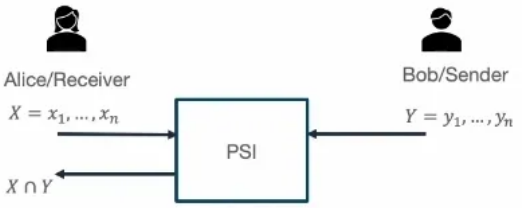
第4讲 隐语PSI 功能及使用介绍

SPU实现的PSI介绍

PSI定义和种类

- 见第1讲

PSI定义：

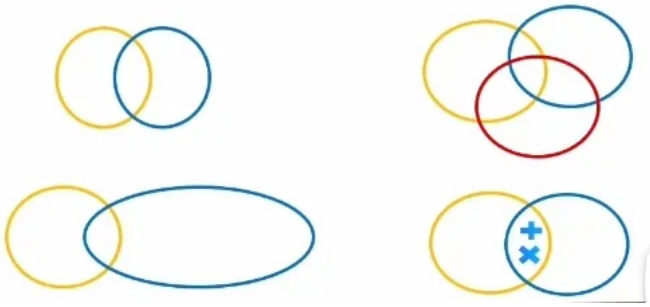


安全求交集：Private Set Intersection（PSI）

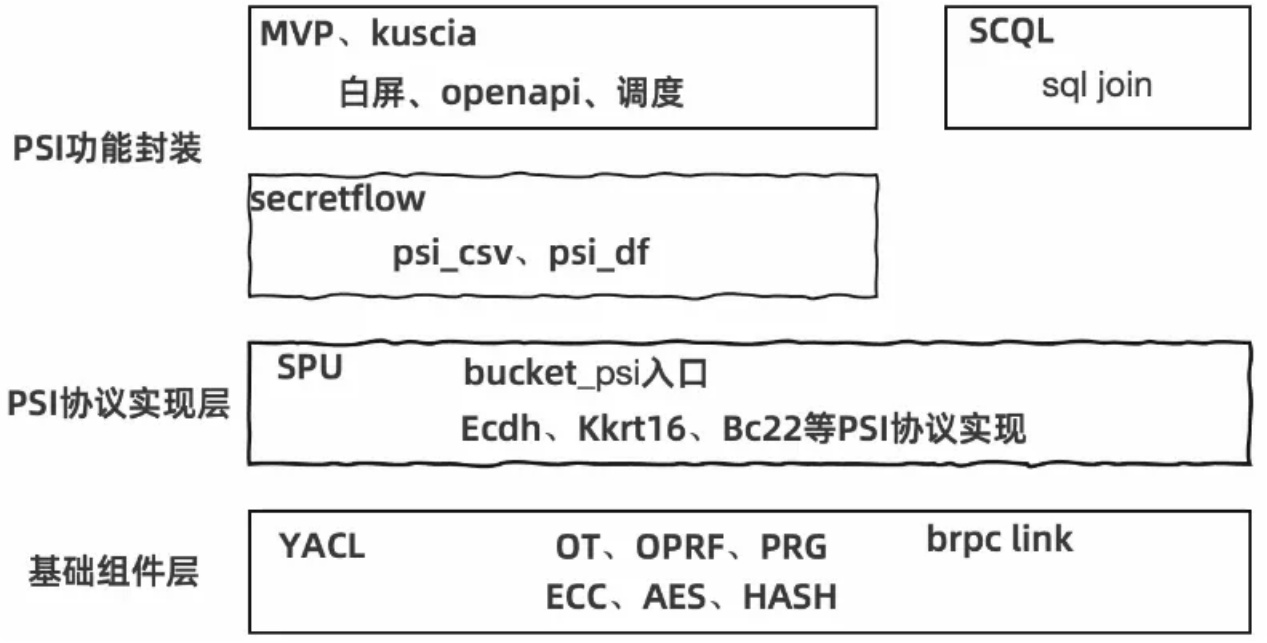
- 一种特殊的安全多方计算（MPC）协议
- Alice持有集合 X，Bob持有集合Y，
- Alice和Bob通过执行PSI协议，得到交集结果$X \cap Y$
- 除交集外不会泄漏交集外的其它信息

PSI分类：

- 2-Party/Multi-Party PSI
- Balanced/Unbalanced PSI
- Semi-honest/Malicious PSI
- PSI with computation：
  - PSI-CA（Cardinality）
  - PSI-Payload Analytics
  - Circuit PSI



