Secure distributed computation

Stanislav Krikunov [1] **Mariia Kopylova** [2] **Alexander Blagodarnyi** [1] **Mikhail Konovalov** [1]

Abstract

In our time the usage of distributed computing is the standard for solving of modern engineering problems. One the of the most popular tasks is the solving of matrix multiplication. In our case the user has two matrices A and B and wants to compute their product AB with the assistance of N servers, without leaking any information about A or B to any server. We suppose that all servers have the necessary properties such as honesty and responsibility, but that they are curious, in that any T of them may collude to try to deduce information about either A or B. The user also wants to optimize the download rate R. For such problem we can use one of the most modern method and in this project we will implement the prototype of GASP codes.

The article (Rafael G. L. D'Oliveira, 2020) was used as state-of-the-art pattern.
Source code is provided on github (url).

1. Introduction

The problem of constructing polynomial codes for Secure Distributed Matrix Multiplication (SDMM) can be summarized as follows. We partition the matrices A and B as follows:

$$A = \begin{bmatrix} A_1 \\ \vdots \\ A_K \end{bmatrix}, \quad B = \begin{bmatrix} B_1 & \cdots & B_L \end{bmatrix}, \text{ so that}$$

$$AB = \begin{bmatrix} A_1B_1 & \cdots & A_1B_L \\ \vdots & \ddots & \vdots \\ A_KB_1 & \cdots & A_KB_L \end{bmatrix}$$

making sure that all products A_kB_l are well-defined and of the same size.

[1] 1st year MSc student in Skolkovo Institute of Science and Technology, Moscow, Russia. [2] 2nd year MSc student in Skolkovo Institute of Science and Technology, Moscow, Russia. Correspondence to: Mariia Kopylova <Mariia.Kopylova@skoltech.ru>.

Final Project of Introduction to Blockchain course, Skoltech, Moscow, Russian Federation, 2020, December.

Clearly, computing the product AB is equivalent to computing all subproducts A_kB_k. One then constructs a polynomial $h(x)$ whose coefficients encode the submatrices A_kB_l, and has N servers compute evaluations $h(a1), \ldots, h(aN)$. The polynomial h is constructed so that every T-subset of evaluations reveals no information about A or B, but so that the user can reconstruct all of AB given all N evaluations.

One can view the parameters K and L as controlling the complexity of the matrix multiplication operations the servers must perform. Imagine a scenario in which one may hire as many servers N as one wants to assist in the SDMM computation, but the computational capacity of each server is limited. In this scenario, one may have fixed values of K and L, and then maximizing the rate R becomes a question of minimizing N. This is the general perspective we adopt in the SDMM problem.

2. Preliminaries

2.1. Polynomial Codes

Within the framework of this project we consider A and B be matrices over a finite field F_q, selected by a user independently and uniformly at random from the set of all matrices of their respective sizes so that all products A_kB_l are well-defined and of the same size. Then AB is the block matrix $AB = (A_kB_l)_{1 \le k \le K, 1 \le l \le L}$. A polynomial code is a tool for computing the product AB in a distributed manner, by computing each block A_kB_l. Formally, we define a polynomial code as follows.

The polynomial code PC(K, L, T, N, α ; β) consists of the following data:

- Positive integers K, L, T, N
- $\alpha = (\alpha_1, \ldots, \alpha_{K+T}) \in N^{K+T}$
- $\beta = (\beta_1, \ldots, \beta_{L+T}) \in N^{L+T}$

A polynomial code PC(K, L, T, N, α ; β) is used to securely compute the product AB. The user chooses T matrices R_t over F_q of the same size as the A_k independently and uniformly at random, and T matrices S_t of the same size as the B_l independently and uniformly at random. They define polynomials $f(x)$ and $g(x)$ by

$$f(x) = \sum_{k=1}^{K} A_k x^{\alpha_k} + \sum_{t=1}^{T} R_t x^{\alpha_{K+t}}$$

and

$$g(x) = \sum_{\ell=1}^{L} B_\ell x^{\beta_\ell} + \sum_{t=1}^{T} S_t x^{\beta_{L+t}}$$

let $h(x) = f(x)h(x)$.

Given N servers, a user chooses evaluation points $a_1, \ldots, a_N F_q^r$ in some finite extension F_q^r of F_q. They then send $f(a_n)$ and $g(a_n)$ to server $n = 1, \ldots, N$, who computes the product $f(a_n)g(a_n) = h(a_n)$ and transmits it back to the user. The user then interpolates the polynomial $h(x)$ given all of the evaluations $h(a_n)$, and attempts to recover all products $A_k B_l$ from the coefficients of $h(x)$. We omit the evaluation vector a from the notation PC(K, L, T, N, α ; β) because as we will shortly show, it does not really affect any important analysis of the polynomial code.

