CRYPTFLOW2: Practical 2-Party Secure Inference

Deewashwer Rathee  Mayank Rathee  Nishant Kumar  Nishanth Chandran
Microsoft Research  Microsoft Research  Microsoft Research  Microsoft Research
t-dee@microsoft.com  t-may@microsoft.com  nishant.kr10@gmail.com  nichandr@microsoft.com
Divya Gupta  Aseem Rastogi  Rahul Sharma
Microsoft Research  Microsoft Research  Microsoft Research
divya.gupta@microsoft.com  aseemr@microsoft.com  rahsha@microsoft.com

ABSTRACT

We present CRYPTFLOW2, a cryptographic framework for secure inference over realistic Deep Neural Networks (DNNs) using secure 2-party computation. CRYPTFLOW2 protocols are both correct – i.e., their outputs are bitwise equivalent to the cleartext execution – and efficient – they outperform the state-of-the-art protocols in both latency and scale. At the core of CRYPTFLOW2, we have new 2PC protocols for secure comparison and division, designed carefully to balance round and communication complexity for secure inference tasks. Using CRYPTFLOW2, we present the first secure inference over ImageNet-scale DNNs like ResNet50 and DenseNet121. These DNNs are at least an order of magnitude larger than those considered in the prior work of 2-party DNN inference. Even on the benchmarks considered by prior work, CRYPTFLOW2 requires an order of magnitude less communication and 20–30× less time than the state-of-the-art.

KEYWORDS
Privacy-preserving inference; deep neural networks; secure two-party computation

1 INTRODUCTION

The problem of privacy preserving machine learning has become increasingly important. Recently, there have been many works that have made rapid strides towards realizing secure inference [4, 6, 13, 17, 19, 22, 31, 43, 48, 49, 51, 55, 57]. Consider a server that holds the weights \( w \) of a publicly known deep neural network (DNN), \( F \), that has been trained on private data. A client holds a private input \( x \); in a standard machine learning (ML) inference task, the goal is for the client to learn the prediction \( F(x, w) \) of the server’s model on the input \( x \). In secure inference, the inference is performed with the guarantee that the server learns nothing about \( x \) and the client learns nothing about the server’s model \( w \) beyond what can be deduced from \( F(x, w) \) and \( x \).

A solution for secure inference that scales to practical ML tasks would open a plethora of applications based on MLaaS (ML as a Service). Users can obtain value from ML services without worrying about the loss of their private data, while model owners can effectively monetize their services with no fear of breaches of client data (they never observe private client data in the clear). Perhaps the most important emerging applications for secure inference are in healthcare where prior work [4, 45, 55] has explored secure inference services for privacy preserving medical diagnosis of chest diseases, diabetic retinopathy, malaria, and so on.

Secure inference is an instance of secure 2-party computation (2PC) and cryptographically secure general protocols for 2PC have been known for decades [32, 63]. However, secure inference for practical ML tasks, e.g., ImageNet scale prediction [24], is challenging for two reasons: a) realistic DNNs use ReLU activations\(^1\) that are expensive to compute securely; and b) preserving inference accuracy requires a faithful implementation of secure fixed-point arithmetic. All prior works [6, 31, 43, 48, 49, 51] fail to provide efficient implementation of ReLUs. Although ReLUs can be replaced with approximations that are more tractable for 2PC [22, 31, 49], this approach results in significant accuracy losses that can degrade user experience. The only known approaches to evaluate ReLUs efficiently require sacrificing security by making the untenable assumption that a non-colluding third party takes part in the protocol [7, 45, 50, 56, 61] or by leaking activations [12]. Moreover, some prior works [45, 49–51, 61] even sacrifice correctness of their fixed-point implementations and the result of their secure execution can sometimes diverge from the expected result, i.e. cleartext execution, in random and unpredictable ways. Thus, correct and efficient 2PC protocols for secure inference over realistic DNNs remain elusive.

1.1 Our Contributions

In this work, we address the above two challenges and build new semi-honest secure 2-party cryptographic protocols for secure computation of DNN inference. Our new efficient protocols enable the first secure implementations of ImageNet scale inference that complete in under a minute! We make three main contributions:

- First, we give new protocols for millionaires’ and DReLU\(^2\) that enable us to securely and efficiently evaluate the non-linear layers of DNNs such as ReLU, Maxpool and Argmax.

- Second, we provide new protocols for division. Together with new theorems that we prove on fixed-point arithmetic over shares, we show how to evaluate linear layers, such as convolutions, average pool and fully connected layers, faithfully.

- Finally, by providing protocols that can work on a variety of input domains, we build a system\(^3\) CRYPTFLOW2 that supports two different types of Secure and Correct Inference (SCI) protocols where linear layers can be evaluated using either homomorphic encryption (SCI\(_{HE}\)) or through oblivious transfer (SCI\(_{OT}\)).

We now provide more details of our main contributions.

\(^{1}\)ReLU(x) is defined as \( \max(x, 0) \).

\(^{2}\)DReLU is the derivative of ReLU, i.e., \( \text{DReLU}(x) = 1 \) if \( x \geq 0 \) and \( 0 \) otherwise.

\(^{3}\)Implementation is available at https://github.com/mpc-mars/EdPC.
New millionaires’ and DReLU protocols. Our first main technical contribution is a novel protocol for the well-known millionaires’ problem [63], where parties $P_0$ and $P_1$ hold $t$-bit integers $x$ and $y$, respectively, and want to securely compute $x < y$ (or, secret shares of $x < y$). The theoretical communication complexity of our protocol is $\approx 3x$ better than the most communication efficient prior millionaires’ protocol [21, 29, 32, 62, 63]. In terms of round complexity, our protocol executes in $\log t$ rounds (e.g. 5 rounds for $t = 32$ bits); see Table 1 for a detailed comparison and [21] for a detailed overview of the costs of other comparison protocols.

Using our protocol for millionaires’ problem, we build new and efficient protocols for computing DReLU for both $t$-bit integers (i.e., $\mathbb{Z}_L, L = 2^t$) and general rings $\mathbb{Z}_n$. Our protocol for DReLU serves as one of the main building blocks for non-linear activations such as ReLU and Maxpool, as well as division over both input domains. Providing support for $t$-bit integers $\mathbb{Z}_L$ as well as arbitrary rings $\mathbb{Z}_n$, allows us to securely evaluate the linear layers (such as matrix multiplication and convolutions) using the approaches of Oblivious Transfer (OT) [8, 51] as well as Homomorphic Encryption (HE) [30, 43, 49], respectively. This provides our protocols great flexibility when executing over different network configurations.

Fixed-point arithmetic. The ML models used by all prior works on secure inference are expressed using fixed-point arithmetic; such models can be obtained from [39, 42, 45, 52]. A faithful implementation of fixed-point arithmetic is quintessential to ensure that the secure computation is correct, i.e., it is equivalent to the cleartext computation for all possible inputs. Given a secure inference task $F(x, w)$, some prior works [45, 49–51, 61] give up on correctness when implementing division operations and instead compute an approximation $F'(x, w)$. In fixed-point arithmetic, each multiplication requires a division by a power-of-2 and multiplications are used pervasively in linear-layers of DNNs. Moreover, layers like average-pool require division for computing means. Loss in correctness is worrisome as the errors can accumulate and $F'(x, w)$ can be arbitrarily far from $F(x, w)$. Recent work [49] has shown that even in practice the approximations can lead to significant losses in classification accuracy.

As our next contribution, we provide novel protocols to compute division by power-of-2 as well as division by arbitrary integers that are both correct and efficient. The inputs to these protocols can be encoded over both $t$-bit integers $\mathbb{Z}_L$ as well as $\mathbb{Z}_n$, for arbitrary $n$. To the best of our knowledge, the only known approach to compute division correctly is via garbled circuits which we compare with in Table 3. While garbled circuits based protocols require communication which is quadratic in $t$ or $\log n$, our protocols are asymptotically better and incur only linear communication. Concretely, for average pool with 7 $\times$ 7 filters and 32-bit integers, our protocols have $\approx 54 \times$ less communication.

Scaling to practical DNNs. These efficient protocols, help us securely evaluate practical DNNs (first made in [29]). Let $x = x_1 || x_0$ and $y = y_1 || y_0$ (where $||$ denotes concatenation and $x_1, y_1$ are strings of the same length). Then,

\begin{align*}
\text{OT vs HE.} & \quad \text{Through our evaluation, we also resolve the OT vs HE conundrum: although the initial works on secure inference [48, 51] used OT-based protocols for evaluating convolutions, the state-of-the-art protocols [43, 49], which currently provide the best published inference latency, use HE-based convolutions. HE-based secure inference has much less communication than OT but HE requires more computation. Hence, at the onset of this work, it was not clear to us whether HE-based convolutions would provide us the best latency for ImageNet-scale benchmarks.}
\end{align*}

To resolve this empirical question, we implement two classes of protocols, $\text{SCI}_{\text{OT}}$ and $\text{SCI}_{\text{HE}}$, in CRYPTO2. In $\text{SCI}_{\text{OT}}$, inputs are in $\mathbb{Z}_L$ ($L = 2^t$, for a suitable choice of $t$). Linear layers such as matrix multiplication and convolution are performed using OT-based techniques [8, 51] while the activations such as ReLU, Maxpool and Avgpool are implemented using our new protocols over $\mathbb{Z}_n$. In $\text{SCI}_{\text{HE}}$, inputs are encoded in an appropriate prime field $\mathbb{Z}_n$ (similar to [43, 49]). Here, we compute linear layers using homomorphic encryption and the activations using our protocols over $\mathbb{Z}_n$. In both $\text{SCI}_{\text{OT}}$ and $\text{SCI}_{\text{HE}}$ faithful divisions after linear layers are performed using our new protocols over corresponding rings. Next, we evaluate ImageNet-scale inference tasks with both $\text{SCI}_{\text{OT}}$ and $\text{SCI}_{\text{HE}}$. We observe that in a WAN setting, where communication is a bottleneck, HE-based inference is always faster and in a LAN setting OT and HE are incomparable.

1.2 Our Techniques

Millionaires’. Our protocol for securely computing the millionaires’ problem (the bit $x < y$ is based on the following observation (first made in [29]). Let $x = x_1 || x_0$ and $y = y_1 || y_0$ (where $||$ denotes concatenation and $x_1, y_1$ are strings of the same length). Then,
Protocol

<table>
<thead>
<tr>
<th>Layer</th>
<th>Protocol</th>
<th>Comm. (bits)</th>
<th>Rounds</th>
</tr>
</thead>
<tbody>
<tr>
<td>Millionaires’ on ${0,1}^t$</td>
<td>GC [62, 63]</td>
<td>$4\lambda t$</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>GMW/ GSV [29, 32]</td>
<td>$\approx 6\lambda t$</td>
<td>log $t + 3$</td>
</tr>
<tr>
<td></td>
<td>SC$^3$ [21]</td>
<td>$&gt; 3\lambda t$</td>
<td>$\approx 4\log^2 \lambda$</td>
</tr>
<tr>
<td>This work ($m = 4$)</td>
<td>$&lt; \lambda t + 14\ell$</td>
<td>log $\ell$</td>
<td></td>
</tr>
<tr>
<td>Millionaires’ example $t = 32$</td>
<td>GC [62, 63]</td>
<td>16384</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>GMW/ GSV [29, 32]</td>
<td>23140</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td>SC$^3$ [21]</td>
<td>13016</td>
<td>15</td>
</tr>
<tr>
<td>This work ($m = 7$)</td>
<td>2950</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>This work ($m = 4$)</td>
<td>3844</td>
<td>5</td>
<td></td>
</tr>
</tbody>
</table>

Table 1: Comparison of communication with prior work for millionaires’ problem. For our protocol, $m$ is a parameter. For concrete bits of communication we use $\lambda = 128$.

OT protocols for leaf comparisons and equality, multiple 1-out-of-$2^m$ OT instances can be combined to reduce communication. Next, recursing up from the leaves to the root, requires securely computing the AND functionality$^6$ that uses Beaver bit triples [8]. We observe that the same secret value is used in 2 AND instances. Hence, we construct correlated pairs of bit triples using 1-out-of-8 OT protocols [44] to reduce this cost to $\lambda + 8$ bits (amortized) per triple, where $\lambda$ is the security parameter and typically 128. Some more work is needed for the above technique to work efficiently for the general case when $m$ does not divide $t$ or $\ell/m$ is not a power of 2. Finally, by picking $m$ appropriately, we obtain a protocol for millionaires’ whose concrete communication (in bits) is nearly 5 times better than prior work.

