SecFloat: Accurate Floating-Point meets Secure 2-Party Computation

本次介绍Rathee等人发表在 S&P'22 的面向浮点数计算的安全两方计算论文，论文链接如下

0. Background, Design, & Contribution

0.0 Background

现在大多数的方案在处理浮点数计算的时候采取如下两条技术路线：1）将浮点数计算电路表示为布尔电路，然后利用GC或者GMW协议按比特进行计算；2）将浮点数近似为满足一定精度的定点数，然后利用算术电路和布尔电路混合协议进行数据计算。方法1）会造成巨大的开销，而方案2）则会有一定程度上的精度损失，有些时候这些损失是非常巨大的甚至不可接受。而要保证和浮点数非常接近的精度，虽然可以通过增加比特位长的方式来实现，但是这种方法大约需要512比特的大整数。这无疑给整体开销带来了非常大的负担。

本文面向32比特的单精度浮点数，涉及了安全两方计算（2PC）下的计算库SecFloat。和已有的方案，例如ABY-F和MP-SPDZ相比，SecFloat在精度和开销上都得到了大幅度的提升。SecFloat的主要贡献在于提供了用于浮点数高精度计算的2PC函数。为了达到这个目标，本文对于浮点数的各部分（指数、尾数等）利用并优化了一系列整数下的2PC库。而已有的安全计算方案在精度上不足，而明文精确计算库在2PC下开销巨大（not crypto-friendly），本文的提供的SecFloat在精度和2PC性能二者之间架起了一座桥梁。

0.1 High-Level SecFloat's Design

0.1.1 IEEE Standard & Intel's MKL

IEEE标准中定义的单精度浮点数具有 $p = 8$ 比特的指数部分和 $q = 23$ 比特的尾数部分，详细数据格式可以参考之前的博客(xxxx)。Intel的数据计算库MKL计算标准数学函数分三步：range reduction, polynomial approximations, output compensation。以 $z = 2^x$ （$x \in [-2^{126}, 2^{127}]$）为例：

1. range reduction: 首先计算 N 和 f 满足 $2^x = 2^N 2^f$，其中N是整数，$f \in [0, 1)$;
2. polynomial approximations：对于 $0 \leq f < 1$ 利用多项式近似 2^f;
3. output compensation：计算 $z = y \cdot 2^N$。

标准明文库利用高比特位来计算中间值得到精确的计算结果，但是如果直接将该方法迁移到2PC下则会造成巨大的通信和计算开销。

0.1.2 Design of Math Functions

SecFloat也遵循MKL的三步设计计算数学函数，但是出于2PC和明文CPU下计算的性能指标完全不同的考量，SecFloat设计了完全不同的函数以便在2PC下实现高效的函数计算：

1. range reduction: 之前的方案在$q = 52$ 比特尾数上利用多项式近似进行rangle reduction，这会带来巨大的开销。本文在满足精度的前提下，使用 $q = 27$ 比特尾数进行计算，大大减少开销；
2. MKL使用的多项式在2PC下计算开销很大，本文生成了crypto-friendly 分段函数来进行性能优化；
3. 本文提供了一种全新的方法来实现高效安全计算分段函数；
4. 为了进一步提升性能，本文在满足精度的前提下，使用了满足累计误差约束的低精度基本原语算子。

0.1.3 Bitwidth Optimizations

承袭SIRNN的设计思想，本文使用了un-uniform bitwidth技术来优化一些基本原子计算。

0.2 主要贡献

总结一下，SecFloat的主要贡献如下：

1. 高精度的高效2PC计算库，达到了Intel的明文数值计算库的精度要求：SecFloat构造了高精度的精确函数在2PC下实现range reduction, polynimal approximations, 和 output compensation。支持正确的浮点基本原语：加减乘除和比较；提供了精确的数学函数计算，包括三角函数、指数和对数函数。并且支持任意浮点表示；
2. 构造了面向32比特单精度浮点数的SecFloat库，精度比已有方案高6个数量级，效率高3个数量级；
3. 实现了隐私保护下的距离感应测试：该应用对于精度非常敏感，和之前的方案比SecFloat精度提升了4个数量级，并且减少通信 $6.8 - 11\times$。 本文验证了之前定点数下的安全预测无法满足隐私保护广告推广服务，而SecFloat可以满足这一应用。

1. Premilaries

1.1 Floating-Point Representation

本文关于秘密分享和基本整数计算原语的定义和SIRNN相同，在此不再赘述。对于会单数在本文中的表示，介绍如下：
给定浮点数 α，在浮点数被参数 $p, q \in \mathbb{Z}^+$ 规定下，表示为四元组 (z, s, e, m)其中：

