Scalable Multi-Party Computation Protocols for Machine Learning in the Honest-Majority Setting

这次介绍的是Fengrun Liu等人发表在IEEE S&P'24上关于诚实大多数下抵抗半诚实敌手的大规模面向机器学习的安全多方计算协议框架，论文链接如下：

https://www.usenix.org/system/files/sec24summer-prepub-278-liu-fengrun.pdf

0. 背景与基础知识

本文以Damgård-Nielsen协议(Crypto'07)为基础，在Mersenne素数域（$p = 2^\ell - 1$）上构建了抵抗诚实大多数场景下的半诚实敌手的安全多方计算协议支撑隐私保护机器学习。本文主要针对截断和比较操作，提出了多种优化技术来减少神经网络安全计算中的通信和轮数开销。本文可以实现63方下的4层卷机神经网络计算，在局域网和广域网分别耗时0.1秒和4.6秒。

Mersenne素数的数学表示为$M_\ell = 2^\ell - 1$，其中ℓ是一个素数，例如$\ell = 31, 61, 127$。本文用$p = 2^\ell - 1$表示该数，且大多数协议是定义在该域上的。E表示集合$\{-1, 0, 1\}$，$2E$表示集合$\{0, \pm 1, \pm 2\}$。

0.1 Shamir秘密分享

本文使用(t, n)-Shamir秘密分享构造安全多方计算协议，其中$n \geq 2t + 1$。$[x]_p$表示度为t的多项式定义的秘密分享，$\langle x \rangle_p$是度为$2t$的多项式定义的秘密分享。$x \leftarrow \Pi_{\text{Reveal}}([x]_p)$表示数据公开操作：所有的计算方将自己的份额发送给$P_{\text{pking}}$，$P_{pking}$执行Shmair秘密分享恢复操作构造$x$，然后向其他所有参与方广播$x$。上述一次操作成为1轮。

0.2 DN乘法协议

Damgård和Nielsen提出的DN协议适合在诚实大多数场景下求乘法，该协议可以大致分为4部分：预处理，输入，在线计算和输出。本文着重介绍预处理和在线计算：

- **预处理:** 在预处理阶段，多方需要生成足够多的秘密随机数用于中间结果转化，DN协议中需要度数为t的共享随机数$[r]_p \leftarrow \Pi_{\text{rand}}$，另外还需要同时共享度数为$t$和$2t$的随机数对$([r]_p, \langle r \rangle_p) \leftarrow \Pi_{\text{DoubleRand}}$。生成上述两种随机数的均摊通信开销分别为每方$2$和$4$个域元素。

- **在线计算：** 在线计算$[xy]_p \leftarrow \Pi_{\text{Mult}}([x]_p, [y]_p)$则需要消耗上述随机数对，具体来说所有参与方首先计算$\langle xy + r \rangle_p = [x]_p \cdot [y]_p + \langle r \rangle_p$，然后$P_{\text{pking}}$收集秘密分享份额并恢复$xy + r$，并广播给所有参与方；最后，所有参与方本地计算$[xy]_p = (xy + r) - [r]_p$。

另外，结果公开的乘法记作 $xy \leftarrow \Pi_{\text{MultPub}}([x]_p, [y]_p)$。

0.3 多输入前缀积

多输入前缀积旨在计算$[b_1]_p, \ldots, [b_t]_p \leftarrow \Pi_{\text{PreMult}}([a_1]_p, \ldots, [a_t]_p)$，使得对于$i = 1, ;t$有$b_i = \Pi_{j=1}^{i} a_j$。该协议构造如下：

- **预处理：** 该阶段需要构造t对关联随机数对$([r_i]_p, [r'_i]_p)$满足$r'_1 = r_1^{-1}$，当$i > 1$时$r'_i = r_{i-1} r_i^{-1}$。如此，则有$r_i \Pi_{j=1}^{i} r'_i = 1$。
- **在线计算：** 在线计算分为两步，首先对于$i = 1, \ldots, t$，计算$c_i \leftarrow \Pi_{\text{MultPub}}([a_i]_p, [r'_i]_p)$；最终，各方本地计算$[b_i]_p = [r_i]_p \cdot \Pi_{j=1}^{i} c_j$。

