The Cost of IEEE Arithmetic in Secure Computation

本次分享的是发表在LatinCrypt'21上的论文 The Cost of IEEE Arithmetic in Secure Computation。

目前常见的安全多方计算方案在处理浮点数的时候，都是将浮点数转化为定点数，然后在定点数上进行安全计算。本文主要量化了IEEE754格式的浮点数在两种MPC框架：基于线性秘密分享和基于布尔电路方案，下的性能开销。为之后的研究提供了指导意义。

0. Notation

首先介绍一下本文设计的一些标记：$\mathbb{Z}_{\langle k\rangle}=\{x\in\mathbb{Z}:-2^{k-1}\le x\le 2^{k-1}-1\}$，进一步，可以将 $\mathbb{Z}_{\langle k\rangle}$ 映射到 $\mathbb{F}_p$: $x\rightarrow x \pmod p$ ($k<\log_2 p$)。$[n]=\{1,\dots,n\}$。

1. 基础知识

1.1 MPC-Black Box

本文主要涉及了线性秘密分享（LSSS-Based MPC）和多方下的混乱电路（GC-Based n-party MPC），前者计算在算术电路上（$\mathbb{F}_p$，$[x]_p$），后者作用在布尔电路上（$\mathbb{F}_2$，$[x]_2$）。

LSSS-Based MPC: 对于LSSS下的方案，最简单的是全门限结构：$x=\sum x_i \pmod p$，进一步还可能有对于x 的认证（MAC），典型的方案SPDZ；而在 $t<n/2$ 门限结构下，Shamir's Secret Sharing是常用的另一种方案；当 $p=2$ 时，本文采用replicated secret sharing技术。

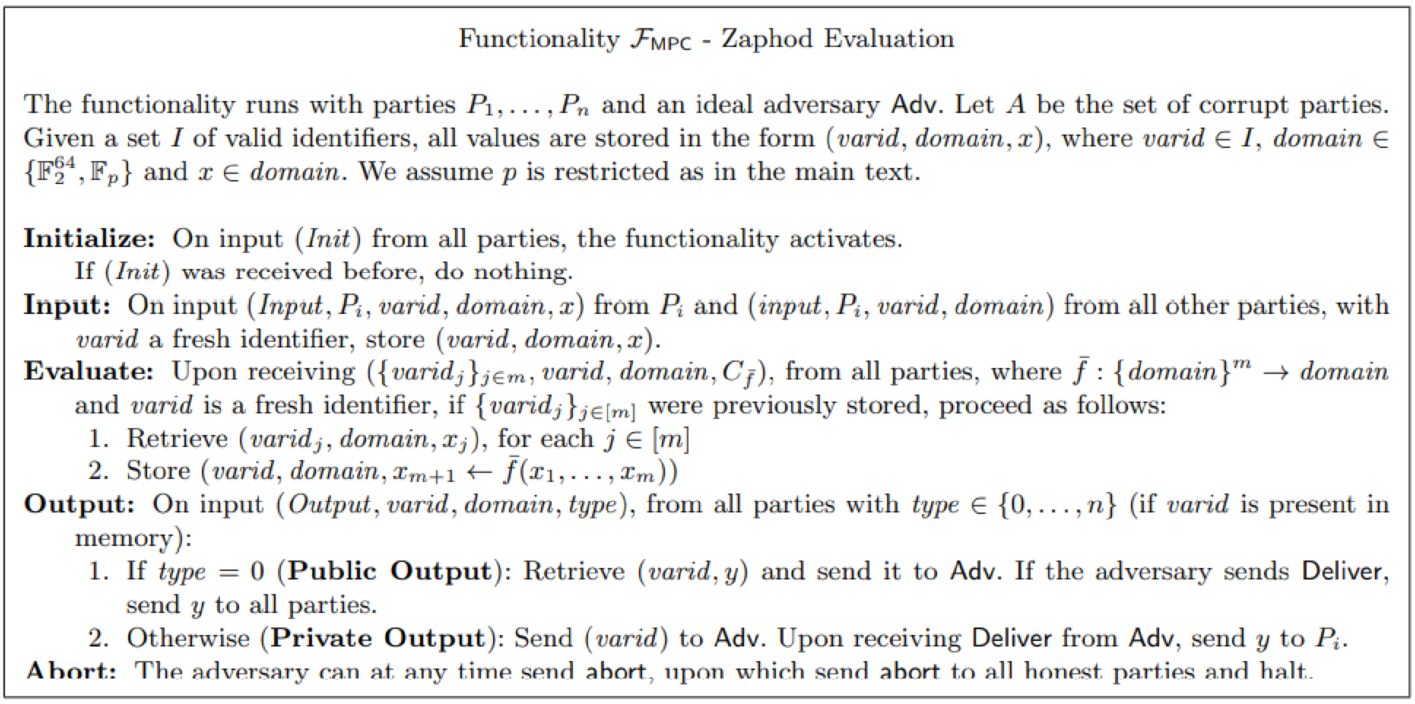
GC-Based n-party MPC: 在全门限结构下不能利用replicated secret sharing的一些性质，因此本文采用了n-party下的GC-Based方案。进一步，为了实现认证秘密分享，本文还引进了BDOZ-Style MAC。

