Secure Error Correction Using Multiparty Computation

Mohammad G. Raeini and Mehrdad Nojoumian
Department of Computer & Electrical Engineering and Computer Science
Florida Atlantic University, Boca Raton, FL, USA
{mghasemineja2017, mnojoumian}@fau.edu

***Abstract*—In the last couple of decades, error correction techniques play a prominent role in various scientific and engineering fields such as information theory, communication, networking, to name a few. These techniques are mainly utilized to locate and fix corrupted data over noisy channels. The data might be corrupted due to various reasons, for instance, communication failures, noise, or adversarial activities. On the other hand, data-privacy has been in the center of attention by many researchers in recent years. As such, it's important to be able to use error correction techniques over private data. This paper therefore proposes a *secure error correction* method by using secure multiparty computation (MPC). To the best of our knowledge, our proposed approach is the first solution in the literature. In secure MPC protocols, parties first share their private inputs by cryptographic primitives in order to jointly compute a function without revealing those private inputs. At the end of the protocol, only the function value will be revealed to all parties. Our secure MPC protocol efficiently implements the error-locator function of Berlekamp-Welch algorithm. After locating errors, we utilize another cryptographic technique, named *enrollment protocol*, to fix the errors.**

keywords: multiparty computation; error correcting codes; Reed-Solomon codes; Berlekamp-Welch algorithm.

I. Introduction

The secure multiparty computation is a distributed computation tool that allows a set of parties to compute a function over their private inputs without revealing any information about those inputs. Secure MPC was first introduced by Yao in his Millionaire's problem [1]. The problem states that two millionaires intend to know who is richer but they do not want to reveal the value of their assets. After Yao's paper, many researchers have conducted research on secure MPC and extended the original idea to a general purpose secure multiparty computation protocol.

Specifically, researchers focused on how they can evaluate a function over *n* pieces of private inputs, holding by *n* parties, without revealing those pieces. It has been shown that any function can be computed using secure multiparty computation. Two general approaches have been proposed in the literature to implement secure MPC, i.e., using boolean circuits or arithmetic circuits. In the former case, *n* parties share their data among themselves, for instance by Shamir's secret sharing [2], and then the parties compute the targeted function through boolean circuits. In this approach, the parties perform bit-wise calculations. As communication complexity is an important factor in secure multiparty computation, this approach is not efficient from that perspective. In the latter case, the parties use, for instance secret sharing, to share their data as finite field elements, and then perform function evaluation over their inputs. As this idea is based on polynomials and finite field calculations, it is more efficient in terms of communication and computational complexities.

In both scenarios, all parties need to communicate and share their data. On the other hand, adversaries may interrupt data communication or create noise while players are performing calculations. As such, if they can use an error detection and correction technique, they can increase the reliability of their protocols. In this paper, we propose a secure error correction technique that can be added to a secure multiparty computation for detecting errors that are caused by adversaries or communication channels. The idea is based on Berlekamp-Welch algorithm and we assume parties have created their shares based on Reed-Solomon codes. The reason that we use these two specific techniques is that, they are based on polynomials over finite fields that are efficient for implementation.

The rest of the paper is organized as follows. We first present the preliminaries materials on secure multiparty computation, Reed-Solomon codes, and their decoding using Berlekamp-Welch algorithm. Then the previous works are briefly reviewed. After that, the main part of this paper will be presented. Finally, technical discussions and concluding remarks are illustrated.

II. Related Works

The secure multiparty computation was first introduced by Yao [1] as the millionaires problem, in which two millionaires intend to determine who is richer without revealing anything about their assets. In other words, they want to determine whether $x > y$ or $x < y$ without revealing x and y, where x and y are their assets. Later the idea was extended to a more general case in which a group of players (or parties) can evaluate a function based on their private inputs without revealing those private values [3], [4], [5]. Many researchers have conducted research on secure multiparty computation and its applications [6], [7]. In this section we briefly review some of the related works in this domain.

Secure multiparty computation has a wide range of applications in sealed-bid auctions, privacy-preserving data mining, private information retrieval, machine learning, to name a few

[6], [8]. As the first set of fundamental applications, we can refer to *integer comparison* [9]; *equality test* [20], [19]; and *interval test* [19]. Note that these operation can be utilized to implement a general purpose secure MPC protocol for evaluating an arbitrary function on a set of private values, owned by a group of parties.