0.4 随机比特秘密分享

协议$([r]_p, [r]_{\text{B}}) \leftarrow \Pi_{\text{SolvedBits}}$生成随机数$[r]_p$和其对应的比特分解向量$[r]_{\text{B}} = ([r_0]_p, \ldots, [r_{t-1}]_p)$，满足$r = \sum_{i=0}^{t-1} 2^i r_i$。

1. Fixed-Point Multiplication

在隐私保护机器学习中，需要将浮点数表示为定点数，然后将定点数编码为$\mathbb{F}_p$内的整数进行安全协议的计算。给定ℓ比特的Mersenne素数p，我们将整数$x \in (-2^{\ell-1}, 2^{\ell-1})$表示为：$x \pmod p \in [0, p-1]$。

1.1 Truncation with Only 1-Bit Gap

给定上述编码后的整数x，使得$|x| = x_1 \cdot 2^d + x_2$且$0 \leq x_2 < 2^d$，那么截断$d$比特的结果就是：1）当$x \geq 0$时截断结果为$x_1$；2）当$x < 0$时结果为$p - x_1$。形式化定义如下：

定义1: 给定$x \in \mathbb{F}_p$，其截断结果$\text{Trunc}_d(x) \in \mathbb{F}_p$定义为：

$$\text{Trunc}_d(x) = \begin{cases} \lfloor x / 2^d \rfloor, & 0 \leq x \leq (p-1)/2 \\ p - \lfloor (p-x)/2^d \rfloor, & (p-1)/2 < x \leq p-1 \end{cases}$$

其中，$0 \leq x < (p-1)/2$表示非负数，$(p-1)/2 < x \leq p-1$表示负数，很容易验证$\text{Trunc}_d(x) = p - \text{Trunc}_d(-x) = -\text{Trunc}_d(-x) \in \mathbb{F}_p$。在Mersenne素数域中，一个非常好的性

质是符号为只取决于最高有效位，边界判断简单，因此本文提出了更加高效截断算法：丢弃后d比特，右移，然后空出来的高d比特填充为最高有效位。形式化定理如下：

定理1: 令$p = 2^\ell - 1$表示Mersenne素数，$x \in \mathbb{F}_p$且$x_0, x_1, \ldots, x_{\ell-1}$表示$x$的比特分解使得$x = \sum_{i=0}^{\ell-1} x_i \cdot 2^i$，那么有

$$\mathrm{Trunc}_d(x) = \sum_{i=d}^{\ell-1} x_i \cdot 2^{i-d} + \sum_{i=\ell-d}^{\ell-1} x_{\ell-1} \cdot 2^i$$

上述定理详细证明参考原文。和之前的方案比（例如ABY3、SecureML等），本文截断方法一个很重要的优化就是不再需要秘密值和域的大小差距很大（仅需要1比特差距即可）。因此，本文截断方法可以大大减少底层秘密分享的位长。形式化来说，本文的安全截断刻画如下：

定理2：在$\mathbb{F}_p, p = 2^\ell - 1$内，令$z \in [0, (p-1)/2]$且$z$可以被分解为$x = z + r \pmod p$ 和 $y = p - r$使得$x + y = 2 \pmod p$，其中$r \in \mathbb{F}_p$内任意元素，那么有

$$\mathrm{Trunc}_d(x) + \mathrm{Trunc}_d(y) + \delta(r, x) \cdot (2^{\ell-d} - 1) \in \mathrm{Trunc}_d(z) + E,$$

其中当且仅当$r_{msb} \wedge x_{msb} = 1$时$\delta(r, x) = 1$，否则$\delta(r, x) = 0$。

针对x, r的正负性，本文从四方面讨论了上述定理2的正确性。详细证明请参考原文。另外，上述截断只是面性$[0, (p-1)/2]$内的非负数，对于负数可以加上一个偏移将其移动到非负数范围内再用定理2截断，截断结果再减去便宜的对应截断结果。如此，引出如下针对截断操作的统一推论：

推理1：在$\mathbb{F}_p, p = 2^\ell - 1$内，令$a \in [0, 2^{\ell-2}) \cup [p - 2^{\ell-2}, p)$ 且 $b = a + 2^{\ell-2}$满足$b \in [0, (p-1)/2]$，那么有$\mathrm{Trunc}_d(b) - 2^{\ell-d-2} \in \mathrm{Trunc}_d(a) + E$。

基于上述推论有如下截断协议：

Protocol 3.1. Π_{Trunc} *(Newly added!)*

Input: $[a]_p$ where $a \in [-2^{\ell-2}, 2^{\ell-2})$.
Output: $[f]_p \in [\mathsf{Trunc}_d(a)]_p + 2E$.
Common Randomness: $([r]_p, [r]_{\mathsf{B}}) \leftarrow \mathcal{F}_{\mathsf{SolvedBits}}$.