Combining the Binary and Large Prime Variants: 本文使用Zaphod技术实现布尔和算术（模素数）下的转化。因为 $x\in\mathbb{Z}_{\langle 64\rangle}$，因此所有的数据比特位长 ≤ 64。故而，在从布尔转化到算术隐藏 x 的时候

$$x+\sum_{i=0}^{\lceil\log_2 p\rceil} b_i\cdot 2^i \pmod p$$

需要 $64 + sec < \log_2 p$，其中sec是统计安全参数。

Zaphod-Evaluation的functionality如下：


Functionality $\mathcal{F}_{\mathsf{MPC}}$ - Zaphod Evaluation

The functionality runs with parties $P_1, \ldots, P_n$ and an ideal adversary Adv. Let A be the set of corrupt parties. Given a set I of valid identifiers, all values are stored in the form $(varid, domain, x)$, where $varid \in I$, $domain \in \{\mathbb{F}_2^{64}, \mathbb{F}_p\}$ and $x \in domain$. We assume p is restricted as in the main text.

Initialize: On input $(Init)$ from all parties, the functionality activates.
If $(Init)$ was received before, do nothing.

Input: On input $(Input, P_i, varid, domain, x)$ from P_i and $(input, P_i, varid, domain)$ from all other parties, with $varid$ a fresh identifier, store $(varid, domain, x)$.

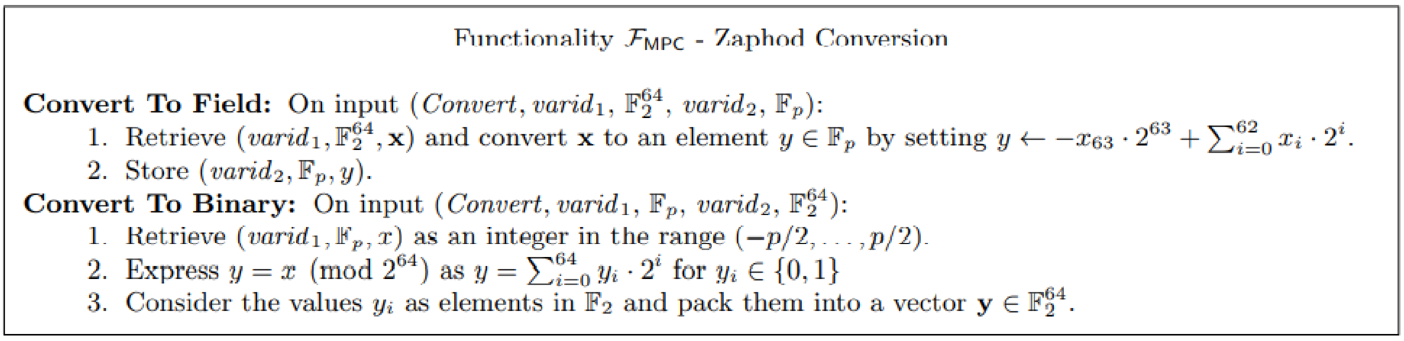
Evaluate: Upon receiving $(\{varid_j\}_{j \in m}, varid, domain, C_{\bar{f}})$, from all parties, where $\bar{f} : \{domain\}^m \rightarrow domain$ and $varid$ is a fresh identifier, if $\{varid_j\}_{j \in [m]}$ were previously stored, proceed as follows:
1. Retrieve $(varid_j, domain, x_j)$, for each $j \in [m]$
2. Store $(varid, domain, x_{m+1} \leftarrow \bar{f}(x_1, \ldots, x_m))$

Output: On input $(Output, varid, domain, type)$, from all parties with $type \in \{0, \ldots, n\}$ (if $varid$ is present in memory):
1. If $type = 0$ (**Public Output**): Retrieve $(varid, y)$ and send it to Adv. If the adversary sends Deliver, send y to all parties.
2. Otherwise (**Private Output**): Send $(varid)$ to Adv. Upon receiving Deliver from Adv, send y to P_i.

Abort: The adversary can at any time send abort, upon which send abort to all honest parties and halt.