ReLU

$DReLU$. Let $\alpha$ be additively secret shared as $a_0, a_1$ over the appropriate ring. $DReLU(a)$ is 1 if $a \geq 0$ and 0 otherwise; note that $a \geq 0$ is defined differently for $t$-bit integers and general rings. Over $\mathbb{Z}_L$, where values are encoded using 2’s complement notation, $DReLU(a) = 1 \oplus MSB(a)$, where $MSB(a)$ is the most significant bit of $a$. Moreover, $MSB(a) = MSB(a_0) \oplus MSB(a_1) \oplus carry$. Here, $carry = 1$ if $a_0 + a_1 \geq 2^{t-1}$, where $a_0, a_1$ denotes the integer represented by the lower $t-1$ bits of $a_0, a_1$. We compute this carry bit using a call to our millionaires’ protocol. Over $\mathbb{Z}_n$, $DReLU(a) = 1$ if $a \in [0, \lfloor n/2 \rfloor)$. Given the secret shares $a_0, a_1$, this is equivalent to $(a_0 + a_1) \in [0, \lfloor n/2 \rfloor) \cup [n, \lfloor 3n/2 \rfloor)$ over integers. While this can be naively computed by making 3 calls to the millionaires’ protocol, we show that by carefully selecting the inputs to the millionaires’ protocol, one can do this with only 2 calls. Finally, we set things up so that the two calls to millionaires’ have correlated inputs that reduces the overall cost to $\approx 1.5$ instances of millionaires’ over $\mathbb{Z}_n$.

Division and Truncation. As a technical result, we provide a correct decomposition of division of a secret ring element in $\mathbb{Z}_L$ or $\mathbb{Z}_n$ by a public integer into division of secret shares by the same public integer and correction terms (Theorem 4.1). These correction terms consist of multiple inequalities on secret values. As a corollary, we also get a much simpler expression for the special case of truncation, i.e., dividing $t$-bit integers by a power-of-2 (Corollary 4.2). We believe that the general theorem as well as the corollary can be of independent interest. Next, we give efficient protocols for both general division (used for Avgpool, Table 3) as well as division by a power-of-2 (used for multiplication in fixed-point arithmetic). The inequalities in the correction term are computed using our new protocol for millionaires’ and the division of shares can be done locally by the respective parties. Our technical theorem is the key to obtaining secure implementation of DNN inference tasks that are bitwise equivalent to cleartext fixed-point execution.

1.3 Other Related Work

Perhaps the first work to consider the secure computation of machine learning inference algorithms was that of [14]. SecureML [51] was the first to consider secure neural network inference and training. Apart from the works mentioned earlier, other works include:

$x < y$ is the same as checking if either $x_1 < y_1$ or $x_1 = y_1$ and $x_0 < y_0$. Now, the original problem is reduced to computing two millionaires instances over smaller length strings ($x_1 < y_1$ and $x_0 < y_0$) and one equality test ($x_1 = y_1$). By continuing recursively, one could build a tree all the way where the leaves are individual bits, at which point one could use 1-out-of-2 OT-based protocols to perform the comparison/equality. However, the communication complexity of this protocol is still quite large. We make several important modifications to this approach. First, we modify the tree so that the recursion is done log($t/m$) times to obtain leaves with strings of size $m$, for a parameter $m$. We then use 1-out-of-$2^m$ OT to compute the comparison/equality at the leaves, employing the lookup-table based approach of [25]. Second, we observe that by carefully setting up the receiver’s and sender’s messages in the

$^6$This functionality takes as input shares of bits $x, y$ from the two parties and outputs shares of $x$ AND $y$ to both parties.
those that considered malicious adversaries [20, 36, 64] (for simpler ML models like linear models, regression, and polynomials) as well as specialized DNNs with 1 or 2 bit weights [4, 55, 57]. Recently, [26] gave protocols for faithful truncation (but not division) over \( \ell \)-bit integers and prime fields in various adversarial settings. For 2-party semi-honest setting, our protocols have up to 20x less communication for the truncations required in our evaluation.

1.4 Organisation

We begin with the details on security and cryptographic primitives used in Section 2 on preliminaries. In Section 3 we provide our protocols for millionaires’ (Section 3.1) and ReLU (Section 3.2, 3.3), over both \( \mathbb{Z}_L \) and general ring \( \mathbb{Z}_n \). In Section 4, we present our protocols for division and truncation. We describe the various components of DNN inference in Section 5 and show how to construct secure protocols for all these components given our protocols from Sections 3 and 4. We present our implementation details in Section 6 and our experiments in Section 7. Finally, we conclude and discuss future work in Section 8.

2 PRELIMINARIES

Notation. For a set \( W, w \in W \) denoting an element \( w \), uniformly at random from \( W \). \( \{ \ell \} \) denotes the set of integers \( \{0, \ldots, \ell - 1\} \). Let \( \mathbf{1}(b) \) denote the indicator function that is \( 1 \) when \( b \) is true and \( 0 \) when \( b \) is false.

2.1 Threat Model and Security

We provide security in the simulation paradigm [18, 32, 47] against a static semi-honest probabilistic polynomial time (PPT) adversary \( \mathcal{A} \). That is, a computationally bounded adversary \( \mathcal{A} \) corrupts either \( P_0 \) or \( P_1 \) at the beginning of the protocol and follows the protocol specification honestly. Security is modeled by defining two interactions: a real interaction where \( P_0 \) and \( P_1 \) execute the protocol in the presence of \( \mathcal{A} \) and the environment \( \mathcal{E} \) and an ideal interaction where the parties send their inputs to a trusted functionality that performs the computation faithfully. Security requires that for every adversary \( \mathcal{A} \) in the real interaction, there is an adversary \( S \) (called the simulator) in the ideal interaction, such that no environment \( \mathcal{E} \) can distinguish between real and ideal interactions. Many of our protocols involve multiple sub-protocols and we describe these using the hybrid model. This is similar to a real interaction, except that sub-protocols are replaced by the invocations of instances of corresponding functionalities. A protocol invoking a functionality \( F \) is said to be in “\( F \)-hybrid model.”

2.2 Cryptographic Primitives

2.2.1 Secret Sharing Schemes. Throughout this work, we use 2-out-of-2 additive secret sharing schemes over different rings [11, 59]. The 3 specific rings that we consider are the field \( \mathbb{F}_L \), the ring \( \mathbb{Z}_L \), where \( L = 2^t \) (\( t = 32 \), typically), and the ring \( \mathbb{Z}_n \), for a positive integer \( n \) (this last ring includes the special case of prime fields used in the works of [43, 49]). We let \( \text{Share}^{\ell}(x) \) denote the algorithm that takes as input an element \( x \) in \( \mathbb{Z}_L \) and outputs shares over \( \mathbb{Z}_L \), denoted by \( (x)_L^0 \) and \( (x)_L^1 \). Shares are generated by sampling random ring elements \( (x)_L^0 \) and \( (x)_L^1 \), with the only constraint that \( (x)_L^0 + (x)_L^1 = x \) (where + denotes addition in \( \mathbb{Z}_L \)). Additive secret sharing schemes are perfectly hiding, i.e., given a share \( (x)_L^0 \) or \( (x)_L^1 \), the value \( x \) is completely hidden. The reconstruction algorithm \( \text{Reconst}^\ell((x)_L^0, (x)_L^1) \) takes as input the two shares and outputs \( x = (x)_L^0 + (x)_L^1 \). Shares (along with their corresponding \( \text{Share}() \) and \( \text{Reconst}() \) algorithms) are defined in a similar manner for \( \mathbb{Z}_n \) and \( \mathbb{F}_n \) with superscripts \( B \) and \( n \), respectively. We sometimes refer to shares over \( \mathbb{Z}_L \) and \( \mathbb{Z}_n \) as arithmetic shares and shares over \( \mathbb{F}_n \) as boolean shares.

2.2.2 Oblivious Transfer. Let \( (\ell) \cdot \text{OT}_\ell \) denote the 1-out-of-k Oblivious Transfer (OT) functionality [16] (which generalizes 1-out-of-2 OT [27, 54]). The sender’s inputs to the functionality are the \( k \) strings \( m_0, \ldots, m_{k-1} \), each of length \( \ell \) and the receiver’s input is a value \( i \in [k] \). The receiver obtains \( m_i \) from the functionality and the sender receives no output. We use the protocols from [44], which are an optimized and generalized version of the OT extension framework proposed in [9, 41]. This framework allows the sender and receiver, to “reduce” \( \ell \) number of oblivious transfers to \( \lambda \) “base” OTs. We also use the notion of correlated 1-out-of-2 OT [5], denoted by \( (\ell) \cdot \text{COT}_\ell \). In our context, this is a functionality where the sender’s input is a ring element \( x \) and the receiver’s input is a choice bit \( b \). The receiver sends a random ring element \( r \) as output and the receiver obtains either \( r \) or \( x + r \) as output depending on \( b \). The protocols for \( (\ell) \cdot \text{OT}_\ell \) [44] and \( (\ell) \cdot \text{COT}_\ell \) [5] execute in 2 rounds and have total communication \( 2\ell + k\ell \) bits, respectively. Moreover, simpler \( (\ell) \cdot \text{OT}_\ell \) has a communication of \( \lambda + 2\ell \) bits [5, 41].

2.2.3 Multiplexer and B2A conversion. The functionality \( \mathcal{F}^\eta_{\text{MUX}} \) takes as input arithmetic shares of a \( n \) and boolean shares of choice bit \( c \) from \( P_0, P_1 \), and returns shares of \( a \) if \( c = 1 \), else returns shares of \( 0 \) over the same ring. A protocol for \( \mathcal{F}^\eta_{\text{MUX}} \) can easily be implemented by 2 simultaneous calls to \( (\ell) \cdot \text{OT}_\ell \) and communication complexity is \( 2(\lambda + 2\ell) \), where \( \eta = \lceil \log n \rceil \).

The functionality \( \mathcal{F}^\eta_{\text{MUX}} \) (for boolean to arithmetic conversion) takes boolean (i.e., over \( \mathbb{Z}_2 \)) shares as input and gives out arithmetic (i.e., over \( \mathbb{Z}_n \)) shares of the same value as output. It can be realized via one call to \( (\ell) \cdot \text{COT}_\eta \) and hence, its communication is \( \lambda + \eta \). For completeness, we provide the protocols realizing \( \mathcal{F}^\eta_{\text{MUX}} \) as well as \( \mathcal{F}^\eta_{\text{B2A}} \) formally in Appendix A.3 and Appendix A.4, respectively.

2.2.4 Homomorphic Encryption. A homomorphic encryption of \( x \) allows computing encryption of \( f(x) \) without the knowledge of the decryption key. In this work, we require an additively homomorphic encryption scheme that supports addition and scalar multiplication, i.e., multiplication of a ciphertext with a plaintext. We use the additively homomorphic scheme of BFV [15, 28] (the scheme used in the recent works of Gazelle [43] and Delphi [49]) and use the optimized algorithms of Gazelle for homomorphic matrix-vector products and homomorphic convolutions. The BFV scheme uses the batching optimization [46, 60] that enables operation on plaintext vectors over the field \( \mathbb{Z}_n \), where \( n \) is a prime plaintext modulus of the form \( 2KN + 1 \), \( K \) is some positive integer and \( N \) is scheme parameter that is a power-of-2.

The protocol of \( (\ell) \cdot \text{OT}_\ell \) [44] incurs a communication cost of \( \lambda + k\ell \). However, to achieve the same level of security, their security parameter needs to be twice that of \( (\ell) \cdot \text{COT}_\ell \). In concrete terms, therefore, we write the cost as \( 2\lambda + k\ell \).
3 MILLIONAIRES’ AND DReLU PROTOCOLS

In this section, we provide our protocols for millionaires’ problem and DReLU(a) when the inputs are \( t \) bit signed integers as well as elements in general rings of the form \( \mathbb{Z}_n \) (including prime fields). Our protocol for millionaires’ problem invokes instances of \( \mathcal{F}_{\text{AND}} \) that take as input boolean shares of values \( x, y \in \{0, 1\} \) and returns boolean shares of \( x \land y \). We discuss efficient protocols for \( \mathcal{F}_{\text{AND}} \) in Appendix A.1 and A.2.

3.1 Protocol for Millionaires’

In the Yao millionaires’ problem, party \( P_0 \) holds \( x \) and party \( P_1 \) holds \( y \) and they wish to learn boolean shares of \( 1 \cdot (x < y) \). Here, \( x \) and \( y \) are \( t \)-bit unsigned integers. We denote this functionality by \( \mathcal{F}_{\text{MILL}}. \) Our protocol for \( \mathcal{F}_{\text{MILL}} \) builds on the following observation that was also used in \([29]\):

\[
1 \cdot (x < y) = 1 \cdot (x < y_0) \cdot 1 \cdot (x < y_0) \cdot 1 \cdot (x < y_0),
\]

where \( x = x_1 || x_0 \) and \( y = y_1 || y_0 \).

**Intuition.** Let \( m \) be a parameter and \( M = 2^m \). First, for ease of exposition, we consider the special case when \( m \) divides \( t \) and \( q = \lfloor t/m \rfloor \) is a power of 2. We describe our protocol for millionaires’ problem in this setting formally in Algorithm 1. We use Equation 1 above, recursively \( \log q \) times to obtain \( q \) leaves of size \( m \) bits. That is, let \( x = x_{q-1} || ... || x_0 \) and \( y = y_{q-1} || ... || y_0 \) (where every \( x_i, y_i \in \{0, 1\}^m \)). Now, we compute the shares of the inequalities and equalities of strings at the leaf level using \((M)_1\)-OT1 (steps 9 and 10, resp.). Next, we compute the shares of the inequalities (steps 14 & 15) and equalities (step 16) at each internal node upwards from the leaf using Equation 1. Value of inequality at the root gives the final output.

