MOTION-A Framework for Mixed-Protocol Multi-Party Computation

本次介绍的是Braun等人发表在TOPS'22上的MOTION。论文链接MOTION。
之前的 ABY 和 ABY2.0 实现了在两方下的三种秘密分享（**A**rithmetic-Sharing，**B**oolean-Sharing 和 **Y**ao's-Sharing）的转化，在MOTION中，Braun等人实现了面向多方下dishonest majority场景的三种秘密分享的转化，并提出了一些协议和实现优化来提升性能，和之前的方案的对比见表1。

Table 1. Related mixed-protocol MPC frameworks with N parties, threshold t, and active (●) or passive (◑) security. We denote the license of unpublished source code as '—'.

Framework	N	t	Security	Protocols	License
ABY [32]	2	1	◑	$A/B/Y$	LGPL-3.0
PrivC [43]	2	1	◑	A/Y	—
TASTY [44]	2	1	◑	A/Y	no license
EzPC [21]	2	1	◑	A/Y	MIT
OPA Mixing [48]	2	1	◑	any 2 of $A/B/Y$	MIT
ABY^3 [60]	3	1	● or ◑	$A/B/Y$	MIT
Sharemind [14]	3	1	● or ◑	A/B	payware[3]
ASTRA [22]	3	1	● or ◑	A/B	—
BLAZE [64]	3	1	● or ◑	A/B	—
Trident [66]	4	1	●	$A/B/Y$	—
SCALE-MAMBA [2]	≥ 2	$N-1$	●	A/Y	MIT-like
MP-SPDZ [50]	≥ 2	$N-1$	● or ◑	A/B or Y	MIT-like
MOTION (this work)	≥ 2	$N-1$	◑	$A/B/Y$	MIT

大致上，MOTION做了如下主要工作：

- MOTION使用面向多方的GMW协议的Arithmetic和Boolean变体实现A-Sharing和B-Sharing，而对于多方的Y-Sharing，则使用BMR协议；
- 进一步，MOTION实现了A/B/Y三种秘密分享的转化；
- 而对于底层的Beaver Triples (MTs) 生成等基础协议，则使用基于OT的方式来实现。

本次，我们将介绍的重点放在三种技术的介绍和协议层面的优化。对于实现层面的优化，则不会过多的涉及。

1. MPC Building Blocks

本文用到的notations都是MPC方案中常见的表达形式，在这里不做过多的展开。我们从原文的Section 5 MPC Building Blocks部分直接开始。

1.1 Oblivious Transfer (OT)

本文利用OT构造底层的基本模块，使用OT Extension来提升性能。并且，本文还使用了Random-OT和C-OT来加速MTs的生成和乘法的计算。关于OT的上述相关知识详见知乎不经意传输(OT)-总结。

这里，我们介绍一下本文提出的对C-OT的优化：Braun等人发现，在C-OT中，虽然根据$\mathcal{R}$的修正比特，$\mathcal{S}$的传输消息的计算方式不一样，但是两种计算的结果却是一样的。为了方便描述C-OT的预计算过程，此处作者用R-OT进行描述。具体来说：

- 双方首先计算R-OT，其中$\mathcal{R}$的选择比特是 $r \in_R \{0,1\}$；
- 对于$\mathcal{R}$的实际输入 c，如果 $r=c$，$\mathcal{R}$ 令 $p=0$，否则 $p=1$。并将 p 发送给 $\mathcal{S}$;
- $\mathcal{S}$ 获得 p 之后，如果 $p=1$ 则调换自己两条信息的位置，否则位置不变。进而发送消息给对方。

但是在C-OT中，$\mathcal{S}$在 $p=0/1$两种情况下发送的消息是完全一样的：

$$y_i = H(i, \boldsymbol{V}'[i]) \odot x_i \odot H(i, \boldsymbol{V}'[i] \oplus c) = H(i, \boldsymbol{V}'[i] \oplus c) \odot x_i \odot H(i, \boldsymbol{V}'[i])$$

