preprocessing phase, we generate a secret-shared pair $([r], [(r \mid p)])$ of a random non-zero class r together with its Legendre symbol. In the online (input-dependent) phase, we securely multiply $[a] \cdot [r]$, open the result and then compute

$$[(a \mid p)] = (ar \mid p)\,[(r \mid p)]\,.$$

Note that the security of the protocol requires that $a \not\equiv 0 \pmod{p}$, which should be taken into account when using this protocol.

Blum Primes. A prime p for which $p \equiv 3 \pmod{4}$ is called a *Blum prime.* By Euler's criterion, -1 is a quadratic non-residue modulo p if and only if p is a Blum prime. Hence, for any Blum prime p, the map $x \mapsto (x \mid p)$ is an odd function for $x \in [-\lfloor p/2 \rfloor, \lfloor p/2 \rfloor]$ (which follows immediately from the multiplicativity property of the Legendre symbol), i.e., it enjoys the same symmetry around the origin as the sign function.

3 Evaluating the Sign Function Using Legendre Symbols

3.1 Redundancy Property of the Sign Function

In this section we show that the sign function enjoys a "local redundancy" property, which lets us correct sign-flip errors by means of majority-decoding as long as those errors occur sparsely (in a sense defined below).

Definition 1. *Let $k \geq 0$ be an integer, and let $\mathcal{T} = [t_1, t_2]$ be an interval of integers with $t_2 - t_1 \geq 2k$. We say that a function $e : \mathcal{T} \to \{0, 1\}$ is an* error function on $\mathcal{T}$ admissible for k *if $e(x) = 0$ for all $x \in [-(k+1), k+1] \cap \mathcal{T}$ and if $\sum_{i=-k}^{k} e(y+i) \leq k$ holds for all $y \in [t_1 + k, t_2 - k]$.*

Lemma 1. *Let k and $\mathcal{T}$ be as in Definition 1, and let e be an error function on $\mathcal{T}$ admissible for k. Then,*

$$\operatorname{sgn}\Bigg(\sum_{i=-k}^{k}(-1)^{e(x+i)}\operatorname{sgn}(x+i)\Bigg)=\operatorname{sgn}(x)$$

holds for all $x \in [t_1+k, t_2-k]$.

The proof will clarify why we require in Definition 1 that an admissible error function $e(x)$ has an "error-free" region around $x = 0$; informally speaking, the reason is that the sign function undergoes its sign change at $x = 0$, which means that there is "less room" for errors under majority-decoding in this region.

Proof. We will prove the statement for $\mathcal{T} = [-a, a]$ where $a \geq k$ is any integer. This implies the claim for any subinterval of $\mathcal{T}$ of cardinality at least $2k+1$. Note that because of symmetry (in the sign function as well as in the definition of an admissible error function), it suffices to prove the statement for $x \geq 0$. We distinguish three cases for x. If $x = 0$, we have $\sum_{i=-k}^{k}(-1)^{e(i)}\operatorname{sgn}(i) = \sum_{i=-k}^{k}\operatorname{sgn}(i) = \operatorname{sgn}(x) = 0$, where the first equality follows because e is admissible for k and the second equality follows from the fact that summing an odd function over an interval symmetric around zero gives the value zero.

Second, if $x > k$, we have

$$\sum_{i=-k}^{k}(-1)^{e(x+i)}\operatorname{sgn}(x+i)=\sum_{i=-k}^{k}(-1)^{e(x+i)}>0,$$

where the equality follows because $\operatorname{sgn}(x+i) = 1$ for all $i \in [-k, k]$ and the inequality follows because e is admissible for k.

For the third (and final) case, suppose that $x \in [1, k]$. We have

$$\begin{aligned}\sum_{i=-k}^{k}(-1)^{e(x+i)}\operatorname{sgn}(x+i) &= \sum_{i=-k}^{k-x+1}\operatorname{sgn}(x+i)+\sum_{i'=k-x+2}^{k}(-1)^{e(x+i')}\operatorname{sgn}(x+i')\\ &= \sum_{j=x-k}^{k+1}\operatorname{sgn}(j)+\sum_{j'=k+2}^{k+x}(-1)^{e(j')}\operatorname{sgn}(j')\\ &= 1+x+\sum_{j'=k+2}^{k+x}(-1)^{e(j')}\\ &\geq (1+x)+(1-x)=2\end{aligned}$$

$\square$

3.2 The Legendre Symbol as a "Noisy" Sign

Suppose that p is a Blum prime. We can view the Legendre symbol $(x \mid p)$ for $x \in \mathbb{F}_p$ as a "noisy" version of the sign of x:

$$(x \mid p) = (-1)^{e(x)}\operatorname{sgn}(x), \tag{1}$$

where $e(x)$ is the error function that is determined by p. If we now plug (1) into Lemma 1, we can conclude that we may compute the sign of x as the sign of the sum of the Legendre symbols of positions in a length-$(2k+1)$ interval centered at x, for all $x \in [t_1 + k, t_2 - k]$, if e is an error function on the interval $[t_1, t_2]$ admissible for k.

Because p is a Blum prime, the pattern of Legendre symbols has odd symmetry, which implies that we can w.l.o.g. define $\mathcal{T}$ such that it is symmetric around zero. A natural question, for a given Blum prime p, non-negative integer k, and $\mathcal{T} = [-d, d]$ for a positive integer $d \geq k$, is how large d can maximally be such that e is an error function on $\mathcal{T}$ that is admissible for k. This gives rise to the following equivalent definition, in which we leave the error function implicit.

Definition 2. *Let k be a non-negative integer, and let $p > 2k + 1$ be a Blum prime. We define the* k-range *of p, denoted $d_k(p)$, to be the largest integer d such that for all integers x with $1 \leq x \leq d$ it holds that*

$$\sum_{i=-k}^{k} (x + i \mid p) > 0, \tag{2}$$

and we set $d_k(p) = 0$ if no such d exists.

Note that $d_0(p)$ tells us the maximum size of Yu's "Consecutive Quadratic Residues and Non-Residues Sign Module" for a given prime p, i.e., in Yu's terminology and notation: a Blum prime p *qualifies* for $\pm\ell$-CQRN for all $\ell \leq d_0(p)$.

Lower Bound on $d_k(p)$. If $p > 2k + 1$ and $d_0(p) > k$, then $d_k(p) \geq d_0(p)$.

