A Framework for Constructing Fast MPC over Arithmetic Circuits with Malicious Adversaries and an Honest-Majority

本次分享的论文是Yehuda Lindell和Ariel Nof发表在ACM CCS'17的诚实大多数场景下，满足security with abort安全性的MPC方案。该方案下，恶意敌手能够控制 $t < n/2$ 个参与方，本文基于Beaver triples构造了两种验证计算正确性的方案，分别面向大量计算参与方和少量计算参与方的场景。论文链接。

1. Background

首先，我们介绍一下本文恶意MPC方案用到的一些基础知识：

1. 诚实大多数场景下，考虑的门限关系一般是 $t < n/2$ 和 $t < n/3$。一般恶意敌手越少，方案也就越高效。本文面向$t < n/2$场景，即临界条件是 $2t + 1 = n$；
2. Beaver triples 在半诚实方案下被用来进行乘法门的计算，生成和使用的方式已经分享过多次。在本文中，Lindell等人基于Beaver triples构造了高效的验证方案。虽然之前也有类似方案，但是之前的方案需要在生成Beaver triples的时候验证其正确性，因此效率较低。本文的方案能够在不需要要求Beaver triples的条件下实现正确性检测，从而预计算更加高效。
3. 本文采用了两种秘密分享：对于>3方的场景，采用Shamir's Secret Sharing；对于3PC则采用Replicated Secret Sharing。本文的乘法协议基于满足security up to additive attacks in presence of malicious adversary半诚实乘法协议构造：即在恶意敌手下，攻击者对乘法结果加入选择的错误值d使得最后的结果变为$xy + d$。
4. 本文的协议是面向Field $\mathbb{F}$ 构造的，不能简单的直接迁移到Ring上。因为检验的方案依赖于乘法逆元(inverse)。

2. Threshold Secret Sharing

Shamir's Secret Sharing 和 Replicated Secret Sharing 都是 Threshold Secret Sharing。关于share(v)、reconstruct$([v], i)$和线性计算（例如加法）的介绍已经很多，在这里不赘述。本文用到了一个关键原语：open$([v])$

- open$([v])$：给定秘密分享$[v]$，open$([v])$或者公开v，或者公开"abort"。显然，任意$t+1$个参与方J都可以计算得到一个v。如果一个秘密分享$[v]$是正确的，那么任意$t+1$个参与方恢复一个唯一的v。且，任意J的超集，只能返回v或者$\perp$（没有其他值）。

接下来，我们定义sharing的正确性：

定义2.1：令$H \subseteq \{P_1, \ldots, P_n\}$表示诚实的参与方，如果对于$[v]$，存在$v' \in \mathbb{F}(v' \neq \perp)$满足对于任意的$J \subseteq H(|J| = t + 1)$都有val$([v])_J = v'$，那么说$[v]$是正确的，其中val$([v])_J$表示$J$中所有参与方恢复得到的关于$[v]$的真实值。

根据上述定义，sharing不正确有如下两种情况：

1. 存在一个$J \subseteq H$，$|J| = t + 1$ 使得 val$([v])_J = \perp$；
2. 存在两个$t + 1$大小的集合$J_1, J_2 \subseteq H$有val$([v])_{J_1} \neq$ val$([v])_{J_2}$。

上述第一种情况验证值的valid，如果发生第一种情况则说明$[v]$是invalid的；第二种情况是确认value-consistent，如果发生第二种情况则说明$[v]$是value-inconsistent的。因此，一个值如果是valid的，也有可能是value-inconsistent的。对于Shamir's Secret Sharing，其一定满足valid；对于Replicated secret Sharing，其可能是invalid的。
进一步，定义一个秘密分享方案是**robustly-linear**的，如果对于invalid-shares $[u]$和$[v]$存在唯一的$\alpha \in \mathbb{F}$使得$\alpha \cdot [u] + [v]$是valid的。为了后续方案的构造，本文提出了如下的**CLAIM**，**COROLLARY**和**LEMMA**：