1. $z \in \{0, 1\}$：如果 $\alpha = 0$, 令 $z = 1$；
2. $s \in \{0, 1\}$：α 的正负性；
3. $e \in \{0, 1\}^{p+2}$：指数部分的有符号表示，取值范围 $[-2^{p-1} + 1, 2^{p-1}]$；
4. $m \in \{0, 1\}^{q+1}$：正则化的尾数无符号表示，取值范围 $[2^q, 2^{q+1} - 1] \cup |\{0\}$，扩大因子为 q（即扩大倍数为 2^q）。

因此，给定 $\alpha = (z, s, e, m)$，其真实值表示为 $(1-z)\cdot(1-2s)\cdot 2^{\mathsf{int}_{p+2}(e)}\cdot\langle m\rangle_{q+1,q}$。

1.2 Secret Sharing

本文使用SIRNN中的2PC协议构建底层的基本算子，这些基本算子和SIRNN中的相同，具体都展现在表1中。

Functionality	Notation		Description	Communication
	Functionality	Protocol		
Multiplexer [65]	$z = c\ ?\ x : y$	$\langle z\rangle^\ell = \Pi^\ell_{\mathsf{MUX}}(\langle c\rangle^B, \langle x\rangle^\ell, \langle y\rangle^\ell)$	$z = x$ if $c = 1$, else $z = y$	$2\lambda + 2\ell$
OR [65]	$z = x \vee y$	$\langle z\rangle^B = \Pi_{\mathsf{OR}}(\langle x\rangle^B, \langle y\rangle^B)$	$z = x \vee y$	$\lambda + 20$
Equality [65]	$e = \mathbf{1}\{x = y\}$	$\langle e\rangle^B = \Pi^\ell_{\mathsf{EQ}}(\langle x\rangle^\ell, \langle y\rangle^\ell)$	Checks if $x = y$, $x, y \in \mathbb{Z}_{2^\ell}$	$< \frac{3}{4}\lambda\ell + 9\ell$
Comparison [65]	$c = \mathbf{1}\{x > y\}$	$\langle c\rangle^B = \Pi^\ell_{\mathsf{GT}}(\langle x\rangle^\ell, \langle y\rangle^\ell)$	Checks if $x > y$, $x, y \in \mathbb{Z}_{2^\ell}$	$< \lambda\ell + 14\ell$
Lookup Table (LUT) [30]	$y = L(x), y \in \mathbb{Z}_{2^n}$	$\langle y\rangle^n = \Pi^{m,n}_{\mathsf{LUT}}(L, \langle x\rangle^m)$	index x, LUT L, $z \in \mathbb{Z}_{2^n}$	$2\lambda + 2^m n$
Zero-Extension [64]	$y = \mathsf{ZXt}(x, n)$	$\langle y\rangle^n = \Pi^{m,n}_{\mathsf{ZXt}}(\langle x\rangle^m)$	$\zeta_n(y) = \zeta_m(x) \bmod 2^n, m \le n$	$\lambda(m+1) + 13m + n$
Truncate-and-Reduce [64]	$y = \mathsf{TR}(x, s)$	$\langle y\rangle^{\ell-s} = \Pi^{\ell,s}_{\mathsf{TR}}(\langle x\rangle^\ell)$	Upper $\ell - s$ bits of x	$\lambda(s+1) + \ell + 13s$
Unsigned Mixed-bitwidth Multiplication [64]	$z = x *_\ell y$	$\langle z\rangle^\ell = \Pi^{m,n,\ell}_{\mathsf{UMult}}(\langle x\rangle^m, \langle y\rangle^n)$	$\zeta_\ell(z) = \zeta_m(x)\cdot\zeta_n(y) \bmod 2^\ell$, $\ell \ge \mathsf{max}(m, n)$	$\lambda(3\mu+\nu) + \mu(\mu+2\nu)$ $+16(m+n)$
Signed Mixed-bitwidth Multiplication [64]	$z = x *'_\ell y$	$\langle z\rangle^\ell = \Pi^{m,n,\ell}_{\mathsf{SMult}}(\langle x\rangle^m, \langle y\rangle^n)$	$\mathsf{int}_\ell(z) = \mathsf{int}_m(x)\cdot\mathsf{int}_n(y) \bmod 2^\ell$, $\ell \ge \mathsf{max}(m, n)$	$\lambda(3\mu+\nu) + \mu(\mu+2\nu)$ $+16(m+n)$
Most Significant Non-Zero Bit [64], [74]	$k, K = \mathsf{MSNZB}(x)$	$\langle k\rangle^\ell, \langle K\rangle^\ell = \Pi^\ell_{\mathsf{MSNZB}}(\langle x\rangle^\ell)$	k, s.t. $x_k = 1 \wedge \forall i > k, x_i = 0$, $K = 2^{\ell-1-k}$	$\le \lambda(5\ell - 4) + \ell^2$

Table I: 2PC building blocks used by SECFLOAT. All communication is in bits. $\mu = \mathsf{min}(m, n), \nu = \mathsf{max}(m, n)$.