Example. Let us illustrate Definition 2 by means of an example. Let us take $p = 23$; note that this is a Blum prime. Below, we have evaluated the first 16 Legendre symbols.

x	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	...
$(x \mid p)$	0	1	1	1	1	−1	1	−1	1	1	−1	−1	1	1	−1	−1	

We can now read off that $d_0(23) = 4$. Furthermore, it is easy to verify that $d_1(23) = 5$, $d_2(23) = 8$, and $d_3(23) = 7$.

3.3 Avoiding Zero by Restricting to Odd Positions

As mentioned in the preliminaries, if we use the single-online-round protocol for securely evaluating the Legendre symbol, we may not evaluate the Legendre symbol on the zero element. A simple trick to avoid zero (also used in [22]) is to restrict to evaluation of odd inputs by using the map $x \mapsto 2x + 1$. Note that this implies that we cannot compute $\operatorname{sgn}(x)$ using the single-online-round protocol; instead we will evaluate $\operatorname{bsgn}(x)$. Removing the conditions on the Legendre symbols at even positions gives rise to the following definition.

Definition 3. *Let k be a non-negative integer, and let $p > 2k + 1$ be a Blum prime. We define $d_k^*(p)$ as the largest integer d such that for all integers x with $1 \le x \le d$ it holds that*

$$\sum_{i=-k}^{k} (2(x+i)+1 \mid p) > 0, \tag{3}$$

and we set $d_k^*(p) = 0$ if no such d exists.

Note that for any Blum prime p for which $d_0(p) > 1$ (which implies that $d_0(p)$ is even), it is easy to see that it holds that $d_0^*(p) = \frac{1}{2}d_0(p) - 1$. For $k > 0$, such simple relations do not seem to exist. This means, for example, that a prime p that gives rise to a high value for $d_1(p)$, does not necessarily give a high value for $d_1^*(p)$, and vice versa.

3.4 Bounds on $d_0(p)$

The value $d_0(p)$ can be interpreted as the position just before the appearance of the first quadratic non-residue. Let $n_1(p)$ denote the smallest quadratic non-residue. Finding bounds on $n_1(p)$ is a well-known problem in number theory, with important contributions from Polyà, Vinogradov and Burgess, among others. The best explicit upper bound that is currently known (for p a Blum prime) is due to Treviño [19]:

$$d_0(p) + 1 = n_1(p) \le 1.1\sqrt[4]{p}\log p.$$

Graham and Ringrose [5] proved an unconditional asymptotic lower bound (improving on a previous result by, independently,[1] Fridlender [4] and Salié [14]), namely, that there exist infinitely many primes for which

$$d_0(p) + 1 = n_1(p) \ge c \cdot \log(p) \cdot \log\log\log p.$$

for some absolute constant c.[2]

Lamzouri *et al.* [10] prove that conditional on the Generalized Riemann Hypothesis, for all primes $p \ge 5$ it holds that

$$d_0(p) + 1 = n_1(p) < (\log p)^2.$$

3.5 Bounds on $d_1(p)$

Hudson [8] proves an upper bound on the least *pair* of quadratic non-residues. Formally, let $n_2(p)$ be the smallest value such that $n_2(p)$ and $n_2(p) + 1$ are quadratic non-residues. For $k = 1$, it must hold that $d_1(p) < n_2(p)$, because an

[1] Ankeny [2] attributes this result to Chowla, but does not provide a reference.

[2] In the literature, this is also written as $n_1(p) = \Omega(\log(p) \cdot \log\log\log p)$, where Ω is Hardy–Littlewood's Big Omega: $f(n) = \Omega(g(n)) \iff \limsup_{n\to\infty} |f(n)/g(n)| > 0$.

"error pattern" consisting of two consecutive quadratic non-residues (such that $n_2(p) \in [1, (p-3)/2]$) cannot be corrected using a majority vote in a window of length $2k+1=3$. Hudson's bound is as follows. For every $p \geq 5$ we have that

$$d_1(p) < n_2(p) \leq (n_1(p) - 1)q_2,$$

where q_2 is the second smallest prime that is a quadratic non-residue modulo p. Hildebrand [6] also proves an upper bound on $n_2(p)$:

$$d_1(p) < n_2(p) \leq p^{1/(4\sqrt{e})+\epsilon} \quad p \geq p_0(\epsilon),$$

for every $\epsilon > 0$ and $p_0(\epsilon)$ a sufficiently large constant depending on ϵ.

Sun [17] gives a construction for generating all elements n in $\mathbb{F}_p$ such that n and $n+1$ are quadratic non-residues.

Lemma 2 **([17]).** *Let p be an odd prime and let g be a primitive root of p. Then,*

$$\begin{aligned}\mathcal{U} &:= \left\{ n \in \mathbb{F}_p \;\middle|\; (n \mid p) = (n+1 \mid p) = -1 \right\} \\ &= \left\{ u_k \in \mathbb{F}_p \;\middle|\; u_k \equiv \frac{(g^{2k-1}-1)^2}{4g^{2k-1}} (mod p), \quad k = 1, \ldots, \left\lfloor \frac{p-1}{4} \right\rfloor \right\}\end{aligned}$$

We can interpret this lemma as giving a collection of upper bounds on $d_1(p)$, that is, $d_1(p) < n_2(p) \leq u_k$ holds for every $k = 1, \ldots, \lfloor (p-1)/4 \rfloor$.

An error pattern that consists of two quadratic non-residues that are separated by one arbitrary position can also not be corrected using a majority vote in a window of length $2k+1 = 3$. Inspired by Sun, we prove the following lemma.

Lemma 3. *Let p be a Blum prime, let $b = \big((2 \mid p) + 1\big)/2 \in \{0, 1\}$ and let g be a primitive root of p. Then,*

$$\begin{aligned}\mathcal{V} &:= \left\{ n \in \mathbb{F}_p \;\middle|\; (n \mid p) = (n+2 \mid p) = -1 \right\} \\ &= \left\{ v_k \in \mathbb{F}_p \;\middle|\; v_k \equiv \frac{(g^{2k-b}-1)^2}{2g^{2k-b}} \mod p, \quad k = 1, \ldots, (p-3)/4 \right\}.\end{aligned}$$

Also this lemma can be viewed as giving a collection of upper bounds on $d_1(p)$. If $(n \mid p) = (n+2 \mid p) = -1$, then a decoding error (under majority decoding with $k=1$) will occur at position $n+1$, hence we have that $d_1(p) \leq v_k$ holds (instead of strict inequality) for every $k = 1, \ldots, (p-3)/4$.