1. $[r_{\mathsf{msb}}]_p \leftarrow [r_{\ell-1}]_p$
2. $[r']_p \leftarrow \sum_{i=d}^{\ell-1} 2^{i-d} \cdot [r_i]_p + \sum_{i=\ell-d}^{\ell-1} 2^i \cdot [r_{\mathsf{msb}}]_p$
3. $[b]_p \leftarrow [a]_p + 2^{\ell-2}$ such that $b \in [0, (p-1)/2]$
4. $\langle c \rangle_p \leftarrow [b]_p + [r]_p$
5. $c \leftarrow \Pi_{\mathsf{Reveal}}([c]_p)$, and compute $\mathsf{Trunc}_d(c)$ and c_{msb}
6. $[e]_p \leftarrow (1 - [r_{\mathsf{msb}}]_p) \cdot c_{\mathsf{msb}}$
7. $[f]_p \leftarrow \mathsf{Trunc}_d(c) - [r']_p + [e]_p \cdot (2^{\ell-d} - 1)$
8. Output $[f]_p - 2^{\ell-d-2}$

1.2 DN-based Fixed-Point Multiplication

有了上述截断协议，可以进一步将截断操作和DN乘法结合起来，在公开中间结果的时候进行截断的计算，具体协议如下：

Protocol 3.2. $\Pi_{\text{Fixed-Mult}}$

Input: $[a]_p$ and $[b]_p$.
Output: $[y]_p \in [\text{Trunc}_d(a \cdot b)]_p + 2E$.
Common Randomness: A random truncation triple $([r']_p, \langle r \rangle_p, [r_{\text{msb}}]_p) \leftarrow \mathcal{F}_{\text{Trunc-Triple}}$ such that $r' = \text{Trunc}_d(r) \in \mathbb{F}_p$ and $r_{\text{msb}} \in \{0,1\}$ is the MSB of r.

1. $\langle c \rangle_p \leftarrow [a]_p \cdot [b]_p + \langle r \rangle_p + 2^{\ell-2}$
2. $c \leftarrow \Pi_{\text{Reveal}}(\langle c \rangle_p)$, and compute $\text{Trunc}_d(c)$ and c_{msb}
3. $[e]_p \leftarrow (1 - [r_{\text{msb}}]_p) \cdot c_{\text{msb}}$
4. $[f]_p \leftarrow \text{Trunc}_d(c) - [r']_p + [e]_p \cdot (2^{\ell-d} - 1) - 2^{\ell-d-2}$
5. Output $[f]_p$

同时乘法协议需要消耗关联随机数，关联随机数的生成协议如下：

Protocol 3.3. $\Pi_{\text{Trunc-Triple}}$

Input: None.
Output: $([r']_p, \langle r \rangle_p, [r_{\text{msb}}]_p)$, where $r' = \text{Trunc}_d(r)$.

1. $([r]_p, [r]_{\text{B}}) \leftarrow \mathcal{F}_{\text{SolvedBits}}$
2. $[r_{\text{msb}}]_p \leftarrow [r_{\ell-1}]_p$.
3. $[r']_p \leftarrow \sum_{i=d}^{\ell-1} 2^{i-d} \cdot [r_i]_p + \sum_{i=\ell-d}^{\ell-1} 2^i \cdot [r_{\text{msb}}]_p$
4. $\langle r \rangle_p \leftarrow \sum_{i=0}^{\ell-1} 2^i \cdot [r_i]_p \cdot [r_i]_p + \langle 0 \rangle_p$

2. Improved Bitwise Primitives

本节则介绍如何实现前缀或门计算和比较协议。

2.1 Prefix-ORs

前缀或门计算$b_j = \vee_{i=1}^{j} a_i$，其中$a_i \in \{0,1\}$，本文利用其对偶问题$\bar{b}_j = \wedge_{i=1}^{j} \bar{a}_i$。该与门的计算可以调用协议$\Pi_{\mathrm{PreMult}}$实现，具体协议如下所示：

Protocol 4.1. Π_{PreOR}

Input: $[a_1]_p, [a_2]_p, \ldots, [a_\ell]_p$ where $a_i \in \{0,1\}$.
Output: All prefix ORs $[a_1]_p, [a_1 \vee a_2]_p, \ldots, [\vee_{i=1}^{\ell} a_i]_p$.