- **CLAIM2.2: 如果$[u]$和$[v]$都是正确的秘密分享，那么对于任意的$\alpha \in \mathbb{F}$，$[w] = \alpha \cdot [u] + [v]$也是正确的。**
- **COROLLARY2.3: 如果$[w] = \alpha \cdot [u] + [v]$不正确，那么$[u]$或者$[v]$不正确。**
- **LEMMA2.4: $][u]$是robustly-linear秘密分享方案的不正确分享，$[v]$是任意分享，那么$[w] = \alpha \cdot [u] + [v]$，$\alpha \in_R \mathbb{F} \setminus \{0\}$的概率$\leq \frac{1}{|\mathbb{F}|-1}$。**

上述CLAIM2.2和COLLARY2.3很直观，LEMMA2.4的证明见原文。

3. SUB-PROTOCOLS & BUILDING BLOCKS

本章节介绍核心的协议模块，包括随机数生成、shares的正确性检验、基于open原语的triples验证和基于mult secure up to additive attacks的triples验证。

3.1 Generating Random Value and Shares

- $\mathcal{F}_{\text{rand}}$: 本文借鉴DN07的方法生成随机数，即每个$P_i$本地选择随机数，并对随机数做秘密分享将份额分发。然后，各方在本地利用Vandermonde矩阵得到$n - t$个随机数的shares。均摊下来，每个随机数生成成每一方发送个2个元素的通信；
- $\mathcal{F}_{\text{coin}}$： 生成一个公开的随机数。该功能可以通过调用$F_{\text{rand}}$然后调用open公开shared的随机数来实现。

3.2 Correctness Check of Shares

给定m个秘密分享的值 $[x_1], \ldots, [x_m]$，批量验证m个秘密分享的值被正确分享了。对于这个问题，本文采用了随机线性组合的方式：首先调用$\mathcal{F}_{\text{coin}}$ 生成m个随机数$\rho_1, \ldots, \rho_m$；然后调用$\mathcal{F}_{\text{rand}}$生成$[r]$；最后计算$[v] = \rho_1 \cdot [x_1] + \cdots + \rho_m \cdot [x_m] + [r]$，并公开$[v]$。如果最后得到abort，则终止；否则正确性验证通过。存在错误的$[x_i]$而该方法却接受的概率是$\leq \frac{1}{|\mathbb{F}|-1}$。具体协议如下：

PROTOCOL 3.1 (Batch Correctness Check of Shares).

- **Inputs:** The parties hold m shares $[x_1], \ldots, [x_m]$.
- **The protocol:**
  (1) The parties call $\mathcal{F}_{\text{coin}}$ to receive random elements $\rho_1, \ldots, \rho_m \in \mathbb{F} \setminus \{0\}$.
  (2) The parties call $\mathcal{F}_{\text{rand}}$ and obtain a sharing $[r]$.
  (3) The parties locally compute
  $$[v] = \rho_1 \cdot [x_1] + \ldots + \rho_m \cdot [x_m] + [r]$$
  (4) The parties run open($[v]$).
  (5) If no abort message was received, then the parties output accept.

3.3 Triple Verfication Based on the Open Procedure

第一种验证triple正确性的方案依赖于open原语。首先，一个三元组$([a],[b],[c])$是正确需要满足如下两个条件：

- $[a],[b],[c]$都是正确的shares；
- $c = a \cdot b$。

本文利用预计算的元组$([a],[b],[c])$ 来验证 $([x],[y],[z])$ 的正确性。具体的协议如下：

PROTOCOL 3.4 (Triple Verification Using Open).