Other cryptographic applications consist of *joint signature* or *decryption schemes* in which a group of parties can sign some documents or decrypt a message whenever a specific number (which is known as threshold) of the parties are present [7]; *shared RSA keys* where some parties can collaborate to generate an RSA key that is shared among them [17]; and *Joint signature* or *decryption schemes* that can be utilized in financial cryptography contexts [7].

Finally, we can refer to electronic auctions with private bids, a.k.a, *sealed-bid auctions*, [11], [12]; *data aggregation* in IoT-enabled smart metering systems [13]; and *private set intersection* [14], [15], in which two parties can compute the intersection of two sets without revealing their set contents. Note that private set intersection has applications in other domains such as privacy-preserving data mining.

III. Preliminaries

In this section, we present some preliminary materials. These include secure multiparty computation (MPC), Reed-Solomon codes, called RS codes, Berlekamp-Welch decoding algorithm for RS codes, called BW algorithm.

A. Secure Multiparty Computation

Secure multiparty computation which its first idea was introduced by Yao [1] is defined as follows. n parties each have a private value and intend to evaluate a public function on their private data in such a way that they don't reveal any information about their private inputs but they all can get the output of the public function. One simple example is that n parties have n integer values and they intend to find the maximum value without revealing their private inputs.

Many scientists have conducted research in this area [19], [20], [22], [5], [21]. In secure multiparty computation, two approaches can be used. One approach performs computation based on bits of shared values, and the other approach, conducts computation based on shared values of secrets in a finite field Z_p, for a prime integer p, [19]. In both cases, secret sharing schemes, such as Shamir secret sharing [2], can be utilized to share secret values. However, both approaches have their own pros and cons, for example, conducting addition or multiplication is efficient in finite field arithmetic, but using boolean circuits it is not efficient anymore. Unfortunately, doing some calculations, such as secret comparisons, is not efficient and trivial in arithmetic circuits, whereas it is trivial in boolean circuit calculations [19]. However, for large integers this task is not efficient when we use boolean circuits.

To overcome the inefficiency of these two scenarios and having an efficient solution, the authors in [20] presented a protocol, called bit-decomposition, that allows parties to convert sharing of finite field elements to sharing of bits. In [19], the authors improved the bit-decomposition protocol by reducing its communication complexity. Another work in this area has been presented in [23] that is based on threshold homomorphic systems.

B. Reed-Solomon Codes

Reed-Solomon codes was introduced in 1960 in [24]. These error-correcting codes are based on polynomials over finite fields and have many applications. In the following discussions, we assume all calculations are done in finite field Z_p for a given prime number p. RS codes encode a message of length k into a codeword of length n, where $k \leq n \leq p$. Mathematically, given a message $m = [m_0, m_1, m_2, ..., m_{k-1}]$, the polynomial P is defined as follows:

$$P(x) = m_0 + m_1 x + m_2 x^2 + ... + m_{k-1} x^{k-1} \tag{1}$$

In which the coefficients are in Z_p. To encode the message m, the polynomial will be evaluated on n different points, say $1, 2, ..., n$, so the encoded message, denoted by c would be:

$$c = [c_0, c_1, c_2, ..., c_{n-1}] = [P(1), P(2), P(3), ..., P(n)] \tag{2}$$

For the Reed-Solomon codes we have the following theorems:

Theorem 1. *The weight of Reed-Solomon codes of length n with a message of length k, $RS(n,k)$, is $n-k+1$.*

Theorem 2. *An $RS(n,k)$ with $n = k + 2e$ can correct e errors.*

A decoding algorithm was developed by Berlekamp and Welch in [25], which we will discuss it in the next section.

C. Berlekamp-Welch Decoding Algorithm

Berlekamp-Welch algorithm is a decoding algorithm for RS codes [25]. As we discussed in the previous section, an $RS(n,k)$ code encodes a message with length k to a codeword of length n. Now, assume that e errors has happened in the codeword $c = [c_0, c_1, c_2, ..., c_{n-1}] = [P(1), P(2), P(3), ..., P(n)]$ and the codeword with errors is as follows: $r = [r_0, r_1, r_2, ..., r_{n-1}]$. That is, $r_i \neq P(i)$ for at most e cases.

