Project Report of Secure Multiparty Computing In The Course of Computer Network and Net-work Security.

安全多方计算（SMPC）项目报告

陈增辉、陈煜堂

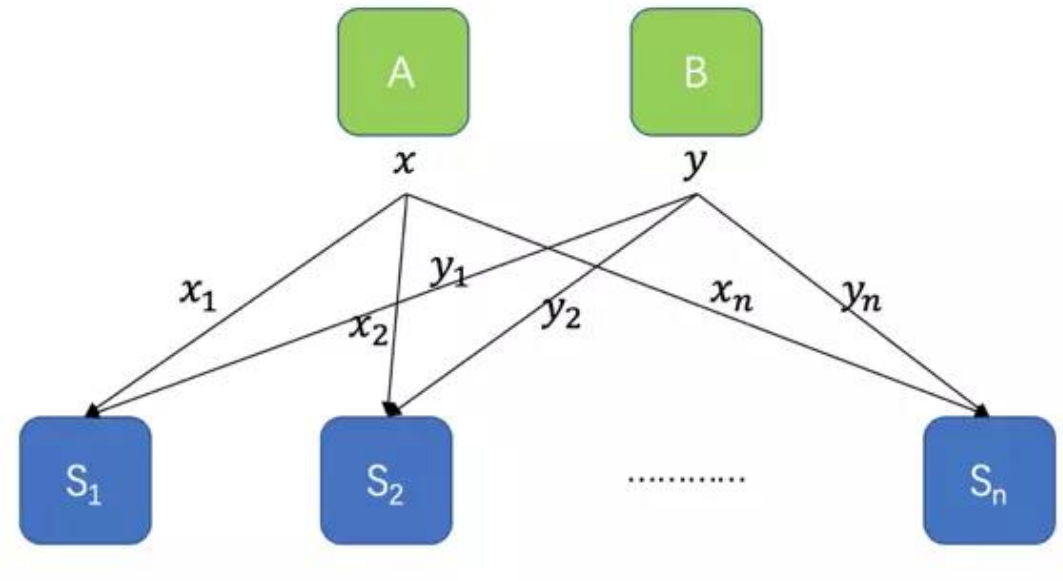
Outline

- 简要介绍：密钥分享和混淆电路
- 初步实践：MP-SPDZ 的安装和冒泡排序的应用
- SMPC 图算法：广度优先搜索（BFS）的实现和评估

Outline

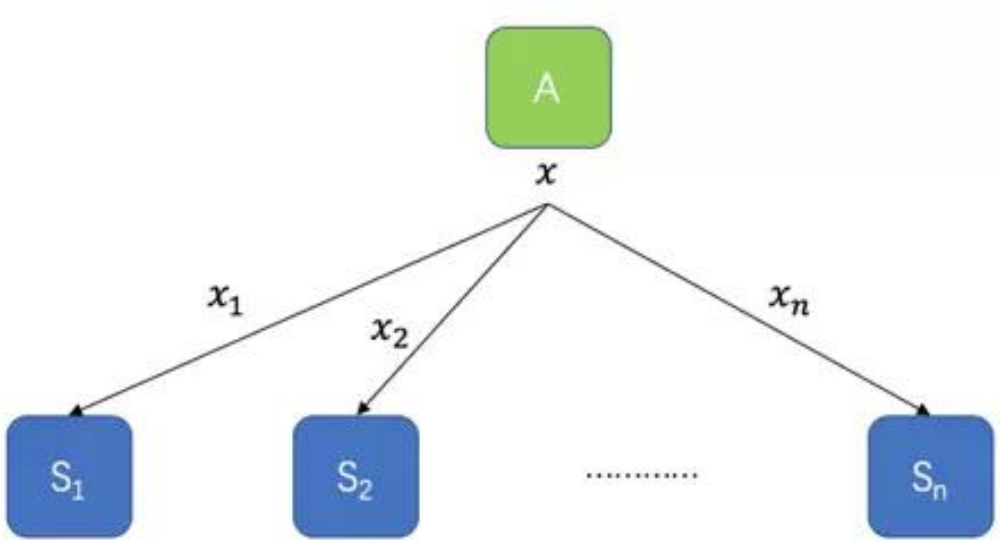
- **简要介绍：密钥分享和混淆电路**
  - **密钥分享**
    - 拟解决的问题
    - 密码学原理
    - 具体实施方法
- 初步实践：MP-SPDZ 的安装和冒泡排序的应用
- SMPC 图算法：广度优先搜索（BFS）的实现和评估

