Fast large-scale honest-majority MPC for malicious adversaries

本次介绍的论文由Chida, Koji等人发表在CCS'18，该方案面向诚实大多数安全模型下 ($t < n/2$) 构造了一种抵抗恶意敌手的Security with Abort方案。

1. Background & Preliminaries

诚实大多数场景下的Security with Abort模型之前的博客详细介绍过，在这里不做过多的赘述。本文面向有限域$\mathcal{F}$构造了基于Message Authentication Code (MAC) 的抵抗恶意敌手方案，实现了诚实大多数下的信息论安全。在一定场景下，本文的恶意方案的开销仅是同场景下半诚实方案开销的两倍。本文用到了如下的基础知识：

- **门限秘密分享：** 主要是Shamir's Secret Sharing 和 Replicated Secret Sharing（3PC）。关于秘密分享的各种原语定义，和之前的方案基本类似。我们在此不做赘述，后续用到再做说明；
- **MAC:** 给定秘密值 $[x]$ 和 随机密钥 $[r]$，定义MAC为 $[r \cdot x]$。该技术在电路计算中用来验证恶意敌手是否作恶。

2. Building Blocks & Sub-Protocols

首先，我们介绍一些基本的构造模块。

2.1 Generating Random Shares ($\mathcal{F}_{\text{rand}}$) & Random Coins ($\mathcal{F}_{\text{coin}}$)

$\mathcal{F}_{\text{rand}}$生成随机的秘密分享，其实现依赖于秘密分享的具体方案：

1. 对于Shamir's Secret Sharing，主要使用基于Hyper Vander Matirx的方法生成（之前的博客已经介绍过）；
2. 对于Replicated Secret Sharing，则可以通过预先设置的随机数种子和伪随机生成器非交互式生成。

而 $\mathcal{F}_{\text{coin}}$ 则生成一个随机的公开值，该功能可以借助 $\mathcal{F}_{\text{rand}}$ 先生成随机秘密分享，然后将分享定义的随机数公开即可。

2.2 Secret Sharing of Inputs ($\mathcal{F}_{\text{input}}$)

$\mathcal{F}_{\text{input}}$实现面向恶意敌手的秘密值的分享，该方法如下：

1. 多方首先生成随机数秘密分享 $[r]$；
2. 将随机数 r 公开给秘密拥有者 P_j；
3. P_j计算并公开 $v - r$；
4. 所有的其他参与方验证 $v - r$ 的一致性（防止P_j作恶），然后本地计算$[v] = (v - r) + [r]$。
   该方法的详细协议如下：

PROTOCOL 3.3 (Secure Sharing of Inputs)

- **Inputs:** Let $v_1, \ldots, v_M \in \mathbb{F}$ be the series of inputs; each v_i is held by some P_j.
- **The protocol:**
  1. The parties call $\mathcal{F}_{\text{rand}}$ M times to obtain sharings $[r_1], \ldots, [r_M]$.
  2. For $i = 1, \ldots, M$, the parties run reconstruct$([r_i], j)$ for P_j to receive r_i, where P_j is the owner of the ith input. If P_j receives $\perp$, then it sends $\perp$ to all parties, outputs abort and halts.
  3. For $i = 1, \ldots, M$, party P_j sends $w_i = v_i - r_i$ to all parties.
  4. All parties send $\vec{w} = (w_1, \ldots, w_M)$, or a collision-resistant hash of the vector, to all other parties. If any party receives a different vector to its own, then it outputs $\perp$ and halts.
  5. For each $i = 1, \ldots, M$, the parties compute $[v_i] = [r_i] + w_i$.
- **Outputs:** The parties output $[v_1], \ldots, [v_M]$.