Theorem 3. *Given a received codeword, generated by RS(n, k) codes, with e errors, then there exists non-zero polynomials $E(x)$ and $Q(x)$ for which we have [26]:*

$$degree(E(x)) \leq e \tag{3}$$

$$degree(Q(x)) \leq k + e - 1 \tag{4}$$

$$Q(i) = r_i E(i) \qquad \forall i = 1, 2, ..., n. \tag{5}$$

Moreover, $\frac{Q(x)}{E(x)}$ will give the polynomial that has generated the codeword, $P(x)$.

Equation 5 is called key equation and solving it gives us the location of the errors in the received codeword, which is guaranteed by theorem 3.

For $i = 1, 2, ..., n$, key equation, equation 5, will produce a system of equations, which we can solve it by different methods in linear algebra, such as Gaussian elimination or Cramer's rule. By finding the solution of the key equation, we can find the locations of the errors, and accordingly, the polynomial $P(x)$ that gives the corrected message.

IV. Secure Error Detection and Correction Using Multiparty Computation

We assume that n parties have n shares and they want to be able to check if any errors has occurred in their data, because in secure multiparty computations, parties constantly exchange shares of their private inputs. We also assume that all the following calculations are done in finite field Z_p where p is a prime number.

In order to be able to detect and correct e errors, we need to have at least $3e+1$ shares. In other words, $3e+1$ parties need to participate. This is due to theorem 2: $n \geq k + 2e$ where k is the message length. Also, we assume that, the number of errors is less than the message length. That is, the entire message has not been altered. In the following section, we will address the problem of error detection, and subsequently, we provide a technique that allows the parties to recover the incorrect shares.

A. Locating One Error at a Time

Each player creates an equation using his secret value (which is denoted by α_i for player i).

$$\sum_{i=0}^{n-2} a_i x^i = \alpha_i (x + b_0) \tag{6}$$

Therefore, we have n equations, each in the hand of one party, by which we can define the following system of equations (consisting of n equations and n unknowns including $a_0, a_1, a_2, ..., a_{n-2}, b_0$).

$$\begin{cases} \sum_{i=0}^{n-2} a_i x^i = \alpha_1 (x + b_0) \\ \sum_{i=0}^{n-2} a_i x^i = \alpha_2 (x + b_0) \\ \qquad ... \\ \sum_{i=0}^{n-2} a_i x^i = \alpha_n (x + b_0) \end{cases} \tag{7}$$

$a_0, a_1, a_2, ..., a_{n-2}$ will be used for the error correction polynomial $Q(x)$ in the BW algorithm and b_0 is error locator as the $E(x)$ polynomial in the BW algorithm. Also, we assume that all calculations are done in Z_p for a public and predefined prime number p. Now, the players evaluate equations with $x = 1, 2, 3, ..., n$, similar to the BW algorithm. As a result, they have:

$$\begin{cases} \sum_{i=0}^{n-2} 1^i a_i = \alpha_1 (1 + b_0) \\ \sum_{i=0}^{n-2} 2^i a_i = \alpha_2 (2 + b_0) \\ \qquad ... \\ \sum_{i=0}^{n-2} n^i a_i = \alpha_n (n + b_0) \end{cases} \tag{8}$$

For the sake of simplicity, we use matrix notation to demonstrate this system of equations:

$$Ax = b \tag{9}$$

Where

$$A = \begin{bmatrix} 1 & 1 & \dots & 1^{n-2} & -\alpha_1 \\ 1 & 2 & \dots & 2^{n-2} & -\alpha_2 \\ 1 & 3 & \dots & 3^{n-2} & -\alpha_3 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 1 & n-1 & \dots & (n-1)^{n-2} & -\alpha_{n-1} \\ 1 & n & \dots & n^{n-2} & -\alpha_n \end{bmatrix} \tag{10}$$

$$x = \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ \vdots \\ a_{n-2} \\ b_0 \end{bmatrix} \quad \text{and} \quad b = \begin{bmatrix} \alpha_1 \\ 2\alpha_2 \\ 3\alpha_3 \\ \vdots \\ (n-1)\alpha_{n-1} \\ n\alpha_n \end{bmatrix} \tag{11}$$