简要介绍：密钥分享拟解决的问题



- Secret：A 与 B （两者可以为同一方的不同数据）需要将数据的运算委托给参与方而不让参与方获取完整的数据信息。
- Share：每个参与方拿到的都是原始数据的一部分，一个或少数几个参与方无法还原出原始数据，只有大家把各自的数据凑在一起时才能还原真实数据。计算结束后的结果仍以secret sharing的方式分散在各参与方那里，并在最终需要得到结果的时候将某些数据合起来。

简要介绍：密钥分享密码学原理



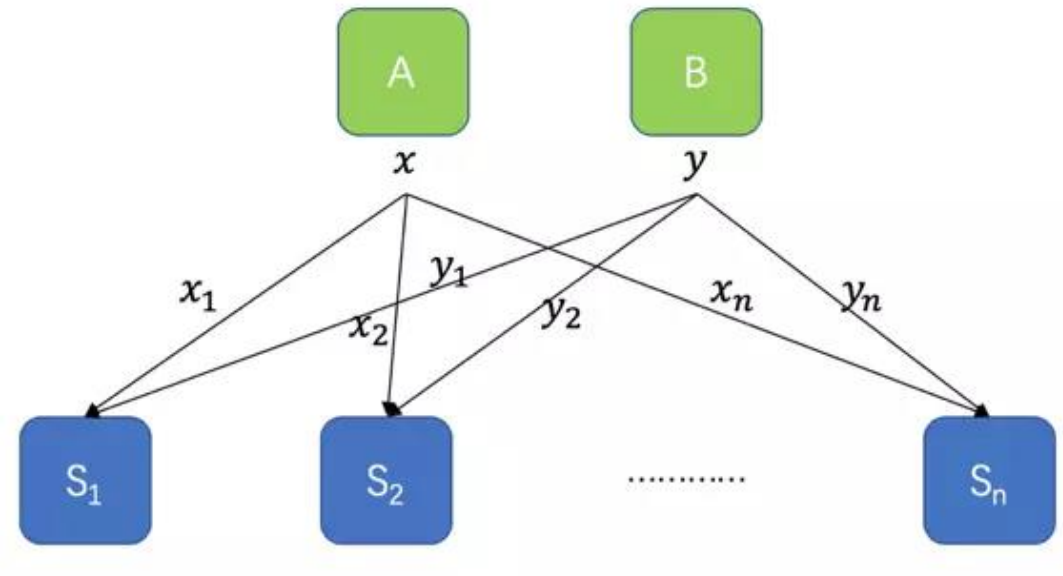
那密钥分享具体是怎么运作的呢？我们先从一个最简单的方法讲起。假设A这个人有一个秘密数字 x ，他想将其分发到$S_1, S_2, \ldots, S_n$那里。那么A首先要做的便是生成 $n-1$ 个随机数 $r_1, r_2, \ldots, r_n$ ，然后计算第 n 个数 $r_n = x - \sum_{i=1}^{n-1} r_i$ ，最后A令 $x_1 = r_1, x_2 = r_2, \ldots, x_n = r_n$ ，并将它们发给$S_1, S_2, \ldots, S_n$。上面这种简单的方法具有如下几条性质：

1. 各个数字 $x_1, x_2, \ldots, x_n$ 都是随机分布的，单独一个或若干个并不泄露任何信息；

2. 当所有$x_1, x_2, \ldots, x_n$合在一起时，可以还原 x ，因为 $x = \sum_{i=1}^{n} x_i$ ；

3. 这种方案具有加法同态的性质，也就是说，各参与方可以在不交换任何数据的情况下直接计算对秘密数据求和。什么意思呢？假设还有另一个人B，他也有一个秘密数字 y ,并且和A一起将数据分发给了$S_1, S_2, \ldots, S_n$，

简要介绍：密钥分享具体实施
阈值密钥分享
(threshold secret sharing)



