Turbospeedz: Double Your Online SPDZ! Improving SPDZ Using Function Dependent Preprocessing

预处理模型在安全多方计算中是一种常用的提升协议在线计算阶段效率的设计方法：预处理阶段，参与方之间生成与输入数据无关的关联随机数，该过程可能需要调用同态加密、茫然传输等开销较大的基础协议；在线计算阶段，参与方利用关联随机数保护实际输入的隐私安全，该过程只需要简单的算术操作。进一步，预处理模型也分为函数无关和函数相关两种，在函数相关预处理中，安全多方计算协议要计算的函数电路是已知的，更有利于提升协议的在线效率。在Turbospeedz中，Ben-Efraim等人利用该技术将SPDZ协议的在线效率提升了2倍。

1. 设计概览

在安全多方计算协议中，任意函数F都可以表示成由加法门和乘法门组成的电路C。加法可以本地计算，乘法则需要调用Beaver三元组。利用Beaver三元组实现乘法时，在线计算阶段需要通信**公开**两个输入的掩码后值，这会造成通信量$O(2)$。本文在函数相关预处理模型下，利用随机掩码技术将**公开**操作从乘法门的*输入*线路转移到*输出*线路，从而减少了2倍的通信。
当然，上述技术会增加预计算的开销，进而增加总的开销。进一步，本文针对SPDZ系列方案中的Overdrive进行了修改，也对预计算进行了优化，使得Turbospeedz的整体通信优于Overdrive。

2. SPDZ回顾

2.1 SPDZ 分享语义（$[[\cdot]]$-分享)

在SPDZ系列协议中有n个参与方，其中最多$n-1$个可以被恶意敌手控制。秘密值$a \in \mathbb{F}$被分享给$\{a_i\}_{i=1}^n$满足$a = \sum_{i=1}^n a_i$；同时秘密全局MAC密钥α也分享为$\alpha = \sum_{i=1}^n \alpha_i$，且$\gamma(a) = \alpha \cdot a$。

$$[[a]] = ((a_1, \ldots, a_n), (\gamma(a)_1, \ldots, \gamma(a)_n))$$

其中P_i拥有$[[a]]_i = (a_i, \gamma(a)_i)$。

2.2 基本运算

给定$a, b, e \in \mathbb{F}$，其中a, b是处于秘密分享$[[a]], [[b]]$，e是公开值：

- $[[a]] + [[b]]$，$e \cdot [[a]]$，$e + [[a]]$是线性运算，即各方可以在本地计算、不需要交互；
- $[[a]] \cdot [[b]]$需要利用Beaver三元组交互计算乘法，在该计算过程中需要公开两个掩码值。

2.3 MACCheck 协议

本文黑盒调用SPDZ的验证协议来保证计算的正确性。该验证失败的概率是$\frac{2}{|\mathbb{F}|}$。当$|\mathbb{F}|$足够大的时候，失败的概率是可忽略的。

2.4 SPDZ 预处理和在线计算

预处理阶段，各方交互生成Beaver乘法三元组$([[a]], [[b]], [[c]])$，平方数对$([[a]], [[c]])$，随机秘密值$[[r]]$，和输入掩码$(r_i, [[r_i]])$。其中r_i公开给P_i。生成上述关联随机数的理想功能简记为$\mathcal{F}_{\text{Prep}}$。
在线计算阶段，输入方P_i首先计算并公开$x_i - r_i$，然后所有参与方本地计算$[[x_i]] \leftarrow [[r_i]] + (x_i - r_i)$。加法门可以本地计算得到$[[z]] = [[x]] + [[y]]$。对于乘法门，多方则需要交互公开$[[\epsilon]] \leftarrow x]] - [[a]]$和$[[\rho]] \leftarrow [[y]] - [[b]]$。进而，每一方在本地计算

$$[[x \cdot y]] \leftarrow [[c]] + \epsilon \cdot [[b]] + \rho \cdot [[a]] + \epsilon \cdot \rho$$