The first $n-2$ columns of the first matrix are public values, which are the same as the columns of the Vandermone matrix. If we use Cramer's rule for solving this system of equations (just for b_0, which determines the location of the error), we will have:

$$b_0 = \frac{det(A_1)}{det(A_2)} \tag{12}$$

where

$$A_1 = \begin{bmatrix} 1 & 1 & \dots & 1^{n-2} & \alpha_1 \\ 1 & 2 & \dots & 2^{n-2} & 2\alpha_2 \\ 1 & 3 & \dots & 3^{n-2} & 3\alpha_3 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 1 & n-1 & \dots & (n-1)^{n-2} & (n-1)\alpha_{n-1} \\ 1 & n & \dots & n^{n-2} & n\alpha_n \end{bmatrix} \tag{13}$$

and

$$A_2 = \begin{bmatrix} 1 & 1 & \dots & 1^{n-2} & -\alpha_1 \\ 1 & 2 & \dots & 2^{n-2} & -\alpha_2 \\ 1 & 3 & \dots & 3^{n-2} & -\alpha_3 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 1 & n-1 & \dots & (n-1)^{n-2} & -\alpha_{n-1} \\ 1 & n & \dots & n^{n-2} & -\alpha_n \end{bmatrix} \tag{14}$$

If we expand the determinant based on the last column, we will have:

$$d_1 = det(A_1) = \sum_{i=1}^{n} (-1)^{i+n} (-\alpha_i) det(A_1^{i,n}) \tag{15}$$

and

$$d_2 = det(A_2) = \sum_{i=1}^{n} (-1)^{i+n} (\alpha_i) det(A_2^{i,n}) \tag{16}$$

where $A^{i,n}$ is the $(n-1) \times (n-1)$ matrix that is created by eliminating the i-th row and n-th column of A. As shown in the above equation, just α_i's are secret values and the second term in the summation is a public value. Therefore, to calculate d_1 and d_2, the players just need to locally multiply their secret values by a public term and send shares of the result to other players, and finally, they perform Lagrange interpolation to find the location of the error. The error detection algorithm is as following, see algorithm 1.

Algorithm 1 Error Detection Protocol

```
1: Each player defines his own equation (with his public ID $i$ and his secret value $\alpha_i$), as follows:
       $$Q(i) = \alpha_i E(i) \quad (17)$$
2: All the players put their public part of their equations in a matrix. They also put $*$ in the last column that is the private value of each player. Note that, in the next step during the Gaussian expansion, this will be eliminated:
       $$A = \begin{bmatrix} 1 & 1 & \dots & 1^{n-2} & * \\ 1 & 2 & \dots & 2^{n-2} & * \\ 1 & 3 & \dots & 3^{n-2} & * \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 1 & n-1 & \dots & (n-1)^{n-2} & * \\ 1 & n & \dots & n^{n-2} & * \end{bmatrix} \quad (18)$$
3: One of the players accepts to calculate $det(A_1^{i,n})$ and $det(A_2^{i,n})$, from $A$ matrix. $A^{i,n}$ means the $(n-1)$ by $(b-1)$ matrix that has been created from $A$ by eliminating its $i$-th row and $n$-th column. After calculation, this player hands out $det(A_1^{i,n})$ and $det(A_2^{i,n})$ to player $i$.
4: Subsequently, each player, who just received his related terms in equations 15 and 16, calculates the multiplication of his private value by the received term locally, we call the result value $det_i$ for player $i$.
5: Each player, shares his $det_i$ between all other players.
6: Each player adds up his received shares, denoted by $s_i$, i.e., $s_i = \sum_{j=1}^{n} det_j$.
7: Finally, players perform Lagrange interpolation on their $s_i$ and get the $d_1$ and $d_2$, as in equation 12. Note that, for $d_1$ and $d_2$, the players need to multiply their private values by the corresponding terms, as defined in equations 15 and 16.
```