Proof. Let $\chi(x) = (x \mid p)$ for all $x \in \mathbb{F}_p$. Jacobsthal [9] proves that for p a Blum prime,

$$\begin{aligned} &\left|\{ n \in \mathbb{F}_p \mid \chi(n) = \chi(n+2) = -1 \;\wedge\; \chi(n+1) = 1 \}\right| = \frac{p-1+2\,(2 \mid p)}{8}, \\ \text{and} \quad &\left|\{ n \in \mathbb{F}_p \mid \chi(n) = \chi(n+1) = \chi(n+2) = -1 \}\right| = \frac{p-5-2\,(2 \mid p)}{8}. \end{aligned}$$

Hence, by summing the cardinalities of the above sets, we get that

$$\left|\{n \in \mathbb{F}_p \mid \chi(n) = \chi(n+2) = -1\}\right| = \frac{p-3}{4}.$$

For $j = 1, 2, \ldots, (p-3)/2$, let $r_j \equiv (g^j - 1)^2/(2g^j) \mod p$. Then, $r_j + 2 \equiv (g^j + 1)^2/(2g^j) \mod p$. It now follows that $\chi(r_j) = \chi(r_j + 2) = (-1)^j \chi(2)$ for all $j = 1, 2, \ldots, (p-3)/2$. Hence, $\chi(r_{2k-(\chi(2)+1)/2}) = \chi(r_{2k-(\chi(2)+1)/2} + 2) = -1$ for all $k = 1, 2, \ldots, (p-3)/4$.

It remains to prove that $r_s \not\equiv r_t \mod p$ for all $s, t \in [1, (p-3)/2]$ with $t \neq s$; for this part we can re-use Sun's proof technique used in the proof of Lemma 2. Namely, for all $s, t \in [1, (p-3)/2]$ with $t \neq s$, we have that $g^{s+t} \not\equiv 1 \mod p$ (since g is a primitive root), which implies that $g^s - g^t \not\equiv (g^s - g^t)/g^{s+t} \mod p$. Hence, $g^s + g^{-s} \not\equiv g^t + g^{-t} \mod p$ from which we obtain that $r_s \not\equiv r_t \mod p$. We can now conclude that

$$\{n \in \mathbb{F}_p \mid \chi(n) = \chi(n+2) = -1\} = \{r_{2k-b} \in \mathbb{F}_p \mid k \in [1, (p-3)/4]\},$$

and the claim follows. $\square$

4 Finding a Prime for a Given k-Range

In order to find a prime that, for given integers k and D_k, gives rise to $d_k(p) \geq D_k$, we could in principle take a naive approach by letting a computer exhaustively enumerate the primes in increasing order and compute the Legendre symbols at $a = 1, \ldots, D_k$, and stop when they are all 1. Although this approach works for small values of k and D_k (say for $D_1 < 200$), for larger D_k this will become intractable.

We can speed up the calculation of d_k by using the multiplicativity of the Legendre symbol, the law of quadratic reciprocity and the Chinese Remainder Theorem (CRT). Moreover, we may speed up the computation by enumerating over values p that already satisfy some conditions on the Legendre symbols, using a *wheel data structure* [13,16]. We will first review the problem for the case $k = 0$ and then extend the method to the case $k = 1$. Our approach also works for arbitrary k, and we supply the relevant extensions, but we note that its practicality rapidly diminishes as k increases.

4.1 Finding Primes with High $d_0(p)$

Recall that finding a prime p' such that $d_0(p') \geq D$, for some D, means that p' must be a Blum prime such that the elements $1, \ldots, D$ are quadratic residues modulo p'. By the complete multiplicativity of the Legendre symbol, it suffices to find a Blum prime p such that all primes $q \leq D$ are quadratic residues modulo p.

Proposition 1. *Let q be an odd prime, and p a Blum prime. Then, it holds that*

$$(q \mid p) = (-p \mid q).$$

Proof. It holds that $(q \mid p) = (p \mid q)^{-1} (-1)^{\frac{p-1}{2}\frac{q-1}{2}} = (p \mid q)(-1)^{\frac{q-1}{2}} = (p \mid q)(-1 \mid q) = (-p \mid q)$, where the first equality holds by the law of quadratic reciprocity, the second holds because p is a Blum prime, the third follows from Euler's criterion and the fourth follows from the multiplicativity property of the Legendre symbol. $\square$

Let $\mathcal{R}_q = \{r \bmod q : (-r \mid q) = 1\}$. Then, q is a quadratic residue modulo p if and only if

$$(p \bmod q) \in \mathcal{R}_q. \tag{4}$$

This represents an (exclusive) disjunction of linear congruences:

$$p \equiv r_1 (\bmod\, q) \vee \ldots \vee p \equiv r_\ell \,(\bmod\ q),$$

where $\mathcal{R}_q = \{r_1 \bmod q, \ldots, r_\ell \bmod q\}$.

Let $q_1, \ldots, q_m$ denote all odd primes that are in $[1, D]$. The condition that all integers $[1, D]$ are quadratic residues modulo x thus gives rise to the following system of *simultaneous disjunctions* of linear congruences:

$$\begin{aligned} x \equiv 7 (\bmod\, 8) \text{ (guarantees that } (-1 \mid x) = -1 \text{ and } (2 \mid x) = 1), \\ (x \bmod q_1) \in \mathcal{R}_{q_1}, \\ (x \bmod q_2) \in \mathcal{R}_{q_2}, \\ \vdots \\ (x \bmod q_m) \in \mathcal{R}_{q_m}. \end{aligned} \tag{5}$$

Suppose for each $i = 1, \ldots, m$ we choose a residue class $a_i \in \mathcal{R}_{q_i}$, and we regard the resulting vector $(a_1, \ldots, a_m)$. We may choose the a_i independently since the q_i are distinct primes. An element $(a_1, \ldots, a_m) \in \mathcal{R}_{q_1} \times \cdots \times \mathcal{R}_{q_m} =: \mathcal{R}$ is in one-to-one correspondence with an arithmetic progression of solutions to the above system of congruences, that is, $x, x+Q, x+2Q, \ldots$ where $Q = 8 \prod_{i \in [m]} q_i$. Linnik's theorem [11] (combined with Xylouris' bound [20]) asserts that there will be a prime in this arithmetic progression whose size is bounded as $O(Q^5)$.

Finding the Smallest Such Prime. Finding *some* prime that satisfies the above system is relatively easy, since we may fix a vector $(a_1, \ldots, a_m) \in \mathcal{R}$. We can then enumerate all positive integers x such that $x \bmod q_i = a_i$ via the constructive proof of the CRT, and output the first solution that is prime. However, finding the *smallest* prime that satisfies the above system is a (much) harder task, as it involves searching over the full set $\mathcal{R}$, whose cardinality is exponential in m.

