Private Multi-Party Neural Network Training over $\mathbb{Z}_{2^k}$ via Galois Rings

Hengcheng Zhou(✉)

Shanghai Jiao Tong University, Shanghai, China
zhc12345@sjtu.edu.cn

Abstract. Secret-sharing-based multi-party computation provides effective solutions for privacy-preserving machine learning. In this paper, we present novel protocols for privacy-preserving neural network training using Shamir secret sharing scheme over Galois rings. The specific Galois ring we use is $GR(2^k, d)$, which contains $\mathbb{Z}_{2^k}$ as a subring. The algebraic structure of $GR(2^k, d)$ enables us to benefit from Shamir scheme while performing modulo operations only on 2^k instead of a prime number, making our protocols more compatible with modern computer architectures. We achieve the parallel processing of training data by embedding different training samples into the different coefficients of the polynomial representing a single Galois ring element, and we show that this embedding can be performed with no additional communication overhead compared to processing only one sample at a time. To evaluate our methods, we conduct private training of neural networks on the MNIST dataset between different numbers of participants. The experimental results indicate the advantages of our protocols compared to existing $\mathbb{F}_p$-based implementations in this domain.

Keywords: Secure multi-party computation · Shamir secret sharing scheme · Galois ring · Neural network training

1 Introduction

Privacy-Preserving Machine Learning (PPML) is a growing field that focuses on protecting sensitive data during the training and inference of machine learning models. Achieving privacy-preserving neural network training is a highly valuable yet challenging problem in PPML. One promising solution is to use secret-sharing-based *multi-party computation* (MPC) [19], which enables mutually untrusting parties to collaboratively compute a function while keeping each party's input confidential. A series of studies [13,12,16,17,10] achieved private training of neural networks for specific numbers of participants, typically two to four parties. Baccarini *et al.* [2] proposed a training protocol for the n-party setting. However, due to the underlying replicated secret sharing scheme [9], their approach suffers from poor scalability as the communication required grows exponentially with the number of participants. Based on Shamir secret

sharing scheme [15], the work in [20] achieved the private training of neural networks between general n parties with $\mathcal{O}(n)$ communication complexity. They enhanced the training efficiency by adopting the SVM loss function, which is more MPC-friendly compared with the cross-entropy loss because it doesn't require division. However, their protocol's performance was significantly impacted due to the modulo p operation introduced by the $\mathbb{F}_p$-based Shamir scheme. The work in [21] employed the packed Shamir secret sharing scheme [8] to reduce the communication overhead of the protocols in [20], but their solution remained limited by the inefficiency of the modulo operation.

In this paper, we utilize Shamir scheme over Galois rings $GR(p^k, d)$ [1] to perform the training of neural networks among general n parties in the information-theoretic setting. When setting $p = 2$, we only need the modulo operation of 2^k rather than a prime number, leading to a more efficient implementation. A Galois ring element can be represented as a polynomial with coefficients in $\mathbb{Z}_{2^k}$. When using multiple coefficients to store data, parallel processing can be achieved. However, this will introduce cross terms after multiplication. We demonstrate that by choosing specific coefficients for embedding, we can avoid the impact of these cross terms. Additionally, we show that this embedding introduces no additional communication overhead compared to storing only one single secret in a ring element. It should be emphasized that our method for handling multiple secrets simultaneously differs from the method based on the packed Shamir secret sharing scheme, as our approach preserves the adversary threshold, whereas the packed scheme does not.

Our main contributions are summarized below:

- We propose privacy-preserving protocols for neural network training among general n parties based on Shamir secret sharing scheme over Galois rings, which can resist t semi-honest adversaries in the honest majority setting.
- We achieve the embedding of multiple data into a single Galois ring element, realizing the parallel computation of multiple secrets. We show that this embedding brings no additional communication overhead.
- We conduct experiments with different numbers of participants, where we train neural networks on the MNIST dataset. The results indicate the advantages of our protocols compared to field-based implementations.

2 Preliminaries

2.1 Shamir Scheme over Commutative Rings

This section describes how to establish Shamir secret sharing scheme over a commutative ring R. We first introduce the following definition from [1].