Figure 1. Functionality $\mathcal{F}_{\mathsf{MPC}}$ - Zaphod Evaluation

$\mathbb{F}_p$ 和 $\mathbb{F}_2$ 之间的转化functionality如下：


Functionality $\mathcal{F}_{\mathsf{MPC}}$ - Zaphod Conversion

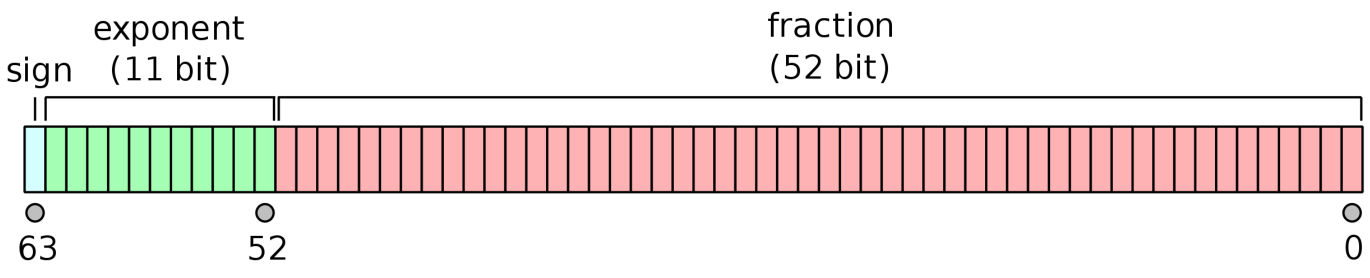
Convert To Field: On input $(Convert, varid_1, \mathbb{F}_2^{64}, varid_2, \mathbb{F}_p)$:
1. Retrieve $(varid_1, \mathbb{F}_2^{64}, \mathbf{x})$ and convert $\mathbf{x}$ to an element $y \in \mathbb{F}_p$ by setting $y \leftarrow -x_{63} \cdot 2^{63} + \sum_{i=0}^{62} x_i \cdot 2^i$.
2. Store $(varid_2, \mathbb{F}_p, y)$.

Convert To Binary: On input $(Convert, varid_1, \mathbb{F}_p, varid_2, \mathbb{F}_2^{64})$:
1. Retrieve $(varid_1, \mathbb{F}_p, x)$ as an integer in the range $(-p/2, \ldots, p/2)$.
2. Express $y = x \pmod{2^{64}}$ as $y = \sum_{i=0}^{64} y_i \cdot 2^i$ for $y_i \in \{0, 1\}$
3. Consider the values y_i as elements in $\mathbb{F}_2$ and pack them into a vector $\mathbf{y} \in \mathbb{F}_2^{64}$.

Figure 2. Functionality $\mathcal{F}_{\mathsf{MPC}}$ - Zaphod Conversion

1.2 IEEE-754

IEEE-754定义了标准的64比特浮点数格式：



从而数据真值可以表示为：

$$(-1)^{b63} \cdot (1 + \sum_{i=1}^{52}(b_{52-i \cdot 2^{-i}})) \cdot 2^{e-1023}$$

其中：

1. $e = 0, m = 0)$：表示0；
2. $e = 2047, m = 0$：表示 ∞；
3. $e = 2047, m \neq 0$：表示 NaN。

1.3 Secure-Floats via $\mathbb{F}_p$-Arithmetic

在MPC中，浮点数利用五元组 (v, p, z, s, err) 表示：

- $v \in [2^{\ell-1}, 2^{\ell})$：表示 $\ell + 1$ 比特的尾数，其最高位拥有置为1；
- $p \in \mathbb{Z}_{\langle k \rangle}$：表示带符号的指数部分；
- z：比特值，表示当前值是否为0；
- s：符号比特；
- err：如果没有舍入或者算术错误，$err = 0$；否则$err \neq 0$。
  其中ℓ 和 k 是公开参数。
  在公开真实值的时候，首先判断 $[b]_p \leftarrow ([err]_p = 0)$，然后公开 $([b]_p \cdot [v]_p, [b]_p \cdot [p]_p, [b]_p \cdot [s]_p, 1 - [b]_p)$。

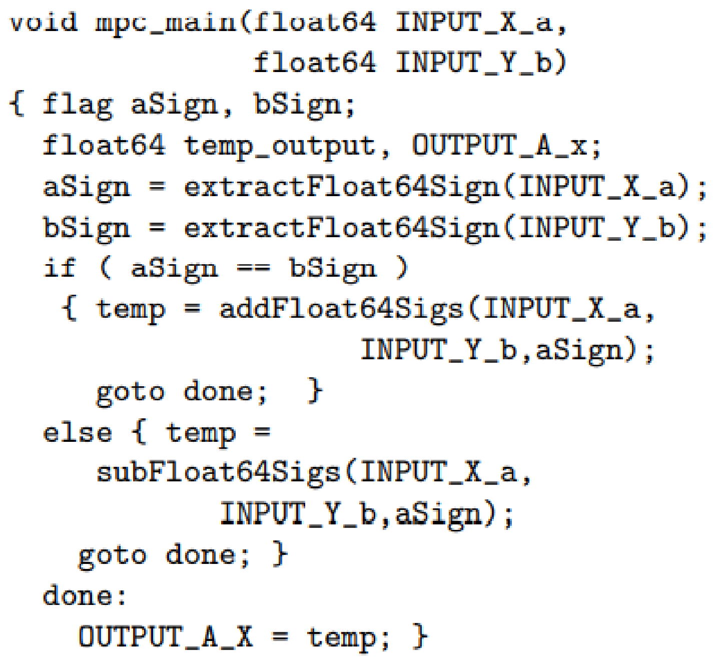
Fixed-Point Representation的表示之前的总结提到很多次，在此不做赘述。