In practice, we may simply enumerate all integers x in ascending order, and check whether x satisfies the system of Eq. (5) rather than computing the Legendre symbols at $1, \ldots, D_0$ explicitly. We can speed up the computation by precomputing the sets $\mathcal{R}_{q_i}$ and storing them in memory. We can check many congruences at once by combining sets of congruences using the CRT. For example, for moduli q, q' we have that $(x \bmod q) \in \mathcal{R}_q$ and $(x \bmod q') \in \mathcal{R}_{q'}$ if and only if $(x \bmod qq') \in \mathcal{R}_{qq'}$,

$$\mathcal{R}_{qq'} := (\mathcal{R}_q + \{0, q', \ldots, (\ell/q' - 1)q'\}) \cap (\mathcal{R}_{q'} + \{0, q, \ldots, (\ell/q - 1)q\}), \tag{6}$$

where $\ell = \mathrm{lcm}(q, q')$ and '+' denotes Minkowski addition. Note that we have abused notation here slightly, and represented the sets $\mathcal{R}_m$ for each modulus m as the set of integers in $[0, m-1]$ that are the canonical lifts of the residue classes mod m. By recursion, the above extends to combining more than two sets of congruences.

4.2 Finding Primes with High $d_1(p)$

For $k > 0$, for $d_k(p) \geq D$ to hold for some positive integer D, it is no longer necessary that p satisfies each disjunction of congruences in Eq. (5); instead, some subsets suffice. For example, for $d_1(p) \geq 6$ we need $(2 \mid p) = 1$ and at least one of $(5 \mid p) = 1$ or $(6 \mid p) = (2 \mid p)\,(3 \mid p) = (3 \mid p) = 1$, otherwise Eq. (2) fails to hold for $a = 5$.

In order for Eq. (2) to hold, we have one set of congruences for every length-$(2k+1)$ subinterval of $[-k, d]$; even for $k = 1$ this quickly grows prohibitively large for non-trivial lower bounds D on $d_k(p)$. While for $k > 0$ the density of primes p satisfying $d_k(p) \geq D$ is greater than for $k = 0$, the search becomes a lot more expensive.

For $k = 1$, we simplify our search for p with $d_1(p) \geq D_1$ with an extra condition: we also require $d_0(p) \geq D_0$ where $D_1 \leq (D_0)^2$. This ensures that each integer in $(D_0, D_1]$ has at most one prime factor greater or equal to D_1. Under this restriction, we get a condition equivalent to $d_1(p) \geq D_1$ which requires fewer computations to check.

Definition 4. *Let D_0, D_1 be non-negative integers with $D_0 < D_1 \leq (D_0)^2$. Let q, q' be distinct primes. We say that $\{q, q'\}$ is a* related pair *on $(D_0, D_1]$ if $D_0 < q, q' \leq D_1$ and there exist positive integers $x, y < D_0$ such that $|xq - yq'| \leq 2$ and $\max\{xq, yq'\} \leq D_1$.*

Proposition 2. *Let D_0, D_1 be non-negative integers with $D_0 < D_1 \leq (D_0)^2$, and let p be a Blum prime with $d_0(p) \geq D_0$. Then $d_1(p) \geq D_1 - 1$ if and only if the following condition holds: for every related pair of primes $\{q, q'\}$ on $(D_0, D_1]$ it holds that $(q \mid p) = 1 \vee (q' \mid p) = 1$.*

Proof. Let a be any positive integer such that $a \leq D_1$. First, we show that $(a \mid p) = -1$ if and only if a has a prime factor $q > D_0$ and $(q \mid p) = -1$. Suppose a has a prime factor $q > D_0$ with $(q \mid p) = -1$. Since $\frac{a}{q} < \frac{a}{D_0} \leq \frac{D_1}{D_0} \leq D_0$, we have $(a/q \mid p) = 1$, hence $(a \mid p) = -1$. If a does not have a prime factor $q > D_0$ with $(q \mid p) = -1$, then taking any prime factor $q' | a$, it must hold that $q' > D_0$, in which case $(q' \mid p) = 1$ by assumption, or $q' \leq D_0$, in which case $(q' \mid p) = 1$ by $d_0(p) \geq D_0$.

We now finish the proof by showing $d_1(p) < D_1 - 1$ if and only if there is some related pair q, q' such that $(q \mid p) = (q' \mid p) = -1$. We have $d_1(p) < D_1 - 1$ if and only if there exists an integer x such that $1 < x \leq D_1 - 1$ and $(x-1 \mid p) + (x \mid p) + (x+1 \mid p) < 0$. This latter inequality holds if and only if at least two of $\{x-1, x, x+1\}$ have Legendre symbol -1. By the above, this holds if and only if two of these numbers have respective prime factors $q, q' > D_0$ and $(q \mid p) = (q' \mid p) = -1$. For these q, q', we have that they constitute a related pair, since they each have a multiple in $\{x-1, x, x+1\}$ and $x+1 \leq D_1$. Conversely, for any related pair there exists such an interval $\{x-1, x, x+1\}$ with $1 < x \leq D_1 - 1$. $\square$

Proposition 2 gives sufficient conditions for $d_1(p) > D_1 - 1$ in terms of related pairs of primes that have to satisfy certain disjunctions of congruences. If we want to include those disjunctions in a system as shown in Eq. (5), we need to represent them in the same form. For every pair of related primes $\{q, q'\}$, the condition that $(q \mid p) = 1 \vee (q' \mid p) = 1$ in Proposition 2 corresponds to taking the *union* of the associated residue sets $\mathcal{R}_q$ and $\mathcal{R}'_q$ of the related primes q and q'. That is, let $\ell = \operatorname{lcm}(q, q')$, then

$$\mathcal{R}_{q,q'} := (\mathcal{R}_q + \{0, q, \ldots, (\ell/q - 1)q\}) \cup (\mathcal{R}'_q + \{0, q', \ldots, (\ell/q' - 1)q'\}),$$

where '+' denotes Minkowski addition, and again we abuse notation and canonically lift residue classes modulo m to the integers $[0, m-1]$. We can now express the related-primes disjunction of congruences as

$$(x \bmod \ell) \in \mathcal{R}_{q,q'}.$$

Since this disjunction of congruences has exactly the same form as the other disjunctions in Eq. (5) we can also take intersections (using Eq. (6)) between a related-primes congruence $\mathcal{R}_{q,q'}$ and another disjunction of congruences.

