Breaking Bicoptor from S&P 2023 Based on Practical Secret Recovery Attack

Jun Xu[1,2], Zhiwei Li[1,2], and Lei Hu[1,2]

[1] Key Laboratory of Cyberspace Security Defense, Institute of Information Engineering, Chinese Academy of Sciences, Beijing, China
{xujun, lizhiwei, hulei}@iie.ac.cn
[2] School of Cyber Security, University of Chinese Academy of Sciences, Beijing, China

Abstract. At S&P 2023, a family of secure three-party computing protocols called Bicoptor was mainly proposed by Huawei Technology in China, which is used to compute non-linear functions in privacy preserving machine learning. In these protocols, two parties P_0, P_1 respectively hold the corresponding shares of the secret, while a third party P_2 acts as an assistant. The authors claimed that neither party in the Bicoptor can independently compromise the confidentiality of the input, intermediate, or output. In this paper, we point out that this claim is incorrect. The assistant P_2 can recover the secret in the DReLU protocol, which is the basis of Bicoptor. The restoration of its secret will result in the security of the remaining protocols in Bicoptor being compromised. Specifically, we provide two secret recovery attacks regarding the DReLU protocol. The first attack method belongs to a clever enumeration method, which is mainly due to the derivation of the modular equation about the secret and its share. The key of the second attack lies in solving the small integer root problem of a modular equation, as the lattices involved are only 3 or 4 dimensions, the LLL algorithm can effectively work. For the system settings selected by Bicoptor, our experiment shows that the desired secret in the DReLU protocol can be restored within one second on a personal computer. Therefore, when using cryptographic protocols in the field of privacy preserving machine learning, it is not only important to pay attention to design overhead, but also to be particularly careful of potential security threats.

Keywords: Secure multiparty computation, privacy-preserving machine learning, secret recovery attack, lattice, the LLL algorithm.

1 Introduction

1.1 Background

Secure Multiparty Computation (MPC) is an important cryptographic protocol that allows multiple parties to compute a function on their private inputs without disclosing any individual input to other parties. Based on this characteristic, MPC has a wide range of applications in many fields, such as in privacy-preserving machine learning (PPML). Recently, PPML based on MPC has received widespread attention from researchers because it combines the utility of machine learning (ML) with the privacy-preserving properties of MPC. However, MPC will cause extra overhead, which is a major constraint on the development of MPC-based PPML. Therefore, finding an MPC protocol with excellent performance is very important. There are already some works aimed at reducing this extra overhead. According to different settings, these works can be divided into three different types: two-party protocols, such as [4,15,13,12,14], three-party protocols, for example, [17,18,16,9,1,6], and four-party protocols, including [5,2].

In the field of MPC-based PPML, a large number of non-linear functions are involved. The basis of the non-linear functions is a sign determination function. Once the sign determination function is constructed, other non-linear functions can be easily implemented. The overhead of evaluating non-linear functions dominates the total overhead. Most existing protocols use preprocessing to improve the online performance. Specifically, after running the input-independent preprocessing phase that typically uses heavy cryptographic mechanisms, once the input is ready, the parties could complete PPML tasks relatively quickly in the online phase. It is worth noting that the total overhead (preprocessing and online) remains unchanged. Although the performance of the online phase is improved, the overhead of the preprocessing phase is usually heavy. For example, at CRYPTO 2020, Escudero et al. put forward a method that could improve online comparison performance through preprocessed materials called "Edabits" [3]. However, the generation of Edabits relies on homomorphic encryption or oblivious transfer, which incurs significant performance overhead.

At S&P 2023, Zhou et al. proposed a family of novel secure three-party computation protocols, called Bicoptor [20], to optimize the overall performance of different non-linear functions used in PPML. The basis of Bicoptor is the Derivative Rectified Linear Unit (DReLU) protocol, which is a sign determination protocol, to determine the sign of the input value, that is, to determine whether the input value is greater than or equal to 0 or less than 0. The DReLU protocol in Bicoptor only requires two commnication rounds, and does not need any preprocessing. Based on this DReLU protocol, Zhou et al. developed other protocols suitable for calculating non-linear functions in PPML. These protocols constitute the so-called Bicoptor. Compared to state-of-the-art works, Edabits at CRYPTO 2020 [3] or Falcon [18] at PETS 2021, Bicoptor performs better in the same settings and enviroment.

The protocols involved in Bicoptor are all scenarios of three-party computation. Briefly speaking, two parties P_0, P_1 hold 2-out-of-2 secret sharing shares, and the third party P_2 is an assistant. Three parties P_0, P_1, P_2 are all static and semi-honest. It is assumed that there are no collusion between any two of three parties. Zhou et al. claimed in [20, Section 2.1] that no party can individually break the input, intermediate or output secrecy. The overview of the DReLU protocol in Bicoptor is as follows. 1). The participants P_0, P_1 locally perform repeated truncations on shares and obtain an array of outcomes $[u_i]$. 2). P_0, P_1

perform some linear operations on array $[u_i]$ to obtain $[w_i]$. 3). P_0, P_1 send array $[w_i]$ to the participant P_2. 4). P_2 reconstructs w_i's and check the existence of the target value.

1.2 Our contributions

In this article, we propose two secret recovery attacks on the DReLU protocol in Bicoptor, with the first attack being an exponential time algorithm for ℓ_x and the second attack being a heuristic polynomial time algorithm for ℓ_x, where ℓ_x is the parameter related to the bit-length of the secret. Once the secret in the DReLU protocol is restored, the security of the remaining protocols in Bicoptor is compromised, as the construction of these protocols is based on DReLU. In these attacks, we assume that the participant P_2 is always a passive adversary. This assumption is reasonable because Bicoptor mentioned that P_0, P_1, and P_2 can be static and semi-honest. Our experiment supports the corresponding theoretical analysis. For the system setting $q = 2^{64}$ and ℓ_x=13 given by Bicoptor, the experimental results show that the secret can be recovered on a personal computer within one second if the second attack is carried out. In addition, we also test some other types of parameter values, and effectively obtain the desired secret.

The essence of the above two attacks lies in the fact that the adversary P_2 is able to derive modular equations related to the secret from the tuples $[w_i]$'s he possesses, and then recover the secret based on a clever enumeration in the first attack and using lattice methods in the second attack. The enumeration operation causes the complexity of the first attack to be exponential with respect to ℓ_x, while the lattice attack method based on LLL is the reason why the complexity of the second attack is polynomial with respect to ℓ_x.

In the first attack, P_2 could obtain a modular equation between the secret and its secret share. It is worth noting that the secret comes from a small interval, i.e. $[0, 2^{\ell_x}) \cup (q - 2^{\ell_x}, q)$, however, its share will fill the entire interval $[0, q)$, i.e. the ring $\mathbb{Z}_q$. Once the modular equation mentioned earlier is derived, P_2 can determine the corresponding share value that originally belongs to the entire interval by enumerating the secret from the small interval. This strategy can greatly reduce the number of candidates of the tuple related to the secret and its shares. Then we provide a filtering method for existing candidate tuples. The idea behind this filtering method is very simple. That is to take the candidate tuples as input for the truncation function in the DReLU protocol, and then observe whether the checking equations are satisfied. For the system setting $q = 2^{64}$ and ℓ_x=13 in Bicoptor, our experiment shows that after two rounds of detection, the desired secret in the DReLU protocol can be uniquely determined.

In the second attack, P_2 first constructs modular equations including partial information about two secret sharing shares of the secret. In order to get these partial information, the properties of the truncation function in the DReLU protocol are utilized. Furthermore, P_2 can also use the array $[w_i]$ to obtain a modular equation between these two secret shares. Once the obtained partial information is imported into this modular equation, a modular equation with a small integer root is generated. If such a small root is found out, then the corresponding secret shares are obtained, which means the secret in the DReLU protocol is restored. In order to identify such small roots, two lattice methods are presented, where the dimensions of the first and second lattices are 4 or 3, respectively. Under the corresponding success conditions, the LLL algorithm can heuristically find the desired result. The success condition required for lattice method II is better than that in lattice method I. It is worth noting that for the parameter values $q = 2^{64}$ and ℓ_x=13 in the DReLU algorithm, the success condition of lattice method I can also be satisfied. It implies that lattice method I works well for such parameter values. Our experimental results also validate this analysis.

1.3 Organization

The rest of this paper is organized as follows. We introduce the preliminaries in Section 2. Section 3 recalls the DReLU protocol in Bicoptor. We present a secret recovery attack against DReLU in Section 4. Section 5 provides an improved secret recovery attack. In Section 6, we show that the security of the remaining protocols in Bicoptor is also broken. Section 7 gives the experimental results. In Section 8, we conclude the paper and provide a future work that can be considered.

2 Preliminary

In subsequent attacks, the following expressions or relationships are often used. For non-negative integers A, B and a positive integer q, the congruence relation $A \equiv B \mod q$ represents that q divides integer $A - B$, and the relation $A = B \mod q$ means that A is the remainder of B divided by q, where $0 \leq A < q$. For two integer vectors α, β with the same dimension, the congruence relation $\alpha \equiv \beta \mod q$ represents that q divides vector $\alpha - \beta$. In other words, q divides every component of vector $\alpha - \beta$. Similarly, the relationship $\alpha = \beta \mod q$ means that the component of α is equal to the remainder after q divides the component at the corresponding position of β. It implies that each component of α is greater than or equal to 0 and less than q.

2.1 Lattice

A lattice $\mathcal{L}$ is a discrete subgroup of $\mathbb{R}^m$. Given n linearly independent (row) vectors $\mathbf{b}_1, \mathbf{b}_2, \cdots, \mathbf{b}_n \in \mathbb{R}^m$, the lattice spanned by these vectors is defined as

$$\mathcal{L}(\mathbf{b}_1, \mathbf{b}_2, \cdots, \mathbf{b}_n) = \left\{ \sum_{i=1}^{n} c_i \mathbf{b}_i | c_i \in \mathbb{Z} \right\}.$$

The vector set $\{\mathbf{b}_1, \mathbf{b}_2, \cdots, \mathbf{b}_n\}$ is called a basis of the lattice $\mathcal{L}$. That is, define B as the $n \times m$ basis matrix whose rows are the basis vectors $\mathbf{b}_1, \mathbf{b}_2, \cdots, \mathbf{b}_n$

which can be written as $B = [\mathbf{b}_1^T, \cdots, \mathbf{b}_n^T]^T$. The dimension and determinant of $\mathcal{L}$ when $n \leq m$ are respectively

$$\dim \mathcal{L} = n, \det \mathcal{L} = \sqrt{\det BB^T}.$$

For the case of $n = m$, the lattice is called full rank and $\det \mathcal{L} = |\det B|$. The celebrated LLL lattice reduction algorithm [7] can output a reduced vector whose Euclidean length satisfies the following condition (see e.g. [8] for the corresponding proof).

Lemma 1 (LLL). *Let $\mathcal{L}$ be an n-dimensional lattice. Within polynomial time, the LLL algorithm outputs the first reduced basis vector $\mathbf{v}_1$ that satisfies*

$$\|\mathbf{v}_1\| \leq 2^{\frac{n-1}{4}} (\det \mathcal{L})^{\frac{1}{n}}.$$

The lattice dimension involved in subsequent attacks is equal to 3 or 4, that is, n=3 or 4. In this case, the relationship $2^{\frac{n-1}{4}} < \sqrt{n}$ always holds. This means that the Euclidean length of v_1 is always smaller than Minkowski's bound, i.e. $\|\mathbf{v}_1\| < \sqrt{n}(\det \mathcal{L})^{\frac{1}{n}}$. Therefore, v_1 is a sufficiently short vector. In practice, the LLL algorithm tends to output the vector whose Euclidean length is much smaller than theoretically predicted. For very low lattice dimensions, such as 3 and 4 dimensions, the LLL algorithm is often able to find the shortest nonzero vector. The Gaussian heuristic gives an approximate Euclidean length of the shortest non-zero vector in $\mathcal{L}$.

Assumption 1 (Gaussian heuristic). *Let $\mathcal{L}$ be a random n-dimensional lattice of $\mathbb{Z}^m$. Then, with overwhelming probability, the Euclidean length of the shortest non-zero vectors in $\mathcal{L}$ is asymptotically close to:*

$$\mathrm{GH}(\mathcal{L}) = \sqrt{\frac{n}{2\pi e}} \det(\mathcal{L})^{\frac{1}{n}}.$$

2.2 Bicoptor and related scheme/functions

In this subsection, we recall the security model and system setting of Bicoptor, the involved secret sharing scheme and the truncation function with errors as well as non-linear functions in PPML. Please refer to the papers [10,20] for more details.

Security model. A three-party computation (3PC) setting is involved in Bicoptor. The two parties P_0, P_1 hold 2-out-of-2 secret shares, and the third party P_2 acts as an assistant. Three participants P_0, P_1, P_2 are static (that is, non-adaptive) and semi-honest (namely, honest-but-curious). It is assumed that there are no collusion between any two of three participants. The authors in Bicoptor claimed that no participant can individually break the input, intermediate or output secrecy.