2.3 Secure Multiplication up to Additive Attacks ($\mathcal{F}_{\text{mult}}$)

本文的乘法协议在半诚实的、满足Security up to Additive Attacks的乘法协议基础上构造。该性质是指半诚实模型下的乘法协议，在恶意敌手下满足：对于输入$[x]$和$[y]$的乘法，敌手只能在最后的乘积中加入误差d使得最后的结果为 $[xy] + d$ 而没有办法进行其他攻击（包括攻破 x 和 y的隐私）。该性质的功能描述如下：

FUNCTIONALITY 3.5 ($\mathcal{F}_{\text{mult}}$ - Secure Mult. Up To Additive Attack)

1. Upon receiving $[x]_H$ and $[y]_H$ from the honest parties, the ideal functionality $\mathcal{F}_{\text{mult}}$ computes x, y and the corrupted parties shares $[x]_{\mathcal{C}}$ and $[y]_{\mathcal{C}}$.
2. $\mathcal{F}_{\text{mult}}$ hands $[x]_{\mathcal{C}}$ and $[y]_{\mathcal{C}}$ to the ideal-model adversary/simulator $\mathcal{S}$.
3. Upon receiving d and $\{\alpha_i\}_{i|P_i \in \mathcal{C}}$ from $\mathcal{S}$, functionality $\mathcal{F}_{\text{mult}}$ defines $z = x \cdot y + d$ and $[z]_{\mathcal{C}} = \{\alpha_i\}_{i|P_i \in \mathcal{C}}$. Then, it runs share$(z, [z]_{\mathcal{C}})$ to obtain a share z_j for each party P_j.
4. The ideal functionality $\mathcal{F}_{\text{mult}}$ hands each honest party P_j its share z_j.

Shamir's Secret Sharing下的DN07乘法和3PC Replicated Secret Sharing都满足该性质，在此不做赘述。

2.4 Checking Equality to 0

在有限域$\mathbb{F}$中，判断$[v]$是否为0，可以将$[v]$与随机数$[r]$相乘并公开$v \cdot r$:

1. 如果 $v = 0$，那么 $v \cdot r = 0$一定成立；
2. 如果 $v \neq= 0$，那么$r = 0$的情况下也有可能得到$v \cdot r = 0$。但是，$Pr(r = 0) = \frac{1}{|\mathbb{F}|}$。当$|\mathbb{F}|$足够大的时候，比如$2^{64}$，该概率可以忽略不计。

本方案在计算$[v] \cdot [r]$的时候调用了$\mathcal{F}_{\text{mult}}$，但是由于不会泄露隐私，因此敌手也只能通过随机猜取d使得最后的$v \cdot r + d = 0$，该概率也是 $\frac{1}{|\mathbb{F}|}$。具体的实现协议如下：

PROTOCOL 3.7 (Checking Equality to 0 in the $(\mathcal{F}_{\text{rand}}, \mathcal{F}_{\text{mult}})$-Hybrid Model)

– **Inputs:** The parties hold a sharing $[v]$.

– **The protocol:**

1. The parties call $\mathcal{F}_{\text{rand}}$ to obtain a sharing $[r]$.
2. The parties call $\mathcal{F}_{\text{mult}}$ on $[r]$ and $[v]$ to obtain $[T] = [r \cdot v]$
3. The parties run open$([T])$. If a party receives $\perp$, then it outputs $\perp$. Else, it continues.
4. Each party checks that $T = 0$. If yes, it outputs accept; else, it outputs reject.

3. Protocol for Large Fields

针对比较大的有限域，比如64比特的素数域，本文的协议构造如下：

PROTOCOL 4.1 (Computing Arithmetic Circuits Over Large Fields)

Inputs: Each party P_j $(j \in \{1,\ldots,n\})$ holds an input $x_j \in \mathbb{F}^\ell$.

Auxiliary Input: The parties hold the description of a finite field $\mathbb{F}$ (with $3/|\mathbb{F}| \leq 2^{-\sigma}$) and an arithmetic circuit C over $\mathbb{F}$ that computes f on inputs of length $M = \ell \cdot n$. Let N be the number of multiplication gates in C.