- **Inputs:** The parties hold a triple $([x], [y], [z])$ to verify and an additional random triple $([a], [b], [c])$.
- **The protocol:**
  (1) The parties call $\mathcal{F}_{\text{coin}}$ to generate a random $\alpha \in \mathbb{F} \setminus \{0\}$.
  (2) Each party locally computes $[\rho] = \alpha \cdot [x] + [a]$ and $[\sigma] = [y] + [b]$.
  (3) The parties run open($[\rho]$) and open($[\sigma]$), as defined in Section 2, to receive ρ and σ. If a party receives $\perp$ in an opening, then it sends $\perp$ to all the other parties and aborts.
  (4) Each party locally computes
  $$[v] = \alpha[z] - [c] + \sigma \cdot [a] + \rho \cdot [b] - \rho \cdot \sigma .$$
  (5) The parties run the open($[v]$) procedure to receive v. If a party receives $\perp$ in the opening, then it sends $\perp$ to all the other parties and aborts.
  (6) Each party checks that $v = 0$. If not, then it sends $\perp$ to the other parties and aborts.
  (7) If no abort messages are received, then the parties output accept.

首先，根据一次一密ρ,σ不会泄露隐私；其次，如果各方能够得到ρ,σ，那么sharings的正确性得到了保证。进一步，则是对triple $([x],[y],[z])$的正确性$z = xy$进行验证：

- 如果$([a],[b],[c])$是一个正确的元组，那么当且仅当$v=0$时才有$z=xy$;
- 如果$([a],[b],[c])$不正确，那么由于α引入的随机性，$v=0$的概率$\leq \frac{1}{|\mathbb{F}|-1}$。

上述验证的主要开销来自三次的**open**调用。

3.4 Triple Verfication Based on Multiplication Secure Up to Additive Attacks

本章节使用multiplication secure up to additive attacks 来减少**open**的调用。回顾协议3.4，核心在于计算

$$v=\alpha\cdot z-c+\sigma\cdot a+\rho\cdot b-\rho\cdot\sigma$$

其中，$\rho=\alpha x+b,\sigma=y+b$。
不公开ρ,σ，那么v的计算就变为;

$$\begin{aligned}[v]&=[\alpha]\cdot[z]-[c]+[\sigma]\cdot[a]+[\rho]\cdot[b]-[\rho]\cdot[\sigma]\\&=[\alpha]\cdot[z]-[c]+[\sigma]\cdot[a]-[\rho]\cdot[y]\end{aligned}$$

然而还有两点需要进一步解决：

- $[\alpha]$是保密的。如果α是公开的，那么敌手如果在计算$z=xy+d$时加入d，那么可以在计算$-[c]+[\sigma]\cdot[a]-[\rho]\cdot[y]$时加入 $-\alpha d$，从而使得最后的验证通过。
- 另一个问题在于可能会泄露y。具体攻击：敌手在计算$[\rho]$的时候加入d使得$\rho=\alpha\cdot x+a+d$，那么最后的计算结果 $v=-d\cdot y$。因此敌手可以得到$y=-\frac{v}{d}$。一个直观的解决方法是利用MPC直接判断$[v]=0$，但是这会泄露$y\stackrel{?}{=}0$。为了这个问题，本文基于 $([x],[y],[z])$生成了一个新的元组 $([x'],[y'],[z'])=([x],[y+\psi],[z+\psi\cdot x])$，其中$\psi\in\mathbb{F}$。从而，基于

  $$z'\stackrel{def}{=}z+\psi\cdot x=x\cdot y+\psi\cdot x=x\cdot(y+\psi)=x'\cdot y'$$

  可以通过验证$([x'],[y'],[z'])$的正确性确认$([x],[y],[z])$的正确性。如此，计算v的公式变为：

  $$\begin{aligned}v&=\alpha\cdot(z+\psi\cdot x)-c+\sigma\cdot a-\rho(y+\psi)\\&=\alpha\cdot(x\cdot+\psi\cdot x)-a\cdot b+(y+\psi+b)\cdot a-(\alpha\cdot x+a)\cdot(y+\psi)\\&=\alpha\cdot x\cdot y+\alpha\cdot x\cdot\psi-a\cdot b+a\cdot y+a\cdot\psi+a\cdot b-\alpha\cdot x\cdot y-a\cdot y-\alpha\cdot x\cdot\psi-a\cdot\psi=0\end{aligned}$$