2. Primitives

本节介绍基本的原语函数。

2.0 Checking for overflows and underflows

如果 α 上溢出，则设置为 NaN；如果下溢出，则设置为0。具体函数如下：

Functionality $\mathcal{F}^{p,q}_{\mathsf{FPCheck}}(\alpha)$

1: $\alpha = (z, s, e, m)$
2: **if** $\mathbf{1}\{e > 2^{p-1} - 1\}$ **then**
3: $\quad m = 2^q;\ e = 2^{p-1}$
4: **if** $\mathbf{1}\{z = 1\} \vee \mathbf{1}\{e < 2 - 2^{p-1}\}$ **then**
5: $\quad m = 0;\ e = 1 - 2^{p-1};\ z = 1$
6: Return (z, s, e, m)

Fig. 1: *Checking for overflows and underflows.*

2.1 Rounding

$x \gg_R r$ 对于ℓ比特、扩大因子为 s 的定点数输入 x，返回比特位长为 $(\ell - r)$、扩大因子为 $(s - r)$ 的结果，并且截断结果向偶舍入（rounding-nearest ties to even）。形式化表示为 $\left|\langle y\rangle_{\ell-r,s-r} - \langle x\rangle_{\ell,s}\right| \le 2^{-(s-r)-1}$。函数如下：

Functionality $\mathcal{F}^{\ell}_{\mathsf{RNTE}}(x, r)$

1: $a = \mathsf{TRS}(x, r-2)$; $\mathsf{idx} = a \bmod 8$
2: $c = L_{\mathsf{RNTE}}(\mathsf{idx})$, $c \in \{0,1\}$
3: Return $\mathsf{TR}(a, 2) + \mathsf{ZXt}(c, \ell - r)$

Fig. 2: *Round Nearest Ties to Even:* $x \gg_R r$.

对于 $x = x_{\ell-r-1} \| d \| g \| x_{r-1}$，其中 $x_{\ell-r-1} \in \{0,1\}^{\ell-r-1}$，$d, g \in \{0,1\}$，$x_{r-1} \in \{0,1\}^{r-1}$。首先将 x_{r-1} 利用TRS替换为比特 f（如果 x_{r-1}中任意比特为1，则$f = 1$）；进一步，令 $idx = d \| g \| f \|$。最后用LUT计算 $L_{RNTE}(d \| g \| f) = g \wedge (d \vee f)$ 实现向偶取整。

2.2 Round&Check

在浮点计算中，需要将正则化的 $Q+1$ 比特尾数 $m \in [2^Q, 2^{Q+1}]$ 从 Q 比特精度减少为 q 比特精度，因此我们需要舍入 $Q-q$ 比特，最终结果是 $q+1$ 比特。然而，如果 $m > 2^{Q+1} - 2^{Q-q-1}$，那么 $\frac{m}{2^{Q-q}}$则会比 $2^{q+1} - 1$更接近 2^{q+1}，造成溢出。这种情况则需要将 e 增加 $e+1$，并将舍入后的尾数部分设为 2^q。函数如下：

Functionality $\mathcal{F}^{p,q,Q}_{\mathsf{Round}^*}(e, m)$

1: **if** $\mathbf{1}\{m \geq 2^{Q+1} - 2^{Q-q-1}\}$ **then**
2: Return $(e+1, 2^q)$
3: **else**
4: Return $(e, m \gg_R (Q-q))$

Fig. 3: *Round mantissa and check for overflow.*

2.3 Multiplication

给定浮点数 α_1 和 α_2，二者的准确乘积 $\alpha = (1 - \alpha.z)(1 - 2\alpha.s)2^{\mathsf{int}(\alpha_1.e)+\mathsf{int}(\alpha_2.e)}$.
$\langle \alpha_1.m \rangle_{q+1,q} \cdot \langle \alpha_2.m \rangle_{q+1,q}$，其中 $\alpha.z = \alpha_1.z \vee \alpha_2.z$ 且 $\alpha.s = \alpha_1.s \oplus \alpha_2.s$。具体函数如下：