The protocol (throughout, if any party receives $\perp$ as output from a call to a sub-functionality, then it sends $\perp$ to all other parties, outputs $\perp$ and halts):

1. *Secret sharing the inputs:*
   (a) For each input v_i held by party P_j, party P_j sends v_i to $\mathcal{F}_{\text{input}}$.
   (b) Each party P_j records its vector of shares $(v_1^j,\ldots,v_M^j)$ of all inputs, as received from $\mathcal{F}_{\text{input}}$. If a party received $\perp$ from $\mathcal{F}_{\text{input}}$, then it sends abort to the other parties and halts.
2. *Generate randomizing share:* The parties call $\mathcal{F}_{\text{rand}}$ to receive a sharing $[r]$.
3. *Randomization of inputs:* For each input wire sharing $[v_m]$ (where $m \in \{1,\ldots,M\}$), the parties call $\mathcal{F}_{\text{mult}}$ on $[r]$ and $[v_m]$ to receive $[r \cdot v_m]$.
4. *Circuit emulation:* Let $G_1,\ldots,G_N$ be a predetermined topological ordering of the gates of the circuit. For $k = 1,\ldots,N$ the parties work as follows:
   - G_k *is an addition gate:* Given pairs $([x],[r\cdot x])$ and $([y],[r\cdot y])$ on the *left* and *right* input wires respectively, the parties locally compute $([x+y],[r\cdot x]+[r\cdot y]) = ([x+y],[r\cdot(x+y)])$.
   - G_k *is a multiplication-by-constant gate:* Given $([x],[r\cdot x])$ on the input wire and constant $a \in \mathbb{F}$, the parties locally compute $([a\cdot x],[r\cdot(a\cdot x)])$.
   - G_k *is a multiplication gate:* Given pair $([x],[r\cdot x])$ and $([y],[r\cdot y])$ on the *left* and *right* input wires respectively:
     (a) The parties call $\mathcal{F}_{\text{mult}}$ on $[x]$ and $[y]$ to receive $[x\cdot y]$.
     (b) The parties call $\mathcal{F}_{\text{mult}}$ on $[r_i\cdot x]$ and $[y]$ to receive $[r_i\cdot x\cdot y]$.
5. *Verification stage: Before* the secrets on the output wires are reconstructed, the parties verify that all the multiplications were carried out correctly, as follows. Let $\{([z_k],[r\cdot z_k])\}_{k=1}^N$ be the pairs on the output wires of all multiplication gates and let $\{([v_m],[r\cdot v_m])\}_{m=1}^M$ be the pairs on the input wires of the circuit.
   (a) The parties call $\mathcal{F}_{\text{coin}}$ to receive $\alpha_1,\ldots,\alpha_N,\beta_1,\ldots,\beta_M \in \mathbb{F}$.
   (b) The parties locally compute
   $$[u] = \sum_{k=1}^{N} \alpha_k\cdot[r\cdot z_k] + \sum_{m=1}^{M} \beta_m\cdot[r\cdot v_m] \quad \text{and} \quad [w] = \sum_{k=1}^{N} \alpha_k\cdot[z_k] + \sum_{m=1}^{M} \beta_m\cdot[v_m].$$
   (c) The parties run open$([r])$ to receive r.
   (d) Each party locally computes $[T] = [u] - r\cdot[w]$.
   (e) The parties call $\mathcal{F}_{\text{checkZero}}$ on $[T]$. If $\mathcal{F}_{\text{checkZero}}$ outputs reject, the parties output $\perp$ and abort. Else, if it outputs accept, the parties proceed to the next step.
6. *Output reconstruction:* For each output wire of the circuit, the parties run reconstruct$([v],j)$, where $[v]$ is the sharing of the value on the output wire, and P_j is the party whose output is on the wire.
   If a party received $\perp$ in any call to the reconstruct procedure, then it sends $\perp$ to the other parties, outputs $\perp$ and halts.

Output: If a party has not output $\perp$, then it outputs the values it received on its output wires.