  因为$y'=0$的概率是可忽略的$\frac{1}{|\mathbb{F}|}$，所以现在判断$v=0$不会再泄露$y'=0$。为了判断$[v]=0$，参与方在预计算阶段调用$\mathcal{F}_{\text{rand}}$生成随机数$[r]$，$r\in\mathbb{F}\setminus\{0\}$。然后计算$[v\cdot r]$，并公开$vr$。如果$vr=0$，那么$v=0$。具体协议如下：

PROTOCOL 3.6 (TRIPLE VERIFICATION BASED ON MULTIPLICATION).

Let π_{mult} be a multiplication protocol that is secure up to additive attack, as described in Section 2.

- **Inputs:** The parties hold a triple $([x], [y], [z])$ to verify, and an additional random triple $([a], [b], [c])$.
- **The protocol:**
  (1) The parties execute $\mathcal{F}_{\text{rand}}$ to generate a random sharing $[\alpha]$.
  (2) The parties execute π_{mult} on $[x]$ and $[\alpha]$ to obtain $[\alpha \cdot x]$.
  (3) Each party locally computes $[\rho] = [\alpha \cdot x] + [a]$ and $[\sigma] = [y] + [b]$.
  (4) The parties execute π_{mult} on $[z]$ and $[\alpha]$ to obtain $[\alpha \cdot z]$.
  (5) The parties execute π_{mult} on $[a]$ and $[\sigma]$ to obtain $[\sigma \cdot a]$.
  (6) The parties execute π_{mult} on $[\rho]$ and $[y]$ to obtain $[\rho \cdot y]$.
  (7) The parties call $\mathcal{F}_{\text{coin}}$ to receive a random $\psi \in \mathbb{F}$.
  (8) The parties run the open($[\alpha]$) procedure to receive α. If a party receives $\perp$ in the opening, then it sends $\perp$ to all the other parties and aborts.
  (9) Each party locally computes

$$[v] = ([\alpha \cdot z] + \alpha\psi \cdot [x]) - [c] + ([\sigma \cdot a] + \psi \cdot [a]) - ([\rho \cdot y] + \psi \cdot [\rho]).$$

  (10) The parties call $\mathcal{F}_{\text{rand}}$ to generate a random sharing $[r]$
  (11) The parties execute π_{mult} on $[r]$ and $[v]$ to obtain $[w] = [r \cdot v]$.
  (12) The parties run the open($[w]$) procedure to receive w. If a party receives $\perp$ in the opening, then it sends $\perp$ to all the other parties and aborts.
  (13) Each party checks that $w = 0$. if not, then it sends $\perp$ to the other parties and aborts.
  (14) If no abort messages are received, then output accept.

上述协议还需要2次 **open**，进一步本文提出了batch verfication的方法，来均摊**open**的开销。batch verfication协议的关键在于计算多个三元组乘法协议对应的$[v]$的一个随机线性组合，然后判断组合的结果是否为0。根据文中分析，batch verfication误判的概率为$\leq \frac{1}{|\mathbb{F}|-1}$。具体协议如下：

PROTOCOL 3.8 (Batch Verification of Triples based on Multiplication).

- **Inputs:** The parties hold a list of triples $\{([x_i],[y_i],[z_i])\}_{i=1}^{L}$ to verify and a list of random triples $\{([a_i],[b_i],[c_i])\}_{i=1}^{L}$.
- **The protocol:**
  (1) The parties call $\mathcal{F}_{\text{rand}}$ to generate a random sharing $[\alpha]$.
  (2) For $i = 1$ to L: The parties run Steps 2-7 of Protocol 3.6 on $[\alpha]$, $([x_i],[y_i],[z_i])$ and $([a_i],[b_i],[c_i])$.
  (3) The parties run open$([\alpha])$.
  (4) For $i = 1$ to L: The parties execute Step 9 of Protocol 3.6 , to obtain $[v_i]$.
  (5) The parties call $\mathcal{F}_{\text{coin}}$ to receive random elements $\rho_1,\ldots,\rho_L \in \mathbb{F}\setminus\{0\}$
  (6) The parties locally compute
  $$[v] = \rho_1 \cdot [v_1] + \ldots + \rho_L \cdot [v_L]$$
  (7) The parties call $\mathcal{F}_{\text{rand}}$ to generate a random sharing $[r]$.
  (8) The parties execute π_{mult} on $[r]$ and $[v]$ to obtain $[w] = [r \cdot v]$.
  (9) The parties run open$([w])$.
  (10) If no abort message was received, then the parties output accept.