System settings. In Bicoptor, all arithmetic operations are worked in an integer ring $\mathbb{Z}_q$, where the bit-length of modulus q is $\ell := \log_2 q$. Considering a secret input $x \in [0, 2^{\ell_x}) \cup (q - 2^{\ell_x}, q)$, where ℓ_x the precision bit length of x satisfying $\ell_x < \ell - 1$. If $x \in [0, 2^{\ell_x})$, then x is positive. If $x \in (q - 2^{\ell_x}, q)$, then x is negative. For the above x, the value ξ is defined as follows:

$$\xi := \begin{cases} x & \text{if } x \in [0, 2^{\ell_x}), \\ q - x & \text{if } x \in (q - 2^{\ell_x}, q). \end{cases} \tag{1}$$

It is easy to see that $\xi \in [0, 2^{\ell_x})$, that is, ξ is positive.

In [20, Section 5.2], Zhou et al. selected $q = 2^{64}$ and $\ell_x = 13$ as the system setting for Bicoptor.

Secret sharing scheme. In Bicoptor, an additive secret sharing scheme with an unbalanced setting is involved. A secret input $x \in \mathbb{Z}_q$ is shared between participants P_0 and P_1, which satisfies the relation

$$x = [x]_0 + [x]_1 \mod q. \tag{2}$$

Here, $[x]_0 = x + R \mod q$ and $[x]_1 = -R \mod q$, where $R \in \mathbb{Z}_q$ is a random number. The participants P_0 and P_1 hold the shares $[x]_0$ and $[x]_1$, respectively. Let the tuple $[x] := ([x]_0, [x]_1)$ represent a two-party secret sharing of x. The secret sharing with an unbalancing model means that the third party P_2 does not obtain any information about the secret x.

For a constant value c in $\mathbb{Z}_q$, the participants P_0 and P_1 hold the shares $[c]_0$ and $[c]_1$, respectively. Here, one of $[c]_0$ and $[c]_1$ is equal to the constant c, and the other is equal to 0. Without loss of generality, we can take $[c]_0 = c$ and $[c]_1 = 0$ in the subsequent analysis. For this case, we could write $[c] = ([c]_0, [c]_1) = (c, 0)$.

The secret sharing has the linear homomorphic property. To be specific, there is the following relations:

$$[x] + [c] \equiv [x + c] \mod q, \tag{3}$$

$$[x_1] + [x_2] \equiv [x_1 + x_2] \mod q, \tag{4}$$

$$c \cdot [x] \equiv [c \cdot x] \mod q, \tag{5}$$

where c is a constant value in $\mathbb{Z}_q$.

The truncation function with errors. The participants P_0 and P_1 have shares $[x]_0$ and $[x]_1$, respectively. Then, P_0 right shifts $[x]_0$ for k bits; and P_1 takes the input negation and then does another negation after k-bit shifting. The above two operations can be rewritten separately as follows.

$$[\mathrm{TRC}(x, k)]_0 := \mathrm{rShift}([x]_0, k), \tag{6}$$

$$[\mathrm{TRC}(x, k)]_1 := q - \mathrm{rShift}(q - [x]_1, k). \tag{7}$$

Here $\mathrm{rShift}(y, k)$ means to shift y in $\mathbb{Z}_q$ by k bits to the right, without padding zero in the left hand. From the perspective of division with residues, if we write $y = y'' \cdot 2^k + y'$, where $0 \le y < q$, $0 \le y' < 2^k$ and $0 \le y'' < \frac{q}{2^k}$, then $\mathrm{rShift}(y, k) = y''$. Equivalently, we have $\mathrm{rShift}(y, k) = \left\lfloor \frac{y}{2^k} \right\rfloor$.

Define $\mathrm{TRC}(x,k) \in \mathbb{Z}_q$ as the k-bit truncation function satisfying

$$\mathrm{TRC}(x,k) = [\mathrm{TRC}(x,k)]_0 + [\mathrm{TRC}(x,k)]_1 \mod q, \tag{8}$$

where P_0 and P_1 hold the shares $[\mathrm{TRC}(x,k)]_0$ and $[\mathrm{TRC}(x,k)]_1$, respectively.

However, the truncation function $\mathrm{TRC}(x,k)$ introduces one-bit error at the least significant bit. The corresponding result is summarized as follows.

Lemma 2 ([20]). *In an integer ring $\mathbb{Z}_q$, let $x \in [0, 2^{\ell_x}) \cup (q - 2^{\ell_x}, q)$, where $\ell_x + 1 < \ell = \log_2 q$. Define ξ as in (1). Then we have the following results with probability $1 - 2^{\ell_x+1-\ell}$:*

- *If $x \in [0, 2^{\ell_x})$, then $\mathrm{TRC}(x,k) = \mathrm{rShift}(\xi, k) + bit$, where $bit = 0$ or 1.*
- *If $x \in (q - 2^{\ell_x}, q)$, then $\mathrm{TRC}(x,k) = q - \mathrm{rShift}(\xi, k) - bit$, where $bit = 0$ or 1.*

Non-linear functions in PPML. A function satisfying $F(x) = a \cdot x + b$ is called a linear function, where a, b are constants. Otherwise, it is called a non-linear function. The common non-linear function used in machine learning is the Rectified Linear Unit (ReLU) function, which can be obtained from the DReLU function. The definitions of these two functions are as follows:

$$\mathrm{ReLU}(x) = \begin{cases} x & \text{if } x \geq 0, \\ 0 & \text{if } x < 0, \end{cases}$$

and

$$\mathrm{DReLU}(x) = \begin{cases} 1 & \text{if } x \geq 0, \\ 0 & \text{if } x < 0. \end{cases}$$

It is easy to see that $\mathrm{ReLU}(x) = \mathrm{DReLU}(x) \cdot x$. The definitions for other non-linear functions were given in [20, Appendix B] (also see Section 6).

3 The DReLU protocol in Bicoptor

In this section, we recall the DReLU protocol in Bicoptor, which is the basis of the Bicoptor family [20]. The specific steps of DReLU are given in Algorithm 1. We provide a detailed explanation for these steps. It is worth noting that subsequent attacks only rely on Steps 1 to 7. Furthermore, why can this protocol be used to determine the sign of input is independent of subsequent attacks. Therefore, we ignore this analysis process. Please refer to [20] for more details.

- In Step 1, the participants P_0 and P_1 jointly generate non-zero random elements $r_*, r_0, r_1, \cdots, r_{\ell_x}$ in the integer ring $\mathbb{Z}_q$. It implies that these elements satisfy $0 < r_*, r_0, r_1, \cdots, r_{\ell_x} < q$.
- In Step 2, P_0 and P_1 set $[x'] = (-1)^t \cdot [x]$, where $x \in [0, 2^{\ell_x}) \cup (q - 2^{\ell_x}, q)$ is a secret input, and $t \in \{0, 1\}$ is a random bit[3]. According to the property

[3] In Step 2 of Algorithm 1, the authors wrote $[x] := (-1)^t \cdot [x]$. The use of two identical symbols x here can easily cause confusion. Therefore, we modify the x on the left side of the expression to x'.

(5) in the secret sharing scheme, we have

$$x' = (-1)^t \cdot x \mod q.$$

It implies that $x' \in [0, 2^{\ell_x}) \cup (q - 2^{\ell_x}, q)$. If we write $[x] = ([x]_0, [x]_1)$ and $[x'] = ([x']_0, [x']_1)$, then we get

$$[x']_0 = (-1)^t \cdot [x]_0 \mod q, \text{ and } [x']_1 = (-1)^t \cdot [x]_1 \mod q.$$

- In Step 3, P_0 and P_1 set $u_*, u_0, u_1, \cdots, u_{\ell_x}$ satisfying $u_* = (-1)^t$, $u_0 = x'$ and u_i is the output of the i-bit truncation function, i.e.

$$u_i = \mathrm{TRC}(x', i), 1 \leq i \leq \ell_x. \tag{9}$$

A two-party secret sharing of u_i is $[u_i] = ([u_i]_0, [u_i]_1)$ for $i \in \{*, 0, 1, \cdots, \ell_x\}$, where $u_i = [u_i]_0 + [u_i]_1 \mod q$. The participants P_0 and P_1 hold shares $[u_i]_0$ and $[u_i]_1$ for $i = *, 0, 1, \cdots, \ell_x$, respectively.
- In Step 4, P_0 and P_1 set[4]

$$[v_*] = [u_*] + 3 \cdot [u_0] - [1] \mod q, \tag{10}$$

$$[v_i] = (\sum_{k=i}^{\ell_x} [u_k]) - [1] \mod q, 0 \leq i \leq \ell_x, \tag{11}$$

where the tuple $[1] = (1, 0)$. A two-party secret sharing of v_i is $[v_i] = ([v_i]_0, [v_i]_1)$ for $i \in \{*, 0, 1, \cdots, \ell_x\}$, where $v_i = [v_i]_0 + [v_i]_1 \mod q$. The participants P_0 and P_1 hold shares $[v_i]_0$ and $[v_i]_1$ for $i = *, 0, 1, \cdots, \ell_x$, respectively.
- In Step 5, the participants P_0 and P_1 first mask $[v_i]$ using non-zero random number r_i in Step 1, where $i = *, 0, 1, \cdots, \ell_x$, and obtain the array $r_* \cdot [v_*], r_1 \cdot [v_0], \cdots, r_{\ell_x} \cdot [v_{\ell_x}]$. Then P_0 and P_1 choose a random permutation Π, and permutate the above array, and get a new array $[w_*], [w_0], \cdots, [w_{\ell_x}]$. A two-party secret sharing of w_i is $[w_i] = ([w_i]_0, [w_i]_1)$ for $i \in \{*, 0, 1, \cdots, \ell_x\}$, where $w_i = [w_i]_0 + [w_i]_1 \mod q$. P_0 and P_1 hold shares $[w_i]_0$ and $[w_i]_1$ for $i = *, 0, 1, \cdots, \ell_x$, respectively.
- In Step 6, P_0 and P_1 send shares $[w_i]_0$ and $[w_i]_1$ to P_2 for $i = *, 0, 1, \cdots, \ell_x$, respectively.
- In Step 7, the participant reconstructs w_i's based on the obtained tuples $([w_i]_0, [w_i]_1)$. Specifically,

$$w_i = [w_i]_0 + [w_i]_1 \mod q \text{ for } i = *, 0, 1, \cdots, \ell_x.$$

If there is a w_i that is equal to 0, set $\mathrm{DReLU}(x)' = 1$. Otherwise, $\mathrm{DReLU}(x)' = 0$.
- In Step 8, P_2 sends the shares $[\mathrm{DReLU}(x)']_0$ and $[\mathrm{DReLU}(x)']_1$ to P_0 and P_1, respectively, where $[\mathrm{DReLU}(x)'] = ([\mathrm{DReLU}(x)']_0, [\mathrm{DReLU}(x)']_1)$.
- In Step 9, P_0 and P_1 execute an XOR operation to obtain the shares of the output $\mathrm{DReLU}(x) := \mathrm{DReLU}(x)' \oplus t$.

Algorithm 1 DReLU protocol.

Input: The shares of x.

Output: The shares of $\mathrm{DReLU}(x)$.

// P_0 and P_1 initialization.

1: P_0 and P_1 generate $\ell_x + 2$ numbers of non-zero random ring elements $\{r_*, r_0, r_1, \cdots, r_{\ell_x}\}$ from $\mathbf{seed}_{01}$.

2: P_0 and P_1 set $[x] := (-1)^t \cdot [x]$.

3: P_0 and P_1 set $[u_*] := [(-1)^t]$, $[u_0] := [x]$, and $[u_i] := [\mathrm{TRC}(x, i)]$, $\forall i \in [1, \ell_x]$.

4: P_0 and P_1 set $[v_*] := [u_*] + 3 \cdot [u_0] - 1$, $[v_i] := (\sum_{k=i}^{\ell_x} [u_k]) - 1$, $\forall i \in [0, \ell_x]$.

5: P_0 and P_1 set $[\{w_i\}] := [\prod\{r_i \cdot v_i\}]$, using the shuffle-seed from $\mathbf{seed}_{01}$.

6: P_0 and P_1 send the shares $[\{w_i\}]$ to P_2.

// P_2 processes.

7: P_2 reconstructs $\{w_i\}$ and sets $\mathrm{DReLU}(x)' = 1$ if there exists zero(s) in $\{w_i\}$; otherwise $\mathrm{DReLU}(x)' = 0$.

8: P_2 shares $\mathrm{DReLU}(x)'$ to P_0 and P_1.

9: P_0 and P_1 output the shares of $t \oplus \mathrm{DReLU}(x)'$.

In order to illustrate the following attack approaches more clearly, we give Fig. 1 for Steps 1-7 of the DReLU protocol from the perspective of participants P_0, P_1 and P_2.