2. Generating Circuits for IEEE Arithmetic

本文使用CBMC-GC将C语言代码编译为MPC代码，然后使用Scale-Mamba执行MPC代码。转化的方式很简单，只需要在C语言代码上做少量修改即可。例子如下：

```
float64 float64_add(float64 a,float64 b)
{ flag aSign, bSign;
  aSign = extractFloat64Sign(a);
  bSign = extractFloat64Sign(b);
  if ( aSign == bSign )
  { return addFloat64Sigs(a,b,aSign); }
  else
  {  return subFloat64Sigs(a,b,aSign); } }
```

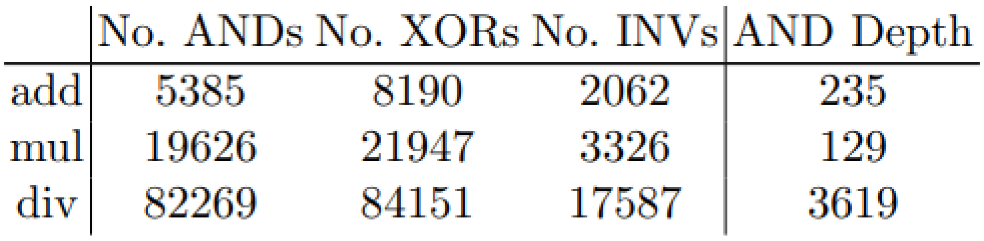
Fig. 3. The original SoftFloat C-code for IEEE-754 double addition



```
void mpc_main(float64 INPUT_X_a,
              float64 INPUT_Y_b)
{ flag aSign, bSign;
  float64 temp_output, OUTPUT_A_x;
  aSign = extractFloat64Sign(INPUT_X_a);
  bSign = extractFloat64Sign(INPUT_Y_b);
  if ( aSign == bSign )
   { temp = addFloat64Sigs(INPUT_X_a,
                  INPUT_Y_b,aSign);
     goto done;  }
  else { temp =
     subFloat64Sigs(INPUT_X_a,
            INPUT_Y_b,aSign);
    goto done; }
  done:
    OUTPUT_A_X = temp; }
```

Fig. 4. SoftFloat C-code after modification for CBMC-GC

转化之后的电路，对于加法、乘法和电路复杂度如下：



	No. ANDs	No. XORs	No. INVs	AND Depth
add	5385	8190	2062	235
mul	19626	21947	3326	129
div	82269	84151	17587	3619

3. Converting Between Representations

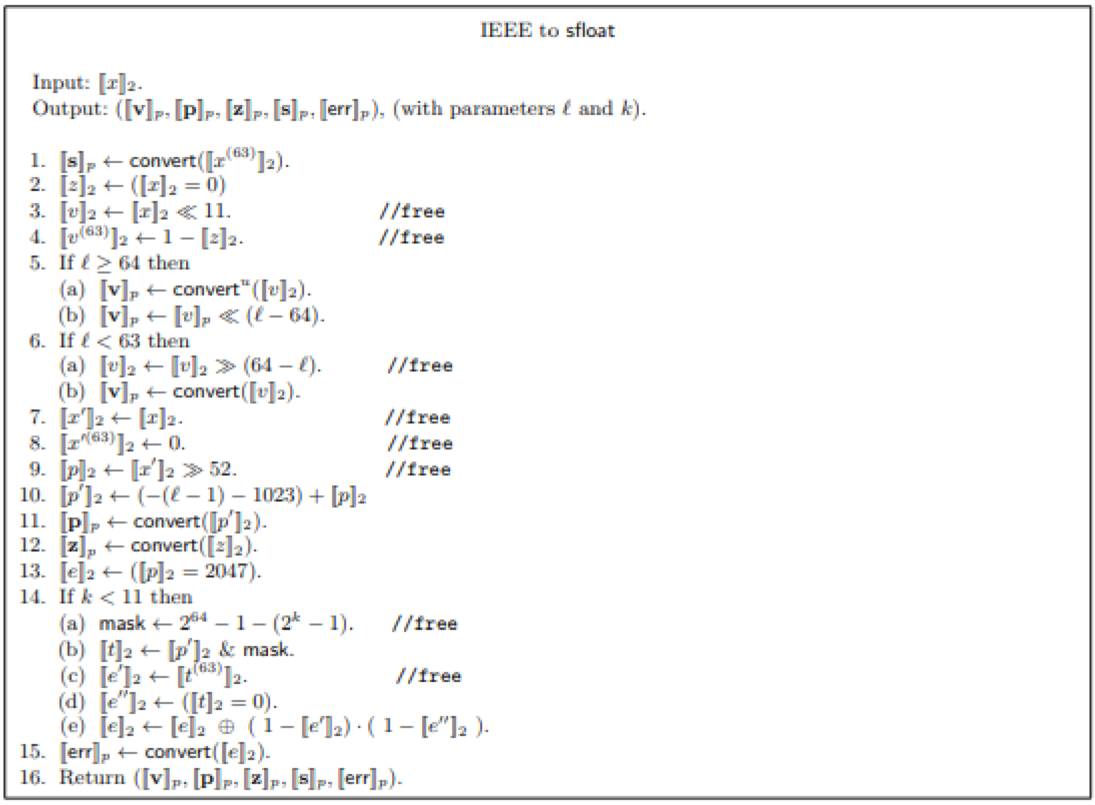
接下来，介绍布尔和算术（模素数）下的转化。本文使用daBits实现转化。首先，对于$[x]_p \leftarrow convert([x]_2)$，转化如下：