B. Correcting One Error at a Time

For the error correction, we use the proposed approach of [28], [29], named *enrollment protocol*. In this paper, the authors provided a new technique, based on Lagrange interpolation, for recovering incorrect shares. After determining the exact location of error, the other parties (who have correct shares) help the player with incorrect share to recover a correct share. The whole idea is that, each player calculates his Lagrange interpolation constant and multiplies it by his share, and then, he sends the shares of the result value to all players. When all players accomplish this phase, they will have portions of the corrected share. By these portions, the player with incorrect share can correct his share. The error correction protocol, algorithm 2, shows the step-by-step process as follows:

Algorithm 2 Error Correction/Enrollment Protocol

```
1: Each party $i$ calculates his Lagrange interpolation constant as following:
       $$\gamma_i = \prod_{1 \le j \le t, i \ne j} \left(\frac{k-j}{i-j}\right) \quad (19)$$
2: Then, each player multiplies his secret value $\alpha_i$ by his Lagrange interpolation constant $\gamma_i$ and splits it into $t$ portions, where $t$ is threshold in secret sharing.
       $$\alpha_i \times \gamma_i = \delta_{1i} + \delta_{2i} + \dots \delta_{ti}. \quad (20)$$
   Next, he sends each portion to one of $t$ players.
3: Each party $j$ receives $t$ portions in total and adds them up as follows:
       $$\sigma_j = \sum_{i=1}^{t} \delta_{ji}. \quad (21)$$
   which $\delta_{ji}$ means the portion that participant $i$ has sent to participant $j$.
4: Party $j$ then sends $\sigma_j$ to party $k$, whose share is corrupted or altered. $t$ players need to send these partial values to player $k$ so that he can recover his correct share.
5: Party $k$ adds up all the $t$ received partial values. As a result, his correct share will be recovered:
       $$\alpha_k^{corrected} = \sum_{j=1}^{t} \sigma_j. \quad (22)$$
```

V. Technical Discussion

In this section we discuss the security analysis as well as computational and communication complexity of the error detection and correction protocols. Note that detection and recovery techniques can be designed based on verifiable secret sharing as shown in [27]. However, in those settings, parties' private inputs are polynomials over finite field rather than single field elements.

A. Security Analysis of Error Correction Protocol

As stated earlier, secure multiparty computation is a distributed setting by which a set of parties can evaluate a function of their secret values without compromising the privacy of their input data. This can be achieved by secret sharing schemes such as Shamir's (t, n)-threshold secret sharing. The security of this scheme is based on the fact that players' secret or private data has been shared among players by adding some randomness using a polynomial of degree $t-1$ with random coefficient. In fact, each secret value is hidden by putting the secret value as the constant term of such a polynomial and defining other coefficients at random. After completing the secure multiparty computation, at least t players need to participate in order to get the final result of computation, which is done using polynomial interpolation. This is similar to (t, n)-threshold secret sharing, in which a group of t players need to cooperate to recover the secret value.

In our secure error correction protocol, t players need to come together to detect and correct an error. To accomplish this, they first create a system of equations. They will then represent it in a matrix format, i.e., $Ax = b$: only one column of matrix A and also vector b consist of secret values of players. As they calculate the determinant of the matrix through Gaussian expansion based on the column containing the secret values, the new sub-matrices are all containing public values, which their determinant will be calculated and distributed among all players by a volunteer player. Subsequently, each player needs to calculate multiplication of his secret value by a public value. Although the player can accomplish such a multiplication by using the multiplication gate of secure MPC, he can also do it locally by utilizing the addition gate of MPC. This is because one value is secret and one value is public. Note that all calculations are done in a finite field such as Z_p for a prime p. After players calculate their corresponding shares in equations 15 and 16, they can then calculate the summation by Lagrange interpolation, in which, at least, t players must participate. This will ensure the security of the error detection protocol.

B. Computational Complexity Discussion

In the error detection protocol, the players construct a joint system of equations, which is distributed among players. In fact, each player has one equation in which his secret value is used according to equation 6. To find the location of the error, they need to calculate d_1 and d_2. They are the determinants of A_1 and A_2 matrices (equations 13 and 14), which have been resulted from the matrix of coefficients. As we know, the worst-case computational complexity of calculating the determinant of a n by n matrix is $O(n^3)$, where n is the number of players.