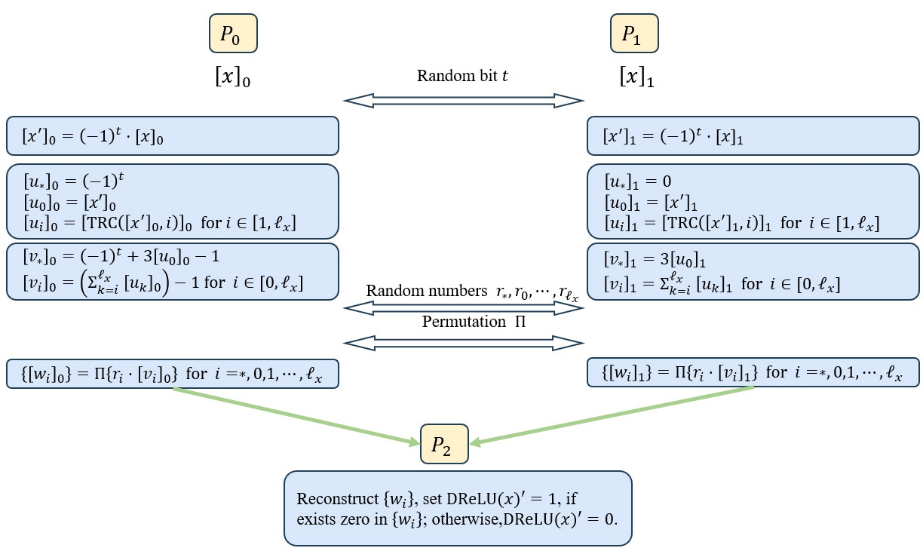

Fig. 1. The description for Steps 1-7 of the DReLU protocol from the perspective of participants.

4 In Step 4 of Algorithm 1, the authors write briefly $[v_*] = [u_*] + 3 \cdot [u_0] - 1$ and $[v_i] = (\sum_{k=i}^{\ell_x} [u_k]) - 1$. In fact, this writing style is not standard, so we modify 1 here to [1].

In the next two secret recovery attacks, we always assume that the participant P_2 is a passive adversary. According to Step 7 in Algorithm 1, P_2 can obtain these tuples $[w_i]$ for $i = *, 0, 1, \cdots, \ell_x$. Based on Step 5, the array $[w_*], [w_0], \cdots, [w_{\ell_x}]$ is a random permutation of the array $r_* \cdot [v_*], r_0 \cdot [v_0], \cdots, r_{\ell_x} \cdot [v_{\ell_x}]$. That is, there is the following relation

$$\{[w_*], [w_0], \cdots, [w_{\ell_x}]\} = \{r_* \cdot [v_*], r_0 \cdot [v_0], \cdots, r_{\ell_x} \cdot [v_{\ell_x}]\}, \tag{12}$$

where $[w_i] = ([w_i]_0, [w_i]_1)$ and $[v_i] = ([v_i]_0, [v_i]_1)$. In the following analysis, we not only consider the case where q is a power of 2 in the system settings of Bicoptor, i.e. $q = 2^{64}$, but also consider the case where q is a random 64-bit prime number.

4 The secret recovery attack on DReLU protocol

In this section, we present an attack to restore the secret in the DReLU protocol.

4.1 Obtaining equations related to x' and $[x']_0$

Based on the expression (12), there is $k \in \{*, 0, 1, \cdots, \ell_x\}$ that makes the relation $[w_k] = r_* \cdot [v_*]$ hold. That is,

$$[w_k]_0 \equiv r_* \cdot [v_*]_0 \mod q, \tag{13}$$

$$[w_k]_1 \equiv r_* \cdot [v_*]_1 \mod q, \tag{14}$$

Hence, P_2 can search up to $\ell_x + 2$ times to find the desired k.

After multiplying both sides of the relation (13) by $[v_*]_1$ and both sides of the relation (14) by $[v_*]_0$, and subtracting the two equations obtained, P_2 gets a new relation

$$[w_k]_0 \cdot [v_*]_1 \equiv [w_k]_1 \cdot [v_*]_0 \mod q. \tag{15}$$

Now we present the following result for the tuple $[v_*] = ([v_*]_0, [v_*]_1)$.

Lemma 3. *Define $x' \in [0, 2^{\ell_x}) \cup (q - 2^{\ell_x}, q)$ and $t \in \{0, 1\}$ as in Step 2 of the DReLU protocol. Define $[v_*] = ([v_*]_0, [v_*]_1)$ as in Step 4 of the DReLU protocol. Then we get*

$$[v_*]_0 =(-1)^t + 3 \cdot [x']_0 - 1 \mod q, \tag{16}$$

$$[v_*]_1 =3 \cdot [x']_1 \mod q. \tag{17}$$

Furthermore, we obtain

$$v_* = 3x' + (-1)^t - 1 \mod q. \tag{18}$$

Proof. From the expression (10), we have $[v_*] \equiv [u_*]+3\cdot[u_0]-[1] \mod q$. Based on the property (3), (4) and (5) of secret sharing, we deduce that

$$\begin{aligned}[v_*]_0 &= [u_*]_0 + 3\cdot[u_0]_0 - 1 \mod q, \\ [v_*]_1 &= [u_*]_1 + 3\cdot[u_0]_1 \mod q.\end{aligned} \tag{19}$$

Note that $u_* = (-1)^t$. Hence, u_* equals ± 1. According to the knowledge of secret sharing, we obtain that $[u_*] = ([u_*]_0, [u_*]_1)$, where

$$[u_*]_0 = (-1)^t, \text{ and } [u_*]_1 = 0. \tag{20}$$

From $u_0 = x'$, $[u_0] = ([u_0]_0, [u_0]_1)$ and $[x'] = ([x']_0, [x']_1)$, we get

$$[u_0]_0 = [x']_0, \text{ and } [u_0]_1 = [x']_1. \tag{21}$$

Plugging (20) and (21) into the above relation (19), we have $[v_*]_0 = (-1)^t + 3\cdot[x']_0 - 1 \mod q$, and $[v_*]_1 = 3\cdot[x']_1 \mod q$. It means that

$$[v_*]_0 + [v_*]_1 \equiv 3\cdot([x']_0 + [x']_1) + (-1)^t - 1 \mod q.$$

Note that $v_* = [v_*]_0 + [v_*]_1 \mod q$ and $x' = [x']_0 + [x']_1 \mod q$. Hence, we deduce the relation (18), that is, $v_* = 3x' + (-1)^t - 1 \mod q$. □

Plugging (16) and (17) into the relation (15), P_2 gets

$$[w_k]_0 \cdot (3\cdot[x']_1) \equiv [w_k]_1 \cdot ((-1)^t + 3\cdot[x']_0 - 1) \mod q. \tag{22}$$

After adding $[w_k]_0 \cdot (3\cdot[x']_0)$ to both sides of the relation (22), P_2 obtains the following equation

$$3[w_k]_0\cdot([x']_0+[x']_1) \equiv [w_k]_1\cdot((-1)^t-1)+3([w_k]_0+[w_k]_1)\cdot[x']_0 \mod q. \tag{23}$$

Plugging the relations

$$x' = [x']_0 + [x']_1 \mod q \text{ and } w_k = [w_k]_0 + [w_k]_1 \mod q$$

into the relation (23), P_2 obtains

$$3[w_k]_0 \cdot x' \equiv [w_k]_1 \cdot ((-1)^t - 1) + 3w_k \cdot [x']_0 \mod q.$$

Based on $t = 0$ or 1, the above relation can be rewritten as

$$\begin{cases} 3w_k\cdot[x']_0 \equiv 3[w_k]_0\cdot x' \mod q & \text{if } t = 0, \\ 3w_k\cdot[x']_0 \equiv 3[w_k]_0\cdot x' + 2[w_k]_1 \mod q & \text{if } t = 1. \end{cases} \tag{24}$$

Let the integer K be the greatest common divisor of integers $3w_k$ and q. That is, $K = \gcd(3w_k, q)$. Because the participant P_2 already knows w_k and q, P_2 can publicly compute K. For the modulus equations in relation (24), after dividing by the above K, P_2 can obtain the following modulus equations

$$\begin{cases} \frac{3w_k}{K} \cdot [x']_0 \equiv \frac{3[w_k]_0 \cdot x'}{K} \mod \frac{q}{K} & \text{if } t=0, \\ \frac{3w_k}{K} \cdot [x']_0 \equiv \frac{3[w_k]_0 \cdot x' + 2[w_k]_1}{K} \mod \frac{q}{K} & \text{if } t=1. \end{cases}$$

Note that $\gcd(\frac{3w_k}{K}, \frac{q}{K}) = 1$. According to the above equations, P_2 obtains the relation:

$$\begin{cases} [x']_0 \equiv (\frac{3w_k}{K})^{-1} \cdot \frac{3[w_k]_0 \cdot x'}{K} \mod \frac{q}{K} & \text{if } t=0, \\ [x']_0 \equiv (\frac{3w_k}{K})^{-1} \cdot \frac{3[w_k]_0 \cdot x' + 2[w_k]_1}{K} \mod \frac{q}{K} & \text{if } t=1. \end{cases} \tag{25}$$

In fact, $x' \in [0, 2^{\ell_x}) \cup (q - 2^{\ell_x}, q)$. It is because that $x' = (-1)^t \cdot x \in \mathbb{Z}_q$, where $t \in \{0, 1\}$ and $x \in [0, 2^{\ell_x}) \cup (q - 2^{\ell_x}, q)$. Therefore, P_2 can obtain all candidates of $[x']_0$ by enumerating the value of x' from (25). In the next subsection, we will give a detailed explanation.

4.2 Obtaining candidate tuples of $(x', [x']_0, [x']_1)$

Let the integer

$$r_t := \begin{cases} (\frac{3w_k}{K})^{-1} \cdot \frac{3[w_k]_0 \cdot x'}{K} \mod \frac{q}{K} & \text{if } t=0, \\ (\frac{3w_k}{K})^{-1} \cdot \frac{3[w_k]_0 \cdot x' + 2[w_k]_1}{K} \mod \frac{q}{K} & \text{if } t=1. \end{cases}$$

The integer r_t is the involved remainder after modulo $\frac{q}{K}$. Clearly, $0 \le r_t < \frac{q}{K}$. For any fixed candidate of x', the corresponding value of r_t is given. Note that $x' \in [0, 2^{\ell_x}) \cup (q - 2^{\ell_x}, q)$ and $t \in \{0, 1\}$. Hence, there are at most 2^{ℓ_x+1} different candidate values for r_t.

Note that $[x']_0$ is a random number in $\mathbb{Z}_q$. According to the expression (25), P_2 can rewrite $[x']_0$ as

$$[x']_0 = r_t + s_t \cdot \frac{q}{K}.$$

Here s_t is an unknown integer satisfying $0 \le s_t \le K-1$. P_2 gets all candidates of $[x']_0$ by enumerating the candidates of integers r_t and s_t. Because there are a maximum of 2^{ℓ_x+1} candidate values for r_t, and $0 \le s_t \le K-1$, the maximum number of candidate values for $[x']_0$ is $2^{\ell_x+1} \cdot K$. Based on the relation $[x']_1 = [x'] - [x']_0 \mod q$, the value of $[x']_1$ is determined by the values of x' and $[x']_0$. It means that the number of the different candidate tuples of $(x', [x']_0, [x']_1)$ is at most $2^{\ell_x+1} \cdot K$.

Finally, let us analyze the size of $K = \gcd(3w_k, q)$. We first discuss the case of w_k. From (13) and (14), we get $[w_k]_0 + [w_k]_1 \equiv r_* \cdot ([v_*]_0 + [v_*]_1) \mod q$.

Plugging the relations $w_k = [w_k]_0 + [w_k]_1 \mod q$ and $v_* = [v_*]_0 + [v_*]_1 \mod q$ into the above relation, we obtain

$$w_k = r_* \cdot v_* \mod q, \tag{26}$$

where r_* is a non-zero random number in $\mathbb{Z}_q$. For a prime q, the greatest common divisor K equals 1 with overwhelming probability. For $q = 2^\ell$, K is a small positive integer with a high probability. The detailed analysis is presented in Appendix A.

4.3 Filtering out the correct tuple

The goal in this subsection is to filter out the correct tuple $(x', [x']_0, [x']_1)$ from multiple candidates. This process may include multiple rounds of filtering.

For any given candidate $(\tilde{x}, [\tilde{x}]_0, [\tilde{x}]_1)$, where $\tilde{x} \in \mathbb{Z}_q$ is a candidate of x', the participant P_2 executes Steps 2 and 3 in Algorithm 1, and calculates

$$[\tilde{v}_{\ell_x}] := [\tilde{u}_{\ell_x}] - [1] \mod q,$$

where $[\tilde{u}_{\ell_x}] := [\mathrm{TRC}(\tilde{x}, \ell_x)]$ and $[1] = (1, 0)$. Note that $\tilde{x}$ is a candidate of x'. Hence $[\tilde{v}_{\ell_x}]$ is the corresponding candidate of $[v_{\ell_x}]$. Then P_2 checks whether there exists $j_1 \in \{*, 0, 1, \cdots, \ell_x\} \setminus \{k\}$ such that the corresponding $[w_{j_1}]$ satisfies the condition

$$[w_{j_1}]_0 \cdot [\tilde{v}_{\ell_x}]_1 \equiv [w_{j_1}]_1 \cdot [\tilde{v}_{\ell_x}]_0 \mod q, \tag{27}$$

where $[w_{j_1}] = ([w_{j_1}]_0, [w_{j_1}]_1)$ and $[\tilde{v}_{\ell_x}] = ([\tilde{v}_{\ell_x}]_0, [\tilde{v}_{\ell_x}]_1)$. It is worth noting that the k here is the integer satisfying (13) and (14). In other words, the involved tuple $[w_k] = ([w_k]_0, [w_k]_1)$ is already satisfied with the following relationship

$$[w_k]_0 \cdot [v_*]_1 \equiv [w_k]_1 \cdot [v_*]_0 \mod q,$$

where $[v_*] = ([v_*]_0, [v_*]_1)$. In the case, P_2 needs to search for such j_1 up to $\ell_x + 1$ times to verify whether the relationship (27) is valid.