- $([r]_2, [r]_p) \leftarrow daBits$;
- $[v]_2 \leftarrow [x]_2 \oplus [r]_2$;
- $v \leftarrow Open([v]_2)$;
- $[x]_p \leftarrow v + [r]_p - 2 \cdot v \cdot [r]_p$。

类似的，$[x]_2 \leftarrow convert([x]_p)$转化如下：

- For $i = 0, ..., sec$ 生成 $([r_i]_2, [r_i]_p) \leftarrow daBits$;
- $[r]_p \leftarrow \sum_{i=0}^{sec} [r_i]_p \cdot 2^i$;
- $[v]_p \leftarrow [x]_p + [r]_p$;
- $v \leftarrow Open([v]_p)$;
- $[\mathbf{x}]_2 \leftarrow x - [\mathbf{r}]_2$.

基于上述两个转化，可以实现IEEE和sfloat之间的转化如下：

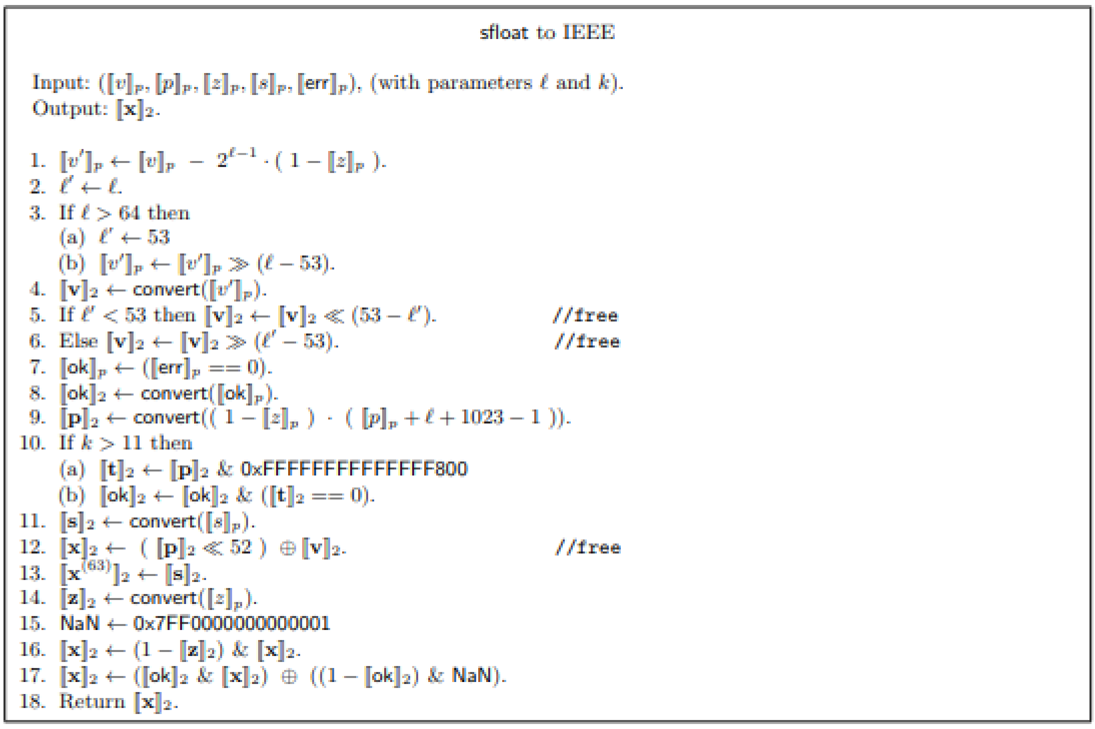


```
IEEE to sfloat

Input: $[x]_2$.
Output: $([\mathbf{v}]_p, [\mathbf{p}]_p, [\mathbf{z}]_p, [\mathbf{s}]_p, [\mathsf{err}]_p)$, (with parameters $\ell$ and $k$).

1. $[\mathbf{s}]_p \leftarrow \mathsf{convert}([x^{(63)}]_2)$.
2. $[z]_2 \leftarrow ([x]_2 = 0)$
3. $[v]_2 \leftarrow [x]_2 \ll 11$.                //free
4. $[v^{(63)}]_2 \leftarrow 1 - [z]_2$.           //free
5. If $\ell \geq 64$ then
   (a) $[\mathbf{v}]_p \leftarrow \mathsf{convert}''([v]_2)$.
   (b) $[\mathbf{v}]_p \leftarrow [v]_p \ll (\ell - 64)$.
6. If $\ell < 63$ then
   (a) $[v]_2 \leftarrow [v]_2 \gg (64 - \ell)$.   //free
   (b) $[\mathbf{v}]_p \leftarrow \mathsf{convert}([v]_2)$.
7. $[x']_2 \leftarrow [x]_2$.                    //free
8. $[x'^{(63)}]_2 \leftarrow 0$.                 //free
9. $[p]_2 \leftarrow [x']_2 \gg 52$.             //free
10. $[p']_2 \leftarrow (-(\ell - 1) - 1023) + [p]_2$
11. $[\mathbf{p}]_p \leftarrow \mathsf{convert}([p']_2)$.
12. $[\mathbf{z}]_p \leftarrow \mathsf{convert}([z]_2)$.
13. $[e]_2 \leftarrow ([p]_2 = 2047)$.
14. If $k < 11$ then
   (a) $\mathsf{mask} \leftarrow 2^{64} - 1 - (2^k - 1)$.   //free
   (b) $[t]_2 \leftarrow [p']_2 \,\&\, \mathsf{mask}$.
   (c) $[e']_2 \leftarrow [t^{(63)}]_2$.              //free
   (d) $[e'']_2 \leftarrow ([t]_2 = 0)$.
   (e) $[e]_2 \leftarrow [e]_2 \oplus (1 - [e']_2) \cdot (1 - [e'']_2)$.
15. $[\mathsf{err}]_p \leftarrow \mathsf{convert}([e]_2)$.
16. Return $([\mathbf{v}]_p, [\mathbf{p}]_p, [\mathbf{z}]_p, [\mathbf{s}]_p, [\mathsf{err}]_p)$.
```