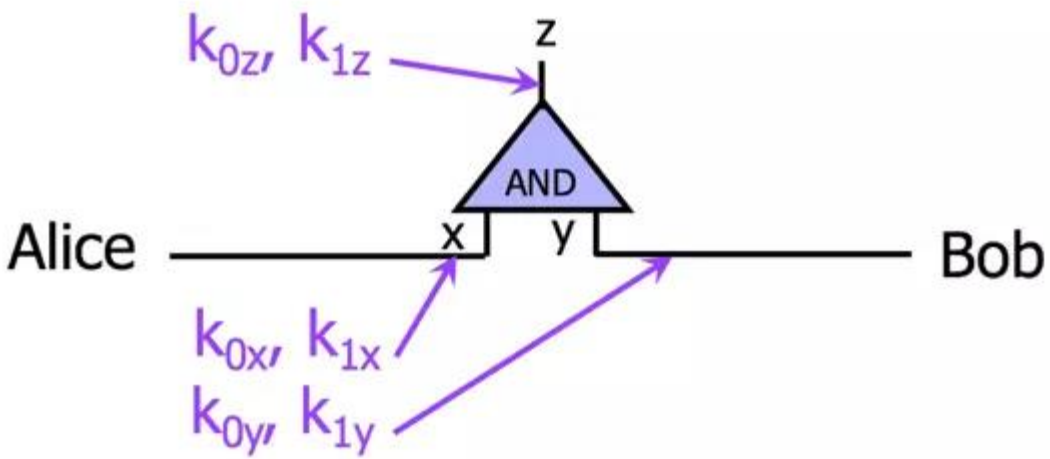
更具体地说，我们可以定义一种名为 (t,n) 阈值密钥分享的方案，此类方案允许任意 t

个参与方将秘密数据解开，但任何不多于 $t-1$ 个参与方的小团体都无法将秘密数据解开。前面提到的那种简单方案其实是 $t=n$ 时的特殊情况。Shamir大神在1979年就提出了阈值密钥分享方案，且该方案支持任意的 t 。该方案运作方式如下：假设A想要使用 (t,n) 阈值密钥分享技术将某秘密数字 s 分享给$S_1,S_2,\ldots,S_n$，那么他首先生成一个 $t-1$ 次多项式多项式 $f(x)=a_0+a_1x+a_2x^2+\cdots+a_{t-1}x^{t-1}$ ，其中 a_0 就等于要分享的秘密数字 s ，而 $a_1,a_2,\ldots,a_{t-1}$ ，则是A生成的随机数。随后A只需将 $s_1=f(1),s_2=f(2),\ldots,s_n=f(n)$ 分别发给$S_1,S_2,\ldots,S_n$即可。到了这一步，稍微有点线性代数基础的同学应该很容易看出来， $f(1),f(2),\ldots,f(n)$ 中任意 t 个凑在一起都可以解出，而任意 $t-1$ 个凑在一起都无法得到 a_0 （即 s ）的确切解。通过这一点便达到了 (t,n) 阈值的要求。Shamir密钥分享方法也是满足加法同态的（因为多项式本身满足这一性质），有兴趣的同学可以自己验证一下。

Outline

- **简要介绍：密钥分享和混淆电路**
  - 密钥分享
  - **混淆电路**
    - 拟解决的问题
      - 一群人各自拥有其隐私数据，他们想把这些数据合起来算点什么，但又不想把数据交给别人
    - 密码学原理
    - 具体实施方法
- 初步实践：MP-SPDZ 的安装和冒泡排序的应用
- SMPC 图算法：广度优先搜索（BFS）的实现和评估

简要介绍：混淆电路密码学原理和具体实施



Encrypted truth table:

$E_{k0x}(E_{k0y}(k_{0z}))$
$E_{k0x}(E_{k1y}(k_{0z}))$
$E_{k1x}(E_{k0y}(k_{0z}))$
$E_{k1x}(E_{k1y}(k_{1z}))$

Garbled truth table:

$E_{k1x}(E_{k0y}(k_{0z}))$
$E_{k0x}(E_{k1y}(k_{0z}))$
$E_{k1x}(E_{k1y}(k_{1z}))$
$E_{k0x}(E_{k0y}(k_{0z}))$

Outline

- 简要介绍：密钥分享和混淆电路
- **初步实践：MP-SPDZ 的安装和冒泡排序的应用**
  - 安装
  - 冒泡排序
  - 遇到的问题
- SMPC 图算法：广度优先搜索（BFS）的实现和评估

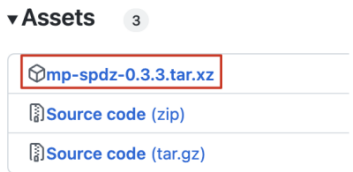
初步实践：MP-SPDZ 的安装

MP-SPDZ

It is a software to benchmark various secure multi-party computation (MPC) protocols in a variety of security models such as honest and dishonest majority, semi-honest/passive and malicious/active corruption.