1. For $i = 1, \ldots, \ell$: compute $[b_i]_p \leftarrow 1 - [a_i]_p$.
2. $([c_1]_p, \ldots, [c_\ell]_p) \leftarrow \mathcal{F}_{\mathsf{PreMult}}([b_1]_p, \ldots, [b_\ell]_p)$
3. Output $(1 - [c_1]_p, \ldots, 1 - [c_\ell]_p)$.

2.2 Bitwise Less Than

基于上述Prefix-OR协议，可以进一步优化比较协议。本文的比较协议时一个公开数a和秘密分享比特串$[b]_B$，计算$(a < b)$。等价的，可以计算$(p - b < p - a)$，$p - a$可以公开计算，且$p - b$的比特分解为$(1 - b_0, \ldots, 1 - b_{\ell-1})$。基于此，本文提出了如下协议：

Protocol 4.2. $\Pi_{\mathsf{Bitwise\text{-}LT}}$

Input: a and $[b]_{\mathsf{B}}$ where $a, b \in \mathbb{F}_p$ and $b = \sum_{i=0}^{\ell-1} 2^i b_i$
Output: $[a < b]_p$

1. Let $a_0, \ldots, a_{\ell-1}$ be the decomposed bits of a.
2. Set $\overline{a}_i = 1 - a_i$ and $\overline{b}_i = 1 - [b_i]_p$ for $0 \leq i \leq \ell - 1$.
3. For $i = 0, \ldots, \ell - 1$: compute $[c_i]_p = \overline{a}_i \oplus [\overline{b}_i]_p$.
4. Run $([d_{\ell-1}]_p, \ldots, [d_0]_p) \leftarrow \mathcal{F}_{\mathsf{PreOR}}([c_{\ell-1}]_p, \ldots, [c_0]_p)$
5. Set $[e_i]_p = [d_i - d_{i+1}]_p$ where $[e_{\ell-1}]_p = [d_{\ell-1}]_p$
6. Output $[a < b]_p = \sum_{i=0}^{\ell-1} \overline{a}_i \cdot [e_i]_p$.

该协议的主要思想还是按比特比较，首先找到不同的比特位置，然后看谁在该位置为1（即数值大）。

3. Functions of Neural Networks

在上述协议的基础上，本文提出了面向神经网络算子的优化。并提出了多个专用优化协议。

3.1 DReLU

DReLU已经介绍过其计算很多次，其等于最高有效位取反。根据SecureNN中的结论在素数环中$a_{\mathrm{msb}} = (2a)_{\mathrm{lsb}}$。为了计算$[(2a)_{\mathrm{lsb}}]_p$，本文首先盲化$y = 2a + r$，然后公开$y$。如果$y \geq r$，那么有$y_{\mathrm{lsb}} = (2a)_{\mathrm{lsb}} \oplus r_{\mathrm{lsb}}$；否则就是$y_{\mathrm{lsb}} = (2a)_{\mathrm{lsb}} \oplus r_{\mathrm{lsb}} \oplus 1$。给定$r$的比特分解的秘密分享，那么很容易调用协议$\Pi_{\mathrm{Bitwise-LT}}$计算$y < r$来判断$2a + r$是否在数值上超过了$p$（超过的则需要进行模$p$计算。）因此，最终有$(2a)_{\mathrm{lsb}} = y_{\mathrm{lsb}} \oplus r_{\mathrm{lsb}} \oplus (y < r >)$。详细协议如下所示：

Protocol 5.1. Π_{DReLU}

Input: $[a]_p$ where $a \in \mathbb{F}_p$
Output: $[\mathsf{DReLU}(a)]_p$
Common Randomness: $([r]_p, [r]_{\mathsf{B}}) \leftarrow \mathcal{F}_{\mathsf{SolvedBits}}$.