Functionality $\mathcal{F}^{p,q}_{\mathsf{FPMul}}(\alpha_1, \alpha_2)$

1: $e = \alpha_1.e + \alpha_2.e$
2: $m = \alpha_1.m *_{2q+2} \alpha_2.m$
3: **if** $\mathbf{1}\{m < 2^{2q+1} - 2^{q-1}\}$ **then**
4: $m = m \gg_R q \bmod 2^{q+1}$
5: **else**
6: $m = m \gg_R (q+1)$; $e = e + 1$
7: $s = \alpha_1.s \oplus \alpha_2.s$; $z = \alpha_1.z \vee \alpha_2.z$
8: Return $\alpha = \mathcal{F}^{p,q}_{\mathsf{FPCheck}}(z, s, e, m)$

Fig. 4: *Floating-Point Multiplication:* $\alpha_1 \boxdot_{p,q} \alpha_2$

首先指数部分计算加法，其次尾数部分计算乘法并扩大位长得到具有 $2q$比特小数部分的定点数 $m \in [2^{2q}, (2^{q+1}-1)^2]$。然后对于位长进行舍入：

- 如果 m 是正则化的，则只需要直接舍入 q 比特；
- 否则需要舍入 $q+1$ 比特，并令 $e+1$。

需要注意的是，如果 $m \in [2^{2q+1}-2^{q-1}, 2^{2q+1}-1]$，则第一种情况的舍入结果溢出。因此第一种情况的舍入条件设置为 $m < 2^{2q+1}-2^{q-1}$。最后验证最终的结果是否溢出。

2.4 Addition

给定浮点数 α_1 和 α_2，令 β_1 表示二者中绝对值较大的，β_2表示较小的。
首先计算二者指数部分的差距$d = \beta_1.e - \beta_2.e$：

- 如果差距过大 $d > q+1$，直接设置结果为 β_1；
- 否则，左移 $\beta_1.m$ 以d比特，使得二者的指数部分都变为 $\beta_2.e$，然后根据二者的符号关系对尾数部分进行加法或者减法得到 m。

进一步，对 m 进行归一化操作，即将 m 左移 $2q+1-k$ 比特，其中 $k = MSNZB(m)$。同时，设置扩大因子为 $2q+1$。同时，从 $e = e-(2q+1-k)+(q+1)$ 以抵消左移和扩大因子的变化。
最后，调用舍入函数将扩大因子从 $2q+1$ 降为 q。并检查最终结果是否溢出。函数如下：

Functionality $\mathcal{F}^{p,q}_{\mathsf{FPAdd}}(\alpha_1, \alpha_2)$

1: $(e_{\mathsf{LT}}, e_{\mathsf{EQ}}) = \mathsf{LT\&EQ}(\alpha_1.e, \alpha_2.e)$
2: $m_{\mathsf{LT}} = \mathbf{1}\{\alpha_1.m < \alpha_2.m\}$
3: $(\beta_1, \beta_2) = e_{\mathsf{LT}} \oplus (e_{\mathsf{EQ}} \wedge m_{\mathsf{LT}})\ ?\ (\alpha_2, \alpha_1) : (\alpha_1, \alpha_2)$
4: $d = \beta_1.e - \beta_2.e$
5: **if** $\mathbf{1}\{d > q+1\}$ **then**
6: Return β_1
7: **else**
8: $m_1 = \beta_1.m *_{2q+2} 2^d$
9: $m_2 = \mathsf{ZXt}(\beta_2.m, 2q+2)$
10: $m_2 = (\beta_1.s \oplus \beta_2.s\ ?\ -m_2 : m_2)$
11: $m = m_1 + m_2;\ e = \beta_2.e$
12: $k, K = \mathsf{MSNZB}(m), K = 2^{2q+1-k}$
13: $m = m *_{2q+2} K;\ e = e + k - q$
14: $(e, m) = \mathcal{F}^{p,q,2q+1}_{\mathsf{Round}^*}(e, m).$
15: $z = \mathbf{1}\{m = 0\};\ s = \beta_1.s$
16: Return $\alpha = \mathcal{F}^{p,q}_{\mathsf{FPCheck}}(z, s, e, m)$

Fig. 5: *Floating-Point Addition:* $\alpha_1 \boxplus_{p,q} \alpha_2$

2.5 Division

给定的分子 α_1 和分母 α_2，得到 $\alpha = \alpha_1.z \cdot (1 - 2\alpha.s) \cdot 2^{\mathsf{int}_{p+2}(\alpha_1.e) - \mathsf{int}_{p+2}(\alpha_2.e)} \cdot (\langle \alpha_1.m \rangle_{q+1,q} / \langle \alpha_2.m \rangle_{q+1,q})$，其中 $\alpha.s = \alpha_1.s \oplus \alpha_2.s$。其中，指数部分的减法很简单：$e = \alpha_1.e - \alpha_2.e$。对于尾数部分：