协议的构造主要思路是构建关于输入的MAC，并对MAC计算和输入一样的电路。最后，通过随机线性组合验证计算中是否有错误。根据本文的分析，最后检测到错误的概率为

$$\frac{2}{|\mathbb{F}|}+(1-\frac{2}{|\mathbb{F}|})\cdot\frac{1}{|\mathbb{F}|}<\frac{3}{|\mathbb{F}|}$$

给定统计安全参数σ，则需要满足 $\frac{3}{|\mathbb{F}|}<2^{-\sigma}$。

优化： 为了减少验证阶段的$\mathcal{F}_{\text{coin}}$的开销，可以利用该协议生成随机数种子，然后利用PRG生成线性组合的参数。

开销： 按照上述协议中的参数设置，协议的开销为

$$(M+2N+1)\cdot\mathcal{F}_{\text{mult}}+(M+3)\cdot\mathcal{F}_{\text{rand}}+(M+L)\cdot\mathsf{reconstruct}+3\cdot\mathsf{open},$$

平均下来，每个乘法门需要 $2\cdot\mathcal{F}_{\text{mult}}$。

4. Protocol for Small Fields

然而对于较小的有限域，条件 $\frac{3}{|\mathbb{F}|}<2^{-\sigma}$ 可能满足不了。因此，需要多做δ次验证，使得$(\frac{3}{|\mathbb{F}|})^{\delta}<2^{-\sigma}$。鉴于此，本文构造了面向任意大小有限域的协议如下：

PROTOCOL 5.3 (Computing Arithmetic Circuits Over Any Finite $\mathbb{F}$)

Inputs: Each party P_j ($j \in \{1,\ldots,n\}$) holds an input $x_j \in \mathbb{F}^\ell$.

Auxiliary Input: The parties hold the description of a finite field $\mathbb{F}$ and an arithmetic circuit C over $\mathbb{F}$ that computes f on inputs of length $M = \ell \cdot n$. Let N be the number of multiplication gates in C.

The protocol:

1. *Parameter computation:* Set δ to be the smallest value for which $\delta \geq \frac{\sigma}{\log(|\mathbb{F}|/3)}$.
2. *Secret sharing the inputs:*
   (a) For each input v_i held by party P_j, party P_j sends v_i to $\mathcal{F}_{\text{input}}$.
   (b) Each party P_j records its vector of shares $(v_1^j,\ldots,v_M^j)$ of all inputs, as received from $\mathcal{F}_{\text{input}}$. If a party received $\perp$ from $\mathcal{F}_{\text{input}}$, then it sends abort to the other parties and halts.
3. *Generate randomizing shares:* For $i = 1$ to δ, the parties call $\mathcal{F}_{\text{rand}}$ to receive a sharing $[r_i]$.
4. *Randomization of inputs:* For each input wire sharing $[v_m]$ (where $m \in \{1,\ldots,M\}$) and for every $i = 1,\ldots,\delta$, the parties call $\mathcal{F}_{\text{mult}}$ on $[r_i]$ and $[v_m]$ to receive $[r_i \cdot v_m]$.
5. *Circuit emulation:* Let $G_1,...,G_N$ be a predetermined topological ordering of the gates of the circuit. For $k = 1,...,N$ the parties work as follows:
   - G_k *is an addition gate:* Given tuples $([x],[r_1 \cdot x],\ldots,[r_\delta \cdot x])$ and $([y],[r_1 \cdot y],\ldots,[r_\delta \cdot y])$ on the *left* and *right* input wires respectively, the parties locally compute $([x+y],[r_1 \cdot (x+y)],\ldots,[r_\delta \cdot (x+y)])$.
   - G_k *is a multiplication-by-a-constant gate:* Given a constant $a \in \mathbb{F}$ and tuple $([x],[r_1 \cdot x],\ldots,[r_\delta \cdot x])$ on the input wire, the parties locally compute $([a \cdot x],[r_1 \cdot (a \cdot x)],\ldots,[r_\delta \cdot (a \cdot x)])$.
   - G_k *is a multiplication gate:* Given tuples $([x],[r_1 \cdot x],\ldots,[r_\delta \cdot x])$ and $([y],[r_1 \cdot y],\ldots,[r_\delta \cdot y])$ on the *left* and *right* input wires respectively:
     (a) The parties call $\mathcal{F}_{\text{mult}}$ on $[x]$ and $[y]$ to receive $[x \cdot y]$.
     (b) For $i = 1$ to δ, the parties call $\mathcal{F}_{\text{mult}}$ on $[r_i \cdot x]$ and $[y]$ to receive $[r_i \cdot x \cdot y]$.
6. *Verification stage:* Let $\{([z_k],[r_1 \cdot z_k],\ldots,[r_\delta \cdot z_k])\}_{k=1}^N$ be the tuples on the output wires of all multiplication gates and let $\{([\beta_{m,1}],\ldots,[\beta_{m,\delta}])\}_{m=1}^M$ be the tuples on the input wires of the circuit.
   (a) For $m = 1,\ldots,M$, the parties call $\mathcal{F}_{\text{rand}}$ to receive $[\beta_{m,1}],\ldots,[\beta_{m,\delta}]$.
   (b) For $k = 1,\ldots,N$, the parties call $\mathcal{F}_{\text{rand}}$ to receive $[\alpha_{k,1}],\ldots,[\alpha_{k,\delta}]$.
   (c) *Compute linear combinations:* For $i = 1,\ldots,\delta$:
      i. The parties call $\mathcal{F}_{\text{product}}$ on vectors $([\alpha_{1,i}],\ldots,[\alpha_{N,i}],[\beta_{1,i}],\ldots,[\beta_{M,i}])$ and $([r_i \cdot z_1],\ldots,[r_i,\cdot z_N],[r_i \cdot v_1],\ldots,[r_i \cdot v_M])$ to receive $[u_i]$.
      ii. The parties call $\mathcal{F}_{\text{product}}$ on vectors $([\alpha_{1,i}],\ldots,[\alpha_{N,i}],[\beta_{1,i}],\ldots,[\beta_{M,i}])$ and $([z_1],\ldots,[z_N],[v_1],\ldots,[v_M])$ to receive $[w_i]$.
      iii. The parties run open$([r_i])$ to receive r_i.
      iv. Each party locally computes $[T_i] = [u_i] - r_i \cdot [w_i]$.
      v. The parties call $\mathcal{F}_{\text{checkZero}}$ on $[T_i]$. If $\mathcal{F}_{\text{checkZero}}$ outputs reject, the parties output $\perp$ and abort. Else, if it outputs accept, they proceed.
7. *Output reconstruction:* For each output wire of the circuit, the parties run reconstruct$([v],j)$, where $[v]$ is the sharing of the value on the output wire, and P_j is the party whose output is on the wire.

If a party received $\perp$ in any call to the reconstruct procedure, then it sends $\perp$

If a party received $\perp$ in any call to the reconstruct procedure, then it sends $\perp$ to the other parties, outputs $\perp$ and halts.

Output: If a party has not aborted, it outputs the values received on its output wires.

和之前的方案相比，此方案主要的不同点在于线性组合的系数 α和 β都是处于秘密分享、而不是明文。如此一来有两个好处：

1. 调用$\mathcal{F}_{\text{rand}}$比$\mathcal{F}_{\text{coin}}$更加高效；
2. 其中，6-(c)-iii)中的可以不用公开r。然而，之前的协议需要公开r防止敌手进行Additive Attacks（因此之前的协议线性组合系数公开）。不用公开r对于reactive computation场景非常友好，也更加高效。

然而，存在的一个问题是求$[u_i]$和$[w_i]$增加了4次$\mathcal{F}_{\text{mult}}$调用。接下来，我们介绍如何优化这部分。

Secure Sum of Products: 对于

$$[u] = \sum_{k=1}^{N} [\alpha_k] \cdot [r \cdot z_k] + \sum_{m=1}^{M} [\beta_m] \cdot [r \cdot v_m]$$

$$[w] = \sum_{k=1}^{N} [\alpha_k] \cdot [z_k] + \sum_{m=1}^{M} [\beta_m] \cdot [v_m]$$