Figure 5. IEEE to sfloat



```
sfloat to IEEE

Input: $([v]_p, [p]_p, [z]_p, [s]_p, [\mathsf{err}]_p)$, (with parameters $\ell$ and $k$).
Output: $[\mathbf{x}]_2$.

1. $[v']_p \leftarrow [v]_p - 2^{\ell-1} \cdot (1 - [z]_p)$.
2. $\ell' \leftarrow \ell$.
3. If $\ell > 64$ then
   (a) $\ell' \leftarrow 53$
   (b) $[v']_p \leftarrow [v']_p \gg (\ell - 53)$.
4. $[\mathbf{v}]_2 \leftarrow \mathsf{convert}([v']_p)$.
5. If $\ell' < 53$ then $[\mathbf{v}]_2 \leftarrow [\mathbf{v}]_2 \ll (53 - \ell')$.   //free
6. Else $[\mathbf{v}]_2 \leftarrow [\mathbf{v}]_2 \gg (\ell' - 53)$.                 //free
7. $[\mathsf{ok}]_p \leftarrow ([\mathsf{err}]_p == 0)$.
8. $[\mathsf{ok}]_2 \leftarrow \mathsf{convert}([\mathsf{ok}]_p)$.
9. $[\mathbf{p}]_2 \leftarrow \mathsf{convert}((1 - [z]_p) \cdot ([p]_p + \ell + 1023 - 1))$.
10. If $k > 11$ then
   (a) $[\mathbf{t}]_2 \leftarrow [\mathbf{p}]_2 \,\&\, \mathsf{0xFFFFFFFFFFFFF800}$
   (b) $[\mathsf{ok}]_2 \leftarrow [\mathsf{ok}]_2 \,\&\, ([\mathbf{t}]_2 == 0)$.
11. $[\mathbf{s}]_2 \leftarrow \mathsf{convert}([s]_p)$.
12. $[\mathbf{x}]_2 \leftarrow ([\mathbf{p}]_2 \ll 52) \oplus [\mathbf{v}]_2$.                  //free
13. $[\mathbf{x}^{(63)}]_2 \leftarrow [\mathbf{s}]_2$.
14. $[\mathbf{z}]_2 \leftarrow \mathsf{convert}([z]_p)$.
15. $\mathsf{NaN} \leftarrow \mathsf{0x7FF0000000000001}$
16. $[\mathbf{x}]_2 \leftarrow (1 - [\mathbf{z}]_2) \,\&\, [\mathbf{x}]_2$.
17. $[\mathbf{x}]_2 \leftarrow ([\mathsf{ok}]_2 \,\&\, [\mathbf{x}]_2) \oplus ((1 - [\mathsf{ok}]_2) \,\&\, \mathsf{NaN})$.
18. Return $[\mathbf{x}]_2$.
```

Figure 6. sfloat to IEEE

转化的时候需要考虑浮点数各个部分的关系和异常处理、错误处理。

4. Evaluation

本文分别做了IEEE-754在布尔电路下的时间开销，sfix在LSSS-Based MPC下的时间开销和sfloat在LSSS-Based MPC下的时间开销。对于计算精度来说，sfix-sfloat-IEEE-754逐步上升，而效率却是逐步下降。因此，在实际应用中需要平衡二者之间的关系。