- 如果 $\alpha_1.m < \alpha_2.m$，则 $m_1 = \alpha_1.m \cdot 2$，并令 $e - 1$；
- 进而计算 $m1/\alpha_2.m$

如此，则结果的尾数是正则化的。为了求商，先利用牛顿迭代法近似计算 $r_t \approx \frac{1}{\alpha_2.m}$ 的倒数，从而得到 $m' = m_1 \cdot r_t$。最后的结果需要在 m' 和 $m' + 1$ 中选择，这一步则需要比较 m_1 分别到 $m' \cdot \alpha_2.m$ 和 $(m' + 1) \cdot \alpha_1 \cdot m$ 的距离。具体函数如下。

Functionality $\mathcal{F}^{p,q}_{\mathsf{FPDiv}}(\alpha_1, \alpha_2)$

1: $m_1 = \mathsf{ZXt}(\alpha_1.m, q + 2)$; $m_2 = \alpha_2.m$; $e = \alpha_1.e - \alpha_2.e$
2: **if** $\mathbf{1}\{\alpha_1.m < \alpha_2.m\}$ **then**
3: $\quad m_1 = 2m_1$; $e = e - 1$
4: $t = 2$; $g = \lceil \frac{q+1}{2^t} \rceil + 1$; $k_0 = g + 1$
5: $h = \mathsf{TR}(m_2, q - g) \bmod 2^g$
6: $r_0 = L_{\mathsf{recp-init}}(h), r_0 \in \{0,1\}^{k_0+2}$
7: **for** $i = 1$ to t **do**
8: $\quad k_i = 2^i \cdot (g - 1) + 3$
9: $\quad f_i = 2^{k_i} - \mathsf{TR}(m_2 *_{k_{i-1}+q+1} r_{i-1}, q + k_{i-1} - k_i)$
10: $\quad r_i = r_{i-1} *_{k_i+2} 2^{k_i - k_{i-1}} + \mathsf{TR}(r_{i-1} *'_{k_i+k_{i-1}+2} f_i, k_{i-1})$
11: $m'' = m_1 *_{k_t+q+2} r_t$; $m' = \mathsf{TR}(m'', k_t)$
12: $y_1 = m_2 *_{q+3} m'$; $y_2 = y_1 + \mathsf{ZXt}(m_2, q + 3)$
13: $y = (m_1 *_{q+3} 2^{q+1}) - (y_1 + y_2)$; $(\mathsf{lt}, \mathsf{eq}) = \mathsf{LT\&EQ}(0, y)$
14: $m = \mathsf{lt} \oplus (\mathsf{eq} \wedge \mathbf{1}\{m \bmod 2 = 1\})\ ?\ m' + 1 : m'$
15: $s = \alpha_1.s \oplus \alpha_2.s$, $z = \alpha_1.z$
16: Return $\alpha = \mathcal{F}^{p,q}_{\mathsf{FPCheck}}(z, s, e, m)$

Fig. 6: *Floating-Point Division:* $\alpha_1 \boxslash_{p,q} \alpha_2$.

3. Math Functions

本节总结SecFloat对于三角函数、对数和指数函数的计算。为了提升性能，在计算数学函数的时候为了提升性能对于加法和除法做了性能上的优化（cheap addtion and division）。虽然这带来了一定的误差，但是依旧满足精度要求。接下来，$\mathsf{Float}_{p,q}(r)$ 对实数 r 进行正确舍入之后得到的由参数 p, q 定义的浮点数。

3.1 Spline Evaluation

一个n段的对于分段函数（splines，or piecewise polynomials）F，可以用分段点 $\mathcal{K} = \{\kappa_1, ..., \kappa_{n+1}\}$ 和 n 个阶为 d 的多项式定义。令所有的系数为 $\Theta = \{\theta_i^{(j)}\}_{i=0,j=1}^{d,n}$，那么对于 $\delta \in [\kappa_1, \kappa_{n+1})$，可以计算

$$F(\delta) = \theta_0^{(j)} + \theta_1^{(j)} \cdot \delta + \cdots + \delta_d^{(j)} \cdot \delta_d, \delta \in [\kappa_j, \kappa_{j+1}).$$