If the relation (27) is satisfied, then the candidate $(\tilde{x}, [\tilde{x}]_0, [\tilde{x}]_1)$ is kept. Otherwise, it is removed. After the above filtering process, if the remaining candidates are not unique. P_2 will continue filtering in the following way.

Let $(\tilde{x}, [\tilde{x}]_0, [\tilde{x}]_1)$ be one of the remaining candidates. P_2 computes

$$[\tilde{v}_{\ell_x - 1}] := ([\tilde{u}_{\ell_x - 1}] + [\tilde{u}_{\ell_x}]) - [1] \mod q,$$

where $[\tilde{u}_t] := [\mathrm{TRC}(\tilde{x}, t)]$ for $\ell_x - 1 \le t \le \ell_x$. Clearly, $[\tilde{v}_{\ell_x - 1}]$ is the corresponding candidate of $[v_{\ell_x - 1}]$. Then P_2 checks whether there exists $j_2 \in \{*, 0, 1, \cdots, \ell_x\} \setminus \{k, j_1\}$ such that the corresponding $[w_{j_2}]$ satisfies the condition

$$[w_{j_2}]_0 \cdot [\tilde{v}_{\ell_x - 1}]_1 \equiv [w_{j_2}]_1 \cdot [\tilde{v}_{\ell_x - 1}]_0 \mod q, \tag{28}$$

where $[w_{j_2}] = ([w_{j_2}]_0, [w_{j_2}]_1)$ and $[\tilde{v}_{\ell_x - 1}] = ([\tilde{v}_{\ell_x - 1}]_0, [\tilde{v}_{\ell_x - 1}]_1)$. In this case, P_2 needs to search for such j_2 up to ℓ_x times to verify whether the relation

(28) is valid. If the relation (28) holds, then the candidate $(\tilde{x}, [\tilde{x}]_0, [\tilde{x}]_1)$ is kept. Otherwise, the candidate is removed.

The above is the case of two rounds of filtering. Next, we will describe the general situation. We assume that a total of m rounds of filtering are required. Clearly, $1 \leq m \leq \ell_x + 1$. In this process, P_2 needs to check whether there exists $j_s \in \{*, 0, 1, \cdots, \ell_x\} \setminus \{k, j_1, \cdots, j_{s-1}\}$ such that the following condition holds:

$$[w_{j_s}]_0 \cdot [\tilde{v}_{\ell_x+1-s}]_1 \equiv [w_{j_s}]_1 \cdot [\tilde{v}_{\ell_x+1-s}]_0 \mod q, \tag{29}$$

where s takes 1, 2, $\cdots, m$ in sequence, $[w_{j_s}] = ([w_{j_s}]_0, [w_{j_s}]_1)$ and $[\tilde{v}_{\ell_x+1-s}] = ([\tilde{v}_{\ell_x+1-s}]_0, [\tilde{v}_{\ell_x+1-s}]_1)$. It is easy to see that the relations (27) and (28) are two cases of (29) for $s = 1$ and 2, respectively. If the relation (29) is satisfied, then the candidate $(\tilde{x}, [\tilde{x}]_0, [\tilde{x}]_1)$ is kept. Otherwise, the candidate is removed. For this general case, if the number of remaining candidates is equal to 1, then we claim that the candidate is the the desired tuple $(x', [x']_0, [x']_1)$ we are looking for. This claim can be analyzed as follows.

If the candidate $(\tilde{x}, [\tilde{x}]_0, [\tilde{x}]_1) = (x', [x']_0, [x']_1)$, then the corresponding candidate $[\tilde{v}_{\ell_x+1-s}] = [v_{\ell_x+1-s}]$ for $1 \leq s \leq m$. Note that the array $[w_i]$ for $i = *, 0, 1, \cdots, \ell_x$ is obtaining by randomly per-mutating the array $r_i \cdot [v_i]$ for $i = *, 0, 1, \cdots, \ell_x$. In other words, the relationship (12) holds. Therefore, the expression (29) is always satisfied when $[\tilde{v}_{\ell_x+1-s}] = [v_{\ell_x+1-s}]$. It means that the desired tuple $([x'], [x']_0, [x']_1)$ is not removed from the candidate set. Hence, if there is only one candidate in the end, then this candidate must be $(x', [x']_0, [x']_1)$.

Let C_s be the candidate set before the s-th round of filtering, where $1 \leq s \leq m \leq \ell_x + 1$. Let $|C_s|$ be the number of elements in the candidate set C_s. Clearly, the number of elements in the corresponding candidate sets is monotonically decreasing, i.e., $1 \leq |C_m| \leq |C_{m-1}| \leq \cdots \leq |C_1| \leq 2^{\ell_x+1} \cdot K$. For this general situation, P_2 needs to compute the relationship (29) up to

$$|C_1| \cdot (\ell_x + 1) + |C_2| \cdot \ell_x + \cdots + |C_m| \cdot (\ell_x - m + 2) = \mathcal{O}(\ell_x 2^{\ell_x})$$

times to determine whether it is met, where the times of calculations is mainly determined by the exponent of ℓ_x.

Remark 1. For the specific parameters $\ell = 64, \ell_x = 13$ in the DReLU protocol, our experiment shows that, after two rounds of filtration, i.e. $m = 2$, it is sufficient to restore the desired tuple $(x', [x']_0, [x']_1)$.

Once the tuple $(x', [x']_0, [x']_1)$ is found out, the involved r_t is also determined in this process. It implies that the corresponding $t \in \{0, 1\}$ is obtained. Therefore, the secret x is revealed by computing $x = (-1)^t \cdot x' \mod q$.

5 The improved attack on DReLU protocol

In this section, we present an improved attack on the DReLU protocol to recover the secret x, the computational complexity of which is a polynomial on ℓ_x. This attack uses a lattice reduction algorithm instead of enumeration, which can be used in general cases. We still assume that the participant P_2 is a passive adversary.

5.1 Generating modular linear equation with small root

According to the expression (12), there is $j \in \{*, 0, 1, \cdots, \ell_x\}$ satisfying the following relation

$$\begin{cases} [w_j]_0 \equiv r_{\ell_x} \cdot [v_{\ell_x}]_0 \mod q, \\ [w_j]_1 \equiv r_{\ell_x} \cdot [v_{\ell_x}]_1 \mod q, \end{cases} \tag{30}$$

where $[w_j] = ([w_j]_0, [w_j]_1)$ and $[v_{\ell_x}] = ([v_{\ell_x}]_0, [v_{\ell_x}]_1)$. Therefore, P_2 needs to search up to $\ell_x + 2$ times to get the desired j. Once such j is obtained, based on the above two relations, P_2 will get the following equation

$$[w_j]_0 \cdot [v_{\ell_x}]_1 \equiv [w_j]_1 \cdot [v_{\ell_x}]_0 \mod q. \tag{31}$$

In order to transform (31) into a modular equation with small roots, the following lemma is utilized.

Lemma 4. *Define $x' \in [0, 2^{\ell_x}) \cup (q - 2^{\ell_x}, q)$ as in Step 2 of the DReLU protocol. Let integers $y := \left\lfloor \frac{[x']_0}{2^{\ell_x}} \right\rfloor$ and $z := \left\lfloor \frac{q-[x']_1}{2^{\ell_x}} \right\rfloor$. Then there is the following relation*

$$[v_{\ell_x}]_0 = y - 1, \text{ and } [v_{\ell_x}]_1 = q - z. \tag{32}$$

Proof. Based on the expression (11), we get $[v_{\ell_x}] = [u_{\ell_x}] - [1]$. According to the property of secret sharing, we obtain that $[v_{\ell_x}]_0 = [u_{\ell_x}]_0 - 1$ and $[v_{\ell_x}]_1 = [u_{\ell_x}]_1$. From the expression (9), we have that $[u_{\ell_x}]_0 = [\mathrm{TRC}(x', \ell_x)]_0$ and $[u_{\ell_x}]_1 = [\mathrm{TRC}(x', \ell_x)]_1$. It implies that

$$[v_{\ell_x}]_0 = [\mathrm{TRC}(x', \ell_x)]_0 - 1, \text{ and } [v_{\ell_x}]_1 = [\mathrm{TRC}(x', \ell_x)]_1. \tag{33}$$

According to (6) and (7), there are the following two relations

$$[\mathrm{TRC}(x', \ell_x)]_0 = \mathrm{rShift}([x']_0, \ell_x), \text{ and } [\mathrm{TRC}(x', \ell_x)]_1 = q - \mathrm{rShift}(q - [x']_1, \ell_x),$$

where $\mathrm{rShift}([x']_0, \ell_x) = \lfloor \frac{[x']_0}{2^{\ell_x}} \rfloor = y$ and $\mathrm{rShift}(q - [x']_1, \ell_x) = \lfloor \frac{q-[x']_1}{2^{\ell_x}} \rfloor = z$. Hence,

$$[\mathrm{TRC}(x', \ell_x)]_0 = y \text{ and } [\mathrm{TRC}(x', \ell_x)]_1 = q - z. \tag{34}$$

Plugging (34) into (33), we deduce that the relation (32) is satisfied. □

Note that $0 \le [x']_0, [x']_1 < q$. Thus $0 \le y = \left\lfloor \frac{[x']_0}{2^{\ell_x}} \right\rfloor < \frac{q}{2^{\ell_x}}$ and $0 \le z = \left\lfloor \frac{q-[x']_1}{2^{\ell_x}} \right\rfloor < \frac{q}{2^{\ell_x}}$. For the case that $\ell = 64$ and $\ell_x = 13$ in the DReLU algorithm, the integers y and z are small compared to the modulus q.

Plugging the relation (32) into (31) and rearranging the obtained equation, P_2 yields

$$[w_j]_1 \cdot y + [w_j]_0 \cdot z \equiv [w_j]_1 \mod q. \tag{35}$$

Once $[w_j]_0$ and $[w_j]_1$ are given, the equation (35) is a modular linear equation with small root (y, z).

5.2 Obtaining candidate tuples of (y, z)

We first present the following lemma.

Lemma 5. *Define $x' \in [0, 2^{\ell_x}) \cup (q - 2^{\ell_x}, q)$ as in Step 2 of the DReLU protocol. Define y and z as in Lemma 4. Then we have the following relation with probability $1 - 2^{\ell_x + 1 - \ell}$:*

$$y = z + \epsilon \mod q, \tag{36}$$

where $\epsilon \in \{-1, 0, 1\}$. If $x' \in [0, 2^{\ell_x})$, then ϵ= 0 or 1. If $x' \in (q - 2^{\ell_x}, q)$, then ϵ = 0 or -1.

Proof. According to (34), i.e. $[\mathrm{TRC}(x', \ell_x)]_0 = y$ and $[\mathrm{TRC}(x', \ell_x)]_1 = q - z$, and the relation $\mathrm{TRC}(x', \ell_x) = [\mathrm{TRC}(x', \ell_x)]_0 + [\mathrm{TRC}(x', \ell_x)]_1 \mod q$, we have

$$\mathrm{TRC}(x', \ell_x) = y - z \mod q. \tag{37}$$

By Lemma 2, we get that

$$\mathrm{TRC}(x', \ell_x) = \begin{cases} \mathrm{rShift}(\xi, \ell_x) + bit & \text{if } x' \in [0, 2^{\ell_x}), \\ q - \mathrm{rShift}(\xi, \ell_x) - bit & \text{if } x' \in (q - 2^{\ell_x}, q), \end{cases}$$

where $bit = 0$ or 1, and

$$\xi = \begin{cases} x' & \text{if } x' \in [0, 2^{\ell_x}), \\ q - x' & \text{if } x' \in (q - 2^{\ell_x}, q). \end{cases}$$

It implies that $0 \leq \xi < 2^{\ell_x}$. Hence, $\mathrm{rShift}(\xi, \ell_x) = \lfloor \frac{\xi}{2^{\ell_x}} \rfloor = 0$. Based on this relation, we get that

$$\mathrm{TRC}(x', \ell_x) = \begin{cases} bit & \text{if } x' \in [0, 2^{\ell_x}), \\ q - bit & \text{if } x' \in (q - 2^{\ell_x}, q). \end{cases} \tag{38}$$

Plugging (37) into (38), we obtain the relation (36), that is, $y = z + \epsilon \mod q$, where $\epsilon \in \{-1, 0, 1\}$. To be specific, $\epsilon = bit \in \{0, 1\}$ if $x' \in [0, 2^{\ell_x})$ and $\epsilon = (-1) \cdot bit \in \{0, -1\}$ if $x' \in (q - 2^{\ell_x}, q)$. □

Plugging the relation (36) into (35) and reorganizing the obtained equation, P_2 has

$$([w_j]_0 + [w_j]_1) \cdot z \equiv [w_j]_1 \cdot (1 - \epsilon) \mod q. \tag{39}$$

For the sake of discussion, let $W := \gcd([w_j]_0 + [w_j]_1, q)$. Since $[w_j]_0$, $[w_j]_1$ and q are known, the integer W can be publicly computed. According to (30), (32) and (36), there is the equation $[w_j]_0 + [w_j]_1 \equiv r_{\ell_x} \cdot (\epsilon - 1) \mod q$. It implies that the great common divisor $W = \gcd(r_{\ell_x} \cdot (\epsilon - 1), q)$, where $\epsilon \in \{-1, 0, 1\}$. Next, we will analyze based on the situation of ϵ.