因此，双方可以独立并行传输自己的消息给对方，从而减少一半的延迟。

1.2 Beaver Triples

Beaver MTs 能够用在A-Sharing和B-Sharing中用来计算乘法/AND门。下面我们以A-Sharing下的A-MTs为例进行介绍。
多方场景下，每一个参与方P_i本地生成两个随机的分享 $\langle a\rangle_i^A$ 和 $\langle b\rangle_i^A \in_R \mathbb{Z}_{2^\ell}$，进而多方计算生成 $\langle c\rangle^A = \langle a\rangle^A \langle b\rangle^A$。因为

$$a \cdot b = (\sum_i \langle a\rangle_i^A) \cdot (\sum_i \langle b\rangle_i^A) = \sum_i (\langle a\rangle_i^A \cdot \langle b\rangle_i^A) + \sum_{i,j\neq i} (\langle a\rangle_i^A \cdot \langle b\rangle_j^A) \pmod{2^\ell},$$

P_i在本地计算 $\sum_i (\langle a\rangle_i^A \cdot \langle b\rangle_i^A)$，然后$P_i$和$P_j$两方计算$\langle a\rangle_i^A \langle b\rangle_j^A$ $(i \neq j)$。两方基于C-OT实现xy，其中$x = \langle a\rangle_i^A, y = \langle b\rangle_j^A$。在第$k$次COT：

- P_i作为发送方输入 x，得到$r_k \in \mathbb{Z}_{2^\ell}$；
- P_j作为接收方，输入比特$\boldsymbol{y}[k]$，得到$r_k + \boldsymbol{y}[k] \cdot x$，其中$\boldsymbol{y}$表示对$y$进行分解之后得到的比特向量；
- 最终，P_i令 $z_i = -\sum k = 1^\ell r_k 2^{k-1} \pmod{2^\ell}$，$P_j$令 $z_j = \sum_{k=1}^{\ell} 2^{k-1}(r_k + y[k] \cdot x) \pmod{2^\ell}$，从而使得 $z_i + z_j = x \cdot y$。

由于模约减，可以在COT中通过舍弃高k比特的方式减少一半的通信量。一共需要两轮通信，总通信量为 $\approx N(N-1)\ell(\kappa + \ell/2)$ 比特。

对于B-MTs，处理方法和A-MTs类似。总通信量为 $N(N-1)(\kappa+1)$比特，一共需要两轮通信。

1.3. Square Pairs (SPs)

SPs是指秘密分享下的 $(\langle a\rangle^A, \langle c\rangle^A)$，满足 $c = a^2$。其生成方法和MTs类似，除了我们只需要任意两方做一次乘法，因此通信量是生成MTs的一半，通信轮数也是2轮。

1.4. Shared Bits (SBs)

SBs指$(\langle b\rangle^B, \langle b\rangle^A)$，其中$b \in \{0,1\}$。给定$\langle b\rangle^A$，可以通过 $\langle b\rangle_i^B = \langle b\rangle_i^A \pmod 2$来计算$\langle b\rangle^B$。我们基于DET+17中的协议 Π_{RandBit}生成SBs。我们利用$(\langle b\rangle^B, \langle b\rangle^A)$实现 $B \to A$的转化。生成SBs需要3轮通信，通信量为$N(N-1)(\ell+2)(\mathrm{K} + \ell/2 + 3)/2$。

1.5. Extended Shared Bits (ESBs)

进一步，ESBs指$(\langle \boldsymbol{r} \rangle^S, \langle r \rangle^A)$，其中 $(S \in \{B, Y\}), r = \sum_{k=0}^{\ell-1} 2^k \boldsymbol{r}[k]$。生成ESBs的方法有两种：

1. 基于SBs：首先生成ℓ个SBs：$(\langle r_k \rangle^B, \langle r_k \rangle^A)$ 其中$k = 0, ..., \ell - 1$。这样一来可以令$\langle \boldsymbol{r} \rangle^B = (\langle r_0 \rangle^B, ..., \langle r_{\ell-1} \rangle^B)$，$\langle r \rangle^A = \sum_{k=0}^{\ell} \langle r_k \rangle^A$。如果要得到 $(\langle \boldsymbol{r} \rangle^Y, \langle r \rangle^A)$，则需要对 $\langle r_k \rangle^B$ 进行 $B \to Y$ 转化。
2. 基于加法电路：在该方法下，首先令P_i本地生成随机$\langle r \rangle_i^A \in_R \mathbb{Z}_{2^\ell}$，从而构成$\langle r \rangle^A = \sum_{j=0}^{N} \rangle r \rangle_j^A$。然后每一方将自己的$\langle r \rangle_j^A$ 进行B-或者Y-Sharing，进而计算$N-1$个布尔电路下的加法得到$\langle \boldsymbol{r} \rangle^B$ 或者 $\langle \boldsymbol{r} \rangle^Y$。进行布尔加法时，不同的电路通信量和通信开销不同，一般常用的有RCA、PPA。