1. $[y]_p \leftarrow [2a + r]_p$ and reveal $y \leftarrow \Pi_{\mathsf{Reveal}}([y]_p)$.
2. $[b]_p \leftarrow y_{\mathsf{lsb}} \oplus [r_0]_p$.
3. Run $[c]_p \leftarrow \mathcal{F}_{\mathsf{Bitwise\text{-}LT}}(y, [r]_{\mathsf{B}})$
4. Output $1 - [b]_p \oplus [c]_p$

3.2 Vectorized Two-Layer DN Multiplication

进一步，本文利用ALTAS中利用一轮通信的计算两层乘法的技术来优化计算。给定输入 $[x]_p, [y_p], \{[z_i]_p\}_{i=1}^m$，计算输出$\{[xyz_i]_p\}_{i=1}^m$，协议计算流程大致如下：

- 遵循标准的DN协议流程在计算$[xy]_p$的过程中公开中间结果$u = xy + r$；
- 利用$0 = z_i + (-z_i)$准备三元组$([r]_p, [-z_i]_p, [-rz_i]_p)$。如此按照下述协议则可以本地计算所有的$[xyz_i]_p$。

Protocol 5.2. $\Pi_{\text{2L-DN}}$

Input: $[x]_p, [y]_p$ and $\{[z_i]_p\}_{i=1}^m$
Output: $\{[xyz_i]_p\}_{i=1}^m$
Common Randomness: $m+1$ pairs of double sharings $([r]_p, \langle r\rangle_p)$ and $\{([r_i]_p, \langle r_i\rangle_p)\}_{i=1}^m$.

1. Compute $\langle u\rangle_p = [x]_p \cdot [y]_p + \langle r\rangle_p$.
   Compute $\langle u_i\rangle_p = [r]_p \cdot [-z_i]_p + \langle r_i\rangle_p$.
2. Reveal $u \leftarrow \Pi_{\text{Reveal}}(\langle u\rangle_p)$ and $u_i \leftarrow \Pi_{\text{Reveal}}(\langle u_i\rangle_p)$.
3. Compute $[c_i]_p = u_i - [r_i]_p$ for $1 \le i \le m$.
4. For $i = 1, \ldots, m$, locally compute:
   $[xyz_i]_p = u \cdot [z_i]_p + [c_i]_p$

值得注意的是，上述协议的第2步两个计算可以并行进行，因此只需要一轮通信。

3.3 ReLU & Maxpool

给定a，$\mathrm{ReLU}(a) = \mathrm{DReLU}(a) \cdot a = (1 - (b-c)\cdot(b-c)) \cdot a = a - a(b-c)(b-c)$。其中的乘法则可以调用上述协议$\Pi_{\mathrm{2L-DN}}$实现。具体如下：

Protocol 5.3. Π_{ReLU}

Input: $[a]_p$ where $a \in \mathbb{F}_p$
Output: $[\mathsf{ReLU}(a)]_p$
Common Randomness: $([r]_p, [r]_{\mathsf{B}}) \leftarrow \mathcal{F}_{\mathsf{SolvedBits}}$.

1. $[y]_p \leftarrow [2a + r]_p$ and reveal $y \leftarrow \Pi_{\mathsf{Reveal}}([y]_p)$.
2. $[b]_p \leftarrow y_{\mathsf{lsb}} \oplus [r_0]_p$.
3. Run $[c]_p \leftarrow \mathcal{F}_{\mathsf{Bitwise\text{-}LT}}(y, [r]_{\mathsf{B}})$
4. Run $[t]_p \leftarrow \mathcal{F}_{\mathsf{2L\text{-}DN}}([b-c]_p, [b-c]_p, \{[a]_p\})$.
5. Output $[a]_p - [t]_p$.

进一步，Maxpool的计算可以规约到求两个值中的最大值：

$$\mathrm{Max}(a, b) = \mathrm{DReLU}(a - b) \cdot a + [1 - \mathrm{DReLU}(a - b)] \cdot b = \mathrm{ReLU}(a - b) + b$$

该计算则可以基于ReLU的安全计算协议实现。对于求m个元素中的最大值，则可以分组之后，调用$m - 1$次ReLU计算协议实现。具体Maxpool安全计算协议如下：

Protocol 5.4. Π_{Maxpool}

Input: $[a_1]_p, \ldots, [a_m]_p$ assuming m is powers of two.
Output: $[a_k]_p$ where $a_k = \text{Maxpool}(a_1, \ldots, a_m)$.