Download and configure on ubuntu:

Download and unpack the distribution (https://github.com/data61/MP-SPDZ/releases) mp-spdz-0.3.3.tar.xz:



then execute the following from the top folder:

Scripts/tldr.sh

MP-SPDZ documentation:

https://github.com/data61/MP-SPDZ#tldr-binary-distribution-on-linux-or-source-distribution-on-macos

https://mp-spdz.readthedocs.io/en/latest/Compiler.html#

MP-SPDZ Publication:

https://ia.cr/2020/521

初步实践：MP-SPDZ 冒泡排序

```
28  d = maopao(a)
29  print_str("%s", d.reveal())

5   def maopao(a):
6       print_ln("hello: %s", a.reveal())
7       n = len(a)
8       @for_range(n)
9       def _(i):
10          @for_range(n)
11          def _(j):
12              @if_((a[i] > a[j]).reveal())
13              def _():
14                  # print_ln("!!!")
15                  tmp = a[i]
16                  a[i] = a[j]
17                  a[j] = tmp
18      return a
```

```
24  @for_range_opt(n)
25  def _(i):
26      a[i] = sfix.get_input_from(i)

101 # use @for_range_opt for balanced optimization
102 # but use Python loops if compile-time numbers are need (e.g., for players)
103
104 @for_range_opt(3)
105 def _(i):
106     for j in range(2):
107         data[i][j] = sfix.get_input_from(j)
```

初步实践：MP-SPDZ 实践中遇到的问题

- 问题：默认只在第0个 party 的命令行中输出 print_ln() 的内容，按官方文档尝试失败，命令如下：
- $./semi-party.x -p 0 -N 2 -IF Player-Data/Input -I merge_and_sort
- 解决：将所有输出全部存放到文件中，命令如下：
- $./semi-party.x -p 0 -N 2 -IF 'Player-Data/Input' -OF Player-Data/Output merge_and_sort

Public Outputs

By default, `print_ln()` and related functions only output to the terminal on party 0. This allows to run several parties in one terminal without spoiling the output. You can use interactive mode with option `-I` in order to output on all parties. Note that this also to reading inputs from the command line unless you specify `-IF` as well. You can also specify a file prefix with `-OF`, so that outputs are written to `<prefix>-P<player>-<thread>`.

Outline

- 简要介绍：密钥分享和混淆电路
- 初步实践：MP-SPDZ 的安装和冒泡排序的应用
- **SMPC 图算法：广度优先搜索（BFS）的实现和评估**
  - **BFS 原理、邻接矩阵和其他数据结构设计**
  - 私密输出、结果校验
  - 安全性和性能

SMPC 图算法：BFS 原理和邻接矩阵 M 、数据结构 C 的设计

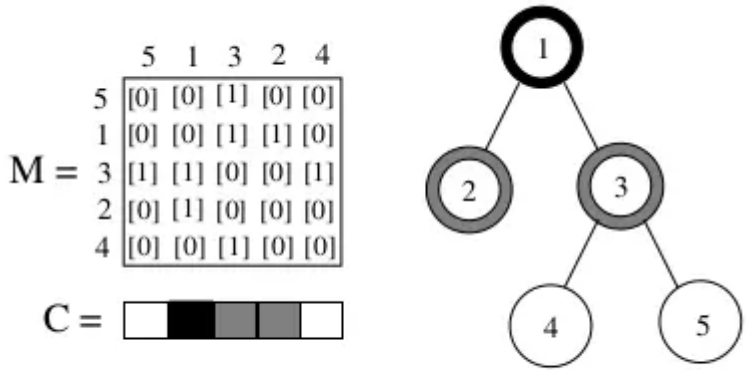


Figure 1: Illustration of the BFS algorithm.

```
39  M = Array(n*n, sfix)
40  V = Array(n*3, sfix)
41  @for_range_opt(n)
42  def _(i):
43      @for_range_opt(n)
44      def _(j):
45          M[i*n+j] = sfix.get_input_from(i)
46
47  @for_range(n)
48  def _(i):
49          V[i*3] = 0
50          V[i*3+1] = 2048
51          V[i*3+2] = 2048
52  V[0] = 1
53  V[1] = 0
54
55  @for_range(n)
56  def _(i):
57          @if_(M[i].reveal()==1)
58          def _():
59              V[i*3+1] = 1
```

