Amortizing Division and Exponentiation

Cong Zhang Shuaishuai Li Dongdai Lin

State Key Laboratory of Information Security, Institute of Information Engineering, Chinese Academy of Sciences
School of Cyber Security, University of Chinese Academy of Sciences

Inscrypt 2022

Outline

Outline

1. Background
2. Preliminaries
3. Correlated Multiplication
4. Amortizing Division and Exponentiation

Multi-Party Computation

- n Parties $P_1, \ldots, P_n$.
- Party P_i holds private input x_i.
- Party P_i obtains output $f_i(x_1, x_2, \ldots, x_n)$.
- Goal: Construct a protocol Π securely implement some functionality f.
- Correctness: P_i learns $f_i(x_1, x_2, \ldots, x_n)$.
- Privacy: P_i only learns $f_i(x_1, x_2, \ldots, x_n)$.

Multi-Party Computation

The general framework of MPC:

Addition $+$

Turing-complete $\Rightarrow$ Any functionality f

Multiplication $\times$

Multi-Party Computation

The general framework of MPC:

Addition $+$

Turing-complete

Any functionality f

Multiplication $\times$

Problem: the commonly used operations such as division, and exponentiation over the integers and/or the finite fields are relatively complex to express by addition and multiplication. More efficient protocols for these basic operations can result in a more efficient protocol for f.

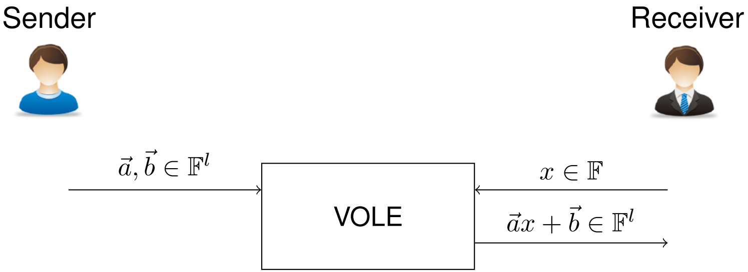
Division and Expoentiation

Our focus in this work:

- Division: the parties have shares of a and b and want to compute the share of $a \cdot b^{-1}$ over a finite field.
- Private Exponentiation: the parties have shares of a and b and want to compute the share of b^a over a finite field.

Outline

Vector Oblivious Linear-Function Evaluation



Cost: To generate length l VOLE, the communication cost is only $O(\kappa \log l)$, using recent works on pseudorandom correlation generator (PCG) [BCGI18, BCG$^+$19a, BCG$^+$19b, SGRR19, WYKW21, CRR21].

Generating Random Shares and Random Coins

Generating Random Shares functionality $\mathcal{F}_{\mathsf{rand}}$

- Input: None.
- Output: A random share $[r]$.
- Implement: Each party select a random value locally.
- Communication cost: 0.

Generating Random Coins functionality $\mathcal{F}_{\mathsf{coin}}$

- Input: None.
- Output: A random value $r \in \mathbb{F}$.
- Implement: Invoke $\mathcal{F}_{\mathsf{rand}}$ to generate $[r]$ and then open it.
- Communication cost: $O(n^2\kappa)$.

- $[r] = (r_1, \ldots, r_n)$ is additive sharing, $r = r_1 + \cdots + r_n$. Party P_i holds r_i.
- Linear: $[r] + [s] = [r + s]$.

Secure Multiplication

Secure Multiplication functionality $\mathcal{F}_{\mathsf{mult}}$

- Input: $[x]$ and $[y]$.
- Output: $[z] = [xy]$.
- Implement: Using Beaver triple [Bea91] $([a], [b], [c])$, where $c = ab$:
  1. The parties compute and open shares of $\sigma = x - a$ and $\rho = y - b$.
  2. The parties compute $[z] := \sigma[b] + \rho[a] + [c] + \sigma\rho$.
- Communication cost: $O(n^2\kappa)$.

Secure Inversion

Secure Inversion functionality $\mathcal{F}_{\text{inver}}$

- Input: $[x]$.
- Output: $[x^{-1}]$.
- Implement: Using the method of [BB89]:
  1. The parties invoke $\mathcal{F}_{\text{rand}}$ functionality to generate a random share $[r]$.
  2. The parties invoke the $\mathcal{F}_{\text{mult}}$ functionality with inputs $[x]$ and $[r]$, and obtain $[xr]$.
  3. All parties open the xr and compute $[x^{-1}] := (xr)^{-1}[r]$.
- Communication cost: $O(n^2\kappa)$.