We can naturally extend the approach for searching primes with high $d_1(p)$ to an approach for finding primes with high $d_k(p)$ for $k > 1$. This involves imposing constraints on sets of $k+1$ distinct primes that are related in a suitably defined way. Note, however, that the use of this for high k is limited, given that we still constrain $D_k \leq (D_0)^2$, and we do not elaborate on this.

4.3 Finding Primes with High $d_k^*(p)$

To find primes p with large $d_k^*(p)$ we use the following results.

Definition 5. *Let D_0, D_k be non-negative integers with $D_0 < D_k \leq (D_0)^2$. Let $Q = \{q_0, \ldots, q_k\}$ be a set of $k+1$ distinct primes. We say that Q is a* $*$-related set *on $(D_0, D_k]$ if $Q \subseteq (D_0, D_k]$ and there exist positive odd integers $x_0, \ldots, x_k < D_0$ such that:*

1. *for any i with $0 \leq i \leq k$ we have $x_i q_i \leq D_k$*
2. *for any i, j with $0 \leq i < j \leq k$ it holds that $|x_i q_i - x_j q_j| \in \{2, 4, 6, \ldots, 4k\}$*

Proposition 3. *Let D_0, D_k be non-negative integers with $D_0 < D_k \leq (D_0)^2$, and let p be a Blum prime with $d_0^*(p) \geq \frac{1}{2}D_0$. Then $d_k^*(p) \geq \frac{1}{2}D_k - k$ if and only if the following condition holds: for every set Q of $k+1$ distinct primes $*$-related on $(D_0, D_k]$, it holds that there exists some $q \in Q$ with $(q \mid p) = 1$.*

4.4 Implementation and Results

We have implemented a search algorithm for primes p with minimal $d_k(p)$ and $d^*_k(p)$ for $k = 0, 1, 2$ using the precomputation of linear congruences as detailed above. We have enumerated all minimal p up to 64 bits with ascending $d_1(p)$ and with $d_0(p) \geq 64$, and likewise for ascending $d^*_1(p)$ with $d^*_0(p) \geq 32$. Our implementation is written in Rust and uses the wheel method from [16]. It is publicly available on GitHub [1].

Table 1 shows results of our search for primes that give rise to as high as possible values of $d^*_1(p)$ and $d^*_2(p)$. Because of space constraints, we refer the reader to the full version of our paper (IACR ePrint, Report 2018/1236) for tables for $d_1(p)$ and $d_2(p)$. See Fig. 1 for a plot of the overall results.

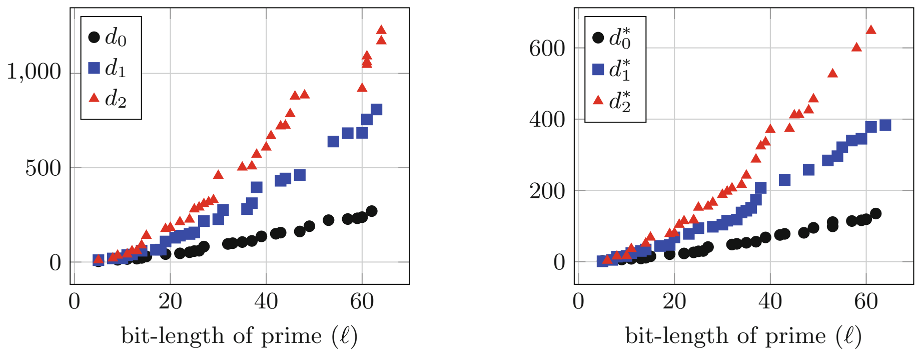

Fig. 1. Graphical comparison of the comparison range achieved by Yu's method (d_0 and d^*_0) vs. our method. The data for d_0 is taken from [12, Table 6.23] (and for all points shown here it holds that $d^*_0 = d_0/2$).

5 Secure Protocols for bsgn

In this section we present three protocols for evaluating the bsgn function, for $k = 1$ and $k = 2$. Note that these immediately imply comparison protocols; from the triangle inequality it follows that correctness for comparison is guaranteed if both inputs lie in $[-\lfloor d/2 \rfloor, \lfloor d/2 \rfloor]$, where $[-d, d]$ is the input range of the bsgn protocol. Throughout this section, we suppose that p is a Blum prime.

We first describe a protocol for securely evaluating the Legendre symbol, which we call $\mathsf{Legendre}$. We describe the protocol in terms of black-box invocations of protocols for sampling a random element from $\mathbb{F}^*_p$ (denoted as $\mathsf{RandomElem}(\mathbb{F}^*_p)$) and for sampling a random bit $\{0, 1\} \subset \mathbb{F}_p$ called $\mathsf{RandomBit}$.

5.1 Secure Medium-Range bsgn Protocol for $k = 1$

In our protocol for $k = 1$, shown as Protocol 2, we compute the binary sign of the sum of the Legendre symbols by means of the multivariate polynomial

$$f(x, y, z) = \frac{x + y + z - xyz}{2},$$

Table 1. Sequence of primes in increasing order (and their bit-lengths ℓ) for which $d_k^*(p)$ is strictly increasing, for $k \in \{1, 2\}$. Primes below the dashed lines have been found via our sieving method, which means that there could exist smaller primes (missed by the sieving method) that give rise to the same or higher values of $d_1^*(p)$ resp. $d_2^*(p)$. For the primes above the dashed lines, it holds that the prime is the smallest possible for a given $d_k^*(p)$. The primes below "$\star\star\star$" are 64 bit primes with the best known k-range.

ℓ	p	$d_1^*(p)$
5	23	1
6	47	4
7	83	5
8	131	7
8	239	8
8	251	14
10	1019	16
11	1091	24
13	4259	30
14	10331	33
14	12011	34
17	74051	42
17	96851	44
19	420731	47
20	831899	52
20	878099	53
20	954971	68
23	5317259	78
25	19127891	79
25	31585979	94
28	140258219	98
30	697955579	104
31	1452130811	112
31	1919592419	115
33	4323344819	116
33	4499001491	117
33	6024587819	118
34	9259782419	138
35	19846138451	143
36	34613840351	151
37	73773096179	153
- - -	- - -	- - -
37	119607747731	174
38	163030664579	182
38	170361409391	207
43	4754588149211	229
48	171772053182831	242