In fact, the players first expand the determinant according to the equation 15, and then, a volunteer player calculates the determinant of $(n-1)$ by $(n-1)$ matrices, which are public values. After calculating d_1 and d_2 determinants, they calculate the b_0 using a division operation in the finite field, which is basically a multiplication gate.

For the error correction protocol, the players calculate the Lagrange interpolation constant (according to equation 19), which is of $O(t^2)$, where t is the threshold of the Shamir's (t, n)-threshold secret sharing. We know that $t \leq n$. The remaining steps in error correction protocol can be done in $O(t)$, see equations 20, 21, 22.

C. Round Complexity Discussion

Generally speaking, one of the drawbacks of secure MPC is its communication complexity, which is due to its distributed nature or multiparty setting. As mentioned earlier, secure MPC protocols utilize standard operations, i.e., logical AND and OR gates, or addition and multiplication gates in a finite field. In the former case, the communication complexity is far more than the latter case. Comparing two gates of the arithmetic circuit (addition and multiplication), the multiplication gate has a larger complexity than the addition gate. Because in multiplication gate, the parties need to execute a degree reduction protocol, which requires re-sharing and a lot of communications. As our proposed protocol is based on the addition gate, its communication complexity is very efficient.

For error detection, players calculate two determinants (d_1 according to equation 15 and d_2 according to equation 16) in which they only do multiplication by some public values. As mentioned before, multiplication by public values can be easily done using the addition gate. In other words, the players can reduce the communication complexity of the error correction protocol by using the addition gate; unlike multiplication gate it requires no re-sharing or communications. They then share their corresponding determinants, algorithm 1. After that, they do a division in a finite field (which is basically a multiplication in the finite field). Finally, they get the error location by doing Lagrange interpolation collaboratively. Overall, they go through 3 rounds of communication, one for sharing their corresponding determinants and two rounds for doing the division.

For error correction, according to the algorithm 2, one round of communication is needed in step 2 and one round in step 3. That is, two rounds of communication in total.

VI. Step-By-Step Example

Assume our selected prime number is $p = 7$, and the players' ID-s are $1, 2, 3, 4$. Also, the private values of four parties are $2, 0, 5, 3$, and the third value has changed to 4 because of an error. As a result, the codeword is $[c_0, c_1, c_2, c_3] = [2, 0, 5, 3]$, and the received vector, as described in BW algorithm, is $[r_0, r_1, r_2, r_3] = [\alpha_0, \alpha_1, \alpha_2, \alpha_3] = [2, 0, 4, 3]$. The key equation of BW algorithm will be as follows:

$$a_0 + a_1 x + a_2 x^2 = \alpha_i (x + b_0) \tag{23}$$

By evaluating this equation at $x = 1, 2, 3, 4$ (here each x is the ID of a player and each equation is at the hand of one player), we will have:

$$\begin{cases} a_0 + a_1 + a_2 = 2(1 + b_0) \\ a_0 + 2a_1 + 4a_2 = 0(2 + b_0) \\ a_0 + 3a_1 + 2a_2 = 4(3 + b_0) \\ a_0 + 4a_1 + 2a_2 = 3(4 + b_0) \end{cases} \tag{24}$$

In fact, each player creates his equation by using his ID, which is a public value. This system of equations can be written as, after doing all calculation in Z_7:

$$\begin{cases} a_0 + a_1 + a_2 + 5b_0 = 2 \\ a_0 + 2a_1 + 4a_2 + 0b_0 = 0 \\ a_0 + 3a_1 + 2a_2 + 3b_0 = 5 \\ a_0 + 4a_1 + 2a_2 + 4b_0 = 5 \end{cases} \tag{25}$$

To find the location of the error, we need to find b_0 as follows, which was explained earlier in section IV:

$$b_0 = \frac{det(A_1)}{det(A_2)} \tag{26}$$

where

$$A_1 = \begin{bmatrix} 1 & 1 & 1 & 2 \\ 1 & 2 & 4 & 0 \\ 1 & 3 & 2 & 5 \\ 1 & 4 & 2 & 5 \end{bmatrix} \quad \text{and} \quad A_2 = \begin{bmatrix} 1 & 1 & 1 & 5 \\ 1 & 2 & 4 & 0 \\ 1 & 3 & 2 & 3 \\ 1 & 4 & 2 & 4 \end{bmatrix} \qquad (27)$$