上述Beaver三元组$([[a]], [[b]], [[c]])$可以利用Overdrive中基于同态的方案生成。验证Beaver常用的方案是**牺牲**另外一个三元组进行验证。MASCOT对该验证方案进行了优化，使得用关联随机三元组$([[a]], [[b]], [[c]])$和$([[a]], [[\hat{b}]], [[\hat{c}]])$即可。

3. 基于黑盒调用SPDZ预处理的在线计算改进

Turbospeedz首先黑盒调用SPDZ的预处理过程，生成需要的关联随机数来加速在线计算。本文需要的关联随机数如下：

- 置换值（permutation element）：对于电路C中的线w，生成随机元素λ_w（其秘密分享记作$[[\lambda_w]]$）。置换值在预计算阶段生成，独立于实际输入，且置换值也独立于Beaver乘法三元组。
- 外部值（external value）：在线计算阶段，秘密拥有者计算$e_w \stackrel{def}{=} v_w + \lambda_w$，并公开$e_w$。

加法门可以很直观的验证其可以本地计算：

$$e_z = v_z + \lambda_z = (v_x + v_y) + (\lambda_x + \lambda_y) = (v_x + \lambda_x) + (v_y + \lambda_y) = e_x + e_y$$

对于乘法门，假设已有Beaver乘法三元组$([[a]], [[b]], [[c]])$，本文首先定义了输入偏移（input offsets）：

$$\widetilde{\lambda_x} \stackrel{def}{=} a - \lambda_x$$
$$\widetilde{\lambda_y} \stackrel{def}{=} b - \lambda_y$$

进一步定义调节外部值如下：

$$\hat{e_x} \stackrel{def}{=} e_x + \widetilde{\lambda_x} = (v_x + \lambda_x) + (a - \lambda_x) = v_x + a$$
$$\hat{e_y} \stackrel{def}{=} e_y + \widetilde{\lambda_y} = (v_y + \lambda_y) + (b - \lambda_y) = v_y + b$$

从上式本文有：

$$v_x v_y = (v_x + a)(v_y + b) - a(v_y + b) - b(v_x + a) + ab = \hat{e_x}\hat{e_y} - \hat{e_y}a - \hat{e_x}b + c$$

上式可以用在在线计算阶段计算乘法输出，进一步设置输出线的置换值为$\lambda_z = c + r$（加入r是为了让λ_z满足独立随机）。

类似的，平方函数可以计算如下：

$$(v_x)^2 = (v_x + a)^2 - 2a(v_x + a) + a^2 = (\hat{e_x})^2 - 2\hat{e_x}a + c$$

上述该计算可以利用预计算关联随机数$([[a]], [[c]])$其中$c = a^2$来实现。

鉴于以上推导，黑盒调用SPDZ预计算实现关联随机数生成的协议如下：

Protocol $\Pi_{\mathbf{FDPrep}}$

Initialize: The parties call Π_{Prep} (i.e., a secure implementation of $\mathcal{F}_{\mathrm{Prep}}$) with the number of multiplication gates, squaring gates, and input wires to receive the desired number of input maskings, multiplication triples, squaring pairs, and random $[[\cdot]]$-shared elements.[a] The parties then perform the following local computations in topological order on the gates of the circuit:
Input Wires: For every $i \in [n]$, for each input wire of party i the parties associate an available masking $(r_i, [[r_i]])$ (i.e., a random $[[\cdot]]$-shared element revealed to party i).
Addition gates: On an addition gate with input permutation element shares $[[\lambda_x]]$ and $[[\lambda_y]]$ the parties locally compute the output permutation element shares $[[\lambda_z]] \leftarrow [[\lambda_x]] + [[\lambda_y]]$.
Multiplication gates: On a multiplication gate with input permutation element shares $[[\lambda_x]]$ and $[[\lambda_y]]$ the parties assign to the gate the next available multiplication triple $([[a]], [[b]], [[c]])$ and the next available $[[\cdot]]$-shared random element $[[r]]$, and locally compute:

- $[[\cdot]]$-shares of the offset values $[[\widetilde{\lambda_x}]] \leftarrow [[a]] - [[\lambda_x]]$ and $[[\widetilde{\lambda_y}]] \leftarrow [[b]] - [[\lambda_y]]$.
- $[[\cdot]]$-shares of the permutation element on the output wire $[[\lambda_z]] \leftarrow [[c]] + [[r]]$.