因此，主要的任务在于对于给定的 δ 从 Θ 中选择合适的系数。本文使用δ的指数部分和尾数部分少量的比特位和 $\mathcal{K}$ 定义 idx (例如 $idx \in \{0,1\}^\ell$) 和 $K = \{k_1, \ldots, k_{n+1}\}$，其中 $k_j \in \{0,1\}^\ell$ 对于 $j \in [n+1]$ 满足 当且仅当 $idx \in [k_j, k_{j+1}]$ 时有 $\delta \in [\kappa_j, \kappa_{j+1})$。
已有的方法有两种：

- 比较 idx 和 $\{k_1, ..., k_{n+1}\}$，根据比较结果选择合适的系数。但是该方法需要n次比较；
- 利用LUT将 idx 映射到正确的系数，该LUT一共有 2^ℓ 大小，每一项包含 $d+1$ 个浮点数参数。由于LUT的开销和表中元素的大小成正比，因此该方法开销也很大。

本文提出的方法如下所示：

$$\mathcal{F}_{\mathsf{GetC}}^{\ell,n,d,p,q}(x, \Theta = \{\theta_i^{(j)}\}_{i=0,j=1}^{d,n}, K = \{k_j\}_{j=1}^{n+1})$$

1: $\{v_j\}_{j\in[n]} = L^K_{\mathsf{ActiveInterval}}(x), x \in \{0,1\}^\ell, v_j \in \{0,1\}$
2: For $j \in [n]$, $V_j = \mathsf{ZXt}(v_j, q+1)$; $V_j' = V_j \bmod 2^{p+2}$
3: **for** $i = 0$ to d **do**
4: $\quad z_i = s_i = e_i = m_i = 0$
5: $\quad$ **for** $j = 1$ to n **do**
6: $\quad\quad z_i = z_i + v_j \cdot \theta_i^{(j)}.z$; $s_i = s_i + v_j \cdot \theta_i^{(j)}.s$
7: $\quad\quad e_i = e_i + V_j' \cdot \theta_i^{(j)}.e$; $m_i = m_i + V_j \cdot \theta_i^{(j)}.m$
8: Return $\{(z_i, s_i, e_i, m_i)\}_{i=0}^d$

Fig. 7: *Retrieve the coefficients of the active piece in an n-piece d-degree spline using an ℓ-bit index x:* $\mathsf{GetC}_{\ell,n,d,p,q}(x, \Theta, K)$.

LUT $L^K_{\mathsf{ActiveInterval}}$ 将 idx 映射为 每一项为 n 比特的one-hot比特向量 $v \in \{0,1\}^n$。v 满足如果 $idx \in [k_j, k_{j+1})$ 则有 $v_j = 1$，否则为 0。在提取 v 之后，则可以通过 $\sum_j v_j \cdot \theta_j$ 计算正确的系数。

从而LUT的每一项和 d 无关，本文使用的分段函数满足 $\ell \leq 8$ 且 $n \leq 64$。具体来说，对于tangent函数只需要两个参数，用简单的LUT性能更好；对于指数函数和对数函数，LUT每一项包含6-8个系数，本文的方法能够降低通信 $2.3 - 3.7\times$；和基于比较的方法相比，本文的方法能够比基于比较的方法降低通信 $3.1 - 3.9\times$。对于多项式计算，本文使用了Horner's rule优化。

3.2 Sine

对于 $sin\pi(\alpha)$，对于：

- 如果 $|\alpha| \geq 2^{23}$，返回 0 因为输入一定是整数；

- 如果 $|\alpha| < 2^{-14}$，近似为 $\pi\alpha$。

对于其他情况，计算如下：

1）Range Reduction (Step 9-14):

$\sin \pi$ 是奇周期函数，因此可以将 $sin\pi(\alpha)$ ($|\alpha| \in [2^{-14}, 2^{23})$)
归约到计算 $sin\pi(\delta)$ ($\delta \in [0, 0.5]$)。令 $s = 1\{\alpha < 0\}$ 和 $\beta = |\alpha| = 2 \cdot K + a + n$ 其中 $K \in \mathbb{N}, a \in \{0, 1\}$ 且 $n \in [0, 1)$。
对于 $\delta \in [0, 0.5]$，定义

$$\delta = \begin{cases} n, n < 0.5; \\ 1 - n, else \end{cases}$$

根据 $sin\pi(1 - x) = sin\pi(x)$，有

$$sin\pi(\alpha) = (-1)^{a+s} \cdot sin\pi(\delta)$$

2) Polynimal Evaluation (Step 15-28):

对于 $\delta < 2^{-14}$，利用 $\pi \cdot \alpha$ 近似；其余的输入范围分为两种情况，我们针对每种情况分别涉及了 $d = 5$ 的分段函数 (函数形式为 $\theta_1\delta + \theta\delta^3 + \theta\delta^5$):