For example, d_1 can be expanded as follows:

$$\begin{aligned} d_1 &= 2(-1)^{1+4} \begin{vmatrix} 1 & 2 & 4 \\ 1 & 3 & 2 \\ 1 & 4 & 2 \end{vmatrix} + 0(-1)^{2+4} \begin{vmatrix} 1 & 1 & 1 \\ 1 & 3 & 2 \\ 1 & 4 & 2 \end{vmatrix} \\ &+ 5(-1)^{3+4} \begin{vmatrix} 1 & 1 & 1 \\ 1 & 2 & 4 \\ 1 & 4 & 2 \end{vmatrix} + 5(-1)^{4+4} \begin{vmatrix} 1 & 1 & 1 \\ 1 & 2 & 4 \\ 1 & 3 & 2 \end{vmatrix} \\ &= 2 \times 5 + 0 \times 6 + 5 \times 1 + 5 \times 2 = 4 \end{aligned} \qquad (28)$$

Similarly, we can calculate d_2, which is equal to 1. After conducting the calculation, we will have $b_0 = 4$, which means the location of the error is 3. Because the error locator polynomial is $x + b_0 = x + 4$, and its root shows the location of the error; its root in Z_7 is 3. Note that the last columns of A_1 and A_2 are consisted of a multiplicative factor, which is the private values of parties. Therefore, the determinant calculation should be expanded based on that column, and after the expansion, the players can use any method to calculate the determinant of the resulting 3×3 matrix.

VII. Concluding Remarks and Future Work

In this paper, we introduced the notion of *secure error correction* that allows a group of parties to detect and correct potential errors in a vector representing their private inputs without revealing any information about those inputs. To the best of our knowledge, our proposed approach is the first solution in the literature. Our idea was based on the Reed-Solomon codes and Berlekamp-Welch decoding algorithm. In other words, we utilized the Berlekamp-Welch algorithm to determine the location of the errors in a privacy-preserving fashion, and then, we used another cryptographic primitive, named *enrollment protocol*, to fix the error.

Our secure MPC protocol efficiently implements the error-locator function of Berlekamp-Welch algorithm by using only addition gates. Note that detection and recovery techniques can be designed based on verifiable secret sharing as shown in [27]. However, in those settings, parties' private inputs are polynomials over finite field rather than single field elements. As our future work, we intend to extend our secure error correction approach to build self-healing MPC protocols that are ideal, i.e., the size of shares is equal to the size of secret.

References

[1] A. Yao, Protocols for Secure Computation, In Proc. 23rd Annual Symp. on Foundations of Computer Science (FOCS), pages 160164. IEEE, 1982.

[2] A. Shamir, How to share a secret, Commun. ACM, 22(11):612613, 1979.

[3] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or a completeness theorem for protocols with honest majority, 19th ACM Symp. on Theory of Computing, pp. 218-229, 1987.

[4] David Chaum, Claude Crepeau, and Ivan Damgard. Multiparty unconditionally secure protocols, In 20th Annual ACM Symposium on Theory of Computing, STOC88, pages 11-19, 1988.

[5] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation, 20th ACM Symposium on Theory of Computing, pp. 1-10, 1988.

[6] Wenliang Du and Mikhail J. Atallah. Secure multi-party computation problems and their applications: a review and open problems. In Workshop on New Security Paradigms, NSPW01, pages 13-22. ACM, 2001.

[7] Shafi Goldwasser. Multi party computations: past and present. In 16th Annual ACM Symposium on Principles of Distributed Computing, PODC97, pages 1-6, 1997

[8] Y. Lindell and B. Pinkas. Secure multiparty computation for privacy-preserving data mining. J. of Privacy and Confidentiality, 1(1):5998, 2009.

[9] J. A. Garay, B. Schoenmakers and J. Villegas. Practical and secure solutions for integer comparison, 10th International Conference on Practice and Theory in Public-Key Cryptography, LNCS 4450, pp. 330-342, 2007.

[10] J. Herranz and G. Saez. Verifiable secret sharing for general access structures, with application to fully distributed proxy signatures. Proceedings of Financial Cryptography, LNCS 2742, pp. 286302 (2003).