2.2. The degree table

For $\alpha \in N^{K+T}$ and $\beta \in N^{L+T}$ we consider the outer sum $\alpha \oplus \beta \in N^{(K+T)} \times (L+T)$ of α and β is defined to be the matrix

$$\alpha \oplus \beta = \begin{bmatrix} \alpha_1 + \beta_1 & \cdots & \alpha_1 + \beta_{L+T} \\ \vdots & \ddots & \vdots \\ \alpha_{K+T} + \beta_1 & \cdots & \alpha_{K+T} + \beta_{L+T} \end{bmatrix}$$

For A we can say that it is a matrix with entries in N. We define the terms of A as a set

$$termsA = \{n \in N : \exists (i,j), A_{ij} = n\}$$

Consider the polynomial code PC(K, L, T, N, α ; β), with associated polynomials

$$f(x) = \sum_{k=1}^{K} A_k x^{\alpha_k} \quad , \quad g(x) = \sum_{\ell=1}^{L} B_\ell x^{\beta_\ell}$$

Then we can express the product $h(x)$ of $f(x)$ and $g(x)$ as $h(x) = f(x)g(x) = \sum_{j \in \mathcal{J}} C_j x^j$, for some matrices $_j$, where $\mathcal{J} = terms(\alpha \oplus \beta)$. Thus, the terms in the outer sum α and β correspond to the terms in the polynomial $h(x) = f(x)h(x)$. Because of this we refer to the table representation of $(\alpha \oplus \beta)$ in Table I. Each $\alpha_k + \beta_l$ in the central block must be distinct from every other entry in $(\alpha \oplus \beta)$. The condition of T – security states that all α_{K+t} in the lower-left block must be pairwise distinct, and all β_{L+t} in the upper-right block must be pairwise distinct.

	β_1	$\cdots$	β_L	β_{L+1}	$\cdots$	β_{L+T}
α_1	$\alpha_1+\beta_1$	$\cdots$	$\alpha_1+\beta_L$	$\alpha_1+\beta_{L+1}$	$\cdots$	$\alpha_1+\beta_{L+T}$
$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
α_K	$\alpha_K+\beta_1$	$\cdots$	$\alpha_K+\beta_L$	$\alpha_K+\beta_{L+1}$	$\cdots$	$\alpha_K+\beta_{L+T}$
α_{K+1}	$\alpha_{K+1}+\beta_1$	$\cdots$	$\alpha_{K+1}+\beta_L$	$\alpha_{K+1}+\beta_{L+1}$	$\cdots$	$\alpha_{K+1}+\beta_{L+T}$
$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
α_{K+T}	$\alpha_{K+T}+\beta_1$	$\cdots$	$\alpha_{K+T}+\beta_L$	$\alpha_{K+T}+\beta_{L+1}$	$\cdots$	$\alpha_{K+T}+\beta_{L+T}$

Table 1. The combinatorial problem of constructing α and β so that $(\alpha \oplus \beta)$ is decodable and T-secure.

2.3. Polynomial Code For Big T

For big T we construct a polynomial code, $GASP_{big}$. Given K, L, and T, define the polynomial code $GASP_{big}$ as follows. Let α and β be give by

$$\alpha_k = \begin{cases} k-1 & if 1 \leq k \leq K \\ KL+t-1 & if k = K+t, 1 \leq t \leq T \end{cases},$$

$$\beta_\ell = \begin{cases} K(\ell-1) & if 1 \leq \ell \leq L \\ KL+t-1 & if \ell = L+t, 1 \leq t \leq T \end{cases}$$

if $L \leq K$, and

$$\alpha_\ell = \begin{cases} K(\ell-1) & if 1 \leq \ell \leq L \\ KL+t-1 & if \ell = L+t, 1 \leq t \leq T \end{cases},$$

$$\beta_k = \begin{cases} k-1 & if 1 \leq k \leq K \\ KL+t-1 & if k = K+t, 1 \leq t \leq T \end{cases}$$

if $K < L$.
Lastly, define $N = |terms(\alpha \oplus \beta)|$. Then $GASP_{big}$ is defined to be the polynomial code PC(K, L, T, N, α ; β). Let $N = |terms(\alpha \oplus \beta)|$. Then N is given by