Unbounded Fan-In Multiplication

Unbounded Fan-In Multiplication functionality $\mathcal{F}^{l}_{\text{unbounded-mult}}$

- Input: $[x_1], \ldots, [x_l]$.
- Output: $[x] = [\Pi_{i=1}^{l} x_i]$.
- Implement: Using the method of [BB89]:
  1. The parties invoke $\mathcal{F}_{\text{rand}}$ to generate $l+1$ random shares $[r_0], [r_1], \ldots, [r_l]$.
  2. The parties invoke $\mathcal{F}_{\text{inver}}$ with inputs $[r_0], \ldots, [r_l]$ and obtain $[r_0^{-1}], \ldots, [r_l^{-1}]$.
  3. For $i = 1, \ldots, l$, the parties invoke $2\ \mathcal{F}_{\text{mult}}$ to compute $[d_i] := [r_{i-1}] \cdot [x_i] \cdot [r_i^{-1}]$. Then the parties open d_i to each other.
  4. The parties invoke $\mathcal{F}_{\text{mult}}$ with input $[r_0^{-1}]$ and $[r_l]$, and compute $[x] = \Pi_{i=1}^{l}[x_i] := \Pi_{i=1}^{l} d_i \cdot [r_0^{-1}] \cdot [r_l]$.
- Communication cost: $O(n^2 \kappa l)$.

Public Base Exponentiation

Public Base Exponentiation functionality $\mathcal{F}_{\text{pbexp}}$

- Input: $[a]$ and b.
- Output: $[b^a]$.
- Implement: Using the method of [AAN18]:
  1. Each party P_i locally computes $c_i := b^{a_i}$, where a_i is the share of a.
  2. P_i shares $[c_i]$ to all parties.
  3. The parties invoke $\mathcal{F}^n_{\text{unbounded-mult}}$ to compute $[\Pi^n_{i=1} c_i] = [b^a]$.
- Communication cost: $O(n^3\kappa)$.

Outline

Correlated Multiplication Triple Generation

The Correlated Multiplication Triple Generation functionality $\mathcal{F}^{l}_{\text{ctriple}}$ is defined as follows:

- Input: None.
- Output: Random shares $[a_1], \ldots, [a_l], [b], [c_1], \ldots, [c_l]$, where $c_i = a_i b$ for $i = 1, \ldots, l$.

Protocol

The main idea is to use VOLE to generate correlated multiplication triples. The formal description of our protocol is as follows:

1. The parties invoke $\mathcal{F}_{\text{rand}}$ to generate $[a_1], \ldots, [a_l], [b]$, where $[a_k] = (a_1^k, \ldots, a_n^k), [b] = (b_1, \ldots, b_n)$ conditioned on $\sum_{j=1}^n a_j^k = a_k, \sum_{j=1}^n b_j = b$ for $k = 1, \ldots, l$.
2. For every distinct $i, j = 1, \ldots, n$, P_j picks $v_{i,j}^k \xleftarrow{R} \mathbb{F}$ for $k = 1, \ldots, l$ and defines $\vec{a_j} := (a_j^1, \ldots, a_j^l) \in \mathbb{F}^l, \vec{v_{i,j}} := (v_{i,j}^1, \ldots, v_{i,j}^l)$. Then P_i and P_j invoke $\mathcal{F}_{\text{vole}}$, where P_i acts as receiver with input b_i and P_j acts as sender with input $(\vec{a_j}, -\vec{v}_{i,j})$. As a result, P_i receives $\vec{u_{i,j}} = \vec{a_j} b_i - \vec{v_{i,j}}$.
3. For $i = 1, \ldots, n, k = 1, \ldots, l$, P_i computes $c_i^k := a_i^k b_i + \sum_{j \neq i} (u_{i,j}^k + v_{j,i}^k)$

Communication Cost: The total communication is $O(n^2 \kappa \log l)$.

Outline

Single Division Case

The secure division functionality $\mathcal{F}_{\text{div}}$ is defined as follows:

- Input: $[x], [y]$.
- Output: $[x^{-1}y]$.