4. PROTOCOLS FOR LARGE FIELDS & SMALL FIELDS

4.1 PROTOCOLS FOR LARGE FIELDS

对于大的Field，本文的计算协议如下：

PROTOCOL 4.2 (COMPUTING AN ARITHMETIC CIRCUIT OVER FINITE FIELDS).

Let π_{mult} be private semi-honest multiplication protocol. If VERSION 2 is used, then π_{mult} must also be secure up to additive attack.

- **Inputs:** Each party P_j ($j \in \{1, \ldots, n\}$) holds an input $x_j \in \mathbb{F}^{\ell}$.
- **Auxiliary Input:** The parties hold a description of an arithmetic circuit C that computes f on inputs of length $\ell \cdot n$. Let N be the number of multiplication gates in C. In addition, the parties hold a statistical security parameter σ.
- **The protocol:**
  (1) *Precomputation:* Each party sets δ to be the smallest value for which $\delta \geq \sigma / \log(|\mathbb{F}| - 1)$. The parties then run δ executions of Protocol 4.1 with input N, and obtain vectors $\vec{d}_1, \ldots, \vec{d}_\delta$ of N triples.
  (2) *Sharing the inputs:* For each input wire with an input v, the parties run share(v) with the dealer being the party whose input is associated with that wire.
  (3) *Correctness checking of inputs:* Let $[v_1], \ldots, [v_m]$ be the shares on the input wires, generated in the previous step. Repeat δ times: The parties run Protocol 3.1 on $[v_1], \ldots, [v_m]$.
  If there exists an execution in which a party did not output accept, it sends $\perp$ to the other parties and halt.
  (4) *Circuit emulation:* Let $G_1, \ldots, G_L$ be a predetermined topological ordering of the gates of the circuit. For $k = 1, \ldots, L$ the parties work as follows:
    - *If G_k is an addition gate:* Given shares $[x]$ and $[y]$ on the input wires, the parties locally compute $[x + y]$.
    - *If G_k is a multiplication-by-a-constant gate:* Given share $[x]$ on the input wire and a public constant $a \in \mathbb{F}$, the parties locally compute $[a \cdot x]$.
    - *If G_k is a multiplication gate:* Given shares $[x]$ and $[y]$ on the input wires, the parties run π_{mult} on $[x]$ and $[y]$, and define the result as their share on the output wire.

  (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 $\{([x_k], [y_k], [z_k])\}_{k=1}^{N}$ be the triples generated by computing multiplication gates (i.e., $[x_k]$ and $[y_k]$ are the shares on the input wires of the kth multiplication gate and $[z_k]$ is the share on the output wire), and let $\vec{d}_i = \left\{([a_k^i], [b_k^i], [c_k^i])\right\}_{k=1}^{N}$ be the triples generated in ith iteration of the offline phase.
  For $i = 1$ to δ, the parties work as follows:
    - **VERSION 1:**
      For $k = 1, \ldots, N$: The parties run Protocol 3.4 on input $([x_k], [y_k], [z_k])$ and $([a_k^i], [b_k^i], [c_k^i])$ to verify $([x_k], [y_k], [z_k])$. (Observe that all executions of Protocol 3.4 can be run in parallel).
    - **VERSION 2:**
      The parties run Protocol 3.8 on $\{([x_k], [y_k], [z_k])\}_{k=1}^{N}$ and $\left\{([a_k^i], [b_k^i], [c_k^i])\right\}_{k=1}^{N}$ to verify $\{([x_k], [y_k], [z_k])\}_{k=1}^{N}$.