1. If $m = 1$: Output $[a_1]_p$
2. Set $\boldsymbol{a}^1 \leftarrow ([a_1]_p, \ldots, [a_m]_p)$ and $s \leftarrow m$.
3. For $i = 1, \ldots, \log m$:
   a. $s \leftarrow s/2$.
   b. Set $([b_1]_p, \ldots, [b_s]_p, [c_1]_p, \ldots, [c_s]_p) \leftarrow \boldsymbol{a}^i$.
   c. For $j = 1, \ldots, s$: $[e_j]_p \leftarrow \mathcal{F}_{\text{ReLU}}([b_j - c_j]_p) + c_j$.
   d. If $s = 1$: Output $[e_1]_p$.
      Else: Set $\boldsymbol{a}^{i+1} \leftarrow ([e_1]_p, \ldots, [e_l]_p)$

4. Evaluations

首先，本文给出了各个协议的理论通信和轮数复杂度如下：

Protocols	Rounds		Communication	
	Online	Prep.	Online	Prep.
$\Pi_{\text{Fixed-Mult}}$	1	2	2	3ℓ
Π_{PreMult}	1	2	2ℓ	7ℓ
Π_{PreOR}	1	2	2ℓ	7ℓ
$\Pi_{\text{Bitwise-LT}}$	1	2	2ℓ	7ℓ
Π_{DReLU}	3	2	$4+2\ell$	$1+10\ell$
$\Pi_{\text{2L-DN}}$	1	1	$2(m+1)$	$m+1$
Π_{ReLU}	3	2	$6+2\ell$	$2+10\ell$
Π_{Maxpool}	$3\log m$	2	$(m-1)(6+2\ell)$	$(m-1)(2+10\ell)$

Table 1: Round and communication complexity of building blocks. Numbers of communication are reported in field elements per party.

进一步，本文在$p = 2^{31} - 1$的素数域上进行了安全协议实现，定点数小数保留位数为12比特。所有的计算基于32比特整数实现。安全预测的具体时间和通信开销如下：

#PC	Network-A		Network-B		Network-C	
	Online	Prep.	Online	Prep.	Online	Prep.
3PC	0.006	0.006	0.007	0.02	0.02	0.178
7PC	0.01	0.008	0.01	0.027	0.03	0.254
11PC	0.011	0.01	0.011	0.038	0.033	0.335
21PC	0.008	0.017	0.01	0.059	0.035	0.508
31PC	0.011	0.023	0.013	0.077	0.047	0.632
63PC	0.024	0.048	0.025	0.149	0.096	1.208

Table 2: Online and preprocessing time for oblivious inference in the LAN setting. All numbers are reported in seconds.

#PC	Network-A		Network-B		Network-C	
	Online	Prep.	Online	Prep.	Online	Prep.
3PC	0.37	0.244	0.387	0.392	1.222	2.189
7PC	0.373	0.213	0.404	0.576	1.327	2.768
11PC	0.376	0.216	0.41	0.414	1.346	2.777
21PC	0.386	0.233	0.444	0.57	1.718	4.812
31PC	0.398	0.268	0.478	0.631	2.027	6.65
63PC	0.478	0.52	0.68	1.885	4.573	10.69

Table 3: Online and preprocessing time for oblivious inference in the WAN setting. All numbers are reported in seconds.

#PC	Network-A		Network-B		Network-C	
	Online	Prep.	Online	Prep.	Online	Prep.
3PC	0.047	0.319	0.2	1.34	1.92	12.806
7PC	0.061	0.403	0.257	1.69	2.466	16.154
11PC	0.065	0.425	0.273	1.783	2.615	17.043
21PC	0.068	0.444	0.286	1.86	2.738	17.776
31PC	0.069	0.45	0.291	1.887	2.782	18.035
63PC	0.07	0.457	0.296	1.916	2.829	18.309

Table 4: Online and preprocessing communication size per party of oblivious inference. All numbers are reported in MB.

Protocol	Network-A		Network-B		Network-C	
	LAN	WAN	LAN	WAN	LAN	WAN
Falcon[45]	0.009	0.532	0.011	0.535	0.04	2.171
This	0.006	0.37	0.007	0.387	0.02	1.222
Times	1.5×	1.4×	1.6×	1.4×	1.9×	1.8×

Table 5: Comparing the performance of Falcon and our protocol.