Single Division Case

Recall the inversion protocol [BB89]:

1. The parties invoke $\mathcal{F}_{\mathsf{rand}}$ functionality to generate a random share $[a]$.
2. The parties invoke the $\mathcal{F}_{\mathsf{mult}}$ functionality with inputs $[x]$ and $[a]$, and obtain $[ax]$.
3. All parties open the ax and compute $[x^{-1}] := (ax)^{-1}[a]$.

Main observation: If we compute $[ax]$ and $[ay]$ *at the same time*, then open ax as before, the division $[x^{-1}y]$ can be obtained directly from $(ax)^{-1}[ay]$.

Single Division Case

Our single division protocol:

1. The parties invoke $\mathcal{F}^2_{\text{ctriple}}$ to generate $[a_1], [a_2], [b], [c_1], [c_2]$, where $a_1 b = c_1, a_2 b = c_2$.
2. The parties compute $[\rho] := [x - a_1], [\sigma] := [y - a_2]$ locally and open them.
3. The parties compute $[r] = [bx] := \rho[b] + [c_1]$ and $[s] = [by] := \sigma[b] + [c_2]$.
4. The parties open r and compute $[x^{-1}y] := r^{-1}[s]$.

Correctness. The correctness of π_{div} follows from:
$r^{-1}s = (\rho b + c_1)^{-1}(\sigma b + c_2) = ((x - a_1)b + c_1)^{-1}((y - a_2)b + c_2) = x^{-1}y$.
Cost. The cost is 3 openings and one instance of length 2 correlated multiplication triple generation, which is strictly less than 2 independent multiplication triple generation.

Batch Division Case

The secure division functionality in batch setting $\mathcal{F}_{\mathsf{bdiv}}$ is defined as follows:

- Input: $[x]$ and $[y_1], \ldots, [y_l]$.
- Output: $[x^{-1} y_i]$ for $i = 1, \ldots, l$.

Batch Division Case

Our batch division protocol:

1. The parties invoke $\mathcal{F}^{l+1}_{\text{ctriple}}$ to generate $[a_0], [a_1], \ldots, [a_l], [b], [c_0], [c_1], \ldots, [c_l]$, where $a_i b = c_i$ for $i = 0, 1, \ldots, l$.
2. The parties compute $[\rho] := [x - a_0], [\sigma_i] := [y_i - a_i]$ locally for $i = 1, \ldots, l$ and open them.
3. The parties compute $[r] = [bx] := \rho[b] + [c_0]$ and $[s_i] = [by_i] := \sigma_i[b] + [c_i]$ for $i = 1, \ldots, l$.
4. The parties open r and compute $[x^{-1}y_i] := r^{-1}[s_i]$ for $i = 1, \ldots, l$.

Correctness. Similar to single division case.
Cost. The cost is $l + 2$ openings and one instance of length $l + 1$ correlated multiplication triple generation. Since the open step does not need computation, and the cost of $l + 1$ correlated multiplication triple generation is only about $O(\log l)$ multiplication triple generation. The cost of our protocol is almost the same as a single division instance (except for a logarithmic factor).

Batch Exponentiation Case

The batch private exponentiation functionality $\mathcal{F}_{\mathsf{bexp}}$ is defined as follows:

- Input: $[y], [x_1], \ldots, [x_l]$.
- Output: $[y^{x_1}], \ldots, [y^{x_l}]$.

Batch Exponentiation Case

Our batch exponentiation protocol:

1. The parties invoke $\mathcal{F}_{\text{coin}}$ to generate a random generator $g \in \mathbb{F}$.
2. The parties invoke $\mathcal{F}^{l}_{\text{ctriple}}$ to generate $[a_1], \ldots, [a_l], [b], [c_1], \ldots, [c_l]$.
3. The parties invoke $\mathcal{F}_{\text{pbexp}}$ with inputs $g, [b]$ to obtain $[t] = [g^b]$.
4. The parties invoke $\mathcal{F}_{\text{mult}}$ with inputs $[y], [t]$ to obtain $[p] = [t \cdot y]$.
5. The parties open p.
6. For $i = 1, \ldots, l$, the parties invoke $\mathcal{F}_{\text{pbexp}}$ with inputs $p, [x_i]$ to obtain $[q_i] = [p^{x_i}]$.
7. For $i = 1, \ldots, l$, the parties compute $[\sigma_i] := [x_i - a_i]$ locally and open σ_i. Then the parties compute $[r_i] := -\sigma_i[b] - [c_i]$.
8. For $i = 1, \ldots, l$, the parties invoke $\mathcal{F}_{\text{pbexp}}$ with input $g, [r_i]$ to obtain $[s_i] = [g^{r_i}]$.
9. For $i = 1, \ldots, l$, the parties invoke $\mathcal{F}_{\text{mult}}$ with input $[q_i]$ and $[s_i]$ to obtain $[y^{x_i}] := [q_i s_i]$.