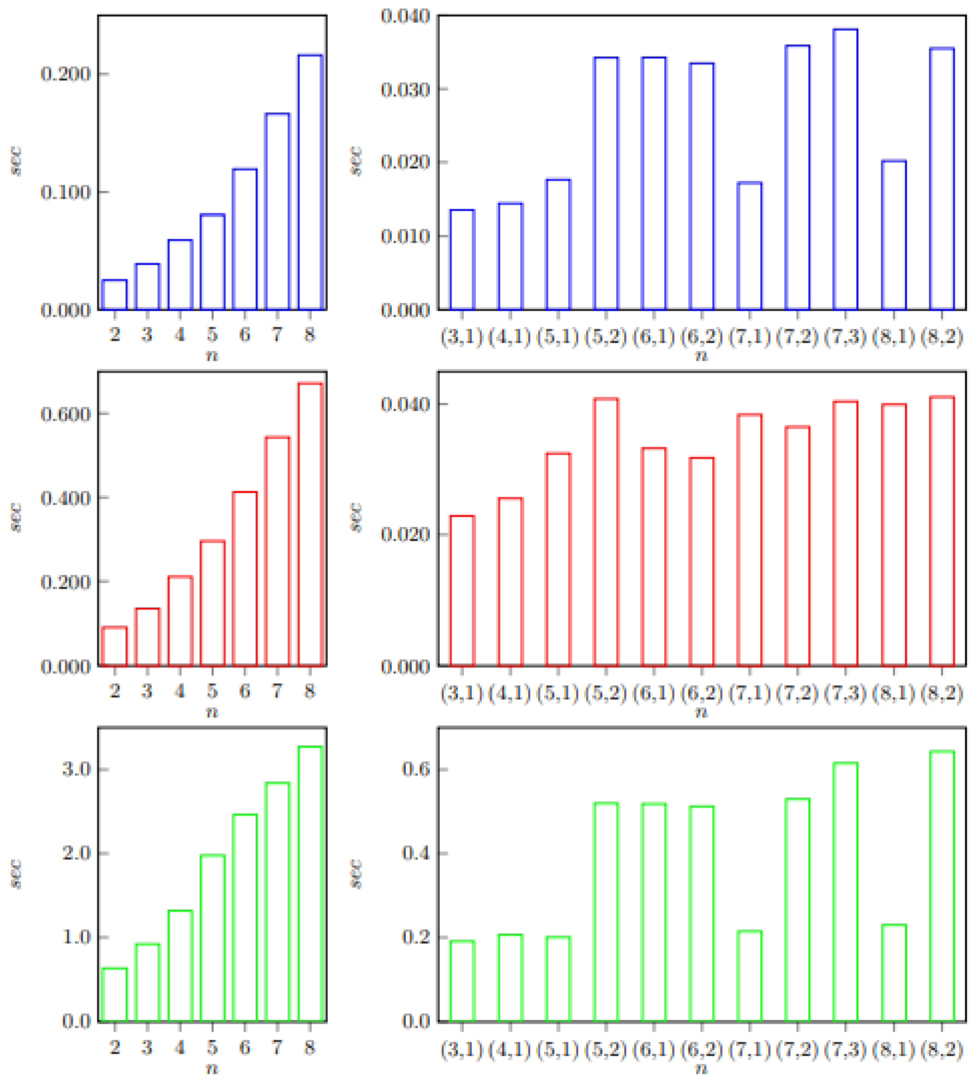


Fig. 7. Execution time in sec to execute the binary circuit based IEEE-754 operations addition (blue), multiplication (red), division (green) for Full Threshold access structure with n players (left column) and Shamir access structures (n, t) (right column)