Definition 1. *Let $\alpha_1, \ldots, \alpha_n \in R$. We say that these points form an exceptional sequence if for each pair of integers $1 \leqslant i, j \leqslant n$ with $i \neq j$ it holds that $\alpha_i - \alpha_j \in R^*$, where R^* is the subset of R consisting of all elements that are invertible under multiplication. We define the Lenstra constant of R to be the maximum length of an exceptional sequence in R.*

Let $\beta_1, \ldots, \beta_n$ be n distinct nonzero elements in an exceptional sequence of R. To share a secret $u \in R$ among n participants $P_1, \ldots, P_n$, the dealer first generates a polynomial $f(x)$ of degree t over $R[x]$ such that $f(0) = u$ and all other coefficients are uniformly random, and then send the share $u^i = f(\beta_i)$ to party P_i, $1 \leqslant i \leqslant n$. The vector $(u^1, \ldots, u^n)$ is called a t-sharing of u, which is denoted by $\langle u \rangle_t$. When it is clear from the context, we just write $\langle u \rangle$. We call $\beta_1, \ldots, \beta_n$ the evaluation points. The reconstruction of a t-sharing $\langle u \rangle_t$ requires at least $t+1$ shares and can be computed through the Lagrange interpolation method as: $u = \sum_{i=1}^{n} \prod_{j=1, j\neq i}^{n} \frac{\beta_j}{\beta_j - \beta_i} u_i$. The divisions involved are well-defined, since $\beta_1, \ldots, \beta_n$ form an exceptional sequence such that the inverse of $\beta_j - \beta_i$ exists. We use Π_{Reveal} to denote the protocol for secret reconstruction. Since we consider semi-honest adversaries, we can let a designated party perform the calculation of secret reconstruction. It is worth noting that the number of parties n that this scheme can support is limited by the Lenstra constant of R, since the evaluation points are associated with each participant.

2.2 Galois Ring

A Galois ring is a ring of the form $\left(\mathbb{Z}/p^k\mathbb{Z}\right)[x]/(h(x))$, where p is a prime, k is a positive integer, and $h(x) \in \left(\mathbb{Z}/p^k\mathbb{Z}\right)[x]$ is a non-constant, monic polynomial such that its reduction modulo p is an irreducible polynomial in $\mathbb{F}_p[x]$. We denote by $GR(p^k, d)$ the Galois ring over $\mathbb{Z}_{p^k}$ of degree d. The Lenstra constant of $GR(p^k, d)$ is p^d. Let $R = GR(p^k, d)$, we have $R/(p) \cong \mathbb{F}_{p^d}$ and there exists a non-zero element $\xi \in R^*$ of multiplicative order $p^d - 1$. Given the set $T = \left\{0, 1, \xi, \ldots, \xi^{p^d-2}\right\}$, then any $g \in R$ can be uniquely written as $g = g_0 + g_1 p + g_2 p^2 + \cdots + g_{k-1} p^{k-1}$, where $g_0, \ldots, g_{k-1} \in T$. Moreover, g is a unit if and only if $g_0 \neq 0$. We can see that T forms an exceptional sequence. So we can get a Shamir scheme over R between n parties when $n < p^d$ by choosing evaluation points from $T \setminus \{0\}$. See [18] for more details about Galois rings. In this paper, we call the sharing generated by Shamir scheme over R a Galois sharing.

3 Protocol Constructions

3.1 Notation

Let n participants $P_1, \ldots, P_n$ ($n \geqslant 3$) participating in the calculation, of which at most t participants can be passively corrupted. In the rest of this paper, we use R to denote the Galois ring $GR(2^k, d) = \left(\mathbb{Z}/2^k\mathbb{Z}\right)[x]/(h(x))$. Consider an element $a \in R$, then a can be expressed as a polynomial: $a = a_0 + a_1 x + a_2 x^2 + \cdots + a_{d-1} x^{d-1} + (h(x))$, where $a_0, \ldots, a_{d-1} \in \mathbb{Z}_{2^k}$. Addition and multiplication over R are the addition and multiplication of the corresponding polynomials. We write a as $[a_0, a_1, \ldots, a_{d-1}]$ and abbreviate it as $[a_0, a_1, \ldots, a_j]$ if $a_i = 0$ for $j < i < d$. Particularly, we can write $[a_0]$ just as a_0. We use $[\![a_0, a_1, \ldots, a_{d-1}]\!]$ to denote the Galois sharing $\langle a \rangle = \langle [a_0, a_1, \ldots, a_{d-1}] \rangle$, and we denote the share corresponding to P_i as $a^i = [a_0^i, a_1^i, \ldots, a_{d-1}^i]$. For a set $\mathcal{S}$, $u \xleftarrow{\$} \mathcal{S}$ denotes sampling a uniformly random element u from $\mathcal{S}$.