Outline

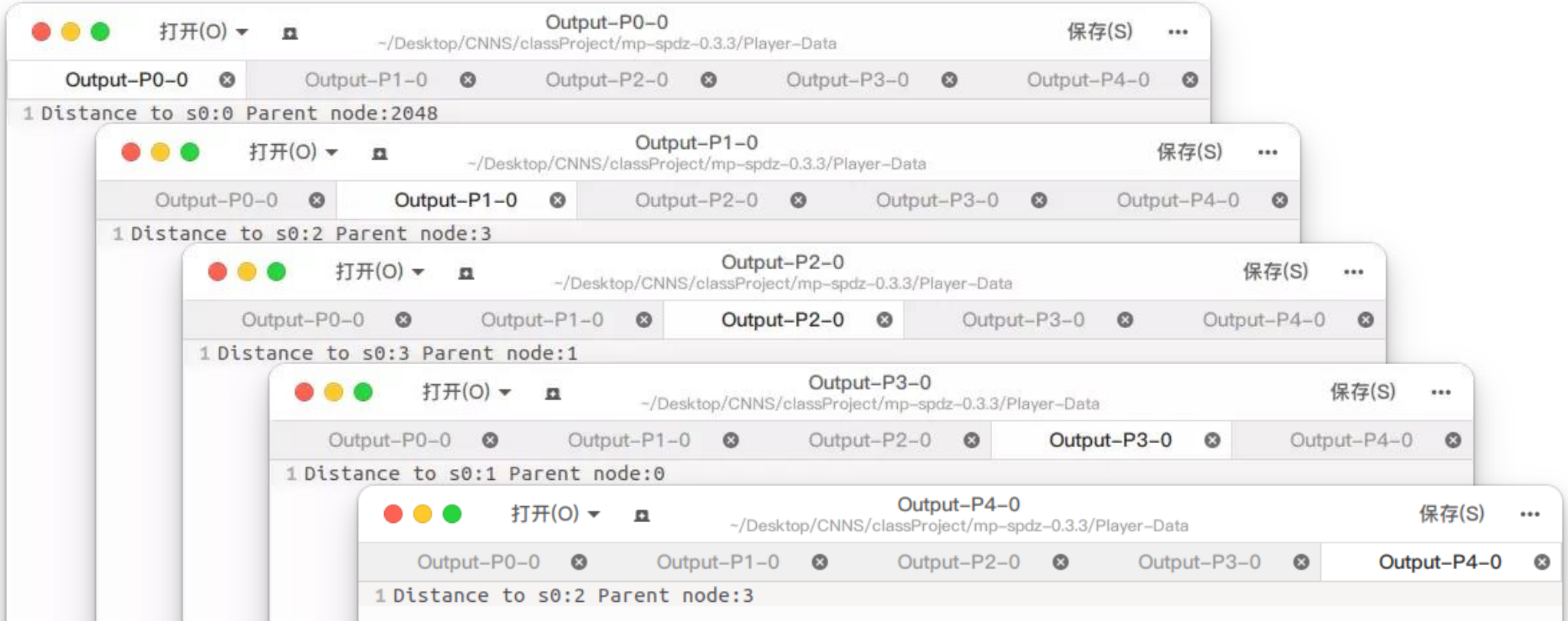
- 简要介绍：密钥分享和混淆电路
- 初步实践：MP-SPDZ 的安装和冒泡排序的应用
- **SMPC 图算法：广度优先搜索（BFS）的实现和评估**
  - BFS 原理、邻接矩阵和其他数据结构设计
  - **私密输出、结果校验**
  - 安全性和性能

SMPC 图算法：私密输出（**Private Outputs**）

Private Outputs to Computing Parties

Some types provide a function to reveal a value only to a specific party (e.g., `Compiler.types.sint.reveal_to()`). It can be used conjunction with `print_ln_to()` in order to output it.

```
68  @for_range(n)
69  def _(i):
70      print_ln_to(i,"Distance to s0:%s Parent node:%s",V[i*3+1].reveal(),V[i*3+2].reveal())
```



Outline

- 简要介绍：密钥分享和混淆电路
- 初步实践：MP-SPDZ 的安装和冒泡排序的应用
- **SMPC 图算法：广度优先搜索（BFS）的实现和评估**
  - BFS 原理、邻接矩阵和其他数据结构设计
  - 私密输出、结果校验
  - **安全性和性能**

SMPC 图算法：性能测试与评估

算法共有四个步骤：

1. 从 Input-P…… 读入信息生成邻接矩阵，此处分内循环和外循环，循环 n*n 次，如下：

```
@for_range_opt(n)
def _(i):
  @for_range_opt(n)
    def _(j):
      M[i*n+j] = sfix.get_input_from(i)
```