ℓ	p	$d_2^*(p)$
6	47	3
7	83	6
8	131	8
8	179	15
10	1019	16
11	1091	26
11	1427	31
11	1811	36
14	9539	51
15	19211	68
19	334619	78
20	717419	80
21	1204139	104
22	2808251	114
24	8774531	116
24	11532611	117
25	18225611	152
27	98962211	155
28	247330859	166
30	738165419	174
30	1030152059	188
31	1456289579	197
32	2451099251	206
34	11159531291	207
34	13730529419	216
35	17221585499	219
35	19186524419	232
35	26203369331	242
37	92830394411	248
37	128808841619	287
38	232481520059	324
39	408727560491	335
40	807183995411	370
- - -	- - -	- - -
44	15869813229371	373
45	19379613618119	411
46	46760546950211	412

(*continued*)

Table 1. (*continued*)

48	178774759690511	243
48	205152197251811	258
52	2950193919326891	259
52	3705750905778011	284
54	10624213337944379	296
55	26259748609914431	321
57	141840650661890879	340
59	321961111376298371	345
61	1158960903343074191	348
61	1561357330831673339	378
64	9409569905028393239	383
48	240160967391791	425
49	294269750529611	456
53	8755197891979139	526
57	85283169141238571	528
58	148892345027857499	599
61	1915368196138563011	648
	⋆⋆⋆	
64	10807930853257193939	623

Protocol 1. Legendre($[x]$)

```
    *Offline Phase*
1: $[a] \leftarrow \mathsf{RandomElem}(\mathbb{F}_p^*)$
2: $[b] \leftarrow \mathsf{RandomBit}()$
3: $[s] \leftarrow 2[b] - 1$
4: $[r] \leftarrow [s] \cdot [a^2]$

    *Online Phase*
5: $c \leftarrow [x] \cdot [r]$
6: $[z] \leftarrow (c \mid p) \cdot [s]$
7: **return** $[z]$
```

which can be evaluated securely in two rounds using ordinary secure multiplication. It is easy to verify that f correctly computes the sign of the sum of $x, y, z \in \{-1, +1\}$.

Protocol 2. bsgn1Simple($[a]$), $|a| \leq d_1^*(p)$

```
1: $[x] \leftarrow \mathsf{Legendre}(2[a] - 1)$, $[y] \leftarrow \mathsf{Legendre}(2[a] + 1)$, $[z] \leftarrow \mathsf{Legendre}(2[a] + 3)$
2: $[s] \leftarrow ([x] + [y] + [z] - [x][y][z])/2$
3: **return** $[s]$
```

Decreasing the Round Complexity in the Online Phase. Protocol bsgn1Simple requires three rounds in the online phase. We can bring this down to a single round by premultiplying the random Legendre symbols produced in the offline phase of the Legendre protocol. This is shown in Protocol 3. The random bit protocol has been concretely instantiated in the offline phase of Protocol 3 to show that the product of the three random Legendre symbols can be computed in parallel to the preparation of their corresponding random elements. The offline phase requires two rounds in addition to the round complexity of securely sampling random elements of $\mathbb{F}_p^*$.

Protocol 3. bsgn1SingleRound($[a]$), $|a| \leq d_1^*(p)$

```
   Offline Phase
1: for $i \in \{1,2,3\}$ do $[t_i] \leftarrow$ RandomElem($\mathbb{F}_p^*$);  $[u_i] \leftarrow$ RandomElem($\mathbb{F}_p^*$)
2: for $i \in \{1,2,3\}$ do $[v_i] \leftarrow [t_i] \cdot [t_i]$;  $w_i \leftarrow [u_i] \cdot [u_i]$
   $[m] \leftarrow [u_1] \cdot [u_2]$
3: for $i \in \{1,2,3\}$ do $[r_i] \leftarrow [v_i] \cdot [u_i] \cdot w_i^{-1/2}$;  $[s_i] \leftarrow [u_i] \cdot w_i^{-1/2}$
   $[n] \leftarrow [m] \cdot [u_3] \cdot \prod_{i=1}^{3} w_i^{-1/2}$
4: return $([r_1], [s_1], [r_2], [s_2], [r_3], [s_3], [n])$

   Online Phase
5: for $i \in \{1,2,3\}$ do $c_i \leftarrow (2[a] - 3 + 2i) \cdot [r_i]$
6: return $2^{-1} \left(\sum_{i=1}^{3} [s_i] \cdot (c_i \mid p) - [n] \cdot \prod_{i=1}^{3} (c_i \mid p)\right)$
```

5.2 Secure Medium-Range bsgn Protocol for $k = 2$

In our protocol for $k = 2$, shown as Protocol 4, we compute the binary sign of the sum of the five Legendre symbols by means of another invocation of Legendre. In the latter (outer) invocation of Legendre, we need not apply the $x \mapsto 2x + 1$ map because we sum an odd number of values in $\{-1, +1\}$ which cannot become zero. Note that this requires that $d_0(p) \geq 5$ for correctness of the protocol.

Protocol 4. bsgn2($[a]$), $|a| \leq d_2^*(p)$, $d_0(p) \geq 5$

```
1: $[x_1] \leftarrow$ Legendre($2[a] - 3$), $[x_2] \leftarrow$ Legendre($2[a] - 1$), $[x_3] \leftarrow$ Legendre($2[a] + 1$)
   $[x_4] \leftarrow$ Legendre($2[a] + 3$), $[x_5] \leftarrow$ Legendre($2[a] + 5$)
2: $[s] \leftarrow$ Legendre($[x_1] + [x_2] + [x_3] + [x_4] + [x_5]$)
3: return $[s]$
```

6 Application: Fast Neural Network Evaluation in MPC

In this section we demonstrate the usefulness of our secure binary-sign evaluation technique for securely evaluating a neural network.

6.1 Binarized Multi-layer Perceptron for MNIST

For our experiments, we take the binarized multi-layer perceptron of Courbariaux et al. for recognizing handwritten digits from the well-known MNIST benchmark data set [7], which we refer as BMLP below. The BMLP network uses the sign function as its non-linear activation function, and is designed to be evaluated using integer arithmetic only, which allows for a natural MPC implementation.

The MNIST data set contains images of 28-by-28 pixels, where the intensity of each pixel is represented by a byte, i.e., an integer in $\mathcal{B} := [0, 255]$ (0 represents black, 255 represents white, and the values in between represent shades of gray). For the BMLP network, an input image is represented as a byte *vector* $\boldsymbol{x} \in \mathcal{B}^{784}$. Note that by reshaping a two-dimensional image into a (one-dimensional) vector the spatial structure is lost, but this is not a problem for multi-layer perceptrons (as opposed to convolutional neural networks, for instance).