- 如果 $\delta \in [2^{-14}, \frac{1}{32})$，我们设计了9段的分段函数$F^1$，其在LUT中用指数部分的低4比特决定；
- 如果 $\delta \in [\frac{1}{32}, 0.5]$，我们设计了34段的分段函数 F^2，指数函数的低2比特和尾数的高5比特可以定位F^2对于δ的活跃区间（除了$\delta = 0.5$，因为区间划分左闭右开）。对于 $\delta = 0.5$我们置idx_2对应最后一个区间 $[\frac{63}{128}, 0.5)$ (Step 24)。

3）Output Compensation (Setp 29)

根据1）中公式，如果$a \oplus s = 0$ 输出 $sin\pi(\delta)$，否则返回 $-sin\pi(\delta)$。具体函数如下。

Functionality $\mathcal{F}^{8,23}_{\mathsf{FPsin}\pi}(\alpha)$

$p = 8;\ q = 23;\ Q = 27$
$\alpha' = (\alpha.z, 0, \alpha.e, \alpha.m *_{Q+1} 2^{Q-q})$
if $\mathbf{1}\{\alpha.e \geq 23\}$ **then**
  Return $\mathtt{Float}_{p,q}(0)$
else if $\mathbf{1}\{\alpha.e < -14\}$ **then**
  $\gamma = \mathtt{Float}_{p,Q}(\pi) \boxdot_{p,Q} \alpha'$
  Return $(\gamma.z, \gamma.s, \mathcal{F}^{p,q,Q}_{\mathsf{Round}*}(\gamma.e, \gamma.m))$
else
  $m = \alpha.m *_{q+15} 2^{\alpha.e+14}$
  $a = \mathsf{TR}(m, q+14);\ n = m \bmod 2^{q+14}$
  $f = (\mathbf{1}\{n > 2^{q+13}\}\ ?\ 2^{q+14} - n : n)$
  $k, K = \mathsf{MSNZB}(f);\ f = f *_{q+15} K$
  $z = \mathbf{1}\{f = 0\};\ e = (z\ ?\ -2^{p-1} + 1 : k - q - 14)$
  $\delta = (z, 0, e, \mathsf{TR}(f, q + 14 - Q))$
  if $\mathbf{1}\{\delta.e < -14\}$ **then**
    $\mu = \mathtt{Float}_{p,Q}(\pi) \boxdot_{p,Q} \delta$
  else
    if $\mathbf{1}\{\delta.e < -5\}$ **then**
      $\mathsf{idx}_1 = \delta.e + 14 \bmod 2^4$
      $(\theta_1, \theta_3, \theta_5) = \mathsf{GetC}_{4,9,5,p,Q}(\mathsf{idx}_1, \Theta^1_{\mathsf{sin}}, K^1_{\mathsf{sin}})$
    else
      $\mathsf{idx}_2 = 32 \cdot (\delta.e + 5 \bmod 2^7)$
      $\mathsf{idx}_2 = \mathsf{idx}_2 + \mathsf{ZXt}(\mathsf{TR}(\delta.m, Q-5) \bmod 32, 7)$
      $\mathsf{idx}_2 = \mathbf{1}\{\delta.e = -1\}\ ?\ 127 : \mathsf{idx}_2$
      $(\theta_1, \theta_3, \theta_5) = \mathsf{GetC}_{7,34,5,p,Q}(\mathsf{idx}_2, \Theta^2_{\mathsf{sin}}, K^2_{\mathsf{sin}})$
    $\Delta = \delta \boxdot_{p,Q} \delta$
    $\mu = ((\theta_5 \boxdot_{p,Q} \Delta) \boxplus^*_{p,Q} \theta_3) \boxdot_{p,Q} \Delta$
    $\mu = (\mu \boxplus^*_{p,Q} \theta_1) \boxdot_{p,Q} \delta$
  Return $(\mu.z, a \oplus \alpha.s, \mathcal{F}^{p,q,Q}_{\mathsf{Round}*}(\mu.e, \mu.m))$.

Fig. 8: *Floating-point* $\mathsf{sin}\pi_{8.23}(\alpha)$.