**Correctness and security.** Correctness is shown by induction on the depth of the tree starting at the leaves. First, by correctness of \((M)_1\)-OT1 in step 9, \((\langle x_i \rangle_0)_{\text{eq}}^B = \langle x_i \rangle_0 \oplus 1 \cdot (x_i = y_i) \). Similarly, \((\langle x_i \rangle_0)_{\text{eq}}^B = \langle x_i \rangle_0 \oplus 1 \cdot (x_i = y_i) \). This proves the base case. Let \( q = 2^q \). Also, for level \( i \) of the tree, parse \( x = x^{(i)} || x^{(i)} \) and \( y = y^{(i)} || y^{(i)} \). Assume that for \( i \) it holds that \( \langle x_i \rangle_0 \oplus \langle y_i \rangle_0 = 1 \cdot (x_i < y_i) \) and \( \langle x_i \rangle_0 \oplus \langle y_i \rangle_0 = 1 \cdot (x_i < y_i) \). Then, we prove the same for \( i + 1 \) as follows: By correctness of \( \mathcal{F}_{\text{AND}} \) for \( j \in \{0, \ldots, q_i - 1\} \), \( \langle x_{i+1} \rangle_0 \oplus \langle y_{i+1} \rangle_0 = \langle x_{i+1} \rangle_0 \oplus \langle y_{i+1} \rangle_0 = 1 \cdot (x_{i+1} < y_{i+1}) \sqcap 1 \cdot (x_{i+1} < y_{i+1}) \sqcap 1 \cdot (x_{i+1} < y_{i+1}) = 1 \cdot (x_{i+1} < y_{i+1}) \) (using Equation 1). The induction step for \( \mathcal{F}_{\text{AND}} \) holds in a similar manner, thus proving correctness.

Given uniformity of \((\langle x_i \rangle_0)_{\text{eq}}^B \) for all \( j \in \{0, \ldots, q - 1\} \), security follows easily in the \((M)_1\)-OT1, \( \mathcal{F}_{\text{AND}} \)-hybrid.

**General case.** When \( m \) does not divide \( t \) and \( q = \lceil t/m \rceil \) is not a power of 2, we make the following modifications to the protocol. Since \( m \) does not divide \( t \), \( q-1 \in \{0, 1\}^t \), where \( t = \text{mod} \ m \). When doing the compute for \( q_{t-1} \) and \( y_{t-1} \), we perform a small optimization and use \((\langle y \rangle_0)_{\text{eq}}^B \text{OT}_{1^t}, \) in steps 9 and 10, where \( R = 2^t \). Second, since \( q \) is not a power of 2, we do not have a perfect binary tree of recursion and we need to slightly change our recursion/tree traversal. In the general case, we construct maximal possible perfect binary trees and connect the roots of the same using the relation in Equation 1. Let \( a \) be such that \( 2^a < q \leq 2^{a+1} \). Now, our tree has a perfect binary sub-tree with \( 2^a \) leaves and we have remaining \( q' = q - 2^a \) leaves. We recurse on \( q' \). In the last step, we obtain our tree with \( q \) leaves by combining the roots of perfect binary tree with \( 2^a \) leaves and tree with \( q' \) leaves using Equation 1. Note that value at the root is computed using \([\log q]\) sequential steps starting from the leaves.

3.1.1 Optimizations. We reduce the concrete communication complexity of our protocol using the following optimizations that are applicable to both the special and the general case.

- Combining two \((M)_1\)-OT1 calls into one \((M)_2\)-OT2: Since the input of \( P_1 \) (OT receiver) to \((M)_1\)-OT1 in steps 9 and 10 is same, i.e., \( y_j \), we can collapse these steps into a single call to \((M)_2\)-OT2 where \( P_0 \) and \( P_1 \) input \( \{s_{j,k}, t_{j,k}\} \) and \( y_j \), respectively. \( P_1 \) sets its output as \((\langle y_j \rangle_0^B \text{eq})_{\text{eq}}^B \). This reduces the cost from \( 2(2\lambda + M) \) to \( (2\lambda + 2M) \).
In Algorithm 2, we describe our protocol for $F_{\text{DReLU}}^{\text{int},\ell}$ that takes as input arithmetic shares of $a$ and returns boolean shares of DReLU($a$). Note that DReLU($a$) = (1$\circ$MSB($a$)), where MSB($a$) is the most significant bit of $a$. To compute shares of MSB($a$) as follows: Let carry = 1 if $(x_0 + x_1) > 2^{\ell-1} - 1$. Then, MSB($a$) = msb$_0 \oplus$ msb$_1$ \oplus carry. We compute boolean shares of carry by invoking an instance of $F_{\text{MILL}}$.

**Correctness and security:** By correctness of $F_{\text{MILL}}$, Reconstr$^B$((carry)$^B$, (carry)$^B$) = 1 if $(2^{\ell-1} - 1 - x_0) < x_1 = 1 if $(x_0 + x_1) > 2^{\ell-1} - 1$. Also, Reconstr$^B$((DReLU$^B$) / (DReLU$^B$)) = msb$_0 \oplus$ msb$_1$ \oplus carry + 1 = MSB($a$) \oplus 1. Security follows trivially in the $F_{\text{MILL}}$ hybrid.

**Communication complexity** In Algorithm 2, we communicate the same as in $F_{\text{MILL}}^{\ell-1}$ that is $\langle (\ell + 1)(\ell - 1) \rangle$ by using $m = 4$.

---

3.3 Protocol for DReLU for general $\mathbb{Z}_n$

We describe a protocol for $F_{\text{DReLU}}^{\text{ring},n}$ that takes arithmetic shares of $a$ over $\mathbb{Z}_n$ as input and returns boolean shares of DReLU($a$). For integer rings $\mathbb{Z}_n$, DReLU($a$) = 1 if $a < \lceil \eta / 2 \rceil$ and 0 otherwise. This notation includes the case of prime fields considered in the works of [43, 49]. Below, we formally discuss the case of rings of odd number of elements and omit the analogous case of even rings. We first describe a (simplified) protocol for DReLU over $\mathbb{Z}_n$ in Algorithm 3 with protocol logic as follows: Let arithmetic shares of $a \in \mathbb{Z}_n$ be $(\langle a \rangle^a)$ and $(\langle a \rangle^a)$. Define wrap = 1 if $(\langle a \rangle^a > n - 1)$. 1 if $(\langle a \rangle^a + \langle a \rangle^a > n + (n - 1)/2)$ and rt = 1 if $(\langle a \rangle^a + \langle a \rangle^a > n + (n - 1)/2)$. Then, DReLU($a$) = 1 (1 if wrap = 0, else it is (1 \oplus rt)). In Algorithm 3, steps 1,2,3, compute these three comparisons using $F_{\text{MILL}}$. Final output can be computed using an invocation of $F_{\text{MUX}}$.

**Optimizations.** We describe an optimized protocol for $F_{\text{DReLU}}^{\text{ring},n}$ in Algorithm 4 that reduces the number of calls to $F_{\text{MILL}}$ to 2. First, we observe that if the input of $P_1$ is identical in all three invocations, then the invocations of OT in Algorithm 1 (steps 9&10) can be done together for the three comparisons. This reduces the communication for each leaf OT invocation in steps 9&10 by an additive factor of $4\ell$. To enable this, $P_0$, $P_1$ add $(n - 1)/2$ to their inputs to $F_{\text{MILL}}$ in steps 1.3 ($\eta = \lceil \log n \rceil$). Hence, $P_1$'s input to
Algorithm 4 Optimized Integer ring DReLU, $\Pi_{\text{DReLU}}^{\text{ring,}n}$

Input: $P_0, P_1$ hold $(a_0^n)$ and $(a_1^n)$, respectively, where $a \in \mathbb{Z}_n$. Let $\eta = \lceil \log n \rceil$.

Output: $P_0, P_1$ get $(\text{DReLU}(a_0^n))_B$ and $(\text{DReLU}(a_1^n))_B$.

1. $P_0$ & $P_1$ invoke an instance of $\mathcal{T}_{\text{Mill}}^{\eta+1}$ where $P_0$’s input is $(3(n-1)/2 - (a_0^n))$, and $P_1$’s input is $(n-1)/2 + (a_1^n)$. For $b \in \{0, 1\}$, $P_b$ learns $\langle \text{wrap} \rangle_B$ as output.
2. $P_0$ sets $x = \left\lfloor 2n - 1 - (a_0^n) \right\rfloor$ if $(a_0^n) > (n-1)/2$, else $x = \left\lfloor n - 1 - (a_0^n) \right\rfloor$.
3. $P_0$ & $P_1$ invoke an instance of $\mathcal{T}_{\text{Mill}}^{\eta+1}$, where $P_0$’s input is $x$ and $P_1$’s input is $(n-1)/2 + (a_1^n)$. For $b \in \{0, 1\}$, $P_b$ learns $\langle \text{xt} \rangle_B$ as output.
4. $P_0$ samples $(z)_B \overset{s}{\leftarrow} \{0, 1\}$.
5. for $j = \{00, 01, 10, 11\}$ do
6. $P_0$ parse $j$ as $j_0||j_1$ and sets $t_j = 1 \oplus \langle \text{xt} \rangle_B \oplus j_0$.
7. if $(a_0^n) > (n-1)/2$ then
8. $P_0$ sets $s_j' = t_j \land \langle \text{wrap} \rangle_B \oplus j_1$.
9. else
10. $P_0$ sets $s_j' = t_j \land \langle \text{wrap} \rangle_B \oplus j_1$.
11. end if
12. $P_0$ sets $s_j = s_j' \oplus (z)_B$.
13. end for
14. $P_0$ & $P_1$ invoke an instance of $(\eta)\text{-OT}_1$ where $P_0$ is the sender with inputs $\langle s_j \rangle$ and $P_1$ is the receiver with input $\langle \text{xt} \rangle_B \oplus (\text{wrap} \rangle_B$. $P_1$ sets its output as $(z)_B$.
15. For $b \in \{0, 1\}$, $P_b$ outputs $(z)_B$.

$\mathcal{T}_{\text{Mill}}^{\eta+1}$ is $(n-1)/2 + (a_0^n)$ in all invocations and $P_0$’s inputs are $(3(n-1)/2 - (a_0^n)), (n-1)/2 + (a_1^n), 2n - 1 - (a_0^n)$ in steps 1, 2, 3, respectively.

Next, we observe that one of the comparisons in step 2 or step 3 is redundant. For instance, if $(a_0^n) > (n-1)/2$, then the result of the comparison $lt = (a_0^n) + (a_1^n) > (n-1)/2$ done in step 2 is always 1. Similarly, if $(a_0^n) \leq (n-1)/2$, then the result of the comparison $rt = 1\langle (a_0^n) + (a_1^n) > n + (n-1)/2 \rangle$ done in step 3 is always 0. Moreover, $P_0$ knows based on her input $(a_0^n)$ which of the two comparisons is redundant. Hence, in the optimized protocol, $P_0$ and $P_1$ always run the comparison to compute shares of wrap and one of the other two comparisons. Note that the choice of which comparison is omitted by $P_0$ need not be communicated to $P_1$, since $P_1$’s input is same in all invocations of $\mathcal{T}_{\text{Mill}}$. Moreover, this omission does not reveal any additional information to $P_1$ by security of $\mathcal{T}_{\text{Mill}}$. Finally, $P_0$ and $P_1$ can run a $(\frac{\eta}{2})\text{-OT}_1$ to learn the shares of DReLU(U,a). Here, $P_1$ is the receiver and her choice bits are the shares learnt in the two comparisons. $P_0$ is the sender who sets the 4 OT messages based on her input share, and two shares learnt from the comparison protocol. We elaborate on this in the correctness proof below.

Correctness and Security. First, by correctness of $\mathcal{T}_{\text{Mill}}^{\eta+1}$ (step 1), wrap $= \text{Reconst}^{\text{B}}(\langle \text{wrap} \rangle_B)$, $(\langle \text{wrap} \rangle_B)_1 = 1\langle (a_0^n) + (a_1^n) > n - 1 \rangle$. Then, $j_f = \langle x \rangle_B \land (\langle \text{wrap} \rangle_B)_1$. Then, $j_f = 1 \oplus x$. We will show that $s_j' = \text{DReLU}(a)$, and hence, by correctness of $(\frac{\eta}{2})\text{-OT}_1, z = \text{Reconst}^{\text{B}}(\langle z \rangle_B, (z)_B) = \text{DReLU}(a)$. We have the following two cases.

When $(a_0^n) > (n-1)/2, \text{lt} = 1, \text{DReLU}(a) = \text{wrap} \land (1 \oplus \text{rt})$. Here, by correctness of $\mathcal{T}_{\text{Mill}}^{\eta+1}$ (step 3), $x = \text{Reconst}^{\text{B}}(\langle x \rangle_B, (x)_B)$. Then, $s_j' = \text{DReLU}(a)$. Hence, $s_j' = t_j \land (\langle \text{wrap} \rangle_B \oplus j_1) = (1 \oplus \text{rt}) \land \text{wrap}$.