  If a party did not output accept in every execution, it sends $\perp$ to the other parties and outputs $\perp$.
  (6) If any party received $\perp$ in any of the previous steps, then it outputs $\perp$ and halts.
  (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.
  (8) 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.

而预计算生成随机三元组的协议如下：

PROTOCOL 4.1 (GENERATING RANDOM MULTIPLICATION TRIPLES).

Let π_{mult} be a private semi-honest multiplication protocol. If VERSION 2 is used in the main protocol, then π_{mult} must also be secure up to additive attack.

- **Inputs:** The parties have the number N of triples to generate.
- **The protocol:**
  (1) The parties call $\mathcal{F}_{\text{rand}}$ to obtain $2N$ random sharings, arranged in a list of the form $\{([a_i], [b_i])\}_{i=1}^{N}$.
  (2) For $i = 1$ to N: the parties execute π_{mult} on $[a_i]$ and $[b_i]$ to obtain $[c_i]$.
- **Outputs:** The parties output $\{([a_i], [b_i], [c_i])\}_{i=1}^{N}$.

两种验证方式的整体开销如下：

- **Open-based:** $(1+\delta) \cdot t(\pi_{\text{mult}}) + 2\delta \cdot t(\mathcal{F}_{\text{rand}}) + 3\delta \cdot t(\text{open})$
- **Mult-based:** $(1+5\delta) \cdot t(\pi_{\text{mult}}) + 3\delta \cdot t(\mathcal{F}_{\text{rand}})$

当$|\mathbb{F}| > 2^{\sigma}$（例如，$\mathbb{F}_p$中$p$是40比特素数），则可以令$\delta = 1$。如此，Open-based的方法开销为$2 \cdot t(\pi_{\text{mult}}) + 2 \cdot t(\mathcal{F}_{\text{rand}}) + 3 \cdot t(\text{open})$，而Mult-based的方法开销为$6 \cdot t(\pi_{\text{mult}}) + 3 \cdot t(\mathcal{F}_{\text{rand}})$。当$4 \cdot$

$t(\pi_{\mathrm{mult}}) + 1 \cdot t(\mathcal{F}_{\mathrm{rand}})$比$3 \cdot t(\mathrm{open})$更加高效时，Mult-based的方法效率更优。

4.2 PROTOCOLS FOR SMALL FIELDS

对于比较小的FIELD，生成预计算随机三元组采用的是EUCRYPTO'17的cut-and-choose的方法，具体请参考我们之前的分享链接。

6. INSTANTIATIONS

6.1 Shamir's Secret Sharing-based MPC

对于多方计算，本文使用基于Shamir's Secret Sharing的协议实现底层的乘法等原语。为了高效，考虑到参与方的多少，有两种实现($\delta = 1$)：

1. Small Parties：在这种场景下，使用PRSS方法生成随机数，使用GRR-Mult计算乘法门，使用Open-based的方法实现验证。整体的通信每个乘法门需要开销$5(n-1)$个元素；
2. Large Parties：使用基于Vandermonde矩阵的方法生成随机数，使用DN07的方法计算乘法门，使用Mult-based的方法实现验证。整体每个乘法门需要开销42个元素。
3. 因此当$5(n-1) < 42 \Leftrightarrow n \leq 9$时，第一种方法更优；$n \geq 10$时第二种方法性能好。

6.2 Replicated Secret Sharing-based 3PC

对于三方的特殊情况，本文使用了Open-based triple verfication的方法来验证，该方法主要利用RSS的冗余性在三方之间验证view。具体协议如下：

PROTOCOL C.1 (Triple Verification - Three-Parties and Replicated Secret Sharing).