[11] Michael Harkavy, J. D. Tygar, and Hiroaki Kikuchi. Electronic auctions with private bids. In 3rd Conference on USENIX Workshop on Electronic Commerce, WOEC98, pages 61-74. USENIX Association, 1998.

[12] M. Nojoumian and D. R. Stinson, Efficient Sealed-Bid Auction Protocols Using Verifiable Secret Sharing. 10th International Conference on Information Security Practice and Experience (ISPEC), Springer LNCS 8434, pp. 302-317, Fuzhou, China, 2014.

[13] S. Tonyali, K. Akkaya, N. Saputro, A. S. Uluagac and M. Nojoumian, Privacy-Preserving Protocols for Secure and Reliable Data Aggregation in IoT-Enabled Smart Metering Systems. Future Generation Computer Systems (FGCS), Elsevier, vol. 78, part 2, pp. 547-557, 2018.

[14] E. De Cristofaro and G. Tsudik. Practical private set intersection protocols with linear complexity. In Financial Cryptography and Data Security (FC10), volume 6052 of LNCS, pages 143159. Springer, 2010.

[15] Y. Huang, D. Evans, and J. Katz. Private set intersection: Are garbled circuits better than custom protocols? In Network and Distributed Security Symposium (NDSS12). The Internet Society, 2012.

[16] Chor, B., Goldwasser, S., Micali, S., and Awerbuch, B. Verifiable secret sharing and achieving simultaneity in the presence of faults. In FOCS (1985), IEEE, pp. 383-395.

[17] Dan Boneh and Matthew Franklin. Efficient generation of shared RSA keys. Journal of ACM, 48(4):702-722, 2001.

[18] Liu X, Li S, Liu J, Chen X, Xu G. Secure multiparty computation of a comparison problem, SpringerPlus, 2016, 5(1):1489. doi:10.1186/s40064-016-3061-0.

[19] T. Nishide and K. Ohta, Multiparty computation for interval, equality, and comparison without bit-decomposition protocol, in Proc. 2007 PKC, pp. 343-360.

[20] I. Damgard, M. Fitzi, E. Kiltz, J.B. Nielsen, and T. Toft, Unconditionally secure constant-rounds multi-party computation for equality, comparison, bits and exponentiation, Proc. 3rd Theory of Cryptography Conference, LNCS 3876, pp.285-304, Springer Verlag, 2006.

[21] O. Goldreich, S. Micali, and A. Wigderson, How to play any mental game or a complete theorem for protocols with honest majority, Proc. 19th STOC, pp.218 229, 1987.

[22] D. Chaum, C. Crepeau, and I. Damgard, Multi-party unconditionally secure protocols, Proc. ACM STOC88, pp.1119, 1988.

[23] 21. B. Schoenmakers and P. Tuyls, Efficient binary conversion for Paillier encrypted values, EUROCRYPT06, LNCS 4004, pp.522537, Springer Verlag, 2006.

[24] I. S. Reed and G. Solomon, Polynomial codes over certain finite fields, 1. SIAM, vol 8, no 2, June 1960, pp 300-304

[25] Lloyd R. Welch and Elwyn R. Berlekamp. Error correction for algebraic block codes, December 30 1986. US Patent 4,633,470.

[26] https://math.berkeley.edu/ mhaiman/math55/reed-solomon.pdf

[27] M. Nojoumian, Unconditionally Secure Proactive Verifiable Secret Sharing Using New Detection and Recovery Techniques. 14th IEEE Annual Conference on Privacy, Security and Trust (PST), pp. 269-274, Auckland, New Zealand, 2016.

[28] M. Nojoumian, D. R. Stinson and M. Grainger, Unconditionally secure social secret sharing scheme. IET Information Security (IFS), Special Issue on Multi-Agent and Distributed Information Security, vol. 4, issue 4, pp. 202-211, 2010.

[29] M. Nojoumian and D. R. Stinson, Brief Announcement: Secret Sharing Based on the Social Behaviors of Players. 29th ACM Symposium on Principles of Distributed Computing (PODC), pp. 239-240, Zurich, Switzerland, 2010.