When $(a_0^n) \leq (n-1)/2, \text{rt} = 0, \text{DReLU}(a) = 1 \land \text{rt}$ if wrap is 0, else 1. It can be written as $(1 \oplus \text{lt}) \land (\text{lt} \land \text{wrap})$. In this case, by correctness of $\mathcal{T}_{\text{Mill}}^{\eta+1}$ (step 3), $x = \text{Reconst}^{\text{B}}(\langle x \rangle_B, (x)_B) = \text{lt}$. Hence, $s_j' = t_j \land (\langle \text{wrap} \rangle_B \oplus j_1) = (1 \oplus \text{lt}) \land \text{wrap}$. Since $(z)_B$ is uniform, security follows in the $(\mathcal{T}_{\text{Mill}}^{\eta+1}, (\frac{\eta}{2})\text{-OT}_1)$-hybrid.

Communication Complexity. With the above optimization, the overall communication complexity of our protocol for DReLU in $\mathbb{Z}_n$ is equivalent to $2$ calls to $\Pi_{\text{Mill}}^{\eta+1}$ where $P_1$ has same input plus $2\lambda + 4$ (for protocol for $(\frac{\eta}{2})\text{-OT}_1$). Two calls to $\Pi_{\text{Mill}}^{\eta+1}$ in this case (using $m = 4$) cost $< \frac{4\lambda}{2}(\eta + 1) + 28(\eta + 1) + 1$ bit. Hence, total communication is $< \frac{4\lambda}{2}(\eta + 1) + 28(\eta + 1) + 1$. We note that the communication complexity of simplified protocol in Algorithm 3 is approximately $3$ independent calls to $\Pi_{\text{Mill}}^{\eta+1}$ which cost $3(\lambda\eta + 14\eta)$ bits, plus $2\lambda + 4$ bits for $\mathcal{T}_{\text{Mill}}^{\eta+1}$. Thus, our optimization gives almost $2\times$ improvement.

4 DIVISION AND TRUNCATION

We present our results on secure implementations of division in the ring by a positive integer and truncation (division by power-of-2) that are bitwise equivalent to the corresponding clear-text computation. We begin with closed form expressions for each of these followed by secure protocols that use them.

4.1 Expressing general division and truncation using arithmetic over secret shares

Let $\text{idiv} : \mathbb{Z} \times \mathbb{Z} \rightarrow \mathbb{Z}$ denote signed integer division, where the quotient is rounded towards $-\infty$ and the sign of the remainder is the same as that of divisor. We denote division of a ring element by a positive integer using $\text{rdiv} : \mathbb{Z}_n \times \mathbb{Z} \rightarrow \mathbb{Z}_n$ defined as

$$\text{rdiv}(a, d) = \text{idiv}(a - \left\lfloor \frac{a}{d} \right\rfloor \cdot n, d) \mod n.$$  \hspace{1cm} (2)

Then, the integer $a_0 \in \{0, 1, \ldots, n-1\}$ is the unsigned representation of $a \in \mathbb{Z}_n$ lifted to integers and $0 < d < n$. For brevity, we use $x \equiv a \pmod{n}$ to denote $x \mod n = y \mod n$.

**Theorem 4.1.** (Division of ring element by positive integer). Let the shares of $a \in \mathbb{Z}_n$ be $(a_0^n, (a_1^n) \in \mathbb{Z}_n$, for some $n = n^1 \cdot d + n^0 \in \mathbb{Z}$, where $n^0, n^1, d \in \mathbb{Z}$, and $0 \leq n^0 < d < n$.

Let the unsigned representation of $a, (a_0^n, (a_1^n) \in \mathbb{Z}_n$ lifted to integers be $a_0, a_0, a_1, \in \{0, 1, \ldots, n-1\}$, respectively, such that $a_0 = a_0^n \cdot d + a_0^0$ and $a_1 = a_1^n \cdot d + a_1^0, \text{where } a_0^0, a_0^0, a_1^0, a_1^0 \in \mathbb{Z}$ and $0 \leq a_0^0, a_1^0 < d$. Let $n' = \left\lceil \frac{n}{2} \right\rceil \in \mathbb{Z}$. Define $corr, A, B, C \in \mathbb{Z}$,
4.2 Protocols for division

In this section, we describe our protocols for division in different settings. We first describe a protocol for the simplest case of truncation for \( b \)-bit integers followed by a protocol for general division in \( \mathbb{Z}_n \) by a positive integer (Section 4.2.2). Finally, we discuss another simpler case of truncation, which allows us to do better than general division for rings with a special structure (Section 4.2.3).

4.2.1 Protocol for truncation of \( b \)-bit integers. Let \( F_{\text{Trunc}}^{\text{int,}f,s} \) be the functionality that takes arithmetic shares of an input and returns arithmetic shares of an output. In this work, we give a protocol (Algorithm 5) that realizes the functionality \( F_{\text{Trunc}}^{\text{int,}f,s} \) correctly building on Corollary 4.2.

\begin{algorithm}[!ht]
\caption{Truncation, \( F_{\text{Trunc}}^{\text{int,}f,s} \)}
\begin{algorithmic}[1]
\Require For \( b \in \{0,1\} \), \( P_b \) holds \((a)_{b}^{L} \), where \( a \in \mathbb{Z}_L \).
\Ensure For \( b \in \{0,1\} \), \( P_b \) learns \((z)_{b}^{L} \) s.t. \( z = a \gg s \).
\State For \( b \in \{0,1\} \), let \( a_0, a_1 \in \mathbb{Z} \) be as defined in Corollary 4.2.
\State For \( b \in \{0,1\} \), \( P_b \) invokes \( F_{\text{Trunc}}^{\text{int,}f,s} \) with input \((a)_{b}^{L} \) to learn output \((z)_{b}^{L} \).
\State For \( b \in \{0,1\} \), \( P_b \) sets \((m)_{b}^{L} = (a)_{b}^{R} \).
\State For \( b \in \{0,1\} \), \( P_b \) sets \( x_b = \text{MSB}((a)_{b}^{L}) \).
\State \( P_0 \) samples \((c)_{b}^{L} \rightleftarrows Z_{xf} \).
\For {\( j \in \{00,01,10,11\} \)}
\State \( P_0 \) computes \( t_j = ((m)_{b}^{R} \oplus j_{0} \oplus x_0) \land ((m)_{b}^{R} \oplus j_{1} \oplus x_1) \) s.t. \( j = (j_{0} || j_{1}) \).
\If {\( t_j \land \{ x_0 = 0 \} \)}
\State \( P_0 \) sets \( s_j = -(c)_{b}^{R} - 1 \).
\ElseIf {\( t_j \land \{ x_0 = 1 \} \)}
\State \( P_0 \) sets \( s_j = -(c)_{b}^{R} + 1 \).
\Else
\State \( P_0 \) sets \( s_j = (c)_{b}^{R} \).
\EndIf
\EndFor
\State \( P_0 \) and \( P_1 \) invoke an instance of \((\delta_{b}^{L})\) \(-\text{OT}_{T} \), where \( P_0 \) is the sender with inputs \{\( s_j \}_j \) and \( P_1 \) is the receiver with input \( ((m)_{b}^{R} || x_1) \) and learns \((c)_{b}^{L} \).
\State \( P_0 \) and \( P_1 \) invoke an instance of \( F_{\text{B2A}}^{\text{int,}f,s} \) with input \((c)_{b}^{R} \) and learns \((d)_{b}^{L} \).
\State For \( b \in \{0,1\} \), \( P_b \) invokes an instance of \( F_{\text{B2A}}^{\text{int,}f,s} \) with input \((c)_{b}^{R} \) and learns \((d)_{b}^{L} \).
\State \( P_b \) outputs \((z)_{b}^{L} = ((a)_{b}^{L} \gg s) + (c)_{b}^{L} \cdot 2^{l-s} + (d)_{b}^{L} \), \( b \in \{0,1\} \).
\end{algorithmic}
\end{algorithm}

for \( b \in \{0,1\} \), and let \( f' = ((m)_{b}^{R} || x_1) \). Then, \( t_j' = ((m)_{b}^{R} \oplus \langle \langle m \rangle_{b}^{R} \oplus x_0 \rangle \land (\langle m \rangle_{b}^{R} \oplus \langle m \rangle_{b}^{R} \oplus x_1) = (\text{MSB}(a) \oplus x_0) \land (\text{MSB}(a) \oplus x_1) \). Now, \( t_j = 1 \) implies that we are in one of the first two cases of expression for \( c \) which case we are in can be checked using \( x_0 \) steps (7 & 9). Now it is easy to see that \( s_j = -(c)_{b}^{R} \gg s + (c)_{b}^{L} \).

Next, by correctness of \( F_{\text{B2A}}^{\text{int,}f,s} \) in Corollary 4.2, \( z = 1(a_0^{L} + a_1^{L} \geq 2^s) \). Given boolean shares of \( c \), \( \text{step 17} \), creates arithmetic shares of the same using an instance of \( F_{\text{B2A}}^{\text{int,}f,s} \). Since \( (c)_{b}^{L} \) is uniformly random, security of our protocol is easy to see in \( (F_{\text{B2A}}^{\text{int,}f,s}) \cdot \text{OT}_{T} \), \( F_{\text{B2A}}^{\text{int,}f,s} \) and \( F_{\text{B2A}}^{\text{int,}f,s} \).

Communication complexity: \( |F_{\text{B2A}}^{\text{int,}f,s}(\cdot)\text{-OT}_{T} \) and \( |F_{\text{B2A}}^{\text{int,}f,s}(\cdot)\text{-OT}_{T} \). Hence, communication required is \( < \lambda \cdot 2^{2s} + 19\lambda \) for communication for \( F_{\text{B2A}}^{\text{int,}f,s} \) that depends on parameter \( s \). For \( \lambda = 32 \) and \( s = 12 \), our concrete communication is 4310 bits (using \( m = 7 \) for \( F_{\text{B2A}}^{\text{int,}f,s} \) and \( P_{\text{MILL}}^{\text{int,}f,s} \) inside \( F_{\text{B2A}}^{\text{int,}f,s} \)) as opposed to 2464 bits for garbled circuits.

4.2.2 Protocol for division in ring. Let \( F_{\text{DIV}}^{\text{ring,}n,d} \) be the functionality for division that takes arithmetic shares of an input and returns arithmetic shares of \( \text{rdiv}(a,d) \) as output. Our protocol builds on our closed form expression from Theorem 4.1. We note that \( b \)-bit integers is a special case of \( Z_n \) and we use the same protocol for
division of an element in $\mathbb{Z}_L$ by a positive integer.

**Intuition.** This protocol is similar to the previous protocol for truncation and uses the same logic to compute shares of corr term. Most non-trivial term to compute is $C$ that involves three signed comparisons over $\mathbb{Z}$. We emulate these comparisons using calls to $F^\text{Direlu}_{\text{B2A}}$ where $\delta$ is large enough to ensure that there are no overflows or underflows. It is not too hard to see that $-2d + 2 \leq A \leq 2d - 2$ and hence, $-3d + 2 \leq A - d, A + d \leq 3d - 2$. Hence, we set $\delta = \lceil \log 6d \rceil$. Now, with this value of $\delta$, the term $C$ can be re-written as $(\text{DRelu}(A - d) \oplus 1) + (\text{DRelu}(A) \oplus 1) + (\text{DRelu}(A + d) \oplus 1)$, which can be computed using three calls to $F^\text{DRelu}_{\text{B2A}}$ (Step 19) and $F^\text{B2A}_{\text{COT}}$ (Step 20) each. Finally, note that to compute $C$ we need arithmetic shares of $A$ over the ring $\mathbb{Z}_A, A = 2^6$. And this requires shares of corr over the same ring. Hence, we compute shares of corr over both $\mathbb{Z}_n$ and $\mathbb{Z}_A$ (Step 15). Due to space constraints, we describe the protocol formally in Appendix D along with its communication complexity. Also, Table 3 provides theoretical and concrete communication numbers for division in both $\mathbb{Z}_L$ and $\mathbb{Z}_n$, as well as a comparison with garbled circuits.

#### 4.2.3 Truncation in rings with special structure

It is easy to see that truncation by $s$ in general rings can be done by performing a division by $d = 2^s$. However, we can omit a call to $F^\text{DRelu}_{\text{B2A}}$ and $F^\text{B2A}_{\text{COT}}$ when the underlying ring and $d$ satisfy a relation. Specifically, if we have $2 \cdot d^0 \leq d = 2^s$, then $A$ is always greater than equal to $-d$, where $n^0, A \in \mathbb{Z}$ are as defined in Theorem 4.1. Thus, the third comparison ($A < -d$) in the expression of $C$ from Theorem 4.1 can be omitted. Moreover, this reduces the value of $\delta$ needed and $\delta = \lceil \log 4d \rceil$ suffices since $-2d \leq A - d, A \leq 2d - 2$.

Our homomorphic encryption scheme requires $n$ to be a prime of the form $2KN + 1$ (Section 2.2.4), where $K$ is a positive integer and $N \geq 8192$ is a power-of-2. Thus, we have $n^0 = n \mod 2^s = 1$ for $1 \leq s \leq 14$. For all our benchmarks, $s \leq 12$ and we use this optimization for truncation in $\text{SCI}_{\text{HE}}$.