- **Inputs:** The parties hold a triple $([x], [y], [z])$ to verify and an additional random triple $([a], [b], [c])$.
- **The protocol:**
  (1) The parties call $\mathcal{F}_{\mathrm{coin}}$ to receive a random element $\alpha \in \mathbb{F} \setminus \{0\}$.
  (2) Each party locally computes $[\rho] = \alpha \cdot [x] + [a]$ and $[\sigma] = [y] + [b]$.
  (3) The parties run open($[\rho]$) and open($[\sigma]$) as defined in Section 6.2, to receive ρ and σ. If any of the parties received $\perp$ in one of the executions, then it sends $\perp$ to the other parties and aborts.
  (4) Each party locally computes
  $$[v] = \alpha[z] - [c] + \sigma \cdot [a] + \rho \cdot [b] - \rho \cdot \sigma.$$
  Denote by (r_j, s_j) the share of v held by party P_j.
  (5) The parties run the compareview$(r_j + s_j)$ by having each P_j sending $r_j + s_j$ to P_{j+1}. Upon receiving $r_{j-1} + s_{j-1}$ from P_{j-1}, party P_j checks that $r_j = -(r_{j-1} + s_{j-1})$. If yes, it outputs accept. Else, it sends $\perp$ to all the other parties and outputs $\perp$.
  (6) If no abort messages are received, then output accept.

整体的电路计算协议类似协议4.2。

利用PRSS等优化，每个乘法门的通信开销是每方传输4个元素。

7. EVALUATIONS

实验主要验证了不同算法对于不同场景的适用性和效率，展示如下。

Protocol version	3	5	7	9	11	30	50	70	90	110
replicated (3 party)	**513**	-	-	-	-	-	-	-	-	-
PRSS_GRR_open	1229	**1890**	**3056**	6719	18024	-	-	-	-	-
van_GRR_open	1428	2104	3214	**4009**	**5187**	20855	45902	79655	124353	177621
van_DN_open	1999	2661	3463	4426	5694	**15954**	**28978**	**44599**	63522	83815
van_DN_mult	3218	4521	5924	7279	8570	21437	34832	47379	**58966**	**72096**

Table 1: Execution time in milliseconds of the circuit with a 61-bit prime, for different numbers of parties. The best time for each number of parties is highlighted.

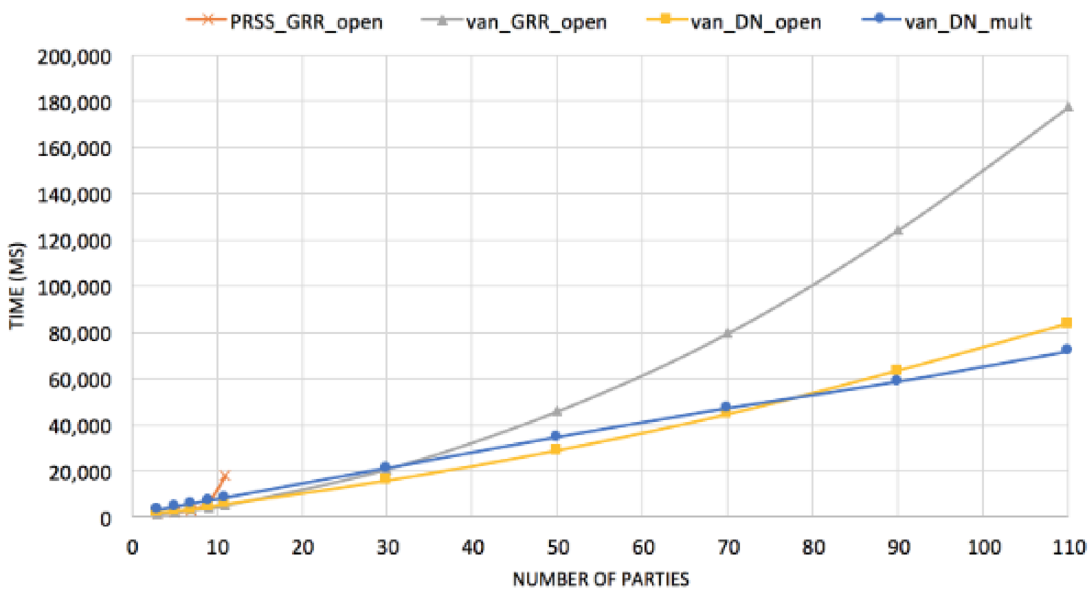

Figure 1: A comparison of the 4 different Shamir-based protocols with a 61-bit prime