The case of ϵ = 0 or -1. For a prime q, we have that $\gcd(r_{\ell_x}, q) = 1$ for a non-zero random number $r_{\ell_x} \in \mathbb{Z}_q$, and $\gcd(\epsilon - 1, q) = 1$ for $\epsilon = 0$ or 1. It implies that $\gcd(r_{\ell_x} \cdot (\epsilon - 1), q) = 1$. That is, the greatest common divisor $W = 1$. When

q is a power of 2, i.e. $q = 2^\ell$, we obtain that $\gcd(\epsilon - 1, q) = 1$ for $\epsilon = 0$ and $\gcd(\epsilon - 1, q) = 2$ for $\epsilon = -1$. Notice that $r_{\ell_x} \in \mathbb{Z}_q$ is a non-zero random number. Without loss of generality, we can write $r_{\ell_x} = 2^r \cdot \delta$, where $0 \le r \le \ell_x$ and δ is an odd number satisfying $0 < \delta < 2^{\ell-r}$. Thus the probability that $\gcd(r_{\ell_x}, q) = 2^r$ is equal to $\frac{2^{\ell-r-1}}{q-1} \approx \frac{1}{2^{r+1}}$, where $q = 2^\ell$. Specifically, if r is greater than $\ell_x - 1$, then the corresponding probability is less than $\frac{1}{2^{\ell_x}}$, which is negligible for the parameter $\ell_x = 13$ in the DReLU algorithm. When $\gcd(r_{\ell_x}, q) = 2^r$, the great common divisor $W = 2^r$ or 2^{r+1} for $\epsilon = 0$ or -1, respectively. According to the above analysis, the probability that $\frac{q}{W} \ge \frac{q}{2^{\ell_x}}$ is very close to 1.

For the modulus equation in (39), after dividing by the great common divisor W, the participant P_2 obtains the following equation

$$\frac{[w_j]_0 + [w_j]_1}{W} \cdot z \equiv \frac{[w_j]_1 \cdot (1 - \epsilon)}{W} \mod \frac{q}{W}.$$

From $\gcd(\frac{[w_j]_0+[w_j]_1}{W}, \frac{q}{W}) = 1$, P_2 produces

$$z \equiv (\frac{[w_j]_0 + [w_j]_1}{W})^{-1} \cdot \frac{[w_j]_1 \cdot (1 - \epsilon)}{W} \mod \frac{q}{W}.$$

Note that $0 \le z < \frac{q}{2^{\ell_x}}$ and $\frac{q}{2^{\ell_x}} \le \frac{q}{W}$ with an overwhelming probability. In this case, the relation $z < \frac{q}{W}$ is satisfied. It implies that

$$z = (\frac{[w_j]_0 + [w_j]_1}{W})^{-1} \cdot \frac{[w_j]_1 \cdot (1 - \epsilon)}{W} \mod \frac{q}{W}.$$

Once z is computed, P_2 can recover y by computing $y = z + \epsilon \mod q$, which is based on the relation (36).

In the case of $\epsilon = 0$ or -1, a maximum of $2 \cdot (\ell_x + 2)$ tuples (y, z) will be generated. It is worth noting that we can use the relationship of $0 \le y, z < \frac{q}{2^{\ell_x}}$ to filter candidate tuples. Simply put, if the candidate y and z satisfy this relationship, we keep this tuple, otherwise we discard it.

The case of $\epsilon = 1$. Notice that $W = \gcd(r_{\ell_x} \cdot (\epsilon - 1), q)$. Hence $W = q$. It means that P_2 cannot get any information on y from (39). For this case, P_2 can utilize the following way to determine the values of y and z.

Based on the expression (12), there is $l \in \{*, 0, 1, \cdots, \ell_x\} \setminus \{j\}$ satisfying

$$\begin{cases} [w_l]_0 \equiv r_{\ell_x - 1} \cdot [v_{\ell_x - 1}]_0 \mod q, \\ [w_l]_1 \equiv r_{\ell_x - 1} \cdot [v_{\ell_x - 1}]_1 \mod q, \end{cases} \tag{40}$$

where $[w_l] = ([w_l]_0, [w_l]_1)$ and $[v_{\ell_x-1}] = ([v_{\ell_x-1}]_0, [v_{\ell_x-1}]_1)$. Hence, P_2 needs to search at most $\ell_x + 1$ times to get the wanted l. From the relation (40), P_2 yields

$$[w_l]_0 \cdot [v_{\ell_x-1}]_1 \equiv [w_l]_1 \cdot [v_{\ell_x-1}]_0 \mod q. \tag{41}$$

Lemma 6. *Define y and z as in Lemma 4. Then we have*

$$\begin{aligned} [v_{\ell_x-1}]_0 &= 3y - 1 + \varepsilon_0 \mod q, \\ [v_{\ell_x-1}]_1 &= -3z - \varepsilon_1 \mod q, \end{aligned} \tag{42}$$

where $\varepsilon_0, \varepsilon_1 \in \{0, 1\}$ are unknown integers.

Proof. According to (11), there is the relation

$$\begin{aligned} [v_{\ell_x-1}]_0 &= [u_{\ell_x-1}]_0 + [u_{\ell_x}]_0 - 1 \mod q, \\ [v_{\ell_x-1}]_1 &= [u_{\ell_x-1}]_1 + [u_{\ell_x}]_1 \mod q, \end{aligned} \tag{43}$$

where $[u_{\ell_x}]_0 = \left\lfloor \frac{[x']_0}{2^{\ell_x}} \right\rfloor$, $[u_{\ell_x}]_1 = q - \left\lfloor \frac{q-[x']_1}{2^{\ell_x}} \right\rfloor$, $[u_{\ell_x-1}]_0 = \left\lfloor \frac{[x']_0}{2^{\ell_x-1}} \right\rfloor$, and $[u_{\ell_x-1}]_1 = q - \left\lfloor \frac{q-[x']_1}{2^{\ell_x-1}} \right\rfloor$. Note that $y = \left\lfloor \frac{[x']_0}{2^{\ell_x}} \right\rfloor$ and $z := \left\lfloor \frac{q-[x']_1}{2^{\ell_x}} \right\rfloor$. Hence we have

$$[u_{\ell_x}]_0 = y, \text{ and } [u_{\ell_x}]_1 = q - z. \tag{44}$$

Next, we reorganize $[u_{\ell_x-1}]_0$ and $[u_{\ell_x-1}]_1$. It is not hard to deduce that there are some integers $\varepsilon_0, \varepsilon_1 \in \{0, 1\}$ such that $\left\lfloor \frac{[x']_0}{2^{\ell_x-1}} \right\rfloor = 2 \cdot \left\lfloor \frac{[x']_0}{2^{\ell_x}} \right\rfloor + \varepsilon_0$, and $\left\lfloor \frac{q-[x']_1}{2^{\ell_x-1}} \right\rfloor = 2 \cdot \left\lfloor \frac{q-[x']_1}{2^{\ell_x}} \right\rfloor + \varepsilon_1$. It implies that

$$[u_{\ell_x-1}]_0 = 2y + \varepsilon_0, \text{ and } [u_{\ell_x-1}]_1 = q - (2z + \varepsilon_1). \tag{45}$$

Then we deduce the relation (42) by plugging (44) and (45) into (43). ☐

Plugging (42) and (36) into (41) and organizing the resulting equation, where the involved $\epsilon = 1$, P_2 obtains the following relation

$$3 \cdot ([w_l]_0 + [w_l]_1) \cdot z \equiv -[w_l]_0 \cdot \varepsilon_1 - [w_l]_1 \cdot (\varepsilon_0 + 2) \mod q. \tag{46}$$

For the sake of discussion, let the great common divisor $V := \gcd(3 \cdot ([w_l]_0 + [w_l]_1), q)$. When q is a prime or a power of 2, it is easy to see $\gcd(3, q) = 1$. Therefore $V = \gcd([w_l]_0 + [w_l]_1, q)$. Now we analyze the case of $[w_l]_0 + [w_l]_1$ modulo q. Combining the relations (40), (42) and (36), P_2 gets the relation

$$[w_l]_0 + [w_l]_1 \equiv r_{\ell_x-1} \cdot (2 + \varepsilon_0 - \varepsilon_1) \mod q.$$

It implies that $V = \gcd(r_{\ell_x-1} \cdot (2+\varepsilon_0-\varepsilon_1), q)$, where r_{ℓ_x-1} is a non-zero random number in $\mathbb{Z}_q$, and $\varepsilon_0, \varepsilon_1 \in \{0, 1\}$. Hence, $2+\varepsilon_0-\varepsilon_1$ can only take one value from 1, 2, or 3. Similar to the analysis of W mentioned above, we can deduce that, for a prime q, the greatest common divisor $V = 1$; for the situation of $q = 2^\ell$, the probability of $\frac{q}{V} \geq \frac{q}{2^{\ell_x}}$ is very close to 1, where the involved $\ell_x \geq 13$.

After dividing by the above V for the modulus equation in (46), P_2 obtains the following equation

$$\frac{3 \cdot ([w_l]_0 + [w_l]_1)}{V} \cdot z \equiv -\frac{[w_l]_0 \cdot \varepsilon_1 + [w_l]_1 \cdot (\varepsilon_0 + 2)}{V} \mod \frac{q}{V}.$$

Due to $\gcd(\frac{3 \cdot ([w_l]_0 + [w_l]_1)}{V}, \frac{q}{V}) = 1$, P_2 yields

$$z \equiv -(\frac{3 \cdot ([w_l]_0 + [w_l]_1)}{V})^{-1} \cdot \frac{[w_l]_0 \cdot \varepsilon_1 + [w_l]_1 \cdot (\varepsilon_0 + 2)}{V} \mod \frac{q}{V}.$$

Note that $0 \le z < \frac{q}{2^{\ell_x}} \le \frac{q}{V}$ with an overwhelming probability. Hence

$$z = -(\frac{3 \cdot ([w_l]_0 + [w_l]_1)}{V})^{-1} \cdot \frac{[w_l]_0 \cdot \varepsilon_1 + [w_l]_1 \cdot (\varepsilon_0 + 2)}{V} \mod \frac{q}{V},$$

where $\varepsilon_0, \varepsilon_1 = 0$ or 1. Finally, According to the relation (36), i.e., $y = z + \epsilon \mod q$, P_2 can compute all candidates of y.

In case of $\epsilon = 1$, a maximum of $4 \cdot (\ell_x + 1)$ tuples (y, z) will be obtained. Similar to the case of $\epsilon = 0$ or -1, we can also use the relationship $0 \le y, z < \frac{q}{2^{\ell_x}}$ to filter candidate values.

5.3 Recovering the secret x using lattice methods

In this subsection, the goal of P_2 is to recover the secret x. According to the expression (12), P_2 gets the following equation

$$[w_k]_0 \cdot [v_*]_1 \equiv [w_k]_1 \cdot [v_*]_0 \mod q, \tag{47}$$

where $k \in \{*, 0, 1, \cdots, \ell_x\} \setminus \{j, l\}$. The integers j, l satisfy relationships (30) and (40), respectively. According to Lemma 3, there are two relations (16) and (17) on $[v_*]_0$ and $[v_*]_1$, i.e.,

$$[v_*]_0 = 3[x']_0 + (-1)^t - 1 \mod q, \text{ and } [v_*]_1 = 3[x']_1 \mod q.$$

Plugging these two relations into (47), and reorganizing the obtained equation, P_2 gets

$$3[w_k]_1 \cdot [x']_0 - 3[w_k]_0 \cdot [x']_1 = [w_k]_1 \cdot (1 + (-1)^{t+1}) \mod q. \tag{48}$$

Recall that $z = \left\lfloor \frac{q-[x']_1}{2^{\ell_x}} \right\rfloor$ and $y = \left\lfloor \frac{[x']_0}{2^{\ell_x}} \right\rfloor$. Hence P_2 can rewrite $[x']_0$ and $q - [x']_1$ as

$$[x']_0 = 2^{\ell_x} \cdot y + c_1, \text{ and } q - [x']_1 = 2^{\ell_x} \cdot z + c_2, \tag{49}$$

where c_1 and c_2 are unknown integers satisfying $0 \le c_1, c_2 < 2^{\ell_x}$. Plugging the relation (49) into (48), and rearranging the obtained equation, P_2 deduces that

$$O_1 \cdot c_1 + O_2 \cdot c_2 + O_3 = 0 \mod q. \tag{50}$$

Here, O_1, O_2 and O_3 are known integers satisfying

$$\begin{aligned} O_1 =& 3 \cdot [w_k]_1 \mod q, \\ O_2 =& 3 \cdot [w_k]_0 \mod q, \\ O_3 =& 3z \cdot 2^{\ell_x} \cdot [w_k]_0 + (3y \cdot 2^{\ell_x} + (-1)^t - 1) \cdot [w_k]_1 \mod q. \end{aligned}$$

Note that the corresponding expressions for O_1, O_2, O_3 depend on tuples $[w_k]$, (y, z), and a random bit t. Based on the above analysis, we conclude that there are a maximum of $2 \cdot \ell_x \cdot (2(\ell_x + 2) + 4(\ell_x + 1))$ candidate values for tuples (O_1, O_2, O_3).