- if $L \leq K$:

$$N = \begin{cases} (K+T)(L+1)-1 & if T < K \\ 2KL+2T-1 & if T \geq K \end{cases}$$

- if $K < L$:

$$N = \begin{cases} (L+T)(K+1)-1 & if T < L \\ 2KL+2T-1 & if T \geq L \end{cases}$$

2.4. Polynomial Code For Small T

For small T we construct a polynomial code, $GASP_{small}$. Given K, L, and T, define the polynomial code $GASP_{small}$ as follows. Let α and β be given by

- if $L \leq K$:

$$\alpha_k = \begin{cases} k-1 & if 1 \leq k \leq K \\ KL+K(t-1) & if k = K+t, 1 \leq t \leq T \end{cases};$$

$$\beta_\ell = \begin{cases} K(\ell-1) & if 1 \leq \ell \leq L \\ KL+t-1 & if \ell = L+t, 1 \leq t \leq T \end{cases}$$

- if $K < L$:

$$\alpha_\ell = \begin{cases} K(\ell-1) & if 1 \leq \ell \leq L \\ KL+t-1 & if \ell = L+t, 1 \leq t \leq T \end{cases};$$

$$\beta_k = \begin{cases} k-1 & if 1 \leq k \leq K \\ KL+K(t-1) & if k = K+t, 1 \leq t \leq T \end{cases}$$

Let $N = |terms(\alpha \oplus \beta)|$. Then N is given by

- if $L \leq K$:

$$N = \begin{cases} 2K+T^2 & if L=1, T<K \\ KT+K+T & if L=1, T \geq K \\ KL+K+L & if L \geq 2, 1=T<K \\ KL+K+L+T^2+T-3 & if L \geq 2, 2 \leq T < K \\ KL+KT+L+2T-3-\lfloor \frac{T-2}{K} \rfloor & if L \geq 2, K \leq T \leq K(L-1)+1 \\ 2KL+KT-K+T & if L \geq 2, K(L-1)+1 \leq T \end{cases} \tag{1}$$

- if $K < L$:

$$N = \begin{cases} 2L+T^2 & if K=1, T<L \\ LT+L+T & if K=1, T \geq L \\ KL+K+L & if K \geq 2, 1=T<L \\ KL+K+L+T^2+T-3 & if K \geq 2, 2 \leq T < L \\ KL+LT+K+2T-3-\lfloor \frac{T-2}{L} \rfloor & if K \geq 2, L \leq T \leq L(K-1)+1 \\ 2KL+LT-L+T & if K \geq 2, L(K-1)+1 \leq T \end{cases} \tag{2}$$

2.5. Combining Both Schemes

In this section, we construct a polynomial, $GASP$, by combining both $GASP_{small}$ and $GASP_{big}$. By construction, GASP has a better rate than all previous schemes.
By the definition: Given K, L, and T, we define the polynomial code GASP to be

$$\text{GASP} = \begin{cases} \text{GASP}_{\text{small}} & if T < \min\{K, L\} \\ \text{GASP}_{big} & if T \geq \min\{K, L\} \end{cases}$$

For $L \leq K$ the polynomial code GASP has rate,

$$\mathcal{R} = \begin{cases} \frac{KL}{KL+K+L} & if 1 = T < L \leq K \\ \frac{KL}{KL+K+L+T^2+T-3} & if 2 \leq T < L \leq K \\ \frac{KL}{(K+T)(L+1)-1} & if L \leq T < K \\ \frac{KL}{2KL+2T-1} & if L \leq K \leq T \end{cases}$$

For $K < L$, the rate is given by interchanging K and L.

2.6. Decoding

For explanation of decoding we consider the example, where $T = 2$ and $K = L = 3$. We consider matrix A and matrix B:

$$A = \begin{bmatrix} A_1 \\ A_2 \\ A_3 \end{bmatrix}, \quad B = \begin{bmatrix} B_1 & B_2 & B_3 \end{bmatrix}.$$

For all product AB is consist of $A_k B_l$ and given by

$$AB = \begin{bmatrix} A_1B_1 & A_1B_2 & A_1B_3 \\ A_2B_1 & A_2B_2 & A_2B_3 \\ A_3B_1 & A_3B_2 & A_3B_3 \end{bmatrix}$$