3.3 Cosine, Tangent, Logarithm, Exponentiation & 2PC Protocols for Floating-Point

其余数学函数的计算遵循和Sine函数相同的构造模板，此部分后续有时间逐步补充。

本文主要列了浮点数相乘、check和 rounding 的具体协议如下，该三个协议都比较简单，对照之前介绍的函数设计即可理解，在这里不再展开。

4. Implementation & Evaluation

首先比较了SecFloat和ABY-F、MP-SPDZ的基本原语性能和精度比较：

Technique	Time (in s) for Batch Size 10^3	10^4	10^5	Comm. (KiB)	ULP Error
CMP					
SECFLOAT	0.02	0.07	0.30	1.11	NA
ABY-F	0.05 (2.3x)	0.25 (3.9x)	2.08 (7.0x)	6.93 (6.2x)	NA
MP SPDZ	0.22 (10.7x)	1.24 (19.0x)	12.85 (43.1x)	89.64 (80.7x)	NA
MULT					
SECFLOAT	0.05	0.14	0.69	3.13	0.5
ABY-F	0.34 (7.0x)	3.22 (23.0x)	32.20 (46.7x)	95.74 (30.6x)	0.5
MP SPDZ	0.38 (7.7x)	1.36 (9.7x)	10.28 (14.9x)	72.72 (23.2x)	$> 10^6$
ADD					
SECFLOAT	0.15	0.56	3.91	11.10	0.5
ABY-F	0.12 (0.8x)	1.00 (1.8x)	9.35 (2.4x)	31.33 (2.8x)	0.5
MP SPDZ	3.63 (23.8x)	36.05 (64.4x)	– –	2561.22 (230.7x)	$> 10^6$
DIV					
SECFLOAT	0.14	0.35	2.43	10.27	0.5
ABY-F	0.46 (3.2x)	3.90 (11.2x)	38.36 (15.8x)	119.55 (11.6x)	0.5
MP SPDZ	11.33 (78.4x)	113.25 (326.3x)	– –	2462.68 (239.8x)	$> 10^6$
SIN					
SECFLOAT	0.46	1.59	11.27	42.36	1
ABY-F	0.48 (1.0x)	4.49 (2.8x)	44.77 (4.0x)	163.49 (3.9x)	$> 10^7$
COS					
SECFLOAT	0.50	1.60	11.29	42.41	1
ABY-F	0.46 (0.9x)	4.51 (2.8x)	45.21 (4.0x)	163.86 (3.9x)	$> 10^7$
TAN					
SECFLOAT	0.66	2.21	14.91	60.38	1
ABY-F	1.32 (2.0x)	12.39 (5.6x)	122.37 (8.2x)	445.86 (7.4x)	$> 10^7$
EXP2					
SECFLOAT	0.41	1.51	11.28	37.23	1
ABY-F	0.85 (2.0x)	8.38 (5.6x)	83.61 (7.4x)	304.89 (8.2x)	1
LOG2					
SECFLOAT	0.53	1.97	14.94	51.48	1
ABY-F	0.92 (1.7x)	9.04 (4.6x)	90.53 (6.1x)	330.77 (6.4x)	$> 10^6$

Table II: Comparison of SECFLOAT with ABY-F and MP-SPDZ. The numbers in parentheses show our improvement factor. '-' denotes tool crash.

OP	Rounds ABY-F	Rounds SECFLOAT	OP	Rounds ABY-F	Rounds SECFLOAT
CMP	12	11	SIN/COS	95/98	196
MULT	47	27	TAN	394	268
ADD	59	49	EXP2	100	187
DIV	296	84	LOG2	157	256

Table III: Rounds comparison of SECFLOAT with ABY-F.

进一步是距离感应测试：

Technique	Time (in s) for Batch Size			Comm. (KiB)	Average ULP Error
	10^3	10^4	10^5		
Scenario I: Parties have secret inputs					
SECFLOAT	0.96	2.86	21.42	75.27	33.86
ABY-F	2.29 (2.4x)	24.33 (8.5x)	250.68 (11.7x)	833.98 (11.1x)	
Scenario II: Parties have secret-shares of inputs					
SECFLOAT	2.41	7.59	55.55	211.26	1.05
ABY-F	4.21 (1.7x)	40.27 (5.3x)	415.61 (7.5x)	1437.73 (6.8x)	19612.95

Table IV: Comparison of SECFLOAT with ABY-F on privacy-preserving proximity testing. The numbers in parentheses show our improvement factor.

最后是面向机器学习广告推荐模型的性能测试：

Task	Size	Time (s)	Comm. (GB)	Improvement
Backprop.	1	40.3	10.9	11.3×
Backprop.	32	1020.5	329.6	9.1×
Inference	1	18.3	4.7	8.8×
Inference	32	1036.0	151.58	8.8×

Table V: Secure machine learning with SECFLOAT and its communication improvements over ABY-F.

5. Conclusion

本文设计了面向单精度浮点数的半诚实安全2PC计算库，作者提出接下来面向恶意敌手和64比特浮点数的安全计算库也是一个研究点。