Let n denote the number of neurons per layer. The BMLP network consists of four layers, and uses $n = 4096$. We view each layer L_i, $i \in [1, 4]$, as a map between an input and output vector:

$$\begin{aligned} L_1 &: \quad \mathcal{B}^{784} \to \{-1, +1\}^n, \\ L_i &: \{-1, +1\}^n \to \{-1, +1\}^n, \qquad i \in \{2, 3\} \\ L_4 &: \{-1, +1\}^n \to \mathbb{Z}^{10}. \end{aligned}$$

Let $k_1 = k_2 = k_3 = m_2 = m_3 = m_4 = n$ and $k_4 = 10$ and $m_1 = 784$. In [7], the output of L_i is computed as

$$L_i(\boldsymbol{x}) := \begin{cases} \text{BinarySign}(\text{BatchNorm}^{k_i}_{\Theta_i}(W_i\boldsymbol{x} + \boldsymbol{b}_i)), & i \in \{1, 2, 3\} \\ \text{BatchNorm}^{k_i}_{\Theta_i}(W_i\boldsymbol{x} + \boldsymbol{b}_i) & i = 4. \end{cases}$$

Here $W_i \in \{-1, +1\}^{k_i \times m_i}$ is a matrix of weights, and $\boldsymbol{b}_i \in \mathbb{Z}^{k_i}$ is a vector of bias values. The function BatchNorm, which applies *batch normalization* element-wise, is defined as

$$\begin{aligned} \text{BatchNorm}^{\ell}_{\Theta_i} : \quad \mathbb{Z}^{\ell} &\to \mathbb{Z}^{\ell} \\ (x_1, \ldots, x_\ell) &\mapsto (f_{\Theta_i,1}(x_1), \ldots, f_{\Theta_i,\ell}(x_\ell)) \end{aligned}$$

where $\Theta_i := (\boldsymbol{\mu}_i, \tilde{\boldsymbol{\sigma}}_i, \boldsymbol{\gamma}_i, \boldsymbol{\beta}_i)$ are the batch norm parameters for the ith layer: $\boldsymbol{\mu}_i = (\mu_{i,1}, \ldots, \mu_{i,\ell})$, $\tilde{\boldsymbol{\sigma}}_i = (\tilde{\sigma}_{i,j})_{j \in [1,\ell]}$, $\boldsymbol{\gamma} = (\gamma_{i,j})_{j \in [1,\ell]}$, and $\boldsymbol{\beta} = (\beta_{i,j})_{j \in [1,\ell]}$, and

$$f_{\Theta_i,j}(x) := \gamma_{i,j}\Big(\frac{x - \mu_{i,j}}{\tilde{\sigma}_{i,j}}\Big) + \beta_{i,j}.$$

The function BinarySign applies the bsgn function element-wise,

$$\begin{aligned} \text{BinarySign} : \quad \mathbb{Z}^{n} &\to \{-1, +1\}^n \\ (x_1, \ldots, x_n) &\mapsto (\text{bsgn}(x_1), \ldots, \text{bsgn}(x_n)). \end{aligned}$$

To obtain the final output of the BMLP, which is an integer $y \in [0, 9]$, we apply an (oblivious) argmax operation to the output of L_4:

$$y := \arg\max L_4(L_3(L_2(L_1(\boldsymbol{x})))).$$

Training the Network. We have trained the BMLP on a GPU using Courbariaux' original implementation (described in [7]) which is publicly available on GitHub.

6.2 Eliminating Redundant Parts of Batch Normalization

In layers 1–3, the BinarySign function is applied directly to the output of the BatchNorm function. Because the bsgn function is invariant to multiplying its input by a positive scalar, the BatchNorm function might perform some operations that are immediately undone by the bsgn function. Indeed, it actually turns out that the BatchNorm function (when followed by the BinarySign function) reduces to an additional bias term; the authors of [7] seem to have overlooked this. Formally,

$$f_i(x) = \gamma_i\big(\frac{x-\mu_i}{\tilde{\sigma}_i}\big) + \beta_i = \frac{\gamma_i}{\tilde{\sigma}_i}\left(x - \mu_i + \frac{\beta_i\tilde{\sigma}_i}{\gamma_i}\right),$$

$$\mathrm{bsgn}(f_i(x)) = \mathrm{bsgn}\left(x - \mu_i + \frac{\beta_i\tilde{\sigma}_i}{\gamma_i}\right), \qquad \gamma_i, \tilde{\sigma}_i > 0.$$

Hence, we update the bias vector in all layers except the last as follows,

$$\boldsymbol{b}_i' := \boldsymbol{b}_i - \boldsymbol{\mu}_i + \frac{\boldsymbol{\beta}_i\tilde{\boldsymbol{\sigma}}_i}{\boldsymbol{\gamma}_i}$$

where all operations (addition, subtraction, multiplication, and division) in the above expression are performed element-wise. With this modification, evaluation of the BMLP network simplifies to

$$L_i(\boldsymbol{x}) = \begin{cases} \mathrm{BinarySign}(W_i\boldsymbol{x} + \boldsymbol{b}_i'), & i \in \{1,2,3\} \\ \mathrm{BatchNorm}^{k_i}_{\Theta_i}(W_i\boldsymbol{x} + \boldsymbol{b}_i) & i = 4. \end{cases}$$

6.3 Instantiating the BinarySign Function per Layer

Our aim is to instantiate the BinarySign function using our medium-range bsgn protocols. Nonetheless, for layer L_1, the magnitudes of the elements in the vector $W_1\boldsymbol{x} + \boldsymbol{b}_1'$ for some image $\boldsymbol{x} \in \mathcal{B}^{784}$ will typically be way too large compared to the input range on which our bsgn protocols guarantee a correct answer. Hence, for L_1 we instantiate BinarySign as the element-wise application of an "off-the-shelf" large-range bsgn protocol, such as Toft's comparison protocol [18].

For layers L_2 and L_3 we instantiate BinarySign with (element-wise applications of) Protocol bsgn1Simple using a 64-bit prime modulus p for which $d_1^*(p) = 383$, and, in a separate experiment, with bsgn2 using a 64-bit modulus p' for which $d_2^*(p') = 594$.[3] Also for these layers, there seems to be a mismatch between the input ranges of bsgn1Simple and bsgn2 on which they guarantee correctness, i.e., $[-383, 383]$ and $[-594, 594]$ respectively, and the magnitudes of the elements in the vector $W_i\boldsymbol{y} + \boldsymbol{b}_i'$ for $i \in \{2,3\}$, where $\boldsymbol{y} \in \{-1,+1\}^n$. The first term in this sum (the vector $W_i\boldsymbol{y}$), can have elements with magnitude equal to n in the worst case, where $n = 4096$. Nonetheless, the distribution of values in the vector $W_i\boldsymbol{y} + \boldsymbol{b}_i'$ for all $i \in \{2,3\}$ is strongly concentrated around zero, hence

[3] The prime moduli are $p = 9409569905028393239$ and $p' = 15569949805843283171$.