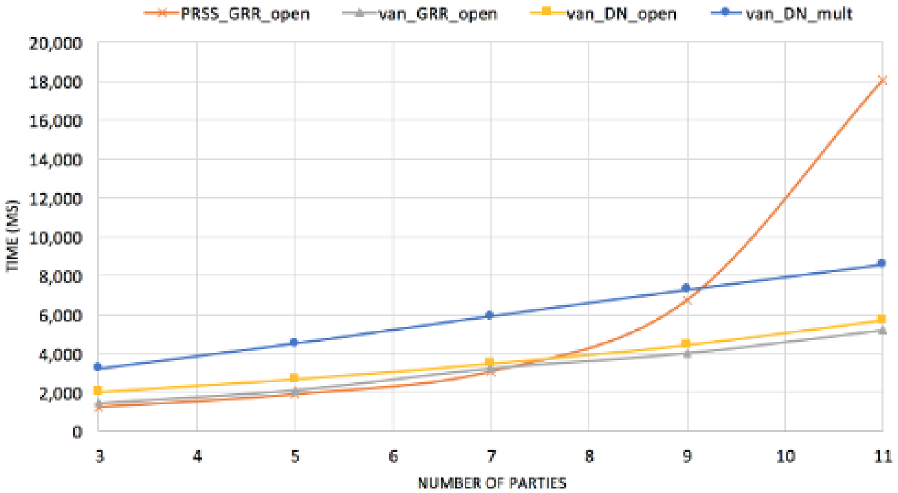

Figure 2: A comparison of the 4 different Shamir-based protocols with a 61-bit prime, for a small number of partiets

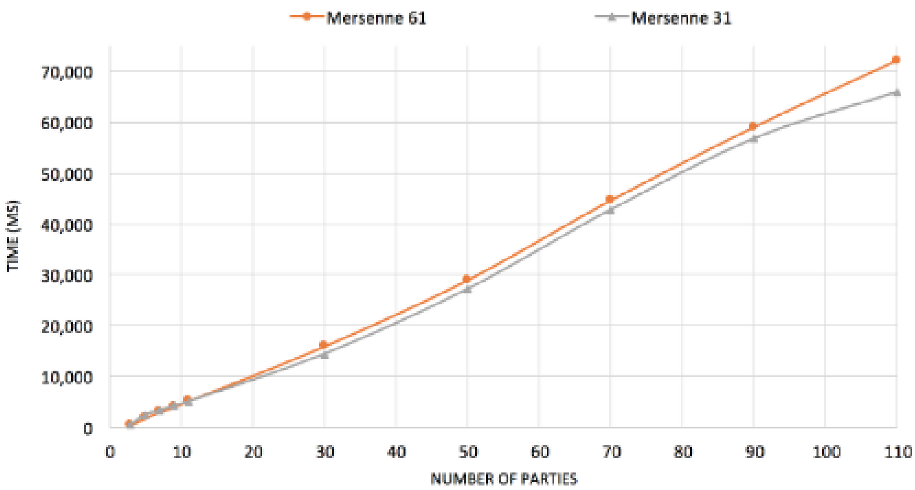

Figure 3: A comparison of the *best* running-times with 31-bit and 61-bit primes, for Shamir-based instantiations

8. Conclusion

本文构造了在Field上的Security with Abort的MPC方案，并做了一系列优化。本次分享只涵盖了整体的协议和构思，以及一些技术关键点。详细的证明和细节优化请参考原文。