3.2 Data Embedding

Data Representation. Before embedding real-life data into Galois rings we need to use fixed-point representation [4] to represent them as integers. A real number $\tilde{x} \in \mathbb{R}$ is transformed to a fixed-point approximation $\bar{x}$ by setting $\bar{x} = \lfloor \tilde{x} \cdot 2^f \rfloor \in \mathbb{Z}$, where f is the precision and $\lfloor \cdot \rfloor$ denotes the floor function. Then, we encode $\bar{x}$ into $\mathbb{Z}_{2^k}$ by computing $x = \bar{x} (\bmod 2^k)$. In this paper, we limit $\bar{x}$ in $[-2^{\ell-1}, 2^{\ell-1}-1]$ and require $k \geqslant \ell + \kappa + 1$, where κ is the statistical security parameter.

Embedding Secrets in R. For $R = GR(2^k, d)$, the polynomial representing an element has d coefficients. As described in [14], we can embed multiple secrets into different coefficients to achieve parallel computation. Specifically, to embed s secrets, we put the i-th secret into the coefficient corresponding to $x^{2^{i-1}-1}$, $1 \leqslant i \leqslant s$, and set other coefficients to 0. When we embed $a_1, \ldots, a_s$ and $b_1, \ldots, b_s$ into two ring elements, after multiplication, the term $a_i b_i$ will be part of the coefficient associated with $x^{2(2^{i-1}-1)}$. From Lemma 1 below, we can see that this coefficient only contains $a_i b_i$. To prevent overflow, we need to ensure $d \geqslant 2^s - 1$. In Section 3.3, we will show how to rearrange the coefficients to put $a_i b_i$ back to the coefficient of $x^{2^{i-1}-1}$. In the rest of the paper, we use s to denote the number of secrets embedded in a ring element. For $x \in \mathbb{Z}$, we use $I(x)$ to represent the value $2^{x-1} - 1$. And we denote by I the set $\{I(i) | 1 \leqslant i \leqslant s\}$.

Lemma 1. *For $m, n, r \in \mathbb{N}$ satisfying $m \neq n$, we have $2^m - 1 + 2^n - 1 \neq 2(2^r - 1)$.*

Proof. Suppose that $2^m - 1 + 2^n - 1 = 2(2^r - 1)$. It is easy to see that r cannot be less than or greater than m, n at the same time. Without loss of generality, we assume $n \leqslant r \leqslant m$. Then $2^m - 1 + 2^n - 1 = 2(2^r - 1) \Leftrightarrow 2^m + 2^n - 2 = 2^{r+1} - 2 \Leftrightarrow 2^m + 2^n = 2^{r+1} \Leftrightarrow 2^{m-n} + 1 = 2^{r+1-n}$. Since $2^{m-n} + 1$ is odd and 2^{r+1-n} is even, this equation cannot hold. We can get that $2^m - 1 + 2^n - 1 \neq 2(2^r - 1)$. □

3.3 Basic Operations

Randomness. Throughout this work, we will need the following functionalities to generate the random sharings necessary for our protocols. We assume that the parameters k, d, and s are set properly.

- $\mathcal{F}_{\text{Ran}}$. This functionality generates a sharing $[\![r_0, r_1, \ldots, r_{d-1}]\!]$ over R, where $r_i \xleftarrow{\$} \mathbb{Z}_{2^k}$ if $i \in I$, otherwise $r_i = 0$.
- $\mathcal{F}_{\text{DouRan}}$. This functionality generates a pair of sharings $\{[\![g_0, g_1, \ldots, g_{d-1}]\!]_{2t}, [\![h_0, h_1, \ldots, h_{d-1}]\!]_t\}$ over R, where $g_i \xleftarrow{\$} \mathbb{Z}_{2^k}$ for $0 \leqslant i \leqslant d-1$, $h_i = g_{2I(j)}$ if $i = I(j)$ for $1 \leqslant j \leqslant s$, otherwise $h_i = 0$.
- $\mathcal{F}_{\text{RanPair}}$. This functionality generates $[\![g_0, g_1, \ldots, g_{d-1}]\!]$ over $GR(2^{k+1}, d)$ along with $[\![h_0, h_1, \ldots, h_{d-1}]\!]$ over $GR(2^k, d)$, where $g_i \xleftarrow{\$} \mathbb{Z}_{2^{k+1}}$ for $0 \leqslant i \leqslant d-1$, $h_i = g_i \bmod 2^k$ if $i \in I$, otherwise $h_i = 0$.
- $\mathcal{F}_{\text{RanBit}}$. This functionality generates a sharing $[\![r_0, r_1, \ldots, r_{d-1}]\!]$ over R, where $r_i \xleftarrow{\$} \{0, 1\}$ if $i \in I$, otherwise $r_i = 0$.