各个子模块的具体开销见表4。

1.6. Other Ooptimizations

本文还使用如下方法进行优化：

- Fixed-Key AES做哈希函数和伪随机函数；
- 在N方下，多个参与方如果要互相公开share，并在本地进行share的累加，在半诚实敌手下则可以通过指定一方做累加，然后将累加结果公布给所有参与方，从而将通信从$N(N-1)\ell$降低到$2(N-1)\ell$比特，但是增加了一轮交互。

Table 4. Total costs of the generation of different kinds of correlated randomness (defined in §5.2 – §5.5) number of symmetric cryptographic operations, number of bits sent by all parties, and number of communication rounds. For ESBs we list the costs of generating them from SBs, and the costs using a summation circuit in either B or Y sharing, which is constructed from different kinds of adders: ripple-carry (RCA), Ladner-Fischer (LFA), optimized Brent-Kung (BKA), and optimized Sklansky (SA) [20].

	Computation [# symm. crypt. ops]	Communication [# bits]	# Rounds
B-MT	$2N(N-1)$	$N(N-1)(\kappa+1)$	2
A-MT	$2N(N-1)\ell$	$N(N-1)\ell(\kappa+\ell/2)$	2
SP	$N(N-1)\ell$	$N(N-1)\ell(\kappa+\ell/2)/2$	2
SB	$N(N-1)(\ell+2)$	$N(N-1)(\ell+2)(\kappa+\ell/2+3)/2$	3
ESB^B from SBs	$N(N-1)\ell(\ell+2)$	$N(N-1)\ell(\ell+2)(\kappa+\ell/2+3)/2$	3
ESB^B w/ RCA in B	$2(N\ell-N+\ell)(N-1)N$	$(N-1)N^2(\kappa+3)(\ell-1)$	$(\ell-1)\lceil\log_2 N\rceil+2$
ESB^B w/ LFA in B	$(\frac{5}{2}N\lceil\log_2 \ell\rceil+2N+2)(N-1)N\ell$	$(N-1)N^2(\kappa+3)\ell(\frac{5}{4}\lceil\log_2 \ell\rceil+1)$	$(2\lceil\log_2 \ell\rceil+1)\lceil\log_2 N\rceil+2$
ESB^B w/ BKA in B	$2(3N+1)(N-1)N\ell$	$3(N-1)N^2(\kappa+3)\ell$	$(2\lceil\log_2 \ell\rceil-1)\lceil\log_2 N\rceil+2$
ESB^B w/ SA in B	$2(N\lceil\log_2 \ell\rceil+1)(N-1)N\ell$	$(N-1)N^2(\kappa+3)\ell\lceil\log_2 \ell\rceil$	$(\lceil\log_2 \ell\rceil+1)\lceil\log_2 N\rceil+2$
ESB^Y w/ RCA in Y	$(9N-8)N^2(\ell-1)$	$(4N^2\kappa\ell-4N^2\kappa+5N\kappa\ell-4N\kappa+N\ell-N+\ell)(N-1)N$	8

2. MPC Protocols

在本章节，我们介绍A-, B-, Y-Sharing三种秘密分享各自的计算协议。

2.1 A-Sharing

给定x，$\text{Share}_i^A(x)$将x分享为 $x = \sum_{j=1}^{N} \langle x \rangle_j^A$。具体来说，拥有$x$的$P_i$可以1）在本地生成份额然后发给其他方，2）也可以使用伪随机函数实现0通信。秘密恢复$\text{Rec}^A(\langle x \rangle^A)$则是需要得到所有份额然后恢复$x$。对于线性操作（包括加法和明文密文乘法），可以类似两方计算不需要交互就、本地计算得到结果。而乘法则需要消耗一个MTs，通过交互得到。具体来说：

- 令 $(\langle a\rangle^A, \langle b\rangle^A, \langle c\rangle^A)$ 表示 MTs，满足$c = ab$;
- 为了计算 $\langle z\rangle^A = \langle x\rangle^A \cdot \langle y\rangle^A$，各方本地计算 $\langle d\rangle_i^A = \langle x\rangle_i^A - \langle a\rangle_i^A$ 和 $\langle e\rangle_i^A = \langle y\rangle_i^A - \langle b\rangle_i^A$，并恢复 d, e;
- 最后，各方在本地计算 $\langle z\rangle_i^A = \langle c\rangle_i^A + e \cdot \langle x\rangle_i^A + d \cdot \langle y\rangle_i^A - d \cdot e$。