We make a scheme for computing of AB via polynomial interpolation. The scheme is private against any $T = 2$ servers colluding to deduce the identities of A and B, and uses a total of $N = 18$ servers. After that we add R_1 and R_2, which are two matrices picked independently and uniformly at random with entries in F_q, both of size equal to the A_k. Similarly, we add S_1 and S_2 with size equal to the B_l. We can define polynomials:

$$f(x) = A_1x^{\alpha_1} + A_2x^{\alpha_2} + A_3x^{\alpha_3} + R_1x^{\alpha_4} + R_2x^{\alpha_5}$$

$$g(x) = B_1x^{\beta_1} + B_2x^{\beta_2} + B_3x^{\beta_3} + S_1x^{\beta_4} + S_2x^{\beta_5}$$

We will recover the products A_kB_l by interpolating the product $h(x) = f(x)g(x)$. Also $h(x)$ can be given by

$$h(x) = \sum_{1 \leq k, \ell \leq 3} A_kB_\ell x^{\alpha_k+\beta_\ell} + \sum_{\substack{1 \leq k \leq 3 \\ 4 \leq \ell \leq 5}} A_kS_\ell x^{\alpha_k+\beta_\ell} +$$

$$+ \sum_{\substack{4 \leq k \leq 5 \\ 1 \leq \ell \leq 3}} B_\ell R_k x^{\alpha_k+\beta_\ell} + \sum_{4 \leq k, \ell \leq 5} R_kS_\ell x^{\alpha_k+\beta_\ell}$$

For this polynomial we consider the degree table:

	$\beta_1 = 0$	$\beta_2 = 3$	$\beta_3 = 6$	$\beta_4 = 9$	$\beta_5 = 10$
$\alpha_1 = 0$	0	3	6	9	10
$\alpha_2 = 1$	1	4	7	10	11
$\alpha_3 = 2$	2	5	8	11	12
$\alpha_4 = 9$	9	12	15	18	19
$\alpha_5 = 12$	12	15	18	21	22

Table 2. The degree table

The polynomial $h(x)$ has the following form:
$h(x) = A_1B_1 + \cdots + A_3B_3x^8 + C_9x^9 + C_{10}x^{10} + C_{11}x^{11} + C_{12}x^{12} + C_{15}x^{15} + C_{18}x^{18} + C_{19}x^{19} + C_{21}x^{21} + C_{22}x^{22}$
Here each C_j is a sum of products of matrices where each sum and has either R_k or S_l as a factor, and thus their precise nature is not important for decoding. for decoding we consider next matrix equation:

$$\begin{pmatrix} 1 & x_1 & x_1^2 & \cdots & x_1^{22} \\ 1 & x_2 & x_2^2 & \cdots & x_2^{22} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & x_{18} & x_{18}^2 & \cdots & x_{18}^{22} \end{pmatrix} \begin{pmatrix} A_1B_1 \\ \vdots \\ R_5S_5 \end{pmatrix} = \begin{pmatrix} h(x_1) \\ \vdots \\ h(x_{18}) \end{pmatrix}$$

We multiply the left and right part of equation by the inverse a Vandermonde matrix. And after that we get the components A_kB_l for solving of AB.
Below is the block-diagram of GASP. Consider the user having 2 matrices A and B. First of all, he encodes these matrices using polynomial coding. Then calculate values of these polynomials in several points N. N depends on the number of servers involved in the calculations. Then user sends out pairs of $f(x)$ and $g(x)$ values that need to be

multiplied to N servers. On the server's side these pairs are multiplied and the result is sent back to the user. The user knows which set of degrees was used to encoding, so he can restore all coefficients using interpolation method.