Lattice method I. For any fixed tuple (O_1, O_2, O_3), the participant P_2 can build a lattice $\mathcal{L}_{O_1,O_2,O_3}$ which is spanned by the row vectors of the matrix

$$\begin{bmatrix} 1 & 0 & 0 & O_1 \\ 0 & 1 & 0 & O_2 \\ 0 & 0 & 2^{\ell_x} & O_3 \\ 0 & 0 & 0 & q \end{bmatrix}.$$

It is not hard to see that the vector $\mathbf{v} := (c_1, c_2, 2^{\ell_x}, 0) \in \mathcal{L}_{O_1,O_2,O_3}$ from (50). Since $0 \le c_1, c_2 < 2^{\ell_x}$, P_2 gets that the Euclidean length of $\mathbf{v}$ satisfies $\|\mathbf{v}\| \le \sqrt{3} \cdot 2^{\ell_x}$.

Note that the determinant of $\mathcal{L}_{O_1,O_2,O_3}$ is $2^{\ell_x} q$, and the dimension is 4. According to Gaussian heuristics, the Euclidean length of the shortest non-zero vector is $\mathrm{GH}(\mathcal{L}_{O_1,O_2,O_3}) = \sqrt{\frac{2\pi e}{4}} \cdot (2^{\ell_x} q)^{\frac{1}{4}} \ge \sqrt{\frac{\pi e}{2}} \cdot 2^{\frac{\ell_x + \ell}{4}}$, where ℓ is the bit-length of q. If the condition

$$\ell > 3\ell_x \tag{51}$$

holds[5], then $\|\mathbf{v}\| < \mathrm{GH}(\mathcal{L}_{O_1,O_2,O_3})$, i.e., the Euclidean length of $\mathbf{v}$ is smaller than the length of the shortest non-zero vector estimated by Gaussian heuristics. In this case, the vector $\mathbf{v}$ is likely to be the shortest nonzero vector in 4-dimensional lattice $\mathcal{L}_{O_1,O_2,O_3}$. For a very low dimensional lattice, the LLL algorithm can heuristically find out the shortest non-zero vector. Once the shortest nonzero vector $\mathbf{v}$ is found, c_1, c_2 will be obtained based on the relation $\mathbf{v} = (c_1, c_2, 2^{\ell_x}, 0)$. Furthermore, the values $[x']_0, [x']_1$ are recovered according to the expression (49).

When the tuple (O_1, O_2, O_3) runs through all possibilities, P_2 can obtain candidates including the wanted tuple $([x']_0, [x']_1)$ through the above method.

The method of filtering out the desired tuple $([x']_0, [x']_1)$ from the candidate set is similar to that in Section 4.3. Simply put, for any given candidate $([\overline{x}]_0, [\overline{x}]_1)$, P_2 calculates $[\overline{v}_0]_0 = (\sum_{i=0}^{\ell_x} [\overline{u}_i]_0) - 1$, $[\overline{v}_0]_1 = \sum_{i=0}^{\ell_x} [\overline{u}_i]_1$, where $\overline{u}_i = \mathrm{TRC}(\overline{x}, i), 0 \le i \le \ell_x$. Then P_2 finds a pair $([w_u]_0, [w_u]_1)$, where $u \in \{*, 0, 1, \cdots, \ell_x\} \setminus \{j, k\}$, which should meet that $[w_u]_0 \cdot [\overline{v}_0]_1 = [w_u]_1 \cdot [\overline{v}_0]_0 \mod q$. During this process, P_2 needs to search up to ℓ_x times to obtain the desired u. If such $([w_u]_0, [w_u]_1)$ is obtained, then the candidate has a high probability of being correct. The experimental results show that this check method can make the probability of success reach 100% for specific parameters in the DReLU protocol.

Once the tuple $([x']_0, [x']_1)$ is obtained, the corresponding O_3 is also determined in this process. It means that the involved $t \in \{0, 1\}$ is known. Therefore, the secret x is recovered by computing $x = (-1)^t \cdot x' \mod q$ and $x' = [x']_0 + [x']_1 \mod q$.

[5] The condition (51) is met for the DReLU protocol, where the involved $\ell = 64$ and $\ell_x = 13$.

Lattice method II. Let the great common divisor $O := \gcd(O_1, O_2, q)$. According to (50), we have $O_3 \equiv -O_1 \cdot c_1 - O_2 \cdot c_2 \mod q$. It implies that O divides the integer O_3. For the equation in (50), after dividing by the integer O, the participant P_2 gets the equation

$$\frac{O_1}{O} \cdot c_1 + \frac{O_2}{O} \cdot c_2 + \frac{O_3}{O} \equiv 0 \mod \frac{q}{O}, \tag{52}$$

where $\frac{O_1}{O}$, $\frac{O_2}{O}$, $\frac{O_3}{O}$ and $\frac{q}{O}$ are integers satisfying $\gcd(\frac{O_1}{O}, \frac{O_2}{O}, \frac{q}{O}) = 1$.

Without loss of generality, we assume that the relation $\gcd(\frac{O_1}{O}, \frac{q}{O}) = 1$ is satisfied.[6] In this case, P_2 can rewrite (52) as

$$c_1 + O_2' \cdot c_2 + O_3' = 0 \mod \frac{q}{O}, \tag{53}$$

where $O_2' = (\frac{O_1}{O})^{-1}(\frac{O_2}{O}) \mod \frac{q}{O}$ and $O_3' = (\frac{O_1}{O})^{-1}(\frac{O_3}{O}) \mod \frac{q}{O}$ are known integers. For any fixed tuple (O_2', O_3'), P_2 can build lattice $\mathcal{L}_{O_2',O_3'}$ which is spanned by the row vectors of the 3-dimensional matrix

$$\begin{bmatrix} 1 & 0 & -O_2' \\ 0 & 2^{\ell_x} & -O_3' \\ 0 & 0 & \frac{q}{O} \end{bmatrix}.$$

It is easy to see that the vector $\mathbf{v}' := (c_2, 2^{\ell_x}, c_1) \in \mathcal{L}_{O_2',O_3'}$ due to (53). Since $0 \le c_1, c_2 < 2^{\ell_x}$, P_2 gets that the Euclidean length of $\mathbf{v}'$ satisfies $\|\mathbf{v}'\| \le \sqrt{3} \cdot 2^{\ell_x}$. Since that the determinant of $\mathcal{L}_{O_2',O_3'}$ is $2^{\ell_x} \cdot \frac{q}{O}$, and the dimension is 3. According to Gaussian heuristics, the Euclidean length of the shortest non-zero vector is $\mathrm{GH}(\mathcal{L}_{O_2',O_3'}) = \sqrt{\frac{2\pi e}{3}} \cdot (2^{\ell_x} \cdot \frac{q}{O})^{\frac{1}{3}} \ge \sqrt{\frac{2\pi e}{3}} \cdot \frac{2^{\frac{\ell_x+\ell}{3}}}{O^{1/3}}$. If the condition

$$\ell > 2\ell_x + \log_2 O \tag{54}$$

is satisfied, then $\|\mathbf{v}'\| < \mathrm{GH}(\mathcal{L}_{O_2',O_3'})$. In other words, the Euclidean length of $\mathbf{v}'$ is smaller than the length estimated by Gaussian heuristics. Hence $\mathbf{v}'$ is likely to be the shortest non-zero vector in the 3-dimensional lattice $\mathcal{L}_{O_2',O_3'}$, which can be heuristically found out by the LLL algorithm. The remaining analysis is similar to that in the lattice method I.

To compare the condition (51) in lattice method I and the condition (54) in lattice method II, we need to consider the size of O. For a prime q, the integer $O = 1$ with an overwhelming probability. For the case of $q = 2^\ell$, the integer O is a small positive integer with a high probability. The detailed analysis is given in Appendix B. To sum up, the condition (54) is better than (51).

[6] For a prime q, if $O = q$, then $\gcd(\frac{O_1}{O}, \frac{q}{O}) = 1$ and $\gcd(\frac{O_2}{O}, \frac{q}{O}) = 1$. If $O = 1$, then $\gcd(\frac{O_1}{O}, \frac{q}{O}) = 1$ or $\gcd(\frac{O_2}{O}, \frac{q}{O}) = 1$. Otherwise, we have $q \mid O_1$ and $q \mid O_2$. It implies that $\gcd(O_1, O_2, q) = O = q$. This is contradictory. For $q = 2^\ell$, we get that $\gcd(\frac{O_1}{O}, \frac{q}{O}) = 1$ or $\gcd(\frac{O_2}{O}, \frac{q}{O}) = 1$. Otherwise, we obtain that $\frac{O_1}{O}$, $\frac{O_2}{O}$ and $\frac{q}{O}$ are all powers of 2, and $\frac{O_1}{O}, \frac{O_2}{O}, \frac{q}{O} \ge 2$. Therefore, $\gcd(\frac{O_1}{O}, \frac{O_2}{O}, \frac{q}{O}) \ne 1$. It is contradictory.

Finally, let us analyze the computational complexity required for the attack method proposed in this section. This problem can be reduced to how many calculations are needed for the LLL algorithm. Whether it is the 4-dimensional lattice in Method I or the 3-dimensional lattice in Method II, the number of candidate lattices depends on the number of tuples (O_1, O_2, O_3). Hence, there are at most $2 \cdot \ell_x \cdot (2(\ell_x + 2) + 4(\ell_x + 1))$ candidates for the desired lattice. Based on the time complexity $n^5(n + \log_2 B)\log_2 B$ of the LLL algorithm [11], where n is the dimension of lattice and $\log B$ is the maximal bit-length of entries in the input lattice matrix, we conclude that the complexity of the LLL algorithm in both Method I and Method II is $O(\ell^2)$, where $\ell = \log_2 q$. Therefore, the overall time complexity in the worst-case scenario is $2 \cdot \ell_x \cdot (2(\ell_x+2)+4(\ell_x+1)) \cdot \mathcal{O}(\ell^2) = \mathcal{O}(\ell^2 \cdot \ell_x^2)$ which is a polynomial on ℓ_x.

6 Attack on the remaining protocols

In this section, we present that the security of other protocols can be also broken.

The case of the Equality protocol. The Equality protocol uses the DReLU protocol to determine whether two input values are equal while maintaining privacy. To be specific, for two inputs x and y, the Equality protocol can be written as the piecewise function:

$$\text{Equality}(x, y) = \begin{cases} 1, & \text{if } x = y, \\ 0, & \text{if } x \neq y. \end{cases} \tag{55}$$

Equivalently, there is the following relation:

$$\text{Equality}(x, y) = 1 - (\text{DReLU}(x - y) \oplus \text{DReLU}(y - x)). \tag{56}$$

The participant P_2 should only perform calculations and does not have access to secret information. However, as mentioned in Sections 4 and 5, the adversary P_2 is capable of recovering the input value of the function DReLU. Therefore, for the Equality protocol, P_2 will recover the input $x - y$ of $\text{DReLU}(x - y)$. Based on (55) or (56), P_2 also knows the value of $\text{Equality}(x, y)$. In fact, these values should be hidden from P_2.

The case of the ReLU protocol. The ReLU protocol requiring two rounds of communication was proposed, and the relationship between the ReLU protocol and DReLU protocol is that

$$\text{ReLU}(x) = \text{DReLU}(x) \cdot x = \begin{cases} x & \text{if } x \geq 0, \\ 0 & \text{if } x < 0. \end{cases}$$

Based on the attack methods in Sections 4 and 5, P_2 can recover the values of x and $\text{DReLU}(x)$. Therefore, P_2 knows the value of $\text{ReLU}(x)$. However, these values should be hidden from P_2.

The case of ABS protocol. The $\text{ABS}(x)$ protocol function can be written as

$$\text{ABS}(x) = \begin{cases} x, & \text{if } x \geq 0, \\ -x, & \text{if } x \leq 0. \end{cases}$$

Equivalently, $\mathrm{ABS}(x) = \mathrm{DReLU}(x) \cdot x + (\mathrm{DReLU}(x) - 1) \cdot x$. Note that P_2 could get the values of x and $\mathrm{DReLU}(x)$. Hence the value of $\mathrm{ABS}(x)$ is obtained by P_2.

The case of Dynamic ReLU protocol. The Dynamic ReLU protocol can be viewed as

$$\text{Dynamic ReLU}(x) = \begin{cases} \alpha_1 \cdot x, & \text{if } x \geq 0, \\ \alpha_0 \cdot x, & \text{if } x \leq 0, \end{cases}$$

where α_0 and α_1 are two constants. The Leaky ReLU, PReLU, and RReLU protocols are the special cases of Dynamic ReLU protocol. For Leaky ReLU, $\alpha_0 = 0.001$, $\alpha_1 = 1$. For PReLU, α_0 is a pre-trained constant and $\alpha_1 = 1$. For RReLU, α_0 is a random constant and $\alpha_1 = 1$. The Dynamic ReLU function can be rewritten as

$$\text{Dynamic ReLU}(x) = \alpha_1 \cdot \mathrm{DReLU}(x) + \alpha_0 \cdot (1 - \mathrm{DReLU}(x)) \cdot x.$$

Using the attack approaches in Sections 4 and 5, P_2 will get the values of x and $\mathrm{DReLU}(x)$. It implies that the value of Dynamic ReLU(x) is revealed by P_2, in other words, none of these protocols are secure anymore.