在恢复d, e的时候，可以采用Sec 1.6 中提到的第二种优化减少通信量（但是增加了轮数）。

平方计算可以利用SPs通过类似的方法得到：$\langle z\rangle^A = \langle c\rangle^A + 2 \cdot d \cdot \langle x\rangle^A - d^2$，其中 $d = x - a$，$(\langle a\rangle^A, \langle c\rangle^A)$是SPs($c = a^2$)。

2.2 B-Sharing

对于B-Sharing， A-Sharing下的加法和乘法替换为$\oplus$和$\wedge$操作，计算方法在比特级别。算法原理类似。

More Efficient AND Gates without MTs: 本文利用优化的COT提出了一种不需要MTs的更加高效的乘法，简单来说是利用COT直接计算AND门下的交叉比特$\wedge$操作。该方法和基于MTs的方法相比，通信量：预计算节省了4比特，在线计算相同；通信轮数，在线计算相同，预计算提升了$2\times$。

2.3 Y-Sharing

多方下的混乱电路协议是BMR协议，该方法的基本思想是多方在预计算交互，构造混乱电路。然后在线计算阶段只需要本地计算即可。和两方计算不同，BMR中任意一方都不能拥有构造混乱表的全部的keys明文。

Setup:
在混乱电路构造阶段，每一个参与方P_i本地生成global key offset $R^i \in_R \{0,1\}^\kappa$，随机置换比特的分享λ_w^i（$\lambda_w = \bigoplus_{j=1}^N \lambda_w^j$），针对输入线$w$两种可能值的两个keys $k_{w,0}^i, k_{w,1}^i$:

- 如果P_j提供实际输入，那么当$i = j$时，有$\lambda_w^i \in_R \{0,1\}$；否则，$\lambda_w^i = 0$;
- 如果w对应公开值，$\lambda_w^i = 0$ ($i=1,...,N);
- 如果w不是XOR门的输出，那么$\lambda_w^i \in_R \{0,1\}$且$k_{w,0}^i \in_r \{0,1\}^\kappa$；如果$w$是XOR的输出且输是$a, b$，那么$\lambda_w^i = \lambda_a^i \oplus \lambda_b^i$且$k_{w,0}^i = k_{a,0}^i \oplus k_{b,0}^i$。两种情况下，都有 $k_{w,1}^i = k_{w,0}^0 \oplus R^i$。

$\mathcal{F}_{GC}$的garbling过程如下：对于每一个AND门g，对于输入线a, b，混乱表项$\widetilde{g}_{\alpha,\beta}^j$计算如下：

$$\widetilde{g}_{\alpha,\beta}^j = (\bigoplus_{i=1}^N F_{k_{a,\alpha}^i, k_{b,\beta}^i}^2 (g \circ j)) \oplus k_{w,0}^j \oplus (R^j \cdot ((\lambda_a \oplus \alpha)(\lambda_b \oplus \beta) \oplus \lambda_w)),$$

对于所有的g和$\alpha, \beta \in \{0,1\}$，其输出$\widetilde{g}_{\alpha,\beta}^1 \circ \widetilde{g}_{\alpha,\beta}^2 \circ \cdots \circ \widetilde{g}_{\alpha,\beta}^N$。
MOTION使用OT实现底层的计算：

- 多方使用1比特的COT计算得到$\lambda_{ab} = \lambda_a \cdot \lambda_b$，从而$P_j$获得share$\lambda_{ab}^j$;
- 进而，每一方本地计算$\lambda_{a\bar{b}w} = \lambda_a \cdot \bar{\lambda_b} \oplus \lambda_w$，该计算可以令$P_j$本地计算$\lambda_{a\bar{b}w}^j = \lambda_{ab}^j \oplus \lambda_a^j \oplus \lambda_w^j$。对于$\lambda_{\bar{a}bw}^j$和$\lambda_{\bar{a}\bar{b}w}^j$计算类似;
- 第三步，多方调用k-比特的COT计算得到$(R^j \cdot ((\lambda_a \oplus \alpha)(\lambda_b \oplus \beta) \oplus \lambda_w))$对于$j = 1, .., N$和$\alpha, \beta \in \{0,1\}$。记该步$P_i$的结果为$\rho_{j,\alpha,\beta}^i$;
- 如此一来，最后的混乱表中，P_j持有的表项$\{\widetilde{g}_{0,0}^j, \widetilde{g}_{0,1}^j, \widetilde{g}_{1,0}^j, \widetilde{g}_{1,1}^j\}$可以构造如下：
  - P_j广播 $F_{k_{a,\alpha}^j, k_{b,\beta}^j}^2 (g \circ j)) \oplus k_{w,0}^j \oplus \rho_{j,\alpha,\beta}^j$;