Output: After performing all the above local computation, the parties partially reveal the offset values $\widetilde{\lambda_x}, \widetilde{\lambda_y}$ for every multiplication gate and $\widetilde{\lambda_x}$ for every squaring gate.
Output verification: This procedure is entered once the parties have finished the above function dependent preprocessing phase.
The parties call the MACCheck protocol with the input being all the opened values so far. If MACCheck fails, they output ϕ and abort, otherwise they accept the partially opened offset values.

[a] Recall that we require an additional $[[\cdot]]$-shared element for each multiplication/squaring gate.

Fig. 1. Our new function dependent preprocessing protocol

而本文提出在线优化协议如下：

Protocol Π_{Online}

Initialize: The parties call $\mathcal{F}_{\text{FDPrep}}$ with the circuit to receive the masking values at each of the input wires and the random elements, the Beaver triples/squaring pairs, and the revealed offsets at each of the multiplication/squaring gates.

Input: To share his input v_{x_i}, party i takes the associated mask value $(r_i, [[r_i]])$ and does the following:

- Broadcast $e_{x_i} = v_{x_i} + r_i$.
- The parties compute $[[v_{x_i}]] \leftarrow e_{x_i} - [[r_i]]$.
- The parties store e_{x_i} as the external value on the wire, and $[[r_i]]$ as shares of the permutation element, λ_{x_i}.

Add: On an addition gate with input wires x, y, input shared values $([[v_x]], [[v_y]])$, input shared permutation elements $([[\lambda_x]], [[\lambda_y]])$, and input external values (e_x, e_y), the parties locally compute the following for the output wire z:

1. $[[v_z]] \leftarrow [[v_x]] + [[v_y]]$
2. $[[\lambda_z]] \leftarrow [[\lambda_x]] + [[\lambda_y]]$[a]
3. $e_z \leftarrow e_x + e_y$

Multiply: On a multiplication gate with input shared values $([[v_x]], [[v_y]])$ and input external values (e_x, e_y), the parties take the associated multiplication triple $([[a]], [[b]], [[c]])$, the partially opened offsets $\widetilde{\lambda_x}, \widetilde{\lambda_y}$, and the associated random $[[\cdot]]$-shared value $[[r]]$ and perform the following steps:

1. Locally compute:
   - $\widehat{e_x} \leftarrow e_x + \widetilde{\lambda_x}$ and $\widehat{e_y} \leftarrow e_y + \widetilde{\lambda_y}$.
   - $[[v_z]] \leftarrow \widehat{e_x}\widehat{e_y} - \widehat{e_y}[[a]] - \widehat{e_x}[[b]] + [[c]]$.[b]
   - $[[\lambda_z]] \leftarrow [[c]] + [[r]]$.[a]
2. Partially open $[[e_z]] \leftarrow [[v_z]] + [[\lambda_z]]$.

Output: This procedure is entered once the parties have finished the circuit evaluation, but still the final output v_z has not been opened.

1. The parties call the MACCheck protocol with the input being all the opened values so far in the online phase. If MACCheck fails, they output ϕ and abort.[c]
2. The parties open v_z and call MACCheck with input v_z, to verify its MAC. If the check fails, they output ϕ and abort, otherwise they accept v_z as a valid output.

---

[a] This computation was performed at the function dependent offline phase.

[b] By Equation (7), v_z holds the value $v_x \cdot v_y$.

[c] ϕ represents that the corrupted parties remain undetected.