The case of Piecewise Linear Unit (PLU). Note that the above protocols can be viewed as piecewise functions with two segments. In general, for the $m+2$ segments, the PLU protocol is as follows.

$$\mathrm{PLU}(x) = \begin{cases} \alpha_{m+1} \cdot x + \beta_{m+1}, & \text{if } \gamma_m \leq x, \\ \alpha_m \cdot x + \beta_m, & \text{if } \gamma_{m-1} \leq x \leq \gamma_m, \\ \cdots & \\ \alpha_j \cdot x + \beta_j, & \text{if } \gamma_{j-1} \leq x \leq \gamma_j, \\ \cdots & \\ \alpha_1 \cdot x + \beta_1, & \text{if } \gamma_0 \leq x \leq \gamma_1, \\ \alpha_0 \cdot x + \beta_0, & \text{if } x \leq \gamma_0, \end{cases}$$

where $\mathrm{PLU}(x)$ has $m+2$ segments, α_i and β_i ($\forall i \in [0, m+1]$) and γ_j ($\forall j \in [0, m]$) are constants. The PLU function can be written as

$$\begin{aligned} \mathrm{PLU}(x) = & (\mathrm{DReLU}(x - \gamma_m) \oplus 0) \cdot (\alpha_{m+1} \cdot x + \beta_{m+1}) \\ & + \cdots \\ & + (\mathrm{DReLU}(x - \gamma_{j-1}) \oplus \mathrm{DReLU}(x - \gamma_j)) \cdot (\alpha_j \cdot x + \beta_j) \\ & + \cdots \\ & + (1 \oplus \mathrm{DReLU}(x - \gamma_0)) \cdot (\alpha_0 \cdot x + \beta_0). \end{aligned}$$

Based on the attack approaches in Sections 4 and 5, P_2 will get the values of $x - \gamma_i$ and $\mathrm{DReLU}(x - \gamma_i)$, for $\forall i \in [0, m]$. Since the values $\gamma_0, \cdots, \gamma_m$ are public, P_2 could recover the secret x.

Note that $\Pr(\gcd(O_1, O_2, q) = 1) \geq \Pr(\gcd(O_1, q) = 1)$. Note that $O_1 = 9 \cdot [x']_1 \mod q$. For a prime q, we get $\gcd(O_1, q) = \gcd([x']_1, q)$. Since $[x']_1$ is a random number in $\mathbb{Z}_q$, we obtain that $\Pr(\gcd([x']_1, q)) = 1/q$.

ReLU6 protocol is a special case for PLU protocol. For ReLU6 protocol, the parameters are $m = 1$, $\alpha_0 = \beta_0 = \beta_1 = \alpha_2 = \gamma_0 = 0$, $\alpha_1 = 1$, $\beta_2 = \gamma_1 = 6$.

$$\mathrm{ReLU6}(x) = \begin{cases} 6, & \text{if } 6 \leq x, \\ x, & \text{if } 0 \leq x < 6. \\ 0, & \text{if } x < 0. \end{cases}$$

Equivalently,

$$\begin{aligned} \mathrm{ReLU6}(x) = & (\mathrm{DReLU}(x-6) \oplus 0) \cdot 6 \\ & + (\mathrm{DReLU}(x) \oplus \mathrm{DReLU}(x-6)) \cdot x \\ & + (1 \oplus \mathrm{DReLU}(x)) \cdot 0. \end{aligned}$$

For ReLU6 protocol, P_2 recovers the input x of $\mathrm{DReLU}(x)$. Then P_2 can calculate the value of $\mathrm{ReLU6}(x)$. In fact, these values should not be exposed to P_2 who only plays a computational role.

MAX2 and MIN2 protocols. For MAX2 and MIN2 protocols, these functions are that:

$$\mathrm{MAX2}(x,y) = \begin{cases} x, & \text{if } x \geq y, \\ y, & \text{if } x \leq y, \end{cases} \text{ and } \mathrm{MIN2}(x,y) = \begin{cases} y, & \text{if } x \geq y, \\ x, & \text{if } x \leq y. \end{cases}$$

Here $\mathrm{MAX2}(x,y) = \mathrm{DReLU}(x-y) \cdot (x-y) + y$ and $\mathrm{MIN2}(x,y) = x - \mathrm{DReLU}(x-y) \cdot (x-y)$. For these two protocols, P_2 could recover the value of $x-y$. Although P_2 does not know the specific values of x and y from $x-y$. However, P_2 knows $x-y$, which will pose a threat to the security of these two protocols.

MAX protocol. As a general protocol, the MAX protocol is used to find the maximum value from multiple inputs $(\phi_1, \cdots, \phi_n)$, and it utilize the $\mathrm{uCMP}(\phi_i,\phi_j)$ protocol, which is defined as $\mathrm{uCMP}(\phi_i,\phi_j) := \mathrm{uDReLU}(\phi_i - \phi_j)$, for $\forall i, j \in [1, n]$ and $i \neq j$. Note that the $\mathrm{uDReLU}(x)$ protocol omits Steps 2 and 9 of the DReLu algorithm (see Alg. 1). Therefore, $\mathrm{uCMP}(\phi_i, \phi_j)$ can be rewritten as

$$\mathrm{uCMP}(\phi_i, \phi_j) = \mathrm{uDReLU}(\phi_i - \phi_j) = \begin{cases} 1, & \text{if } \phi_i \geq \phi_j, \\ 0, & \text{if } \phi_i < \phi_j. \end{cases}$$

Using the attack mathods in Sections 4 and 5, P_2 will know the values of $\phi_i - \phi_j$, for $\forall i, j \in [1, n]$ and $i \neq j$. In fact, P_2 does not get the specific values of ϕ_i and ϕ_j from these values of $\phi_i - \phi_j$. It is worth noting that the values of $\phi_i - \phi_j$ should not be exposed to P_2.

7 Experiments

In this section, we provide the experimental results. The experimental environment is running on a personal computer with 2.40GHz Intel(R) Core(TM) I5-10200h CPU and 16GB RAM, and the operating system is Ubuntu 18.04.6LTS. We use Python and Sage to simulate the running process of the DReLU protocol. Our toolkit is open-source, available at `https://github.com/Halowooder/BBBPSRA`.

In Table 1, we present the specific performance of attack methods in Section 4 and Section 5, where four sets of parameters are tested. The first set of parameters is that q is a random 64-bit prime, and $\ell_x = 13$. The second set of parameters is that $q = 2^{64}$ and $\ell_x = 13$, which is the parameters set used in Bicoptor. The third set of parameters is $q = 2^{64}$ and $\ell_x = 26$, where the value of ℓ_x is twice that of the one in the second parameter set. The last one is that $q = 2^{640}$ and $\ell_x = 130$, where the bit-length of q is ten times that of q in the other sets, and the value of ℓ_x is ten times that of the one in the second parameter set. For different settings, we respectively test the DReLU protocol 10000 times, and then provide the average time required to complete the attack.

Table 1. Comparison of the attack mathods in Sections 4 and 5. The column labeled "Time" means the running time of the attack algorithm in seconds, The column labeled "Success rate" represents success rate of the attack algorithm. The symbol **"-"** means that no result is given.

Attacks of DReLU	q	ℓ_x	**Time**	**Success rate**
The attack in Section 4	64-bit prime	13	8.98	100%
	2^{64}	13	67.95	100%
	2^{64}	26	-	-
	2^{640}	130	-	-
The attack in Section 5 using lattice method I	64-bit prime	13	1.52	100%
	2^{64}	13	0.87	100%
	2^{64}	26	-	-
	2^{640}	130	10.10	100%
The attack in Section 5 using lattice method II	64-bit prime	13	1.65	100%
	2^{64}	13	0.84	100%
	2^{64}	26	0.46	99.78%
	2^{640}	130	2.19	100%

We first explain the attack in Section 4. For the first and second sets of parameters, the experiment shows that it is sufficient to recover the secret in the DReLU algorithm after two rounds of filtration. Note that there is an integer K involved in this attack, which is the greatest common divisor of integers $3w_k$ and q. If q is a 64-bit prime, then $K = 1$ with an overwhelming probability. Therefore, we get the value of $[x']_0$ directly. However, if $q = 2^{64}$, then $K \neq 1$ with a certain probability. In this case, the lifting operation is required to restore $[x']_0$ from the remainder $[x']_0 \mod \frac{q}{K}$. Thus, the involved running time for $q = 2^{64}$ is higher than the situation for a 64-bit prime q. For the third parameter set, where $\ell_x = 26$ is involved. In this case, it is necessary to enumerate 2^{26} times on x' instead of 2^{13} times for the second parameter set. This means that the enumeration time will become relatively long. Therefore, we do not provide relevant experimental results on a personal computer. For the fourth set of parameters, the attack method in Section 4 cannot work effectively because the involved $\ell_x = 130$ is too large.

For the attack in Section 5 using lattice method I, the running time for the first and second parameter sets is roughly the same, because the modulus q is some element in the basis matrices, and its prime or power of 2 does not affect the efficiency of the attack. For the third parameter set, the relationship between ℓ and ℓ_x is that $2 \cdot \ell_x < \ell < 3 \cdot \ell_x$, therefore, the lattice method II could successfully recover the secret, while the lattice method I cannot recover the secret. It means that the lattice method II can work on larger values of ℓ_x compared to the lattice method I. This is consistent with theoretical analysis. For the last parameter set, it is easy to see that the attack time using lattice method I is slightly longer than using lattice method II. The reason here is that the lattice dimension in lattice method I and the size of elements in the lattice basis matrix are slightly larger.

Compared to the approach in Section 4, the approaches in Section 5 take less time to complete the attack, which is because that the attack in Section 4 uses enumeration rather than lattices. The reason why the success rate of attacks using lattice methods does not reach 100% is due to the heuristic nature of the lattice method.

8 Conclusion and future work

Two effective attacks have been proposed to recover the secret in the DReLU protocol. The secret of this protocol was obtained, which also compromised the security of other protocols in the Bicoptor family. In September 2023, the Huawei team announced a new family called Bicoptor 2.0 in ArXiv [19], which improved the probabilistic truncation function used in Bicoptor (see Lemma 2) to a deterministic truncation function. The core member DReLU of Bicoptor 2.0 was designed differently from the one in Bicoptor [20]. The main difference lies in the involvement of the modulo-switch technique and different linear operations. This results in the attack methods proposed in this article not being directly applicable. How to analyze the security of this new family is a future work worth researching.

References

1. Chaudhari, H., Choudhury, A., Patra, A., Suresh, A.: Astra: High throughput 3pc over rings with application to secure prediction. In: Proceedings of the 2019 ACM SIGSAC Conference on Cloud Computing Security Workshop. pp. 81–92 (2019)
2. Dalskov, A., Escudero, D., Keller, M.: Fantastic four:honest-majority four-party secure computation with malicious security. In: 30th USENIX Security Symposium (USENIX Security 21). pp. 2183–2200 (2021)
3. Escudero, D., Ghosh, S., Keller, M., Rachuri, R., Scholl, P.: Improved primitives for mpc over mixed arithmetic-binary circuits. In: Micciancio, D., Ristenpart, T. (eds.) Advances in Cryptology – CRYPTO 2020. pp. 823–852. Springer International Publishing, Cham (2020)