- P_i广播 $F^2_{k^i_{a,\alpha},k^i_{b,\beta}}(g\circ j))\oplus\rho^i_{j,\alpha,\beta}$。

- 上述广播的消息做XOR即可得到$\widetilde{g}^j_{\alpha,\beta}$。
  另外一种优化方法则是通过计算

$$\overbrace{(F^2_{k^j_{a,\alpha},k^j_{b,\beta}}(g\circ j))\oplus k^j_{w,0}\oplus R^j\oplus\rho^j_{j,0,0}\oplus\rho^j_{j,0,1}\oplus\rho^j_{j,1,0})}^{P_j}\bigoplus\underbrace{(F^2_{k^i_{a,\alpha},k^i_{b,\beta}}(g\circ j)\oplus\rho^i_{j,0,0}\oplus\rho^i_{j,0,1}\oplus\rho^i_{j,}}_{P_i}$$

  得到 $\widetilde{g}^j_{1,1}$, 可以不用计算$\rho_{j,1,1}$。

Online: 在线计算阶段，任意参与方P_i持有$\alpha_w=\lambda_w\oplus x$（其中$x$是实际输入），keys $k^j_{w,\alpha}$ $(j=1,..,N)$ 和 分享 λ^i_w。表示为

$$\langle x\rangle^Y=(\overbrace{\lambda^1_x,...,\lambda^N_x}^{private};\underbrace{(\alpha,k^1_{x,\alpha},\ldots,k^N_{x,\alpha})}_{public})$$

在线阶段，秘密拥有者P_i首先广播$\alpha=x\oplus\lambda$；然后每一方广播k^j_α。给定$\langle x\rangle^Y=(\lambda^1_x,...,\lambda^N_x;(\alpha,k^1_{x,\alpha},\ldots,k^N_{x,\alpha})$ 和 $\langle y\rangle^Y=(\lambda^1_y,...,\lambda^N_y;(\beta,k^1_{y,\beta},\ldots,k^N_{y,\beta})$，可以利用free-XOR技术本地计算XOR：

- $\gamma=\alpha\oplus\beta$;
- $k^j_{z,\gamma}=k^j_{x,\alpha}\oplus k^j_{y,\beta}$
- $\langle z\rangle^Y=(\lambda^1_z,...,\lambda^N_z;(\gamma,k^1_{z,\gamma},\ldots,k^N_{z,\gamma})$

AND门则需要解密得到$k^j_{z,\gamma}=\widetilde{g}^j_{\alpha,\beta}\oplus\bigoplus_{i=1}^N F_{k^i_{x,\alpha},k^i_{y,\beta}}(g\circ j)$，通过对比$k^i_{z,\gamma}=k^i_{z,0}/k^i_{z,1}$ 推断γ的值。
其余的优化可以参考原文。

上述各部分的开销如下：

Table 2. Total costs of primitive operations: number of symmetric cryptographic operations, number of bits sent by all parties, and number of communication rounds for one operation.

	Computation [# symm. crypt. ops]		Communication [# bits]		# Rounds	
	Setup	Online	Setup	Online	Setup	Online
ADD^A, XOR^B, XOR^Y	0	0	0	0	0	0
MUL^A	$2\ell N(N-1)$	0	$\ell N(N-1)(\kappa+\ell/2)$	$2\ell N(N-1)$	2	1
AND^B	$2N(N-1)$	0	$N(N-1)(\kappa+1)$	$2N(N-1)$	2	1
AND^Y	$8N(N-1)$	N^2	$N(N-1)((N+1)4\kappa+1)$	0	6	0
Share^A, Share^B	$2(N-1)$	0	0	0	0	0
Share^Y	0	0	0	$(N\kappa+1)(N-1)$	0	2
Rec^A	0	0	0	$\ell N(N-1)$	0	1
Rec^B	0	0	0	$N(N-1)$	0	1
Rec^Y	0	0	$N(N-1)$	0	1	0