Correctness. The correctness of π_{bexp} follows from:
$q_i s_i = p^{x_i} \cdot g^{r_i} = (t \cdot y)^{x_i} \cdot g^{-\sigma_i b - c_i} = g^{b x_i} y^{x_i} \cdot g^{-b x_i} = y^{x_i}$

Batch Exponentiation Case

Cost. The cost of π_{bexp} is as follows:

- In step 1, one opening is needed in $\mathcal{F}_{\mathsf{coin}}$.
- In step 2, length l correlated multiplication triple generation.
- In step 3, one public base exponentition is needed, including $3n+2$ multiplications and $2n+2$ openings.
- In step 4, one multiplication is needed.
- In step 5, one opening is needed.
- In step 6, l public base exponentition is needed, including $(3n+2)l$ multiplications and $(2n+2)l$ openings.
- In step 7, l openings are needed.
- In step 8, l public base exponentitions are needed, including $(3n+2)l$ multiplications and $(2n+2)l$ openings.
- In step 9, l multiplications are needed.

Batch Exponentiation Case

The total costs are $2n + 4 + (4n + 5)l$ openings, a length l correlated multiplication triple generation, $3n + 3 + (6n + 5)l$ multiplications. While in the original single instance protocol of [AAN18], the costs are $6n + 9$ openings and $9n + 9$ multiplications. If we use l instance of [AAN18] to implement the batch private exponentiation, the costs are $(6n + 9)l$ openings and $(9n + 9)l$ multiplications. Since the main costs is multiplication, the costs saved by our protocol is about 33%.

THANK YOU!

Reference

[AAN18] Abdelrahaman Aly, Aysajan Abidin, and Svetla Nikova. Practically efficient secure distributed exponentiation without bit-decomposition. In *Financial Cryptography and Data Security - 22nd International Conference, FC 2018, Nieuwpoort, Curaçao, February 26 - March 2, 2018, Revised Selected Papers*, pages 291–309, 2018.

[BB89] Judit Bar-Ilan and Donald Beaver. Non-cryptographic fault-tolerant computing in constant number of rounds of interaction. In *Proceedings of the Eighth Annual ACM Symposium on Principles of Distributed Computing, Edmonton, Alberta, Canada, August 14-16, 1989*, pages 201–209, 1989.

[BCG+19a] Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, Peter Rindal, and Peter Scholl. Efficient two-round OT extension and silent non-interactive secure computation. In *CCS 2019*, 2019.

[BCG+19b] Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, and Peter Scholl. Efficient pseudorandom correlation generators: Silent OT extension and more. In *Advances in Cryptology - CRYPTO 2019 - 39th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2019, Proceedings, Part III*, pages 489–518, 2019.

[BCGI18] Elette Boyle, Geoffroy Couteau, Niv Gilboa, and Yuval Ishai. Compressing vector OLE. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS 2018, Toronto, ON, Canada, October 15-19, 2018*, pages 896–912, 2018.

[Bea91] Donald Beaver. Efficient multiparty protocols using circuit randomization. In *CRYPTO 1991*, 1991.

[CRR21] Geoffroy Couteau, Peter Rindal, and Srinivasan Raghuraman. Silver: Silent VOLE and oblivious transfer from hardness of decoding structured LDPC codes. In *CRYPTO 2021*, 2021.

[SGRR19] Phillipp Schoppmann, Adrià Gascón, Leonie Reichert, and Mariana Raykova. Distributed vector-ole: Improved constructions and implementation. In *CCS 2019*, pages 1055–1072, 2019.

[WYKW21] Chenkai Weng, Kang Yang, Jonathan Katz, and Xiao Wang. Wolverine: Fast, scalable, and communication-efficient zero-knowledge proofs for boolean and arithmetic circuits. In *42nd IEEE Symposium on Security and Privacy, SP 2021, San Francisco, CA, USA, 24-27 May 2021*, pages 1074–1091, 2021.