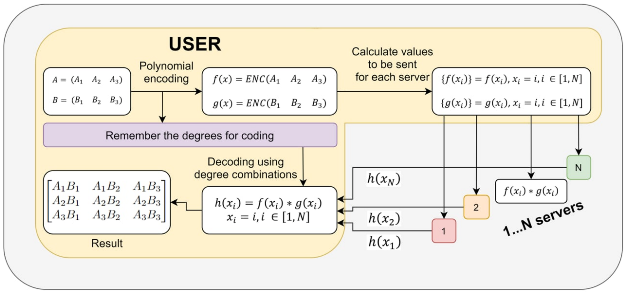

Figure 1. GASP block-diagram

3. Experiment

- Generating matrix A and B with K and L sizes: $A = A[K, 1]$, $B = B[1, L]$
- Generating submatrices A_k and B_l, $k \in [1, K]$, $l \in [1, L]$:
$$A = \begin{bmatrix} A_1 \\ \vdots \\ A_K \end{bmatrix}, B = \begin{bmatrix} B_1 & \cdots & B_L \end{bmatrix}$$
- Generating submatrices S_t and R_t for T - security, $T \in [1, T]$, size A_k = size R_t, size B_k = size S_t:
$$R = \begin{bmatrix} R_1 & \cdots & R_T \end{bmatrix}, S = \begin{bmatrix} S_1 \\ \vdots \\ S_T \end{bmatrix}.$$
- GASP algorithms, which consists of $GASP_{big+small}$ for getting of degree sets and the optimal quantity of servers N:
$$\{\alpha\} = \begin{bmatrix} \alpha_1 & \cdots & \alpha_{\mathrm{T+K}} \end{bmatrix}$$
$$\{\beta\} = \begin{bmatrix} \beta_1 & \cdots & \beta_{\mathrm{T+L}} \end{bmatrix}$$
$$\mathrm{N} = \mathrm{GASP(K, L, T)}$$
- Generating:
  - degree table
  - polynomials $f(x)$ and $g(x)$ for all servers:
$$\mathrm{f}\left(\mathrm{x_i}\right), \mathrm{g}\left(\mathrm{x_i}\right), \mathrm{x_i} = \mathrm{i}, \mathrm{i} \in [1,\ \mathrm{N}]$$
- Distributed multiplication of $f(x)$ by $g(x)$ and getting $h(x)$:
$$\mathrm{h}\left(\mathrm{x_i}\right) = \mathrm{f}\left(\mathrm{x_i}\right) \cdot\ \mathrm{g}\left(\mathrm{x_i}\right), \mathrm{x_i} = \mathrm{i}, \mathrm{i} \in [1,\ \mathrm{N}]$$
- Getting result of multiplication with using inverse Vandermonde matrix to restore result coefficients:
$$\begin{bmatrix} A_1B_1 & A_1B_2 & A_1B_3 \\ A_2B_1 & A_2B_2 & A_2B_3 \\ A_3B_1 & A_3B_2 & A_3B_3 \end{bmatrix}$$

3.1. Comparison of the download rates

Also in our project we consider other codes: Chang (Wei-Ting Chang, 2018) and Kakar (Jaber Kakar & Sezgin, 2018). On the Figure 2 graphs show us the dependency of the download rate for Chang, Kakar and GASP codes on T – security level. As a result we can see that GASP code graph decreases more slowly than the others. It give us the reason to say that the GASP code is more suitable for us than other types of codes.

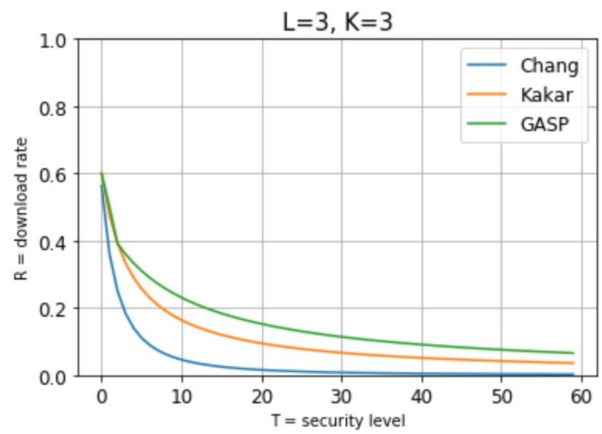

Figure 2. Comparison the Chang, kakar, GASP. We plot the rate of the schemes for $K = 3$ and $L = 3$

3.2. Difficulties

- problems with the justification of mathematical aspects
- the necessary to study additional functionality of used libraries
- problems associated with selection the Vandermond matrix (regular Numpy function `np.vander()` experienced overflow)
- Problem connected with `numpy` library, because numbers are used in exponential form. For example after calculation 22^{22} we get $4142787736421956e{+}29$. But further we calculate the remainder of the division by 29 and it's equals 15. But the remainder of the division 22^{22} by 29 equals 7. As a result we get wrong answers.

4. Conclusions

The task of providing secure computing is actual and not trivial. In our case we had to get the matrix multiplication of A by B with using third-party servers and those servers were not allowed to get the result of this multiplication. The application of GASP codes gives us opportunity to solve this problem of (SDMM) very effective and with necessary security.

References

A source code. URL `https://github.com/MariiaKop/I2BC_final_project.git`.

Jaber Kakar, S. E. and Sezgin, A. Rate-efficiency and straggler-robustness through partition in distributed two-sided secure matrix computation. *arXiv:1810.13006*, 2018.

Rafael G. L. D'Oliveira, Salim El Rouayheb, D. K. Gasp codes for secure distributed matrix multiplication. *arXiv:1812.09962v*, 2020.

Wei-Ting Chang, R. T. On the capacity of secure distributed matrix multiplication. *arXiv:1806.00469*, 2018.