3. MPC Conversions

本章节介绍A/B/Y三种协议多方下的转化：

3.1 B2Y

最直观的方法是利用Y-Sharing的分享方法直接分享$\langle x\rangle_i^B$然后在Y-Sharing下做XOR。本文提出了对于通信量的优化方案：

- P_i广播$\alpha_i = \langle x\rangle_i^B \oplus \lambda_w^i$；
- P_i 计算$\alpha = \bigoplus_{j=1}^N \alpha_j$，并广播$k_{w,\alpha}^i$；
- 令$\langle x\rangle^Y = (\lambda_x^1, ..., \lambda_x^N; (\alpha, k_{x,\alpha}^1, \ldots, k_{x,\alpha}^N)$

不难验证，$\alpha = \bigoplus_{j=1}^N \alpha_j = \bigoplus_{j=1}^N \langle x\rangle_j^B \oplus \lambda_w^i = x \oplus \lambda_w$。如此，通行量提升了$N\times$。

3.2 Y2B

和2PC下一样Y2B也是free的：因为$\alpha = x \oplus \lambda_x$。由于$a$是公开的，$\lambda_x$是B-Sharing的，令$P_1$计算$\langle x\rangle_1^B = \lambda_1 \oplus \alpha$，$P_j$ $(j = 2, ..., N)$设置$\langle x\rangle_j^B = \lambda_j$即可得到$\langle x\rangle^B$。

3.3 B2A

Straightforward: Additive Masking: 该方法直接利用ESBs：$(\langle r\rangle^A, \langle \boldsymbol{r}\rangle^B)$，计算：

- $\langle \boldsymbol{t}\rangle^B = \langle \boldsymbol{x}\rangle^B - \langle \boldsymbol{r}\rangle^B$；
- $t = \mathsf{Rec}^B(\langle \boldsymbol{t}\rangle^B)$；
- $\langle x\rangle^A = t + \langle r\rangle^A$

但是该方法需要在B-Sharing做减法电路，需要$\Omega(\log_2 \ell) + 1$通信轮数。

Optimized: Using Shared Bits: 该方法对于每一比特利一个SBs，将每一比特转化为A-Sharing之后，再本地加权求和即可$\langle x\rangle^A = \sum_{i=0}^{\ell-1} 2^i \cdot \langle x_i\rangle^A$。通信轮数降低为1。

3.4 A2Y

比较直观的方法是每一方将自己的share重新分享为Y-Sharing，然后在Y-Sharing下做$N-1$次加法。本文的优化方法是利用ESBs $(\langle \boldsymbol{r}\rangle^Y, \langle r\rangle^A)$，计算如下：

- $t = \mathsf{Rec}^A(\langle x\rangle - \langle r\rangle^A)$；
- $\langle \boldsymbol{t}\rangle^Y = \mathsf{Share}^Y(t)$；
- $\langle \boldsymbol{x}\rangle^Y = \langle \boldsymbol{t}\rangle^Y + \langle \boldsymbol{r}\rangle^Y$。
  上述方法只需要一次加法电路。

3.5 A2B

两种方法实现A2B：1）使用加法电路直接将在布尔电路下做加法，通信轮数为$O(\log_2 \ell)$；2）先做A2Y，然后做Y2B（free）。

3.6 Y2A

简单是方法是先做Y2B，然后做B2A。但是该方法需要一轮在线通信。本文的优化方法可以利用ESBs $(\langle \boldsymbol{r} \rangle^Y, \langle r \rangle^A)$：

- $\langle \boldsymbol{t} \rangle = \langle \boldsymbol{x} \rangle^Y - \langle \boldsymbol{r} \rangle^Y$;
- $t = \mathsf{Rec}^Y(\langle \boldsymbol{t} \rangle^Y)$;
- $\langle x \rangle^A = t + \langle r \rangle^A$。

  根据BMR的特性，上述Step 1 & 2 不需要在线通信，而A-Sharing下的加法也不要通信。因此，该方法不需要在线通信。

各协议的复杂度如下：

Table 3. Costs of conversion operations: number of symmetric cryptographic operations, number of bits sent by all parties, and number of communication rounds for one conversion. For those marked with "excl. ESB^Y" the costs to generate ESBs needs to be added to optain the total cost in the setup phase. For those marked with "incl. ESB^Y" we used as an example the variant with the optimized Brent-Kung (BKA) in B sharing combined with a $B2Y$ conversion to generate the ESBs.