### 5 SECURE INFERENCE

We give an overview of all the layers that must be computed securely to realize the task of secure neural network inference. Layers can be broken into two categories - linear and non-linear. An inference algorithm simply consists of a sequence of layers of appropriate dimension connected to each other. Examples of linear layers include matrix multiplication, convolutions, Avgpool and batch normalization, while non-linear layers include ReLU, Maxpool, and Argmax.

We are in the setting of secure inference where the model owner, say $P_0$, holds the weights. When securely realizing each of these layers, we maintain the following invariant: Parties $P_0$ and $P_1$ begin with arithmetic shares of the input to the layer and after the protocol, end with arithmetic shares (over the same ring) of the output of the layer. This allows us to stitch protocols for arbitrary layers sequentially to obtain a secure computation protocol for any neural network comprising of these layers. Semi-honest security of the protocol will follow trivially from sequential composibility of individual sub-protocols [18, 32, 47]. For protocols in $\text{SCI}_{\text{OT}}$, this arithmetic secret sharing is over $\mathbb{Z}_l$; in $\text{SCI}_{\text{HE}}$, the sharing is over $\mathbb{Z}_n$, prime $n$. The inputs to secure inference are floating-point numbers, encoded as fixed-point integers in the ring $(\mathbb{Z}_d, \mathbb{Z}_n)$; for details see Appendix E.

#### 5.1 Linear Layers

**Fully connected layers and convolutions.** A fully connected layer in a neural network is simply a product of two matrices - the matrix of weights and the matrix of activations of that layer - of appropriate dimension. At a very high level, a convolutional layer applies a filter (usually of dimension $f \times f$ for small integer $f$) to the input matrix by sliding across it and computing the sum of element-wise products of the filter with the input. Various parameters are associated with convolutions - e.g. stride (a stride of 1 denotes that the filter slides across the larger input matrix beginning at every row and every column) and zero-padding (which indicates whether the matrix is padded with 0s to increase its dimension before applying the filter). When performing matrix multiplication or convolutions over fixed-point values, the values of the final matrix must be scaled down appropriately so that it has the same scale as the inputs to the computation. Hence, to do faithful fixed-point arithmetic, we first compute the matrix multiplication or convolution over the ring $(\mathbb{Z}_L, \mathbb{Z}_n)$ followed by truncation, i.e., division-by-$2^s$ of all the values. In $\text{SCI}_{\text{OT}}$, multiplication and convolutions over the ring $\mathbb{Z}_L$ are done using oblivious transfer techniques and in $\text{SCI}_{\text{HE}}$ these are done over $\mathbb{Z}_n$ using homomorphic encryption techniques that we describe next followed by our truncation method.

**OT based computation.** The OT-based techniques for multiplication are well-known [8, 23, 51] and we describe them briefly for completeness. First consider the simple case of secure multiplication of $a$ and $b$ in $\mathbb{Z}_L$ where $P_0$ knows $a$ and $P_1$ and $P_1$ hold arithmetic shares of $b$. This can be done by invoking $(\hat{1}_i \cdot \text{COT})$ for $i \in \{1, \ldots, t\}$ requiring communication equivalent to $t$ instances of $(\hat{1}_i \cdot \text{COT})^{\mathbb{Z}_L}_{\mathbb{Z}_L}$. Using this, multiplying two matrices $A \in \mathbb{Z}_L^{M \times N}$ and $B \in \mathbb{Z}_L^{N \times K}$ such that $P_0$ knows $A$ and $B$ is arithmetically secret shared requires $MNKt$ instances of $(\hat{1}_i \cdot \text{COT})^{\mathbb{Z}_L}_{\mathbb{Z}_L}$. This can be optimized with structured multiplications inside a matrix multiplication by combining all the COT sender messages when multiplying with the same element, reducing the complexity to that of $NKt$ instances of $(\hat{1}_i \cdot \text{COT})^{\mathbb{Z}_L}_{\mathbb{Z}_L}$. Finally, we reduce the task of secure convolutions to secure matrix multiplication similar to [45, 50, 61].

**HE based computation.** $\text{SCI}_{\text{HE}}$ uses techniques from Gazelle [43] and Delphi [49] to compute matrix multiplications and convolutions over a field $\mathbb{Z}_n$ (prime $n$), of appropriate size. At a high level, first, $P_1$ sends an encryption of its arithmetic share to $P_0$. Then, $P_0$ homomorphically computes on this ciphertext using weights of the model (known to $P_0$) to compute an encryption of the arithmetic share of the result and sends this back to $P_1$. Hence, the communication only depends on the input and output size of the linear layer and is independent of the number of multiplications being performed. Homomorphic operations can have significantly high computational cost - to mitigate this, we build upon the output rotations method from [43] for performing convolutions, and reduce its
number of homomorphic rotations. At a very high level, after performing convolutions homomorphically, ciphertexts are grouped, rotated in order to be correctly aligned, and then packed using addition. In our work, we divide the groups further into subgroups that are misaligned by the same offset. Hence the ciphertexts within a subgroup can first be added and the resulting ciphertext can then be aligned using a single rotation as opposed to subgroup-size many rotations in [43]. We refer the reader to Appendix F for details.

**Faithful truncation.** To correctly emulate fixed-point arithmetic, the value encoded in the shares obtained from the above methods needs to be divided-by-$2^s$, where $s$ is the scale used. For this we invoke $\mathcal{F}_{\text{int,div}}$ in SCI$_{\text{OT}}$ and $\mathcal{F}_{\text{ring,2^s}}$ in SCI$_{\text{HE}}$ for each value of the resulting multiplicative device. With this, the result of secure implementation of fixed-point multiplication and convolutions is **bitwise equal** to the corresponding cleartext execution. In contrast, many prior works on 2PC [49, 51] and 3PC [45, 50, 61] used a **local truncation** method for approximate truncation based on a result from [51]. Here, the result can be arbitrarily wrong with a (small) probability $p$ and with probability $1 - p$ the result can be wrong in the last bit. Since $p$ grows with the number of truncations, these probabilistic errors are problematic for large DNNs. Moreover, even if $p$ is small, 1-bit errors can accumulate and the results of cleartext execution and secure execution can diverge; this is undesirable as it breaks correctness of 2PC.

5.1.2 Avgpool$_d$. The function Avgpool$_d(a_1, \ldots, a_d)$ over a pool of $d$ elements $a_1, \ldots, a_d$ is defined to be the arithmetic mean of these $d$ values. The protocol for computing this function works as follows: $P_0$ and $P_1$ begin with arithmetic shares (e.g. over $\mathbb{Z}_d$ in SCI$_{\text{OT}}$) of $a_i$, for all $i \in [d]$. They perform local addition to obtain shares of $w = \sum_{i=1}^{d} a_i$ (i.e., $P_0$ computes $\langle w \rangle_b^i = \sum_{i=1}^{d} \langle a_i \rangle_b^i$). Then, parties invoke $\mathcal{F}_{\text{DIV}}^{\text{ring},L_d}$ on inputs $\langle w \rangle_b^i$ to obtain the desired output. Correctness and security follow in the $\mathcal{F}_{\text{DIV}}^{\text{ring},L_d}$ hybrid model. Here too, unlike [49], our secure execution of average pool is bitwise equal to the cleartext version.

5.2 Nonlinear Layers

5.2.1 ReLU. Note that ReLU($a$) = $a$ if $a \geq 0$, and 0 otherwise. Equivalently, ReLU($a$) = DReLU($a$) · $a$. For $\mathbb{Z}_d$, first we compute the boolean shares of DReLU($a$) using a call to $\mathcal{F}_{\text{int,div}}$ in SCI$_{\text{HE}}$, and then we compute shares of ReLU($a$) using a call to multiplexer $\mathcal{F}_{\text{MUX}}^L$ (Section 2.2.3). We describe the protocol for ReLU($a$) over $\mathbb{Z}_d$ formally in Algorithm 8, Appendix B (the case of $\mathbb{Z}_d$ follows in a similar manner). For communication complexity, refer to Table 2 for comparison with garbled circuits and Appendix B for details.

5.2.2 Maxpool$_d$ and Argmax$_d$. The function Maxpool$_d(a_1, \ldots, a_d)$ over $d$ elements $a_1, \ldots, a_d$ is defined in the following way. Define $g_t(x, y) = z$, where $w = x - y$ and $z = x$, if $w > 0$ and $z = y$, if $w \leq 0$. Define $z_1 = a_1$ and $z_i = g_t(a_i, z_{i-1})$, recursively for all $2 \leq i \leq d$. Now, Maxpool$_d(a_1, \ldots, a_d) = z_d$.

We now describe a protocol such that parties begin with arithmetic shares (over $\mathbb{Z}_d$) of $a_i$, for all $i \in [d]$ and end the protocol with arithmetic shares (over $\mathbb{Z}_d$) of Maxpool$_d(a_1, \ldots, a_d)$. For simplicity, we describe how $P_0$ and $P_1$ can compute shares of $z = g_t(x, y)$ (beginning with the shares of $x$ and $y$). It is easy to see then how they can compute Maxpool$_d$. First, parties locally compute shares of $w = x - y$ (i.e., $P_0$ computes $\langle w \rangle_b^i = (x)_b^i - (y)_b^i$, for $b \in \{0, 1\}$). Next, they invoke $\mathcal{F}_{\text{int,div}}$ with input $\langle w \rangle_b^i$ to learn output $(a)_b^i$. Now, they invoke $\mathcal{F}_{\text{MUX}}^L$ with input $\langle w \rangle_b^i$ and $(a)_b^i$ to learn output $(t)_b^i$. Finally, parties output $(z)_b^i = (y)_b^i + (t)_b^i$. The correctness and security of the protocol follows in a straightforward manner. Computing Maxpool$_d$ is done using $d - 1$ invocations of the above sub-protocol in $d - 1$ sequential steps.

Argmax$_d(a_1, \ldots, a_d)$ is defined similar to Maxpool$_d(a_1, \ldots, a_d)$, except that its output is an index $i^*$ s.t. $a_{i^*} = \text{Maxpool}_d(a_1, \ldots, a_d)$. Argmax$_d$ can be computed securely similarly to Maxpool$_d(a_1, \ldots, a_d)$.

6 IMPLEMENTATION

We implement our cryptographic protocols in a library and integrate them into the CrypTFlow framework [1, 45] as a new cryptographic backend. CrypTFlow compiles high-level TensorFlow [3] inference code to secure computation protocols using its frontend Athos, that are then executed by its cryptographic backends. We modify the truncation behavior of Athos in support of faithful fixed-point arithmetic. We start by describing the implementation of our cryptographic library, followed by the modifications that we made to Athos.

6.1 Cryptographic backend

To implement our protocols, we build upon the ($1^\ell$)-OT$_T$ implementation from EMP [62] and extend it to ($1^\ell$)-OT$_T$ using the protocol from [44]. Our linear-layer implementation in SCI$_{\text{HE}}$ is based on SEAL/Delphi [2, 58] and in SCI$_{\text{OT}}$ is based on EMP. All our protocol implementations are multi-threaded.

Oblivious Transfer. ($1^\ell$)-OT$_T$ requires a correlation robust function to mask the sender’s messages in the OT extension protocol, and we use AES$_{256}^\text{RK}$ (re-keyed AES with 256-bit key)$^9$ to instantiate it as in [23, 25]. We incorporated the optimizations from [33, 34] for AES key expansion and pipelining these AES$_{256}^\text{RK}$ calls. This leads to roughly 6x improvement in the performance of AES$_{256}^\text{RK}$ calls, considerably improving the overall execution time of ($1^\ell$)-OT$_T$ (e.g. $2.7\times$ over LAN for ($1^\ell$)-OT$_2$).

Millionaires’ protocol. Recall that $m$ is a parameter in our protocol $\Pi^m_{\text{Mill}}$. While we discussed the dependence of communication complexity on $m$ in Section 3.1.2, here we discuss its influence on the computational cost. Our protocol makes $t/m$ calls to ($1^\ell$)-OT$_2$ (after merging steps 9&10), where $M = 2^m$. Using OT extension techniques, generating an instance of ($1^\ell$)-OT$_2$ requires 6 AES$_{256}^\text{RK}$ and $(M+1)$ AES$_{256}^\text{RK}$ evaluations. Thus, the computational cost grows super-polynomially with $m$. We note that for $t = 32$, even though communication is minimized for $m = 7$, empirically we observe that $m = 4$ gives us the best performance under both LAN and WAN settings (communication in this case is about 30% more than when $m = 7$ but computation is $\approx 3 \times$ lower).