Table 2. Classification performance of the BMLP on 10,000 MNIST test images

	Full-range sign	bsgn1Simple	bsgn2
Number of misclassifications	248	227	247
Error rate	0.0248	0.0227	0.0247

we will just ignore the fact that our bsgn-protocols will be invoked a number of times on values outside the range for which they guarantee correctness. As we show quantitatively in Table 2, this does not deteriorate the classification performance compared to a network where the full-range sign protocol is also used in layers L_2 and L_3. (Surprisingly, using bsgn1Simple even slightly improves the performance on the MNIST test set.)

6.4 Experimental Results (Neural Network Evaluation)

We have implemented the neural network in MPyC, a Python framework for secure multiparty computation [15]. For $k = 1$ we used a mixture of Protocols 2 and 3, and for $k = 2$ we used Protocol 4, in all cases expanding the calls to Protocol 1 to parallelize the secure computations of the Legendre symbol as much as possible. As a baseline we use the MPyC built-in secure comparison protocol, which is based on Toft's protocol [18]. For a meaningful performance evaluation, we set the bit length to 10 bits for the built-in comparisons used in layers 2 and 3. We have also vectorized the code for all these comparison protocols, handling $n = 4096$ comparisons at the same time for layers 1–3, which increases the speed considerably.

We have run our experiments on a 3PC-LAN setup (CPUs: Intel four-core 4th generation Core i7 3.6 GHz). A complete evaluation between three parties on a secret-shared input image, using secret-shared weights and bias vectors, runs in 60 s for $k = 1$, in 62 s for $k = 2$, and in 67 s for full-range comparisons. For evaluation of a batch of 10 input images the times are 223, 235, and 302 s, respectively. The times for processing all comparisons in layers 2 and 3 are 20, 34, and 99 s, respectively. Hence, in this experiment the Legendre-based comparisons with $k = 1$ are about 5 times faster than full-range comparisons. Similar speedups may be expected with other MPC frameworks for applications with comparisons restricted to medium-sized integers.

To determine the error rate for our particular BMLP, we have also implemented it in Python (including the Python counterparts of the Protocols bsgn1Simple and bsgn2, producing exactly the same errors outside their input ranges). The results measured for the 10,000 MNIST test images are shown in Table 2.

Acknowledgments. We thank Frank Blom for running all our 3-party experiments on his 3PC-LAN setup. This work has received funding from the European Union's Horizon 2020 research and innovation program under grant agreements No 731583 (SODA) and No 780477 (PRIViLEDGE).

References

1. Abspoel, M.: Search for primes with high d_1, d_2 (2018). https://github.com/abspoel/dk-search
2. Ankeny, N.C.: The least quadratic non residue. Ann. Math. **55**(1), 65–72 (1952)
3. Feige, U., Kilian, J., Naor, M.: A minimal model for secure computation (extended abstract). In: Proceedings of STOC 1994, pp. 554–563 (1994)
4. Fridlender, V.R.: On the least nth power non-residue. Dokl. Akad. Nauk. SSSR **66**, 351–352 (1949)
5. Graham, S.W., Ringrose, C.J.: Lower bounds for least quadratic non-residues. In: Berndt, B.C., et al. (eds.) Analytic Number Theory: Proceedings of a Conference in Honor of Paul T. Bateman, vol. 85, pp. 269–309. Springer, Boston (1990). https://doi.org/10.1007/978-1-4612-3464-7_18
6. Hildebrand, A.: On the least pair of consecutive quadratic nonresidues. Mich. Math. J. **34**(1), 57–62 (1987)
7. Hubara, I., Courbariaux, M., Soudry, D., El-Yaniv, R., Bengio, Y.: Quantized neural networks: training neural networks with low precision weights and activations. J. Mach. Learn. Res. **18**(187), 1–30 (2018)
8. Hudson, R.H.: The least pair of consecutive character non-residues. J. Reine Angew. Math. (281), 219–220 (1976)
9. Jacobsthal, E.: Anwendungen einer Formel aus der Theorie der quadratischen Reste. Ph.D. thesis, Friedrich-Wilhelms-Universität, Berlin, Germany (1906)
10. Lamzouri, Y., Li, X., Soundararajan, K.: Conditional bounds for the least quadratic non-residue and related problems. Math. Comput. **84**(295), 2391–2412 (2015)
11. Linnik, U.V.: On the least prime in an arithmetic progression. I. The basic theorem. Rec. Math. [Mat. Sbornik] N.S. **15**(57), 139–178 (1944)
12. Lukes, R.F.: A very fast electronic number sieve. Ph.D. thesis, University of Manitoba, Winnipeg, Canada (1995)
13. Pritchard, P.: A sublinear additive sieve for finding prime numbers. Commun. ACM **24**(1), 18–23 (1981)
14. Salié, H.: Über den kleinsten positiven quadratischen Nichtrest nach einer Primzahl. Math. Nachr. **3**(1), 7–8 (1949)
15. Schoenmakers, B.: MPyC - secure multiparty computation in Python, v0.4.7. GitHub (2018). https://github.com/lschoe/mpyc
16. Sorenson, J.P.: The pseudosquares prime sieve. In: Hess, F., Pauli, S., Pohst, M. (eds.) ANTS 2006. LNCS, vol. 4076, pp. 193–207. Springer, Heidelberg (2006). https://doi.org/10.1007/11792086_15
17. Sun, Z.H.: Consecutive numbers with the same Legendre symbol. Proc. Am. Math. Soc. **130**(9), 2503–2507 (2002)
18. Toft, T.: Primitives and applications for multi-party computation. Ph.D. thesis, Aarhus Universitet, Denmark (2007)
19. Treviño, E.: The least kth power non-residue. J. Number Theory **149**, 201–224 (2015)
20. Xylouris, T.: Über die Nullstellen der Dirichletschen L-Funktionen und die kleinste Primzahl in einer arithmetischen Progression. Ph.D. thesis, Rheinischen Friedrich-Wilhelms-Universität Bonn, Germany (2011)
21. Yao, A.C.: Protocols for secure computations. In: 23rd Annual Symposium on FOCS 1982, pp. 160–164 (1982)
22. Yu, C.H.: Sign modules in secure arithmetic circuits. Cryptology ePrint Archive, Report 2011/539 (2011). http://eprint.iacr.org/2011/539