2. 初始化数组 C，用于存放每个节点与 s0 的距离，颜色和父亲节点，总共 3*n+2+n 步，如下：

```
@for_range(n)
def _(i):
  V[i*3] = 0
  V[i*3+1] = 2048
  V[i*3+2] = 2048
V[0] = 1
V[1] = 0
@for_range(n)
def _(i):
  @if_(M[i].reveal()==1)
  def _():
    V[i*3+1] = 1
```

3. 广度优先搜索，共 n*(n+n) 步，分外循环、找下一个灰色节点、为节点寻找孩子3部分，代码较长，暂不给出。

4. 输出至指定 party，共 n 步，代码如下：

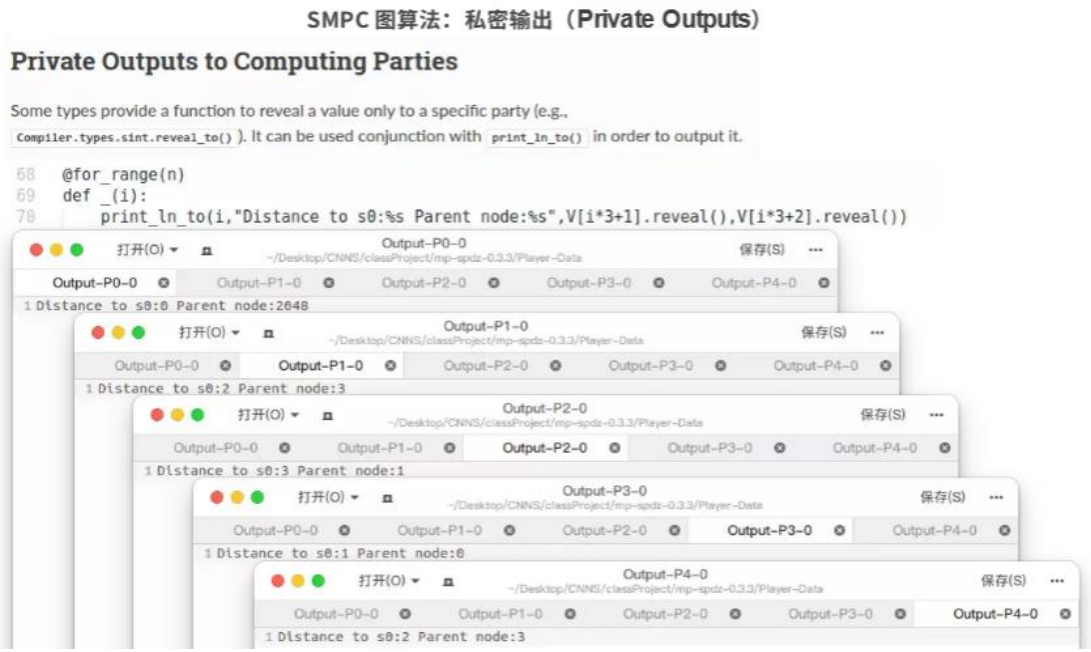
```
@for_range(n)
def _(i):
  print_ln_to(i,"Distance to s0:%s Parent node:%s",V[i*3+1].reveal
(),V[i*3+2].reveal())
```

综上，时间复杂度为 O(n*n + 3*n+2+n + n*(n+n) + n) = O(n*n)

```
The following benchmarks are including preprocessing (offline phase).
Time = 0.27738 seconds
Data sent = 11.1609 MB in ~1560 rounds (party 0)
Global data sent = 22.3217 MB (all parties)
```

```
yunxi@yunxi-virtual-machine:~/Desktop/CNNS/classProject/mp-spdz-0.3.3$ ./semi-pa
rty.x -p 0 -N 5 -IF 'Player-Data/Input' -OF Player-Data/Output bfs
Using security parameter 40
The following benchmarks are including preprocessing (offline phase).
Time = 0.0337145 seconds
Data sent = 0.125796 MB in ~2472 rounds (party 0)
Global data sent = 0.605364 MB (all parties)
```

SMPC 图算法：安全性测试与评估



- 程序全部使用存储 sfix 和 sint 加密类型的数组，对数组数据的读取只能使用 reveal（）方法，数据在运算过程中是保密的，难以被拦截。
- 程序使用 print_ln_to（）进行私密输出，每个节点只接收到自己的数据。

Thank you!