The functionalities $\mathcal{F}_{\text{Ran}}$ and $\mathcal{F}_{\text{DouRan}}$ can be implemented using the randomness extraction technique based on a hyper-invertible matrix (HIM) [3] over R. Since the structure of the sharings we need to generate can only be preserved under $\mathbb{Z}_{2^k}$-linear instead of R-linear transformations, we need to interpret each ring element in the HIM as a matrix over $\mathbb{Z}_{2^k}$, as described in [1]. When considering semi-honest adversaries, it suffices to use a super-invertible matrix (SIM) [6] rather than an HIM. This matrix can be constructed from a Vandermonde matrix with points selected from an exceptional sequence of R, which is feasible since we require $n < 2^d$. For the functionality $\mathcal{F}_{\text{RanPair}}$, we can implement it in a similar way to RAN_p in [5]. We first let each participant generate a pair of required sharings, and then add together the sharings generated by all participants to get the final result.

To realize $\mathcal{F}_{\text{RanBit}}$, we propose Π_{RanBit} based on Π_{RandBit} in [7]. The main difference between Π_{RanBit} and Π_{RandBit} is that our protocol Π_{RanBit} considers the computation over Galois sharings while Π_{RandBit} handles additive sharings. For ease of demonstration, we give the details of Π_{RanBit} when $s = 1$ in Algorithm 1. The analysis of the security and correctness of Π_{RanBit} are similar to that of Π_{RandBit}. To see correctness, we have $(c^{-1}a)^2 = c^{-2}a^2 = e^{-1}e = 1 \bmod 2^{k+1}$, therefore $c^{-1}a = \pm 1 \bmod 2^k$ (Lemma IV.1 in [7]) and $(c^{-1}a + 1)/2 \bmod 2^k$ is a random bit. The randomness comes from the fact that a is a random square root of e. All calculations before Step 8 are performed on sharings over $GR(2^{k+1}, d)$. In Step 8 to Step 11, we use a pair of random sharings to convert the result into a sharing over $GR(2^k, d)$. Since $r_0, r_1, \ldots, r_{d-1}$ are uniformly random in $\mathbb{Z}_{2^{k+1}}$, the revealed value in Step 9 leaks no information. In Step 2, we compute $[\![a]\!] = 2[\![u]\!] + 1$. For the party P_i, his share corresponding to $[\![a]\!]$ is $[a_0^i, a_1^i, \ldots, a_{d-1}^i] = [2u_0^i + 1, 2u_1^i, \ldots, 2u_{d-1}^i]$. Therefore, the share corresponding to $[\![h]\!]$ is $h^i = [2c^{-1}u_0^i + c^{-1} + 1, 2c^{-1}u_1^i, \ldots, 2c^{-1}u_{d-1}^i]$. Since c^{-1} is odd, the division $[\![h]\!]/2$ in Step 7 is well-defined and we let P_i locally compute his share corresponding to $[\![h]\!]/2$ as $[h_0^i/2, h_1^i/2, \ldots, h_{d-1}^i/2]$. Algorithm 1 can be easily extended to handle general s values.

Algorithm 1 Π_{RanBit}

Output: A random shared bit $[\![r]\!]$ over $GR(2^k, d)$, where $r \xleftarrow{\$} \{0, 1\}$;

1: The parties invoke $\mathcal{F}_{\text{Ran}}$ to get a shared random element $[\![u]\!]$ over $GR(2^{k+1}, d)$;
2: $[\![a]\!] = 2[\![u]\!] + [1]$.
3: The parties locally compute $[\![a^2]\!]_{2t} = [\![a]\!][\![a]\!]$.
4: $[e] = \Pi_{\text{Reveal}}([\![a^2]\!]_{2t})$;
5: Let c be the smallest square root of e modulo 2^{k+1}, and c^{-1} be its inverse.
6: $[\![h]\!] = [c^{-1}][\![a]\!] + [1]$;
7: $[\![b]\!] = [\![h]\!]/2$;
8: The parties invoke $\mathcal{F}_{\text{RanPair}}$ to get $[\![r_0, r_1, \ldots, r_{d-1}]\!]$ over $GR(2^{k+1}, d)$ and $[\![v]\!]$ over $GR(2^k, d)$ satisfying $v = r_0 \bmod 2^k$;
9: $[b + r_0, r_1, \ldots, r_{d-1}] = \Pi_{\text{Reveal}}([\![b]\!] + [\![r_0, r_1, \ldots, r_{d-1}]\!])$; $//b + r_0 \in \mathbb{Z}_{2^{k+1}}$
10: The parties set $[g] \in GR(2^k, d)$, where $g = (b + r_0) \bmod 2^k$;
11: $[\![r]\!] = [g] - [\![v]\!]$;