	Computation [# symm. crypt. ops]		Communication [# bits]		# Rounds	
	Setup	Online	Setup	Online	Setup	Online
$Y2B$	0	0	0	0	0	0
$B2Y$	0	0	0	$N(N-1)(\kappa+1)$	0	2
$B2A$ (w/ SBs), $Y2A$ (via B)	$\ell N(N-1)(\ell+2)$	0	$\ell N(N-1)(\ell+2)(\kappa+\ell/2+3)/2$	$N(N-1)\ell$	3	1
$Y2A$ (w/o online comm., excl. ESB^Y)	$2(4N\ell-4N+1)(N-1)$	$N^2(\ell-1)$	$N(N-1)(4N\kappa\ell-4N\kappa+4\kappa\ell-4\kappa+2\ell-1)$	0	7	0
$Y2A$ (w/o online comm., incl. ESB^Y)	$2(3N^2\ell+5N\ell-4N+1)(N-1)$	$N^2(\ell-1)$	$(7N\kappa\ell-4N\kappa+9N\ell+5\kappa\ell-4\kappa+3\ell-1)(N-1)N$	0	$2\lceil\log_2\ell\rceil-1)\lceil\log_2 N\rceil+11$	0
$A2Y$, $A2B$ (via Y, excl. ESB^Y)	$8N(N-1)(\ell-1)$	$N^2(\ell-1)$	$(4N\kappa+4\kappa+1)(N-1)N(\ell-1)$	$N(N-1)(\kappa+1)\ell$	6	2
$A2Y$, $A2B$ (via Y, incl. ESB^Y)	$2(3N\ell+5\ell-4)(N-1)N$	$N^2(\ell-1)$	$(7N\kappa\ell-4N\kappa+9N\ell+5\kappa\ell-4\kappa+2\ell-1)(N-1)N$	$N(N-1)(\kappa+1)\ell$	$2\lceil\log_2\ell\rceil-1)\lceil\log_2 N\rceil+10$	2

4. Experiments

本文做了一系列实验，和ABY，MP-SPDZ等协议对比，验证了子协议和整个系统的性能提升。列举几项如下：

Table 6. Run-times in nanoseconds for one OT, amortized over 10 million parallel evaluations, averaged over 100 runs.

	G-OT		R-OT	$C^{\oplus}$-OT		C^{+}-OT			
Bit size	1	128	128	1	128	8	16	32	64
libOTe [69] LAN	–	120	–	–	130	–	–	–	–
MOTION LAN (this work)	151	196	77	131	147	119	121	126	134
libOTe [69] WAN	–	820	–	–	874	–	–	–	–
MOTION WAN (this work)	1 069	1 221	957	932	973	955	960	972	980

Table 7. Total (online+setup) run-times in seconds for biometric matching, comparing several implementations and protocols over various domains for N parties, bitlength ℓ and multiple database sizes. We benchmarked MOTION and ABY [32] with circuits generated with HyCC [19]. In MOTION, the run-times with SIMD are amortized over 192 / 32 parallel circuits for DB sizes of 1 024 / 4 096. Best run-times are marked in **bold**.

