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**

[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 $k \quad n \quad p$. Mathematically, given a message $m = [m_0; m_1; m_2; :::; m_{k\ \ 1}]$, the polynomial P is defined as follows:

$$P(x) = m_0 + m_1 x + m_2 x^2 + ::: + m_{k\ \ 1} x^{k\ \ 1} \qquad (1)$$

In which the coefficients are in Z_p. To encode the message m, the polynomial will be evaluated on n

For $i = 1, 2, \ldots, n$, 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).

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

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 $a_0, a_1, a_2, \ldots, a_{n-2}, b_0$).

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

$a_0, a_1, a_2, \ldots, a_{n-2}$ 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 $x = 1, 2, 3, \ldots, n$, similar to the BW algorithm. As a result, they have:

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

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

$$Ax = b \qquad (9)$$

Where

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

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

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:

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

where

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

and

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

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

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

and

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

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

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 $\ _i$), as follows:

$$Q(i) = \ _iE(i) \tag{17}$$

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:

$$A = \begin{bmatrix} 1 & 1 & \dots & 1^{n\ \ 2} & \\ 1 & 2 & \dots & 2^{n\ \ 2} & \\ 1 & 3 & \dots & 3^{n\ \ 2} & \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 1 & n\ \ 1 & \dots & (n\ \ 1)^{n\ \ 2} & \\ 1 & n & \dots & n^{n\ \ 2} & \end{bmatrix} \tag{18}$$

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 5164e626r()]TJ/F7 6.9738 Tf 6.227 0 Td [(2 6.1350(all)n)-431(,8 Tf 4.67 -1.46 Tf [(d)]TJ/F7

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