---

$^9$There are two types of AES in MPC applications - fixed-key (FK) and re-keyed (RK) [10, 35]. While the former runs key schedule only once and is more efficient, the latter generates a new key schedule for every invocation and is required in this application.
Implementing linear layers in SCI\textsubscript{HE}. To implement the linear layers in SCI\textsubscript{HE}, we build upon the Delphi implementation \cite{2,49}, that is in turn based on the SEAL library \cite{58}. We use the code for fully connected layers as it is from \cite{2}. For convolution layers, we parallelize the code, employ modulus-switching \cite{58} to reduce the ciphertext modulus (and hence ciphertext size), and implement the strided convolutions proposed in Gazelle \cite{43}. These optimizations resulted in significant performance improvement of convolution layers. E.g. for the first convolution layer\textsuperscript{10} of ResNet50, the runtime decreased from 306s to 18s in the LAN setting and communication decreased from 204 MiB to 76 MiB.

6.2 CrypTFlow integration

We integrate SCI\textsubscript{OT} and SCI\textsubscript{HE} as new cryptographic backends into the CrypTFlow framework \cite{1,45}. CrypTFlow’s TensorFlow frontend Athos outputs fixed-point DNNs that use 64-bit integers and sets an optimal scale using a validation set. CrypTFlow required a bitwidth of 64 to ensure that the probability of local truncation errors in its protocols is small (Section 5.1.1). Since our protocols are correct and have no such errors, we extend Athos to set both the bitwidth and the scale optimally by autotuning on the validation set. The bitwidth and scale leak information about the weights and this leakage is similar to the prior works on secure inference \cite{43,45,48–51,61}.

Implementing faithful truncations using $\Pi_{\text{Trunc}}^{\text{int,fs}}$ requires the parties to communicate. We implement the following peephole optimizations in Athos to reduce the cost of these truncation calls. Consider a DNN having a convolution layer followed by a ReLU layer. While truncation can be done immediately after the convolution, moving the truncation call to after the ReLU layer can reduce the cost of our protocol $\Pi_{\text{Trunc}}^{\text{int,fs}}$. Since the values after ReLU are guaranteed to be all positive, the call to $f_{\text{DReLU}}^{\text{int,f}}$ within it (step 2 in Algorithm 5) now becomes redundant and can be omitted. Our optimization further accounts for operations that may occur between the convolutions and ReLU, say a matrix addition. Moving the truncation call from immediately after convolution to after ReLU means the activations flowing into the addition operation are now scaled by 2s, instead of the usual s. For the addition operation to then work correctly, we scale the other argument of addition by s as well. These optimizations are fully automatic and need no manual intervention.

7 EXPERIMENTS

We empirically validate the following claims:

- We show the first empirical evaluation of two-party secure inference on ImageNet-scale benchmarks (Section 7.3). These results show the trade-offs between OT and HE-based secure DNN inference (Table 7).

We start with a description of our experimental setup and benchmarks, followed by the results.

Experimental Setup. We ran our benchmarks in two network settings, namely, a LAN setting with both machines situated in West Europe, and a transatlantic WAN setting with one of the machines in East US. The bandwidth between the machines is 377 MBps and 40 MBps in the LAN and the WAN setting respectively and the echo latency is 0.3ms and 80ms respectively. Each machine has commodity class hardware: 3.7 GHz Intel Xeon processor with 4 cores and 16 GBs of RAM.

Our Benchmarks. We evaluate on the ImageNet-scale benchmarks considered by \cite{45}: SqueezeNet \cite{40}, ResNet50 \cite{37}, and DenseNet121 \cite{38}. To match the reported accuracies, we need 37-bit fixed-point numbers for ResNet50, whereas 32 bits suffice for DenseNet121 and SqueezeNet (Appendix I). Recall that our division protocols lead to correct secure executions and there is no accuracy loss in going from cleartext inference to secure inference. Appendix G provides a brief summary of these benchmarks.

7.1 Comparison with Garbled Circuits

We compare with EMP-toolkit \cite{62}, the state-of-the-art library for Garbled Circuits (GC). Figure 1 shows the improvement of our ReLU protocols over GC in both LAN and WAN settings. On the x-axis, which is in log-scale, the number of ReLUs range from $2^0$ to $2^{20}$. The histogram shows, using the right y-axis, the cumulative number of layers in our benchmarks (SqueezeNet, ResNet50, DenseNet121) which require the number of ReLU activations given on the x-axis. We observe that these DNNs have layers that compute between $2^{13}$ and $2^{20}$ ReLUs. For such layers, we observe (on the left y-axis) that

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{image.png}
\caption{The left y-axis shows (GC Time). The right y-axis shows the total number of ReLU layers corresponding to each layer size in our benchmark set. The legend entries denote the input domain and the network setting.}
\end{figure}

\end{document}
our protocols are 2×–25× faster than GC – the larger the layers the higher the speedups, and gains are larger in the WAN settings. Specifically, for WAN and > 217 ReLUs, the speedups are much higher than the LAN setting. Here, the cost of rounds is amortized over large layers and the communication cost is a large fraction of the total runtime. Note that our implementations perform load-balancing to leverage full-duplex TCP.

Next, we compare the time taken by GC and our protocols in computing the ReLU activations of our benchmarks in Table 4. Our protocol over \( \mathbb{Z}_2 \) is up to 8× and 18× faster than GC in the LAN and WAN settings respectively, while it is \( 7× \) more communication efficient. As expected, our protocol over \( \mathbb{Z}_n \) has even better gains over GC. Specifically, it is up to 9× and 21× faster in the LAN and WAN settings respectively, and has \( 9× \) less communication.

We also performed a similar comparison of our protocols with GC for the Avgpool layers of our benchmarks, and saw up to 51× reduction in runtime and 41× reduction in communication. We report the concrete performance numbers and discuss the results in more detail in Appendix H.

### 7.2 Comparison with Delphi

In this section, we compare with Delphi [49], which is the current state-of-the-art for 2-party secure DNN inference that outperforms [12, 13, 17, 19, 22, 31, 43, 48, 56] in total time as well as the time taken in online phase. It uses garbled circuits for non-linear layers, and we show that with our protocols, the time taken to evaluate the non-linear layers can be decreased significantly.

For a fair evaluation, we demonstrate these improvements on the benchmarks of Delphi [49], i.e., the MiniONN (CIFAR-10) [48] and ResNet32 (CIFAR-100) DNNs with ReLU activations (as opposed to the ImageNet-scale benchmarks for which Delphi has not been optimized). Similar to Delphi, we perform these computations with a bitwidth of 41 in the LAN setting.

In Table 5, we report the performance of Delphi for evaluating the linear and non-linear components of MiniONN and ResNet32 separately, along with the performance of our protocols for the same non-linear computation\(^{11}\). The table shows that the time to evaluate non-linear layers is the bulk of the total time and our protocols are 20×–30× faster in evaluating the non-linear layers. Also note that we reduce the communication by 12× on MiniONN, and require 9× less communication on ResNet32.

Next, we compare the online time of our protocols with the online time of Delphi in Table 6. In the online phase, linear layers take negligible time and all the time is spent in evaluating the non-linear layers. Here, our protocols are an order of magnitude more efficient than Delphi.

### 7.3 Evaluation on practical DNNs

With all our protocols and implementation optimizations in place, we demonstrate the scalability of CrypTFlow2 by efficiently running ImageNet-scale secure inference. Table 7 shows that both our backends, SCI\(_{OT}\) and SCI\(_{HE}\), are efficient enough to evaluate SqueezeNet in under a minute and scale to ResNet50 and DenseNet121.

In the LAN setting, for both SqueezeNet and DenseNet121, SCI\(_{OT}\) performs better than SCI\(_{HE}\) by at least 20% owing to the higher compute in the latter. However, the quadratic growth of communication with bitlength in the linear-layers of SCI\(_{OT}\) can easily drown this difference if we go to higher bitlengths. Because ResNet50, requires 37-bits (compared to 32 in SqueezeNet and DenseNet121) to preserve accuracy, SCI\(_{HE}\) outperforms SCI\(_{OT}\) in both LAN and WAN settings. In general for WAN settings where communication becomes the major performance bottleneck, SCI\(_{HE}\) performs better than SCI\(_{OT}\) by at least 20% owing to the higher compute in the latter. However, the quadratic growth of communication with bitlength in the linear-layers of SCI\(_{OT}\) can easily drown this difference if we go to higher bitlengths. Because ResNet50, requires 37-bits (compared to 32 in SqueezeNet and DenseNet121) to preserve accuracy, SCI\(_{HE}\) outperforms SCI\(_{OT}\) in both LAN and WAN settings. In general for WAN settings where communication becomes the major performance bottleneck, SCI\(_{HE}\) performs better than SCI\(_{OT}\), SCI\(_{OT}\), and SCI\(_{HE}\), one can choose a specific backend depending on the network statistics [17, 53] to get the best secure inference latency. To the best of our knowledge, no prior system provides this support for OT and HE-based secure DNN inference.

### 8 CONCLUSION AND FUTURE WORK

We have presented secure, efficient, and correct implementations of practical 2-party DNN inference that outperform prior work [49] by an order of magnitude in both latency and scale. We evaluate the first secure implementations of ImageNet scale inference, a task that previously required 3PC protocols [7, 45] (which provide weaker security guarantees) or leaking intermediate computations [12].

---

\(^{11}\)Our non-linear time includes the cost of correct truncation.
the future, we would like to consider ImageNet scale secure training. Even though we can run inference on commodity machines, for training we would need protocols that can leverage specialized compute and networking hardware. Like all prior work on 2PC for secure DNN inference, CrypTFlow2 only considers semi-honest adversaries. In the future, we would like to consider malicious adversaries. Another future direction is to help the server in hiding F from the client when computing a classifier F(x, w). Like [43], SCHE can hide some aspects of F: the filter sizes, the strides, and whether a layer is convolutional or fully connected. Thus, SCHE hides more information than OT-based tools [48] but reveals more information than FHE-based tools [13, 31]. We are exploring approaches to hide more information about F while incurring minimal overhead.

REFERENCES

A SUPPORTING PROTOCOLS

Here, we describe supporting protocols that our main protocols rely on.

A.1 Protocol for regular $F_{\text{AND}}$

Regular $F_{\text{AND}}$ can be realized using bit-triples [8], which are of the form $((b^4)_i, (b^8)_i, (b^{16})_i)$, where $b \in \{0, 1\}$ and $d \land e = f$. Using an instance of $^4Tb_{16}$, the parties can generate two bit-triples [25]. We describe this protocol for generating the first triple, and from there, it will be easy to see how to also generate the second triple.
Algorithm 6 Multiplexer, $\Pi_{\text{MUX}}^n$:

Input: For $b \in \{0, 1\}$, $P_b$ holds $\langle a \rangle^B_b$ and $\langle c \rangle^B_b$.
Output: For $b \in \{0, 1\}$, $P_b$ learns $(z)_{\Pi b}^n$ s.t. $z = a$ if $c = 1$, else $z = 0$.

1. For $b \in \{0, 1\}$, $P_b$ picks $r_b \leftarrow \mathbb{Z}_n$.
2. $P_0$ sets $s_0, s_1$ as follows: If $\langle c \rangle^B_0 = 0$, $(s_0, s_1) = (-r_0, -r_0 + \langle a \rangle^B_0)$. Else, $(s_0, s_1) = (-r_0 + \langle a \rangle^B_0, -r_0)$.
3. $P_0$ & $P_1$ invoke an instance of $(\frac{1}{2})$-OT$_1$ where $P_0$ is the sender with inputs $(s_0, s_1)$ and $P_1$ is the receiver with input $\langle c \rangle^B_1$. Let $P_1$’s output be $x_1$.
4. $P_1$ sets $t_0, t_1$ as follows: If $\langle c \rangle^B_1 = 0$, $(t_0, t_1) = (-r_1, -r_1 + \langle a \rangle^B_1)$. Else, $(t_0, t_1) = (-r_1 + \langle a \rangle^B_1, -r_1)$.
5. $P_0$ & $P_1$ invoke an instance of $(\frac{1}{2})$-OT$_2$ where $P_1$ is the sender with inputs $(t_0, t_1)$ and $P_0$ is the receiver with input $\langle c \rangle^B_0$. Let $P_0$’s output be $x_0$.
6. For $b \in \{0, 1\}$, $P_b$ outputs $(z)_{\Pi b}^n = r_b + x_b$.

A.4 Protocol for B2A

We describe our protocol for realizing $\mathcal{F}_{\text{B2A}}^n$ formally in Algorithm 7. For correctness, we need to show that $d = \text{Recons}_{L}(\langle d \rangle^L_0, \langle d \rangle^L_1) = \langle c \rangle^B + \langle c \rangle^B - 2\langle c \rangle^B \langle c \rangle^B$. By correctness of $(\frac{1}{2})$-OT$_1$, $y_1 = x + \langle c \rangle^B \langle c \rangle^B$. Using this, $\langle d \rangle^L_0 = \langle c \rangle^B + 2x$ and $\langle d \rangle^L_1 = \langle c \rangle^B - 2x^2 - \langle c \rangle^B \langle c \rangle^B$. Security follows from the security of $(\frac{1}{2})$-OT$_1$ and communication required is $\lambda + \eta$ bits.

Algorithm 7 Boolean to Arithmetic, $\Pi_{\text{B2A}}^n$:

Input: $P_0, P_1$ hold $\langle a \rangle^R_0$ and $\langle a \rangle^R_1$, respectively, where $c \in \{0, 1\}$.
Output: $P_0, P_1$ learn $(\langle d \rangle^L_0)$ and $(\langle d \rangle^L_1)$, respectively, s.t. $d = c$.