$\mathbb{Z}_{2^k}$-Linear Operation. Given two sharings $\langle a \rangle$, $\langle b \rangle$ where $a = [a_0, a_1, \ldots, a_{d-1}]$, $b = [b_0, b_1, \ldots, b_{d-1}]$, and two public values $g, h \in \mathbb{Z}_{2^k}$, we can compute locally the sharing $\langle ga + hb \rangle$ by having P_i compute its own share $[ga_0^i + hb_0^i, ga_1^i + hb_1^i, \ldots, ga_{d-1}^i + hb_{d-1}^i]$ locally.

Multiplication. When we embed multiple secrets, the multiplication we need is coefficient-wise multiplication. We propose Π_{Mult} for this kind of multiplication based on the multiplication protocol in [6]. For ease of demonstration, we give the case of $s = 2$ in Algorithm 2. It can be directly extended to deal with the case of embedding a general number of secrets.

Algorithm 2 Multiplication Π_{Mult}

Input: $[\![a_0, a_1]\!]_t$, $[\![b_0, b_1]\!]_t$;
Output: $[\![g_0, g_1]\!]_t$ where $g_0 = a_0 b_0$, $g_1 = a_1 b_1$;
1: The parties invoke $\mathcal{F}_{\text{DouRan}}$ to get $\{[\![r_0, \ldots, r_{d-1}]\!]_{2t}, [\![r_0, r_2]\!]_t\}$;
2: $[\![h_0, \ldots, h_{d-1}]\!]_{2t} = [\![a_0, a_1]\!]_t [\![b_0, b_1]\!]_t$; //$h_0 = a_0 b_0, h_1 = a_1 b_0 + a_0 b_1, h_2 = a_1 b_1$
3: $[\![c_0, \ldots, c_{d-1}]\!]_{2t} = [\![h_0, \ldots, h_{d-1}]\!]_{2t} - [\![r_0, \ldots, r_{d-1}]\!]_{2t}$;
4: The parties invoke Π_{Reveal} to get $[c_0, \ldots, c_{d-1}]$;
5: $[\![g_0, g_1]\!]_t = [c_0, c_2] + [\![r_0, r_2]\!]_t$;

The correctness and security of Π_{Mult} are easy to check. Note that even if one of the inputs is public, such as the multiplication between $[a_0, a_1]$ and $[\![b_0, b_1]\!]$, we still need to invoke Π_{Mult}. However, when $[\![b_0, b_1]\!]$ is a random sharing, in the offline phase we can generate $[\![b_0]\!]$, $[\![b_1]\!]$ along with $[\![b_0, b_1]\!]$ and locally compute the result $[\![a_0 b_0, a_1 b_1]\!] = a_0 [\![b_0]\!] + [0, a_1][\![b_1]\!]$.

Truncation. Truncation serves as a crucial component for achieving private training of neural networks. Algorithm 3 shows the details of our protocol Π_{Trun} for this functionality when $s = 1$. Π_{Trun} is proposed based on the field-based solution Trunc in [4]. Given $[\![a]\!]$ and m where $a \in \mathbb{Z}_{2^k}$ and $1 \leqslant m \leqslant \ell - 1$, Π_{Trun} returns $[\![d]\!]$ satisfying $d = \lfloor a/2^m \rfloor$.

Algorithm 3 Truncation Π_{Trun}

Input: $[\![a]\!]$, m;
Output: $[\![d]\!]$;
1: The parties invoke $\mathcal{F}_{\text{RanBit}}$ to get $[\![r_{\ell+\kappa-1}]\!], \ldots, [\![r_0]\!]$;
2: The parties set $[\![r]\!] = \sum_{i=0}^{\ell+\kappa-1} 2^i [\![r_i]\!]$, and $[\![r']\!] = \sum_{i=0}^{\ell+\kappa-m-1} 2^i [\![r_{i+m}]\!]$;
3: The parties compute $[c] = \Pi_{\text{Reveal}}([2^{\ell-1}] + [\![a]\!] + [\![r]\!])$ and set $[c'] = [c \bmod 2^m]$;
4: $[\![u]\!] = \mathcal{F}_{\text{BitLT}}([c'], \{[\![r_{m-1}]\!], \ldots, [\![r_0]\!]\})$;
5: $[\![d]\!] = ([c] - [c'] - [2^{\ell-1}])2^{-m} - [\![r']\!] - [\![u]\!]$;