直接对于每个k, m都会增加4次$\mathcal{F}_{\text{mult}}$的调用。为了减少开销：

1. 对于Shamir's Secret Sharing，本文先进行本地的local multiplication和加法然后做degree reduction；
2. 对于Replicated Secret Sharing，本文先进行local multiplication然后进行resharing。

开销： 按照上述协议的开销如下：

$$(\delta \cdot M + (1+\delta) \cdot N + 3\delta) \cdot \mathcal{F}_{\text{mult}} + (\delta \cdot (M+N) + 2\delta) \cdot \mathcal{F}_{\text{rand}} + (M+L) \cdot \text{reconstruct} + 2\delta \cdot \text{open}$$

5. Instantiations and Experiments

本文主要和 [22] A Framework for Constructing Fast MPC over Arithmetic Circuits with Malicious Adversaries and an Honest-Majority, CCS'17对比，接下来是对比提升：

	$\mathcal{F}_{\text{mult}}$	$\mathcal{F}_{\text{rand}}$	open	reconstruct
Replicated secret sharing (three parties)	1	0	4	2
Shamir sharing (few parties), $\mathcal{F}_{\text{rand}}$ with PRSS	6	0	$n-1$	1
Shamir sharing (many parties), $\mathcal{F}_{\text{rand}}$ with VAN	6	2	$n-1$	1

Table 1. The communication cost per party for instantiations in [22], written as the number of *field elements* sent.

	Protocol of [22] with $\delta = 1$	Protocol 4.1 (large field)	Protocol 5.3 with $\delta = 1$	Protocol 5.3 with $\delta = 2$
Replicated secret sharing (three parties)	4	2	2	3
Shamir (few parties), $\mathcal{F}_{\text{rand}}$ with PRSS	36	12	12	18
Shamir (many parties), $\mathcal{F}_{\text{rand}}$ with VAN	42	12	14	22

Table 2. The communication cost per party for the instantiations in Table 1 and the protocol of [22], written as the number of *field elements* sent *per multiplication gate*. (Note that Protocol 5.3 with $\delta = 2$ has smaller field elements and thus more elements sent could actually mean less bandwidth.)

Circuit Depth	3 (replicated)	3	5	7	9	11	30	50	70	90	110
20	319	826	844	1,058	1,311	1,377	2,769	4,053	5,295	6,586	8,281
100	323	842	989	1,154	1,410	1,477	3,760	6,052	8,106	11,457	15,431
1,000	424	1,340	1,704	1,851	2,243	2,887	12,144	26,310	33,294	48,927	79,728
10,000	1,631	6,883	7,424	8,504	12,238	16,394	61,856	132,160	296,047	411,195	544,525

Table 3. *LAN configuration* execution times in milliseconds of a circuit with 1,000,000 multiplication gates, for different depths. The first column gives the running time for the replicated secret sharing version; all other columns are the Shamir sharing for different numbers of parties.

	3 (replicated)	3	5	7	9	11	30	50	70	90	110
Protocol 4.1	319	826	844	1,058	1,311	1,377	2,769	4,053	5,295	6,586	8,281
Protocol of [22]	513	1,229	1,890	3,056	4,009	5,187	15,954	28,978	44,599	58,966	72,096
Speedup	161%	149%	224%	289%	306%	377%	576%	715%	842%	895%	871%

Table 4. *LAN configuration* execution times in milliseconds of a circuit with 1,000,000 multiplication gates and depth 20. The times for [22] are for the best protocol for the number of parties.

Circuit Depth	3 (replicated)	3	5	7	9	11	30	50
20	3502	20,492	27,772	28,955	24,482	24,729	87,355	128,366
100	10,712	45,250	53,872	50,719	55,716	56,482	134,860	197,321

Table 5. *WAN configuration* (North Virginia, Germany and India) execution times in milliseconds of a circuit with 1,000,000 multiplication gates, for different depths.