							LAN		WAN	
							DB Size		DB Size	
Implementation	Protocol	Domain	Security	N	Thresh.	ℓ	1 024	4 096	1 024	4 096
ABY [32]	A+B	$\mathbb{Z}_{2^\ell}$	passive	2	1	32	0.26	0.89	2.6	4.1
ABY [32]	A+Y	$\mathbb{Z}_{2^\ell}$	passive	2	1	32	**0.24**	**0.76**	**2.5**	**3.6**
MP-SPDZ [50]	MASCOT [51]	$\mathbb{F}_p$	active	3	2	32	45.78	174.90	1 150.4	4 596.0
MP-SPDZ [50]	MASCOT [51]	$\mathbb{F}_p$	passive	3	2	32	9.56	36.78	935.0	3 746.0
MP-SPDZ [50]	SPD$\mathbb{Z}_{2^k}$ [26]	$\mathbb{Z}_{2^\ell}$	active	3	2	64	57.25	231.53	1 643.8	6 580.2
MP-SPDZ [50]	SPD$\mathbb{Z}_{2^k}$ [26]	$\mathbb{Z}_{2^\ell}$	passive	3	2	64	3.89	13.80	1 126.1	4 500.5
MP-SPDZ [50]	FKOS15 [35]	binary	active	3	2	32	104.76	413.25	3 456.6	13 772.6
MP-SPDZ [50]	OT-based	binary	passive	3	2	32	4.17	14.80	1 346.4	5 289.3
SCALE-MAMBA [2]	Full-Threshold	$\mathbb{F}_p$	active	3	2	32	128.95	253.19	858.9	2 033.0
MOTION (**this work**)	A+B	$\mathbb{Z}_{2^\ell}$	passive	3	2	32	5.74	19.99	15.4	41.3
MOTION (**this work**) w/ SIMD	A+B	$\mathbb{Z}_{2^\ell}$	passive	3	2	32	**0.22**	**1.33**	**1.2**	**5.2**
MOTION (**this work**)	A+Y	$\mathbb{Z}_{2^\ell}$	passive	3	2	32	5.71	21.78	10.2	29.2
MOTION (**this work**) w/ SIMD	A+Y	$\mathbb{Z}_{2^\ell}$	passive	3	2	32	0.26	1.55	1.8	7.5

Table 8. Total and online run-times in milliseconds for the evaluation of the Bristol Fashion circuits [1] for AES-128 with key scheduling and SHA-256 in GMW (B) and BMR (Y) executed with MOTION and MP-SPDZ [50], as well as Choi et al.'s GMW [24]. The run-times are amortized over 512 / 256 executions of AES-128 / SHA-256. For BMR (Y) benchmarks in MP-SDPZ, we used the default setting that allows up to three parties, which is clearly sufficient to analyze the trend of how this implementation scales, and thus we omit the very time-consuming benchmarks with more parties.

			LAN			WAN		
	Implementation		N=2	N=3	N=5	N=2	N=3	N=5
AES	Choi et al. [24]	B	21.7	22.6	26.5	39.8	40.8	44.5
	MP-SPDZ [50]	B	0.7	3.2	6.9	30.6	30.2	33.0
	MP-SPDZ [50]	Y	797.0	2 856.4	—	11 339.7	18 506.0	—
	ABY [32]	B	0.2	—	—	8.5	—	—
	online	B	<0.1	—	—	6.6	—	—
	ABY [32]	Y	0.2	—	—	1.9	—	—
	online	Y	0.1	—	—	0.1	—	—
	MOTION (**this work**)	B	1.9	2.5	3.8	13.8	14.9	18.9
	online	B	0.4	0.5	0.8	7.3	7.7	8.1
	MOTION (**this work**)	Y	4.7	8.0	17.1	61.1	87.8	141.7
	online	Y	0.2	0.2	0.4	0.5	0.7	0.9
SHA	Choi et al. [24]	B	69.2	72.9	85.1	731.3	735.1	769.1
	MP-SPDZ [50]	B	3.7	5.5	9.0	825.2	1 033.8	1 077.2
	MP-SPDZ [50]	Y	2 819.5	30 156.7	—	40 005.9	61 012.6	—
	ABY [32]	B	1.5	—	—	339.8	—	—
	online	B	0.7	—	—	334.1	—	—
	ABY [32]	Y	1.5	—	—	8.1	—	—
	online	Y	0.5	—	—	0.6	—	—
	MOTION (**this work**)	B	8.3	10.8	16.0	500.2	572.4	614.1
	online	B	2.7	3.2	4.5	479.6	547.9	538.4
	MOTION (**this work**)	Y	19.0	29.6	61.8	201.9	279.3	492.6
	online	Y	1.3	1.6	1.8	1.4	2.1	2.7

Table 9. Total run-times in seconds for CryptoNets [36] with HyCC hybrid circuits [19] for N parties and full-threshold security.

Implementation		LAN				WAN			
		N=2	N=3	N=4	N=5	N=2	N=3	N=4	N=5
ABY [32]	A+B	0.5	—	—	—	3.2	—	—	—
	A+Y	0.5	—	—	—	3.4	—	—	—
MOTION (**this work**)	A+B	3.5	4.2	4.9	5.7	6.7	8.0	9.8	13.2
	A+Y	3.6	4.2	4.9	5.8	6.7	8.6	12.6	13.1