The analysis of the security and correctness of Π_{Trun} is similar to that of Trunc. The functionality $\mathcal{F}_{\text{BitLT}}$ invoked in Step 4 compares a public value to

Galois sharings. Given a public value $[a] \in R$ and l sharings $\{[\![b_l]\!], \ldots, [\![b_1]\!]\}$ where $b_i \in \{0, 1\}$, $\mathcal{F}_{\text{BitLT}}$ returns $[\![1]\!]$ if $a < \sum_{i=1}^{l} 2^{i-1} b_i$ and $[\![0]\!]$ otherwise. By using $\mathcal{F}_{\text{RanBit}}$ to generate the required random sharings and following the computational steps of BitLTC1 in [4], we can get the protocol for realizing $\mathcal{F}_{\text{BitLT}}$. One thing to note is that the way BitLTC1 computes unbounded fan-in multiplication depends on field-specific properties. When implementing $\mathcal{F}_{\text{BitLT}}$, we can calculate this kind of multiplication by invoking Π_{Mult} sequentially.

Now we introduce how to extend $\mathcal{F}_{\text{BitLT}}$ to handle general s values. In Step 5 of BitLTC1 [4], the multiplication is performed between public values and secret sharings that cannot be generated offline. When implementing $\mathcal{F}_{\text{BitLT}}$, we can first calculate the multiplication locally, and then utilize the reconstruction operation in the subprotocol for computing the least significant bit (corresponding to Mod2 [4]) to rearrange the coefficients, in a manner similar to Π_{Mult}. On this basis, we can extend $\mathcal{F}_{\text{BitLT}}$ and Π_{Trun} to handle the case of any s without extra communication.

4 Private Neural Network Training

In this paper, we focus on the training of fully connected neural networks using the mini-batch gradient descent algorithm. The purpose of training is to update the target parameters w^l and b^l for the l-th layer. To realize the training, in addition to the protocols proposed above, we also need to implement ReLU and its derivative, as well as the derivative of the loss function with respect to the output of the last layer. Protocols realizing these functionalities can be trivially obtained by using Π_{Reveal}, Π_{Mult}, and Π_{Trun} as the underlying sub-protocols and following the steps of the relevant protocols in [20].

Parallel Training. Similar to the parallel computing method used in packed training [21], we can utilize Galois sharing to compute the gradients corresponding to each sample within a mini-batch in parallel. Once the gradients for each sample are calculated, we need to aggregate and average these individual gradients embedded into different coefficients, and use them as the estimated global gradients to update the target parameters. We propose Π_{Aggre} for this process based on Π_{Trun}. For ease of demonstration, we give the details of Π_{Aggre} when $s = 2$ in Algorithm 4. With Π_{Aggre}, we can implement the parallel training of neural networks in combination with existing protocols. As with packed training, our parallel training does not affect the training results. Note that we need to use different forms of sharing for w^l and b^l. Taking $s = 2$ as an example, we use $[\![w^l]\!]$ and $[\![b^l, b^l]\!]$, respectively. This is because w^l is used in multiplication.

Complexity Analysis. By embedding multiple secrets into a single sharing, we can achieve more efficient communication. When $s = 1$, a total of $2(n-1)kd$ bits of messages are required to reconstruct a secret. When $s > 1$, the amortized communication becomes $2(n-1)kd/s$ bits. Since $d \geqslant 2^s - 1$, increasing s will cause d to increase significantly. From a practical standpoint, setting s to 2 is a