4. Juvekar, C., Vaikuntanathan, V., Chandrakasan, A.: Gazelle: A low latency framework for secure neural network inference. In: 27th USENIX Security Symposium. pp. 1651–1669 (2018)
5. Koti, N., Pancholi, M., Patra, A., Suresh, A.: Swift: Super-fast and robust privacy-preserving machine learning. In: 30th USENIX Security Symposium (USENIX Security 21). pp. 2651–2668 (2021)
6. Kumar, N., Rathee, M., Chandran, N., Gupta, D., Rastogi, A., Sharma, R.: Cryptflow: Secure tensorflow inference. In: 2020 IEEE Symposium on Security and Privacy (SP). pp. 336–353. IEEE (2020)
7. Lenstra, A.K., Lenstra, H.W., Lovász, L.: Factoring polynomials with rational coefficients. Mathematische annalen **261**(ARTICLE), 515–534 (1982)
8. May, A.: New RSA vulnerabilities using lattice reduction methods. Ph.D. thesis, Citeseer (2003)
9. Mohassel, P., Rindal, P.: Aby3: A mixed protocol framework for machine learning. In: Proceedings of the 2018 ACM SIGSAC conference on computer and communications security. pp. 35–52 (2018)
10. Mohassel, P., Zhang, Y.: Secureml: A system for scalable privacy-preserving machine learning. In: 2017 IEEE Symposium on Security and Privacy (SP). pp. 19–38 (2017). `https://doi.org/10.1109/SP.2017.12`
11. Nguyen, P.Q., Stehlé, D.: An LLL algorithm with quadratic complexity. SIAM Journal on Computing **39**(3), 874–903 (2009)
12. Patra, A., Schneider, T., Suresh, A., Yalame, H.: Aby2. 0: Improved mixed-protocol secure two-party computation. In: 30th USENIX Security Symposium (USENIX Security 21). pp. 2165–2182 (2021)
13. Rathee, D., Rathee, M., Kumar, N., Chandran, N., Gupta, D., Rastogi, A., Sharma, R.: Cryptflow2: Practical 2-party secure inference. In: Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security. pp. 325–342 (2020)
14. Riazi, M.S., Weinert, C., Tkachenko, O., Songhori, E.M., Schneider, T., Koushanfar, F.: Chameleon: A hybrid secure computation framework for machine learning applications. In: Proceedings of the 2018 on Asia conference on computer and communications security. pp. 707–721 (2018)
15. Srinivasan, W.Z., Akshayaram, P., Ada, P.R.: Delphi: A cryptographic inference service for neural networks. In: Proc. 29th USENIX Secur. Symp. pp. 2505–2522 (2019)
16. Tan, S., Knott, B., Tian, Y., Wu, D.J.: Cryptgpu: Fast privacy-preserving machine learning on the gpu. In: 2021 IEEE Symposium on Security and Privacy (SP). pp. 1021–1038. IEEE (2021)
17. Wagh, S., Gupta, D., Chandran, N.: Securenn: 3-party secure computation for neural network training. Proc. Priv. Enhancing Technol. **2019**(3), 26–49 (2019)
18. Wagh, S., Tople, S., Benhamouda, F., Kushilevitz, E., Mittal, P., Rabin, T.: Falcon: Honest-majority maliciously secure framework for private deep learning. Proceedings on Privacy Enhancing Technologies **2021**, 188–208 (01 2021). `https://doi.org/10.2478/popets-2021-0011`
19. Zhou, L., Song, Q., Zhang, S., Wang, Z., Wang, X., Li, Y.: Bicoptor 2.0: Addressing challenges in probabilistic truncation for enhanced privacy-preserving machine learning. ArXiv **abs/2309.04909** (2023), `https://api.semanticscholar.org/CorpusID:261682275`
20. Zhou, L., Wang, Z., Cui, H., Song, Q., Yu, Y.: Bicoptor: Two-round secure three-party non-linear computation without preprocessing for privacy-preserving ma-

chine learning. In: 2023 IEEE Symposium on Security and Privacy (SP). pp. 534–551 (2023). https://doi.org/10.1109/SP46215.2023.10179449

A Analysis of the greatest common divisor K

In this section, we analyze the case of $K = \gcd(3w_k, q)$. For the case that q is a prime or a power of 2, integers 3 and q are coprime. Hence $K = \gcd(w_k, q)$. Based on (26), $K = \gcd(r_* \cdot v_*, q)$, where r_* is a non-zero random number in $\mathbb{Z}_q$.

For a prime q, we have $K = 0$ or 1. The probability of $K = 0$ is equal to the probability of $v_* = 0$. By total probability theorem, we get that

$$\Pr(v_* = 0) = \Pr(t = 0) \cdot \Pr(v_* = 0 \mid t = 0) + \Pr(t = 1) \cdot \Pr(v_* = 0 \mid t = 1),$$

where $\Pr(\cdot)$ represents the probability of an event and $\Pr(\cdot \mid \cdot)$ means conditional probability. Due to t is a random bit, $\Pr(t = 0) = \Pr(t = 1) = 1/2$. According to (18), we deduce that $\Pr(v_* = 0 \mid t = 0) = \Pr(x' = 0)$ and $\Pr(v_* = 0 \mid t = 1) = \Pr(x' = 3^{-1} \cdot 2 \mod q)$. Furthermore, $\Pr(x' = 0) = 1/2^{\ell_x+1}$ and $\Pr(x' = 3^{-1} \cdot 2 \mod q) \leq 1/2^{\ell_x+1}$, based on $x' \in [0, 2^{\ell_x}) \cup (q - 2^{\ell_x}, q)$. Therefore, $\Pr(v_* = 0) \leq 1/2^{\ell_x}$. For example, for the parameter $\ell_x = 13$ in the DReLU algorithm, the term $1/2^{\ell_x}$ is negligible. In other words, $\Pr(K = 1) > 1 - 1/2^{\ell_x}$. It implies that, for a prime q, $K = 1$ with overwhelming probability.

For $q = 2^\ell$, based on $K = \gcd(r_* v_*, q)$, we get that K is also a power of 2. For the sake of discussion, we write $K = 2^\gamma$, where $0 \leq \gamma \leq \ell$. Note that $\gcd(r_* v_*, q) = 2^\gamma$ if and only if $\gcd(v_*, q) = 2^i$ and $\gcd(r_*, q) = 2^{\gamma - i}$ for $i = 0, 1, 2, \cdots, \gamma$. Hence, we have that

$$\Pr(\gcd(r_* v_*, q) = 2^\gamma) = \sum_{i=0}^{\gamma} \Pr(\gcd(v_*, q) = 2^i) \cdot \Pr(\gcd(r_*, q) = 2^{\gamma - i}). \quad (57)$$

First, we discuss the probability of $\gcd(v_*, q) = 2^i$. From (18), we have

$$v_* = \begin{cases} 3x' \mod q & \text{if } t = 0, \\ 3x' - 2 \mod q & \text{if } t = 1. \end{cases}$$

Therefore,

$$\gcd(v_*, q) = \begin{cases} \gcd(x', q) & \text{if } t = 0, \\ \gcd(3x' - 2, q) & \text{if } t = 1, \end{cases}$$

where $x' \in [0, 2^{\ell_x}) \cup (q - 2^{\ell_x}, q)$. Observe that $\Pr(\gcd(x', q) = 2^i) = \frac{2^{\ell_x - i}}{2^{\ell_x+1} - 1} \approx 2^{-(i+1)}$ for $0 \leq i \leq \ell_x$, and $\Pr(\gcd(x', q) = 2^i) = 0$ for $i > \ell_x$. Similarly, we can also deduce that $\Pr(\gcd(3x' - 2, q) = 2^i) \approx 2^{-(i+1)}$ for $0 \leq i \leq \ell_x$ and $\Pr(\gcd(3x' - 2, q) = 2^i) = 0$ for $i > \ell_x$. By total probability theorem, we obtain that $\Pr(\gcd(v_*, q) = 2^i)$ equals

$$\Pr(t = 0) \cdot \Pr(\gcd(x', q) = 2^i) + \Pr(t = 1) \cdot \Pr(\gcd(3x' - 2, q) = 2^i),$$

where $\Pr(t=0)=\Pr(t=1)=\frac{1}{2}$. It implies that

$$\begin{cases} \Pr(\gcd(v_*,q)=2^i)\approx 2^{-(i+1)} & \text{if } \ 0\le i\le \ell_x, \\ \Pr(\gcd(v_*,q)=2^i)=0 & \text{if } \ i>\ell_x. \end{cases} \tag{58}$$

Next, we discuss the probability of $\gcd(r_*,q)=2^{\gamma-i}$, where $0\le i\le\gamma$. Note that r_* is non-zero random number in $\mathbb{Z}_q$ and $q=2^\ell$. Hence,

$$\Pr(\gcd(r_*,q)=2^{\gamma-i})=\frac{2^{\ell+i-\gamma}}{q-1}\approx 2^{i-\gamma}. \tag{59}$$

Plugging (58) and (59) into (57), we have

$$\Pr(K=2^\gamma)=\begin{cases} \frac{\gamma}{2^{\gamma+1}}(1+o(1)) & \text{if } \ 0\le\gamma\le\ell_x, \\ \frac{\ell_x}{2^{\gamma+1}}(1+o(1)) & \text{if } \ \gamma>\ell_x. \end{cases}$$

From the above relation, we get that, for the case of $q=2^{\ell_x}$, the great common divisor K is a small positive integer with a high probability.

B Analysis of the greatest common divisor O

In this section, we analyze the case of $O=\gcd(O_1,O_2,q)$, where $O_1=3\cdot[w_k]_1 \mod q$ and $O_2=3\cdot[w_k]_0 \mod q$. From $([w_k]_0,[w_k]_1)=r_*\cdot([v_*]_0,[v_*]_1) \mod q$, $[v_*]_0=3[x']_0+(-1)^t-1 \mod q$, and $[v_*]_1=3[x']_1 \mod q$, we deduce $O_1=9r_*\cdot[x']_1 \mod q$. When q is a prime number or a power of 2, we always have $\gcd(O_1,q)=\gcd(r_*\cdot[x']_1,q)$.

For a prime q, note that r_* is a non-zero number in $\mathbb{Z}_q$, we have $\gcd(O_1,q)=\gcd([x']_1,q)$. Further, $\gcd(O_1,q)=0$ if $[x']_1=0$, and $\gcd(O_1,q)=1$ if $[x']_1\neq 0$. Because $[x']_1$ is a random number in $\mathbb{Z}_q$, we get that $\Pr(\gcd(O_1,q)=1)=1-\frac{1}{q}$. We observe $\Pr(\gcd(O_1,O_2,q)=1)\ge\Pr(\gcd(O_1,q)=1)$. Therefore, the probability that $\gcd(O_1,O_2,q)=1$ is at least $1-\frac{1}{q}$. For a sufficiently large q, $1-\frac{1}{q}$ is negligible. It means that $O=1$ for a prime q with an overwhelming probability.

For $q=2^\ell$, we have $O=\gcd(O_1,O_2,q)$ is a power of 2. For the sake of discussion, we write $O=2^{\Delta}$, where $0\le\Delta\le\ell$. Observe that $\Pr(\gcd(O_1,O_2,q)=2^{\Delta})\le\Pr(\gcd(O_1,q)\ge 2^{\Delta})$. From $\gcd(O_1,q)=\gcd(r_*\cdot[x']_1,q)$ and q is a power of 2, we deduce that $\gcd(O_1,q)=2^s$ if and only if $\gcd(r_*,q)=2^i$ as well as $\gcd([x']_1,q)=2^{s-i}$ for $i=0,1,\cdots,s$, where $\Delta\le s\le\ell$. Note that r_* and $[x']_1$ are independent of each other. Hence, we derive

$$\begin{aligned} \Pr((\gcd(O_1,q)=2^s) &= \sum_{i=0}^{s}\Pr((\gcd(r_*,q)=2^i)\cdot\Pr((\gcd([x']_1,q)=2^{s-i}) \\ &= \sum_{i=0}^{s}\frac{2^{\ell-i}-1}{2^{\ell+1}-2}\cdot\frac{2^{\ell-(s-i)}}{2^{\ell+1}}=\frac{1}{4}\cdot\frac{s+1}{2^s}(1+o(1)). \end{aligned}$$

It is not hard to see that $\Pr(\gcd(O_1, q) \geq 2^{\Delta}) = \sum_{s=\Delta}^{\ell} \Pr((\gcd(O_1, q) = 2^s)$ which is equivalent to $\frac{1}{4} \cdot (1 + o(1)) \cdot \sum_{s=\Delta}^{\ell} \frac{s+1}{2^s}$. Next, we compute $\sum_{s=\Delta}^{\ell} \frac{s+1}{2^s}$. For the sake of discussion, let $A := \sum_{s=\Delta}^{\ell} \frac{s}{2^s}$. Based on $\frac{1}{2} \cdot (A + \sum_{s=\Delta}^{\ell} \frac{1}{2^s}) = A + \frac{\ell+1}{2^{\ell+1}} - \frac{\Delta}{2^{\Delta}}$, we get $A = \sum_{s=\Delta}^{\ell} \frac{1}{2^s} + \frac{\Delta}{2^{\Delta-1}} - \frac{\ell+1}{2^{\ell}}$. Further, $\sum_{s=\Delta}^{\ell} \frac{s+1}{2^s} = A + \sum_{s=\Delta}^{\ell} \frac{1}{2^s} = 2 \sum_{s=\Delta}^{\ell} \frac{1}{2^s} + \frac{\Delta}{2^{\Delta-1}} - \frac{\ell+1}{2^{\ell}}$, which equals $\frac{\Delta+2}{2^{\Delta-1}} - \frac{\ell+3}{2^{\ell-1}}$. Therefore, we deduce $\Pr(\gcd(O_1, q) \geq 2^{\Delta}) = (\frac{\Delta+2}{2^{\Delta+1}} - \frac{\ell+3}{2^{\ell+1}})(1 + o(1))$. For a large ℓ, for example, $\ell = 64$, the term $\frac{\ell+3}{2^{\ell+1}}$ is is negligible. For a small Δ, for example, $\Delta = 6$, the term $\frac{\Delta+2}{2^{\Delta+1}}$ is sufficiently small (it is approximately 0.06 for $\Delta = 6$). From $\Pr(\gcd(O_1, O_2, q) = 2^{\Delta}) \leq \Pr(\gcd(O_1, q) \geq 2^{\Delta})$, we obtain that the probability of $\gcd(O_1, O_2, q) = 2^{\Delta}$ is sufficiently small for a small Δ. In other words, for the case of $q = 2^{\ell}$, the greatest common divisor O is a small positive integer with a high probability.