1. $P_0$ & $P_1$ invoke an instance of $(\frac{1}{2})$-OT$_1$ where $P_0$ is the sender with correlation function $f(x) = x + \langle c \rangle^B$ and $P_1$ is the receiver with input $\langle c \rangle^B$. Party $P_0$ learns $x$ and sets $y_0 = x - n - x_1$ and $P_1$ learns $y_1$.
2. For $b \in \{0, 1\}$, $P_b$ computes $(\langle d \rangle^L_b) = \langle c \rangle^B_b - 2 \cdot y_b$.

Algorithm 8 $L$-bit integer ReLU, $\Pi_{\text{ReLU}}^{\text{int},L}$:

Input: $P_0, P_1$ hold $\langle a \rangle^R_0$ and $\langle a \rangle^R_1$, respectively.
Output: $P_0, P_1$ get $(\text{ReLU}(a))^L_0$ and $(\text{ReLU}(a))^L_1$.

1. For $b \in \{0, 1\}$, $P_b$ invokes $\mathcal{F}_{\text{ReLU}}^{\text{int},L}(\langle a \rangle^R_b)$ to learn output $\langle y \rangle^L_b$.
2. For $b \in \{0, 1\}$, $P_b$ invokes $\mathcal{F}_{\text{MUX}}^n$ with inputs $\langle a \rangle^L_b$ and $\langle y \rangle^L_b$ to learn $(z)_{\Pi b}^L$ and sets $(\text{ReLU}(a))^L_b = (z)_{\Pi b}^L$.

Security trivially follows in $(\frac{1}{2})$-OT$_1$-hybrid. Communication complexity is $2(\lambda + 2\eta)$.

B PROTOCOL FOR ReLU

We describe our ReLU protocol for the case where the input and output shares are over $\mathbb{Z}_n$ in Algorithm 8, and note that the case of $\mathbb{Z}_n$ follows similarly. It is easy to see that the correctness and security of the protocol hold in the $(\mathcal{F}_{\text{ReLU}}^{\text{int},L}, \mathcal{F}_{\text{MUX}}^n)$-hybrid.

Communication complexity. We first look at the complexity of $\Pi_{\text{ReLU}}^{\text{int},L}$, which involves a call to $\mathcal{F}_{\text{ReLU}}^{\text{int},L}$ and $\mathcal{F}_{\text{MUX}}^n$. $\mathcal{F}_{\text{ReLU}}^{\text{int},L}$ has the same communication as $\mathcal{F}_{\text{MILL}}^n$, which requires $\lambda(f - 1) + 13 \cdot (f - 1) - 2\lambda - 22$ bits if we assume $m = 4$ and $m \mid (f - 1)$, and exclude optimization (3.1.1) in the general expression from Section 3.1.2. $\mathcal{F}_{\text{MUX}}^n$ inverts a cost of $2\lambda + 4f$ bits, bringing the total cost to $\lambda^2 + 17\lambda - \lambda - 35 + 2\lambda^2$ bits, which can be rewritten as $< \lambda + 18f$. We get our best communication for $f = 32$ (with all the optimizations) by taking $m = 7$ for the $\Pi_{\text{MILL}}^{\text{int},L}$ invocation inside $\Pi_{\text{ReLU}}^{\text{int},32}$, which gives us a total communication of 3298 bits.

Now, we look at the complexity of $\Pi_{\text{ring}}^n$, which makes calls to $\mathcal{F}_{\text{ReLU}}^{\text{ring},n}$ and $\mathcal{F}_{\text{MUX}}^n$. The cost of $\mathcal{F}_{\text{ring},n}$ is $2\lambda + 4$ bits for $(\frac{1}{2})$-OT$_1$, plus $\frac{1}{2}\lambda(\lambda + 1) + 27\eta + 4\lambda - 44\lambda$ for bit invocations of $\mathcal{F}_{\text{MILL}}^n$, where $P_1$’s input is the same in both invocations and the same assumptions are made as for the expression of $\mathcal{F}_{\text{MILL}}^n$ above. The cost of $\mathcal{F}_{\text{MUX}}^n$ is $2\lambda + 4\eta$ bits, and thus, the total cost is $\frac{1}{2}\lambda(\eta + 1) + 31\eta - 13$, which
can be rewritten as $\frac{3}{2} \lambda(\eta + 1) + 31\eta$. Concretely, we get the best communication for $\eta = 32$ by taking $m = 7$ for the millionaire invocations, getting a total communication of 5288 bits.

C PROOF OF DIVISION THEOREM

Here, we prove Theorem 4.1.

Proof. From Equation 2, we can write $\text{rdiv}((a)_{1}^n, d)$ as:

$$\text{rdiv}((a)_{1}^n, d) = a_{0} \text{idiv}(a_{1} - (1 \cdot a_{1} \geq n') \cdot n, d)$$

$$= a_{0} \text{idiv}(a_{1} \cdot d + a_{0} \cdot 1 \cdot a_{1} \geq n') \cdot (n \cdot d + n') \cdot d)$$

$$= a_{0}^{2} - 1 \cdot a_{1} \geq n') \cdot n \cdot d + \text{idiv}(a_{0}^{2} - 1 \cdot a_{1} \geq n') \cdot n' \cdot d, d)$$

(3)

for $i \in \{0, 1\}$. $a_{0}$ can be expressed as $a_{0} = a_{0} + a_{1} - w \cdot n$, where the wrap-bit $w = 1 \cdot a_{0} + a_{1} \geq n)$. We can rewrite this as:

$$a_{0} = a_{0} + a_{1} - w \cdot n$$

$$= (a_{1} + n \cdot d) \cdot d + (a_{0} + d \cdot w \cdot n')$$

$$= (a_{1} + n \cdot d + k \cdot d + (a_{0} + d \cdot w \cdot n' - k \cdot d), d)$$

(4)

for some integer $k$ such that $0 \leq a_{0} + a_{1} - w \cdot n' - k \cdot d < d$. Similar to Equation 3 and from Equation 4, we can write $\text{rdiv}(a, d)$ as:

$$\text{rdiv}(a, d) = a_{0}^{2} + a_{1} - w \cdot n' + k \cdot d - 1 \cdot a_{1} \geq n') \cdot n$$

$$+ \text{idiv}(a_{0}^{2} - 1 \cdot a_{1} \geq n') \cdot n + (d \cdot w \cdot n') + k \cdot d - 1 \cdot a_{1} \geq n') \cdot n' \cdot d, d)$$

$$= a_{0}^{2} + a_{1} - w \cdot n' \cdot 1 \cdot a_{1} \geq n') \cdot n + \text{idiv}(a_{0}^{2} - 1 \cdot a_{1} \geq n') \cdot n' \cdot d, d)$$

(5)

From Equations 3 and 5, we have the following correction term:

$$c = a_{0} \text{idiv}(a, d) = \text{idiv}(a, d)^{n}$$

$$= a_{0} \cdot 1 \cdot (a_{0} \geq n') \cdot n + \text{idiv}(a_{0}^{2} - 1 \cdot a_{1} \geq n') \cdot n + \text{idiv}(a_{0}^{2} - 1 \cdot a_{1} \geq n') \cdot n' \cdot d)$$

$$= c_{1} \cdot n + c_{0} \cdot B$$

(7)

Let $A'_{1} = \text{idiv}(a_{0}^{2} + a_{1} - i \cdot n', d)$. Then the values of the correction terms $c_{1}$ and $c_{0}$ are as summarized in Table 8.

From the table, we have $c_{1} = \text{corr} \cdot n + c_{0} \cdot B$. Thus, adding $\text{corr} \cdot n + B \mod n$ to $\text{rdiv}(a_{0}^{n}, d) + \text{rdiv}(a_{1}^{n}, d)$ accounts for all the correction terms except $c_{0} \mod n$.

Now all that remains to be proven is that $c_{0} = 1 - C$. Let $C_{0} = 1 \cdot (A < d)$, $C_{1} = 1 \cdot (A < d)$, and $C_{2} = 1 \cdot (A < d)$. Then, we have $C = C_{0} + C_{1} + C_{2}$. Note from the theorem statement that $A = a_{0}^{2} + a_{1}^{2}$ and $A = a_{0}^{2} + a_{0}^{2} - n' - n$ for the cases corresponding to rows 1 and 8 respectively from the table, while $A = a_{0}^{2} + a_{0}^{2} - n' - n$ for the rest of cases. Thus, it is easy to see that $c_{0} = \text{idiv}(A, d)$. Also note that $-2 \cdot d + 2 \leq A \leq 2 \cdot d - 2$, implying that the range of $c_{0}$ is $\{-2, -1, 0, 1\}$. Now we look at each value assumed by $c_{0}$ separately as follows:

- $c_{0} = 2$: In this case, we have $(A < d)$, implying $C_{0} = C_{1} = 1$, and $c_{0} = -2$.
- $c_{0} = 1$: In this case, we have $(A < d)$, implying $C_{0} = C_{1} = 1$, and $c_{0} = -1$.
- $c_{0} = 0$: In this case, we have $(A < d)$, implying $C_{0} = C_{1} = 1$, and $c_{0} = 0$.
- $c_{0} = 1$: In this case, we have $(A < d)$, implying $C_{0} = C_{1} = 1$, and $c_{0} = 1$.

Thus, $c = n \cdot corr \cdot n + 1 \cdot (C - B) = n \cdot \text{rdiv}(a, d) - \text{rdiv}(a_{0}^{n}, d) = \text{rdiv}(a_{1}^{n}, d)$.

D PROTOCOL FOR GENERAL DIVISION

We describe our protocol for general division formally in Algorithm 9. As discussed in Section 4.2.2, our protocol builds on Theorem 4.1 and we compute the various sub-terms securely using our new protocols. Let $\eta = \lfloor \log 6d \rfloor$. We compute the shares of $\text{corr}$ over both $\mathbb{Z}_{2}$ and $\mathbb{Z}_{\Delta}$ (Step 15). We write the term $C$ as $(\text{DReLU}(A - d) \oplus 1) + (\text{DReLU}(A) \oplus 1) + (\text{DReLU}(A + d) \oplus 1)$, which can be computed using three calls to $\mathcal{F}_{\text{DIV}}^{\text{INT},\delta}$ (Step 19) and $\mathcal{F}_{\text{B2A}}^{n}$ (Step 20) each.

Correctness and Security. First, $m = \text{Reconst}^{B}((m)_{0}^{B}, (m)_{1}^{B}) = \text{Reconst}^{B}((a_{0}^{B}, (a_{1}^{B})) = 1 \cdot (a \geq n')$. Next, similar to Algorithm 5, $\text{Reconst}^{B}((\text{corr})_{0}^{B}, (\text{corr})_{1}^{B}) = \text{corr} = \text{Reconst}^{B}((\text{corr})_{0}^{B}, (\text{corr})_{1}^{B})$, where $\text{corr}$ is as defined in Theorem 4.1. Given the bounds on value of $A$ (as discussed above), it easy to see that Steps 16 and 17 compute arithmetic shares of $A$, and $A_{0} = (A - d), A_{1} = A, A_{2} = A + d$. Thus, Now, invocation of $\mathcal{F}_{\text{DIV}}^{\text{INT},\delta}$ on shares of $A_{j}$ (Step 19) returns boolean shares of $\gamma = (1 \oplus \text{MSB}(A_{j}))$ over $\delta$ bit integers, which is same as $1 \oplus (1 \cdot A_{j} < 0)$ over $\mathbb{Z}_{\Delta}$. Hence, $C'_{j} = \text{Reconst}^{B}((\text{corr})_{0}^{B}, (\text{corr})_{1}^{B}) = 1 \cdot (A_{j} < 0)$. By correctness of $\mathcal{F}_{\text{B2A}}^{n}$, step 22 computes arithmetic shares of $C$ as defined in Theorem 4.1. In step 23, $B_{0} + B_{1} = B$ as defined. Hence, correctness holds and $(\text{corr})_{B}^{n}$ are shares of $\text{rdiv}(a, d)$.

Given that $(\text{corr})_{0}^{B}$ and $(\text{corr})_{1}^{B}$ are uniformly random, security of the protocol is easy to see in $(\text{corr})_{B}^{n}$-OT$_{\eta}^{3}, \mathcal{F}_{\text{DIV}}^{\text{INT},\delta}$-hybrid.