Algorithm 4 Π_{Aggre}

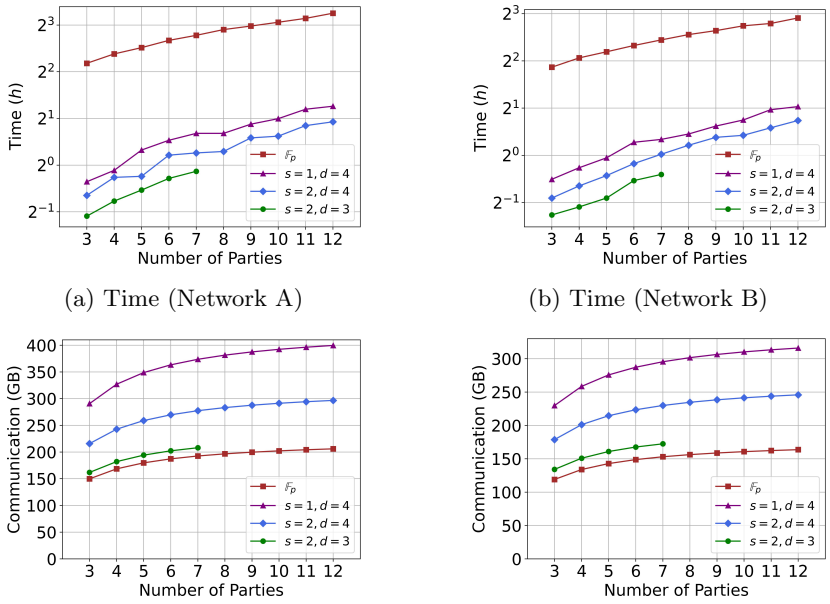
Input: $[\![a_0, a_1]\!]$, m;
Output: $[\![d]\!]$ where $d = \lfloor (a_0 + a_1)/2^m \rfloor$;
1: The parties invoke $\mathcal{F}_{\text{RanBit}}$ to get $[\![r_{\ell+\kappa-1}]\!], \ldots, [\![r_0]\!]$, and invoke $\mathcal{F}_{\text{Ran}}$ to get $[\![g]\!]$;
2: The parties set $[\![r]\!] = \sum_{i=0}^{\ell+\kappa-1} 2^i [\![r_i]\!]$, and $[\![r']\!] = \sum_{i=0}^{\ell+\kappa-m-1} 2^i [\![r_{i+m}]\!]$;
3: The parties compute $[\![v_0, v_1]\!] = [\![r]\!] - [\![g]\!] + [0, 1][\![g]\!]$; $//r = v_0 + v_1$
4: $[c_0, c_1] = \Pi_{\text{Reveal}}(2^{\ell-1} + [\![a_0, a_1]\!] + [\![v_0, v_1]\!])$;
5: $[c'] = [(c_0 + c_1) \bmod 2^m]$;
6: $[\![u]\!] = \mathcal{F}_{\text{BitLT}}([c'], \{[\![r_{m-1}]\!], \ldots, [\![r_0]\!]\})$;
7: $[\![d]\!] = ([c_0 + c_1] - [c'] - 2^{\ell-1})2^{-m} - [\![r']\!] - [\![u]\!]$;

suitable choice, as it requires only $d \geqslant 3$. It should be noted that parallel training does not reduce the communication overhead of the entire training process to $1/s$ of the original, as the communication required for training is not directly proportional to the size of each training batch.

5 Experimental Results

The experiments were carried out on a single server equipped with 2 24-core 2.20GHz Intel Xeon Gold 5220R CPUs and 128GB of RAM under the LAN setting. We accelerated the computation of local matrix multiplication using an NVIDIA RTX 3090 GPU. We set the number of adversaries t to 1 and generated all the random sharings offline. To evaluate our protocols, we experimentally compared them with the protocols in [20], which utilize Shamir scheme over $\mathbb{F}_p$ to achieve neural network training among general n parties. We set the prime p to be $2^{110} - 21$. Due to the security issue introduced in [11], we chose the protocols in [20] with probabilistic truncation replaced with deterministic truncation as the baseline protocols. We train the same two 3-layer DNNs as in [20], Network A and Network B, on the MNIST dataset. We followed the same hyper-parameter settings as [20], and set f to 16 and 13 for Network A and Network B, respectively.

Since our parallel training method does not affect the training results, we focus on the training efficiency. We first conducted experiments using Shamir scheme over $GR(2^{128}, 4)$. The results are shown in Fig. 1. We can see that utilizing Galois sharing with $s = 1$ can lead to an average of 77.1% improvement in training time as compared to the field-based solution. Further improvement can be achieved by setting $s = 2$, which shows an additional improvement of 21.1% on average. Fig. 1 also shows the result of experiments over $GR(2^{128}, 3)$ when $3 \leqslant n \leqslant 7$. In the case of $s = 2$, the training efficiency for $d = 3$ is, on average, 25.2% higher than that of $d = 4$. Taken together, we can get that setting d to 3 results in an average improvement of 87.8% as compared to the implementation on $\mathbb{F}_p$. This substantial gain in efficiency underscores the advantages of our protocols. It is worth noting that our protocols need more communication than $\mathbb{F}_p$-based solutions. The advantage of our protocols lies in computational efficiency, which is achieved by eliminating the need for modulo p operation.



(a) Time (Network A)

(b) Time (Network B)

(c) Comm. (Network A)

(d) Comm. (Network B)

Fig. 1. The time and communication required for an epoch of training

6 Conclusion

We propose protocols for private neural network training that leverage Shamir scheme over $GR(2^k, d)$. By embedding multiple secrets into a single ring element, we can utilize the available coefficients more effectively and reduce the overall communication. Based on Galois rings, our protocols eliminate the modulo p operation typically required in Shamir secret sharing scheme, thereby significantly improving the efficiency of the training process.