Fig. 2. Our new online phase protocol

可以看到，本文在线计算协议只需要公开e_z一个元素。

4. 针对Overdirve的预处理改进

为了提升预计算的效率，本文在Overdirive的基础上提出了如下优化：

- 令输出线的置换元素等于乘法三元组中的c可以节省开销，但是这不安全，因为置换元素必须是独立随机的。从另一个角度，本文做如下优化：如果一条输出线是下一层乘法的输入，那么令输出的随机置换为下一层乘法对应的乘法三元组。因为三元组中的a, b是随机独立的，所以这样是安全的。
- 本文令同一条线的三元组中的a相同，即便该线可能输入不同的乘法门。很显然，该优化不可以再黑盒调用SPDZ，因为SPDZ对于每一个乘法门独立随机生成三元组。
- 鉴于上述优化，第三节中定义的$\hat{\lambda_x}$和$\hat{\lambda_y}$将为0，这会进一步简化协议设计和实现。

基于上述优化的预计算协议如下：

Protocol Π_{FDTriple}

Initialize: Each party P_i randomly chooses $\lambda_{\omega,i} \leftarrow \mathbb{F}$ for every wire ω that is an input wire of the circuit[a] or an output wire of a multiplication gate. Then, in topological order on the circuit, for each addition gate with input wires x, y and output wire z, party P_i computes $\lambda_{z,i} = \lambda_{x,i} + \lambda_{y,i}$.

Multiplication: For each multiplication gate with input wires x, y

1. Each party P_i sets $a_i = \lambda_{x,i}, b_i = \lambda_{y,i}$ and randomly selects $\hat{b}_i \leftarrow \mathbb{F}$.
2. Every unordered pair (P_i, P_j) executes the following 2-party multiplication as in [33]:
   (a) P_i sends P_j the encryption $\text{Enc}_i(a_i)$.[b]
   (b) P_j responds with $C^{(ij)} = b_j \cdot \text{Enc}_i(a_i) - \text{Enc}_i(e^{(ij)})$ for a random $e^{(ij)} \leftarrow \mathbb{F}$.[b]
   (c) P_i decrypts $d^{(ij)} = \text{Dec}_i(C^{(ij)})$.
   (d) The last two steps are repeated with $\hat{b}_j$ to get $\hat{e}^{(ij)}$ and $\hat{d}^{(ij)}$.
3. Each party P_i computes $c_i = a_i b_i + \Sigma_{i \neq j}(e^{(ji)} + d^{(ij)})$, and $\hat{c}_i$ similarly.

Authentication: Each party P_i inputs to $\mathcal{F}_{[[]]}$ the shares $\lambda_{\omega,i}$ for each wire ω that is an input wire of the circuit or an output wire of a multiplication gate, and additionally the shares $c_i, \hat{b}_i$, and $\hat{c}_i$ for each multiplication gate; for each such value the functionality outputs to the parties $[[\lambda_\omega]]$, $[[c]]$, $[[\hat{b}]]$, or $[[\hat{c}]]$, respectively, where $\lambda_\omega = \Sigma_i \lambda_{\omega,i}$, etc.

Sacrifice: The parties do the following for each multiplication triple pair $([[a]], [[b]], [[c]]), ([[a]], [[\hat{b}]], [[\hat{c}]])$:

1. Call $r \leftarrow \mathcal{F}_{\text{RAND}}$.
2. Compute and partially open $[[\rho]] = r[[b]] - [[\hat{b}]]$.[c]
3. Compute and partially open $[[\tau]] = r \cdot [[c]] - [[\hat{c}]] - \rho \cdot [[a]]$.[c] If $\tau \neq 0$ then abort.

MACCheck and Output: Run MACCheck on all opened values. If the check fails then abort. Otherwise, output all non-sacrificed computed triples $([[a]], [[b]], [[c]])$.

---

[a] The value $\lambda_\omega = \Sigma_i \lambda_{\omega,i}$ is the random value used in Π_{Online} for the input distribution, and therefore is later partially revealed to the relevant party.

[b] For simplicity, the details of the zero-knowledge proofs and of the noise drowning have been omitted; the complete details of this 2-party protocol can be found at [33, Figure 7].

[c] Note that a and b are linear combinations of the permutation elements λ_ω input to $\mathcal{F}_{[[]]}$, and thus $[[a]]$ and $[[b]]$ can be locally computed by the parties.

Fig. 3. Our modified triple generation protocol

5. 总结

本文利用电路已知条件，基于掩码秘密分享技术进一步提升了在线计算的性能。一般的，该掩码秘密分享可以进一步迁移到任意秘密分享提升在线计算性能。相关的文章有ABY2.0，Meteor，和ScoinFL等。