Communication complexity: $\Pi_{\text{ring},n,d}^{\text{DIV}}$ involves a single call to $\mathcal{F}_{\text{DIV}}^{\text{ring},n,d}$ and $(\text{corr})_{B}^{n}$-OT$_{\eta}^{3},$ and three calls each to $\mathcal{F}_{\text{DIV}}^{\text{INT},\delta}$ and $\mathcal{F}_{\text{B2A}}^{n}$ From Appendix B, we have the cost of $\mathcal{F}_{\text{ring},n,d}^{\text{DIV}}$ as $\frac{1}{4} \Delta + 27 \eta + \frac{1}{2} - 13$ bits. $(\text{corr})_{B}^{n}$-OT$_{\eta}^{3}$ and $3 \times \mathcal{F}_{\text{B2A}}^{n}$ cost $24 + (\eta + d)$ and $3 \times 3 \eta$ bits respectively. Since the cost of $\mathcal{F}_{\text{DIV}}^{\text{INT},\delta}$ and $\mathcal{F}_{\text{B2A}}^{n}$ is $\lambda \cdot 13 \cdot \eta + 3 \cdot 35 \cdot \frac{1}{2}$ bits (see Appendix B), $3 \times \mathcal{F}_{\text{DIV}}^{\text{INT},\delta}$ requires $3 \cdot \Delta \cdot 13 \cdot \eta - 3 \cdot 35 \cdot \frac{1}{2}$ bits.
Algorithm 9 Integer ring division, $\Pi^{\text{ring},n,d}_{\text{DIV}}$:

**Input:** For $b \in \{0, 1\}$, $P_b$ holds $\langle a \rangle^n_b$, where $a \in \mathbb{Z}_n$.

**Output:** For $b \in \{0, 1\}$, $P_b$ learns $\langle z \rangle^n_b$ s.t. $z = \text{rdiv}(a, d)$.

1. For $b \in \{0, 1\}$, let $a_b, a_0^b, a_1^b \in \mathbb{Z}$ and $n_0, n_1, n^b \in \mathbb{Z}$ be as defined in Theorem 4.1. Let $\eta = \lceil \log(n) \rceil$, $\delta = \lceil \log(6d) \rceil$, and $\Lambda = 2^9$.
2. For $b \in \{0, 1\}$, $P_b$ invokes $T^{\text{ring},n}_{\text{ReLU}}$ with input $\langle a \rangle^n_b$ to learn output $\langle a \rangle^n_b$. Party $P_b$ sets $(m)^B_b = \langle a \rangle^B_b \cdot b$.
3. For $b \in \{0, 1\}$, $P_b$ sets $x_b = 1 \{ \langle a \rangle^n_b \geq n^\lambda \}$.
4. $P_0$ samples $\langle \text{corr} \rangle^n_0 \leftarrow \mathbb{Z}_n$ and $\langle \text{corr} \rangle^\Lambda_0 \leftarrow \mathbb{Z}_\Lambda$.
5. For $j = \{00, 01, 10, 11\}$ do
6. $P_0$ computes $t_j = \langle (m)^B_0 \oplus j_0 \oplus x_0 \rangle \land \langle (m)^B_0 \oplus j_0 \oplus j_1 \rangle$ s.t. $j = (j_0 || j_1)$.
7. if $t_j \land 1 \{x_0 = 0\}$ then
8. $P_0$ sets $s_j = \langle -\langle \text{corr} \rangle^n_0 \rangle + 1$ and $r_j = \langle -\langle \text{corr} \rangle^\Lambda_0 \rangle + 1$.
9. else if $t_j \land 1 \{x_1 = 1\}$ then
10. $P_0$ sets $s_j = \langle -\langle \text{corr} \rangle^n_0 \rangle + 1$ and $r_j = \langle -\langle \text{corr} \rangle^\Lambda_0 \rangle + 1$.
11. else
12. $P_0$ sets $s_j = \langle -\langle \text{corr} \rangle^n_0 \rangle + 1$ and $r_j = \langle -\langle \text{corr} \rangle^\Lambda_0 \rangle + 1$.
13. end if
14. end for
15. $P_0 \& P_1$ invoke an instance of $(\frac{1}{2})\cdot\text{OT}_{\text{ring}}$, where $P_0$ is the sender with inputs $(s_j || j_1)$ and $P_1$ is the receiver with input $(m)^B_0 || x_1$.
16. $P_1$ sets its output as $\langle \text{corr} \rangle^n_1 || \langle \text{corr} \rangle^\Lambda_1$.
17. For $b \in \{0, 1\}$, $P_b$ invokes $T^{\text{int},\delta}_{\text{ReLU}}$ with input $\langle a \rangle_b$ to learn output $\langle y \rangle^B_b$. Party $P_b$ sets $\langle \text{corr} \rangle^\Lambda_b \leftarrow \langle \text{corr} \rangle^\Lambda_0 \cdot b$.
18. For $b \in \{0, 1\}$, $P_b$ invokes an instance of $T^{\text{ring},n}_{\text{B2A}}$ with input $\langle C_b \rangle^B_b$ and learns $\langle C \rangle^B_b$.
19. end for
20. For $b \in \{0, 1\}$, $P_b$ sets $\langle C \rangle^B_b = \langle C_b \rangle^B_b$ and $\langle \text{corr} \rangle^\Lambda_b \cdot n^1 + b - \langle C \rangle^B_b - B_b$, for $b \in \{0, 1\}$.

of communication. Thus, the overall communication of $\Pi^{\text{ring},n,d}_{\text{DIV}}$ is $\frac{3}{2} \lambda \eta$ + $3 \delta$ + $3 \delta$ + $3 \delta$ + $\frac{7}{2}$ + $\frac{7}{2}$, which can be rewritten as $< \frac{3}{2} \lambda + 3 \delta + (\eta + 2 \delta)$. Concretely, we get the best communication for $\Pi^{\text{ring},n,49}_{\text{DIV}}$ ($\eta = 32$) by setting $m = 7$ in all our millionaire invocations, which results in a total communication of 7796 bits.

Note that for the case of $\ell$-bit integers, our division protocol would require a call to $T^{\text{ring},n}_{\text{ReLU}}$ and $(\frac{1}{2})\cdot\text{OT}_{\text{ring}}$, and three calls each to $T^{\text{ring},n}_{\text{ReLU}}$ and $T^{\text{ring},n}_{\text{B2A}}$. The cost of $T^{\text{ring},n}_{\text{ReLU}}$ and $3 \times T^{\text{ring},n}_{\text{B2A}}$ are as mentioned in the previous paragraph, and the cost of $(\frac{1}{2})\cdot\text{OT}_{\text{ring}}$ and $T^{\text{ring},n}_{\text{B2A}}$ are $2 + 4 \cdot (\ell + \delta)$ and $3 \cdot \lambda + 3 \ell$ bits respectively. Thus, the overall communication is $\ell \cdot 3 \delta + 20 \frac{1}{2} \delta + 44 \frac{1}{2} \delta - 7 \lambda - 142$ bits, which can be rewritten as $< \lambda + 21 \cdot (\ell + \delta)$. By setting $m = 8$ in all our millionaire invocations, we get the best communication of 5570 bits for $\Pi^{\text{int},32,49}_{\text{DIV}}$.

E INPUT ENCODING

Neural network inference performs computations on floating-point numbers, whereas the secret-sharing techniques only work for integers in a ring $\mathbb{Z}_n$, for any $n \in \mathbb{N}$.

To represent a floating-point number $x \in \mathbb{Q}$ in the ring $\mathbb{Z}_n$, we encode it as a fixed-point integer $a = [x \cdot 2^q] \mod n$ with scale $s$. Fixed-point arithmetic is performed on the encoded input values (in the secure domain) and the same scale $s$ is maintained for all the intermediate results. The ring size $n$ and the scale $s$ are chosen such that the absolute value of any intermediate result does not exceed the bound $[n/2]$ and there is no loss in accuracy (refer Appendix I).

F IMPROVEMENT TO GAZELLE’S ALGORITHM

Gazelle [43] proposed two methods for computing convolutions, namely, the input rotations and the output rotations method. The only difference between the two methods is the number of (homomorphic) rotations required. In this section, we describe an optimization to reduce the number of rotations required by the output rotations method.

Let $c_i$ and $c_n$ denote the number of input and output channels respectively, and $c_n$ denote the number of channels that can fit in a single ciphertext. At a high level, the output rotations method works as follows: after performing all the convolutions homomorphically, we have $c_i \cdot c_n$ intermediate ciphertexts that are to be accumulated to form tightly packed output ciphertexts. Since most of these ciphertexts are misaligned after the convolution, they must be rotated in order to align and pack them. The intermediate ciphertexts can be grouped into $c_i/c_n$ groups of $c_i$ ciphertexts each, such that the ciphertexts within each group are added (after alignment) to form a single ciphertext. In [43], the ciphertexts within each group are rotated (aligned) individually, resulting in $\approx c_i \cdot c_n$ rotations. We observe that these groups can be further divided into $c_n$ subgroups of $c_i/c_n$ ciphertexts each, such that ciphertexts within a subgroup are misaligned by the same offset. Doing this has the advantage that the $c_i/c_n$ ciphertexts within each subgroup can first be added and then the resulting ciphertext can be aligned using a single rotation. This brings down the number of rotations by a factor of $c_i/c_n$ to $\approx c_n \cdot c_i/c_n$.

With our optimization, the output rotations method is better than the input rotations method when $f^2 \cdot c_i > c_n$, where $f^2$ is the filter size, which is usually the case.

G COMPLEXITY OF OUR BENCHMARKS

The complexity of the benchmarks we use in Section 7 is summarized as follows:

- SqueezeNet: There are 26 convolution layers of maximum filter size $3 \times 3$ and up to 1000 output channels. The activations after linear layers are ReLUs with size of up to 200,704

\[\frac{3}{2} \lambda \eta + 3 \delta + 44 \frac{1}{2} \delta - 4 \frac{1}{2} \lambda - 119 \frac{1}{2} \text{ bits} \]

\[\text{(1)} \cdot \text{(filter size)} \cdot \text{(number of output channels)} \]

\[\text{(number of convolution layers)} \]
Table 10: Summary of the accuracy achieved by fixed-point code vs input TensorFlow (TF) code.

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Bitwidth</th>
<th>Scale</th>
<th>TF Top 1</th>
<th>Fixed Top 1</th>
<th>TF Top 5</th>
<th>Fixed Top 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>SqueezeNet</td>
<td>32</td>
<td>9</td>
<td>55.86</td>
<td>55.90</td>
<td>79.18</td>
<td>79.22</td>
</tr>
<tr>
<td>ResNet50</td>
<td>37</td>
<td>12</td>
<td>76.47</td>
<td>76.45</td>
<td>93.21</td>
<td>93.23</td>
</tr>
<tr>
<td>DenseNet121</td>
<td>32</td>
<td>11</td>
<td>74.25</td>
<td>74.35</td>
<td>91.88</td>
<td>91.90</td>
</tr>
</tbody>
</table>

Table 9: Performance comparison of Garbled Circuits with our protocols for computing Avgpool layers. Runtimes are in seconds and communication numbers are in MiB.

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Garbled Circuits</th>
<th></th>
<th>Our Protocol</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>LAN</td>
<td>WAN</td>
<td>Comm</td>
<td>LAN</td>
</tr>
<tr>
<td>SqueezeNet</td>
<td>0.2</td>
<td>2.0</td>
<td>36.02</td>
<td>0.1</td>
</tr>
<tr>
<td>ResNet50</td>
<td>0.4</td>
<td>3.9</td>
<td>96.97</td>
<td>0.1</td>
</tr>
<tr>
<td>DenseNet121</td>
<td>17.2</td>
<td>179.4</td>
<td>6017.94</td>
<td>0.5</td>
</tr>
</tbody>
</table>

(a) over $\mathbb{Z}_2^\ell$

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Garbled Circuits</th>
<th></th>
<th>Our Protocol</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>LAN</td>
<td>WAN</td>
<td>Comm</td>
<td>LAN</td>
</tr>
<tr>
<td>SqueezeNet</td>
<td>0.2</td>
<td>2.2</td>
<td>39.93</td>
<td>0.1</td>
</tr>
<tr>
<td>ResNet50</td>
<td>0.4</td>
<td>4.2</td>
<td>106.22</td>
<td>0.1</td>
</tr>
<tr>
<td>DenseNet121</td>
<td>19.2</td>
<td>198.2</td>
<td>6707.94</td>
<td>0.6</td>
</tr>
</tbody>
</table>

(b) over $\mathbb{Z}_n$

H GARBLED CIRCUITS VS OUR PROTOCOLS FOR Avgpool

In this section, we compare our protocols with garbled circuits for evaluating the Avgpool layers of our benchmarks, and the corresponding performance numbers are given in Table 9. On DenseNet121, where a total of 176,640 divisions are performed, we have improvements over GC of more than $32 \times$ and $45 \times$ in the LAN and the WAN setting, respectively, for both our protocols. However, on SqueezeNet and ResNet50, the improvements are smaller ($2 \times$ to $7 \times$) because these DNNs only require 1000 and 2048 divisions, respectively, which are not enough for the costs in our protocols to amortize well. On the other hand, the communication difference between our protocols and GC is huge for all three DNNs. Specifically, we have an improvement of more than $19 \times$, $27 \times$, and $31 \times$ on SqueezeNet, ResNet50, and DenseNet121 respectively, for both our protocols.

I FIXED-POINT ACCURACY OF OUR BENCHMARKS

In this section, we show that the accuracy achieved by the fixed-point code matches the accuracy of the input TensorFlow code. Table 10 summarizes the bitwidths, the scales, and the corresponding TensorFlow (TF) and fixed-point accuracy for each of our benchmarks. Since our truncation and division protocols lead to faithful implementation of fixed-point arithmetic, accuracy of secure inference is the same as the fixed-point accuracy.