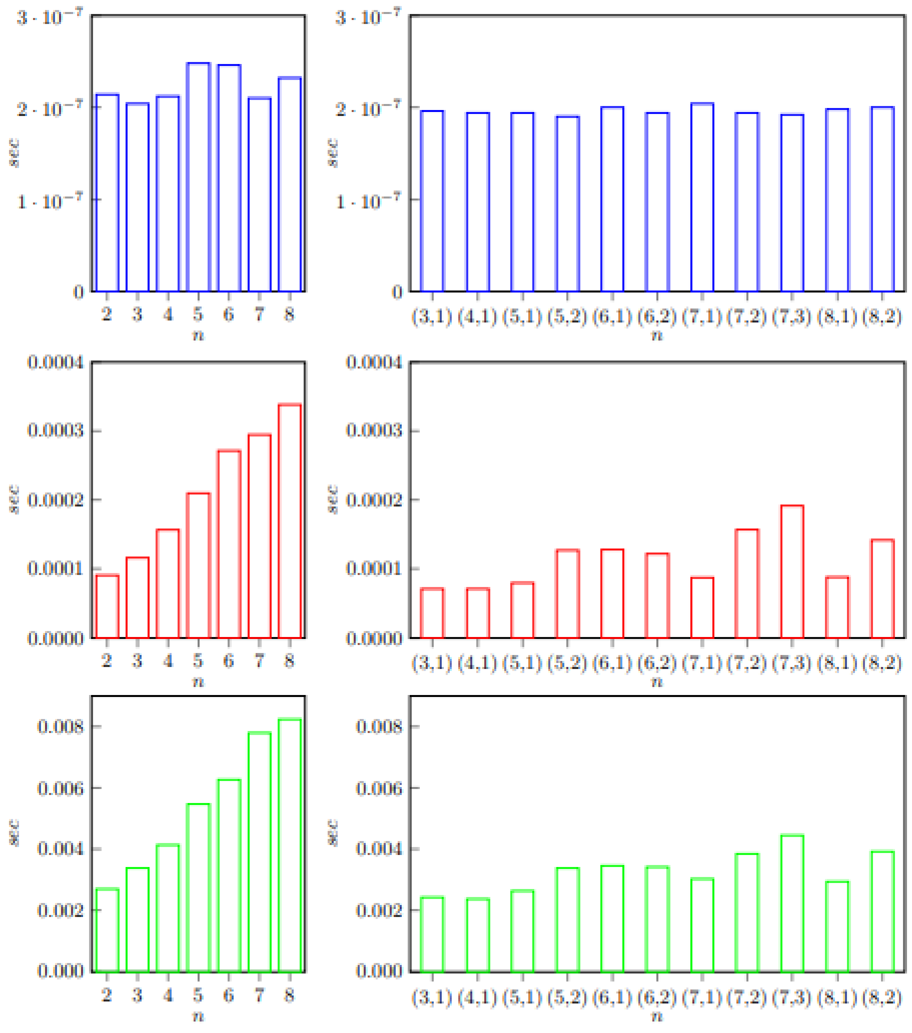


Fig. 8. Execution time in sec to execute the LSSS-based sfix operations addition (blue), multiplication (red), division (green) for Full Threshold access structure with n players (left column) and Shamir access structures (n, t) (right column)

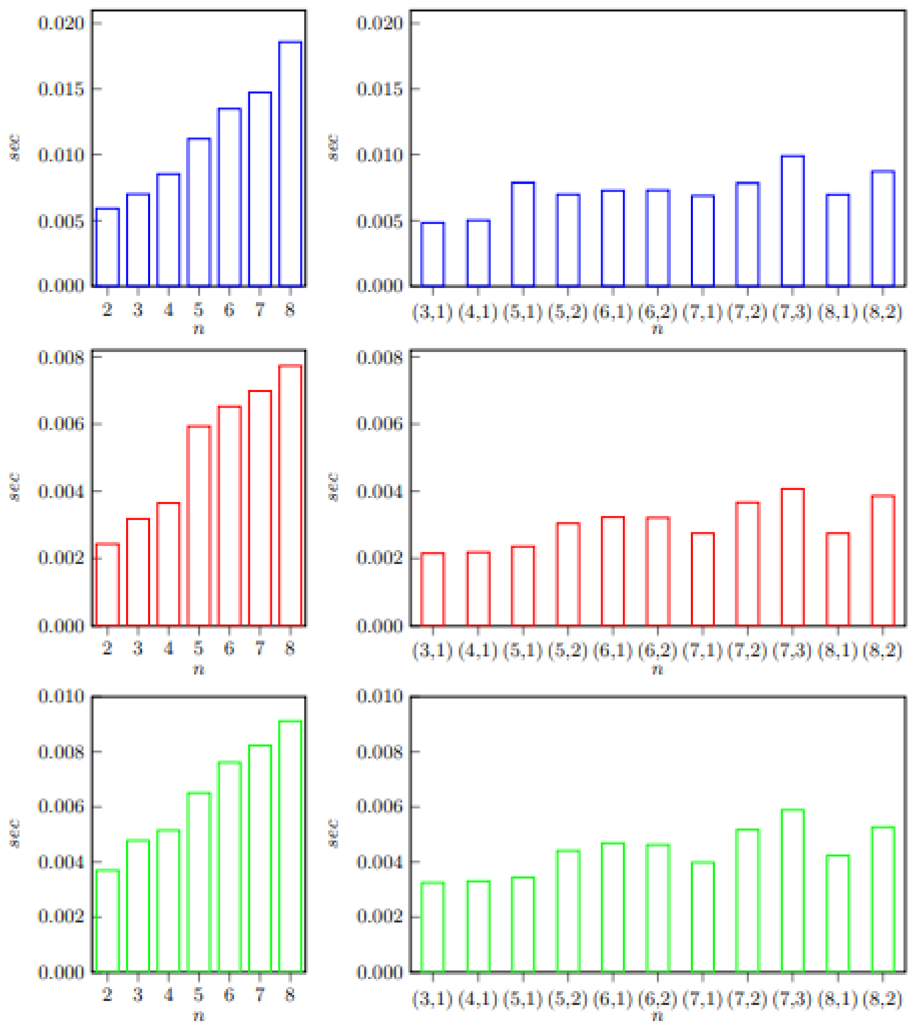


Fig. 9. Execution time in sec to execute the LSSS-based sfloat operations addition (blue), multiplication (red), division (green) for Full Threshold access structure with n players (left column) and Shamir access structures (n, t) (right column)