Acknowledgments. The work was supported in part by the National Key Research and Development (R&D) Program of China under Grant 2022YFA1004900 and in part by the National Natural Science Foundation of China under Grants 12361141818, 12426302, 12031011 and 12271084.

References

1. Abspoel, M., Cramer, R., Damgård, I., Escudero, D., Yuan, C.: Efficient information-theoretic secure multiparty computation over $\mathbb{Z}/p^k\mathbb{Z}$ via Galois rings. In: Proceedings of the Theory of Cryptography Conference. pp. 471–501 (2019)
2. Baccarini, A.N., Blanton, M., Yuan, C.: Multi-party replicated secret sharing over a ring with applications to privacy-preserving machine learning. PoPETs **2023**, 608–626 (2023)

3. Beerliová-Trubíniová, Z., Hirt, M.: Perfectly-secure MPC with linear communication complexity. In: Proceedings of the Theory of Cryptography Conference. pp. 213–230 (2008)
4. Catrina, O., Hoogh, S.: Improved primitives for secure multi-party integer computation. In: International Conference on Security and Cryptography for Networks. pp. 182–199 (2010)
5. Damgård, I., Fitzi, M., Kiltz, E., Nielsen, J., Toft, T.: Unconditionally secure constant-rounds multi-party computation for equality, comparison, bits and exponentiation. In: Proceedings of the 3th Theory of Cryptography Conference (TCC). pp. 285–304 (2006)
6. Damgård, I., Nielsen., J.B.: Scalable and unconditionally secure multiparty computation. In: Annual International Cryptology Conference. pp. 572–590 (2007)
7. Damgård, I., Escudero, D., Frederiksen, T., Keller, M., Scholl, P., Volgushev, N.: New primitives for actively-secure MPC over rings with applications to private machine learning. In: IEEE Symposium on Security and Privacy. pp. 1102–1120 (2019)
8. Franklin, M., Yung, M.: Communication complexity of secure computation (extended abstract). In: Proceedings of the Twenty-Fourth Annual ACM Symposium on Theory of Computing. pp. 699–710 (1992)
9. Ito, M., Saito, A., Nishizeki, T.: Secret sharing scheme realizing general access structure. Electronics and Communications in Japan Part III-Fundamental Electronic Science **72**, 56–64 (1989)
10. Koti, N., Patra, A., Rachuri, R., Suresh, A.: Tetrad: actively secure 4PC for secure training and inference. In: Symposium on Network and Distributed System Security (NDSS) (2022)
11. Li, Y., Duan, Y., Huang, Z., Hong, C., Zhang, C., Song, Y.: Efficient 3PC for binary circuits with application to maliciously-secure DNN inference. In: Proceedings of the 32nd USENIX Conference on Security Symposium (2023)
12. Mohassel, P., Rindal, P.: ABY^3: a mixed protocol framework for machine learning. In: Proceedings of the 2018 ACM SIGSAC conference on computer and communications security. pp. 35–52 (2018)
13. Mohassel, P., Zhang, Y.: SecureML: a system for scalable privacy-preserving machine learning. In: IEEE Symposium on Security and Privacy. pp. 19–38 (2017)
14. Orsini, E., Smart, N.P., Vercauteren, F.: Overdrive2k: Efficient secure MPC over $\mathbb{Z}_{2^k}$ from somewhat homomorphic encryption. In: CT-RSA. pp. 254–283 (2020)
15. Shamir, A.: How to share a secret. Communications of the ACM **22**(11), 612–613 (1979)
16. Wagh, S., Gupta, D., Chandran, N.: SecureNN: 3-party secure computation for neural network training. PoPETs **2019**(3), 26–49 (2019)
17. Wagh, S., Tople, S., Benhamouda, F., Kushilevitz, E., Mittal, P., Rabin, T.: FALCON: honest-majority maliciously secure framework for private deep learning. PoPETs **2021**(1), 188–208 (2021)
18. Wan, Z.: Lectures on Finite Fields and Galois Rings. World Scientific Publishing Company (2003)
19. Yao, A.: Protocols for secure computations. In: Proceedings of the 23rd Annual Symposium on Foundations of Computer Science. pp. 160–164 (1982)
20. Zhou, H.: Information-theoretically secure neural network training with flexible deployment. In: Proceedings of the 32nd International Conference on Artificial Neural Networks. pp. 324–336 (2023)
21. Zhou, H.: Private neural network training with packed secret sharing. In: Proceedings of the 30th International Computing and Combinatorics Conference (2024)