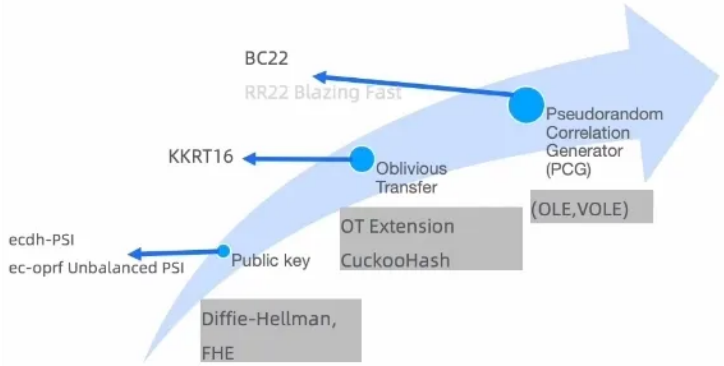
隐语PSI功能分层



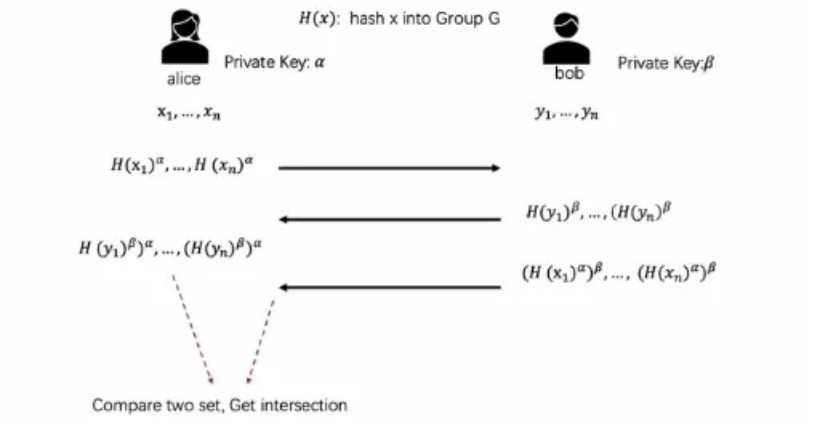
SPU实现的PSI介绍

SPU实现的PSI种类

- 半诚实模型
  - 两方
    - ecdh、kkrt16、bc22(pcg-psi)
    - ec-oprf PSI (Unbalanced PSI)
    - dp-psi
  - 多方
    - ecdh-3-party(可扩展到多方)
- 恶意模型
  - mini-PSI(适合小数据集)



ecdh-PSI介绍



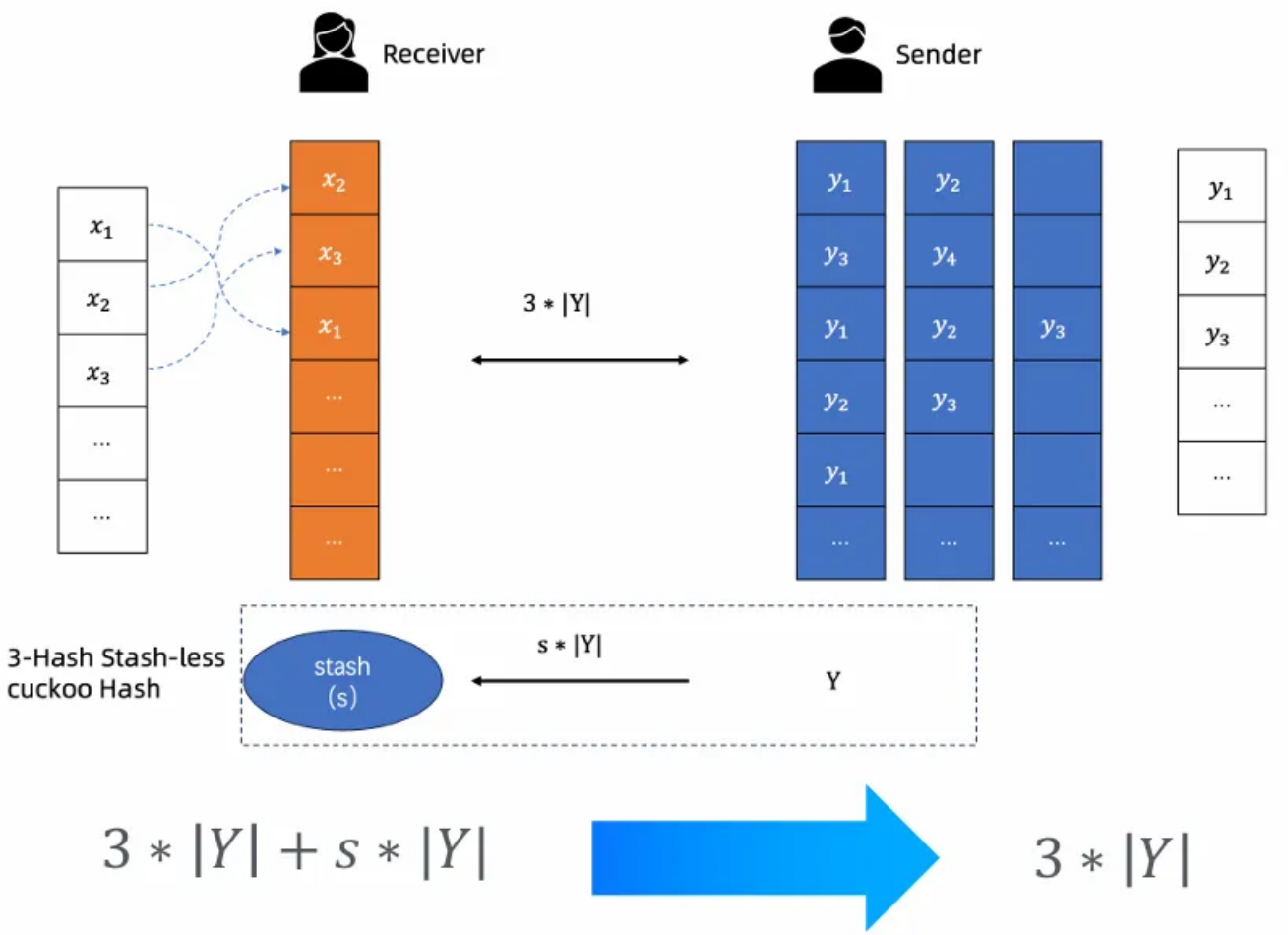
ecdh-psi协议特点：

- Simple to understand and to explain (to your managers)
- Simple to implement
- Best Communication cost, high Computation cost
- Easy to extend
  - Can be modified to compute intersection size（PSI-CA）
  - Google private join and compute；Facebook Private-ID

KKRT16 PSI介绍

- 论文主要贡献
  - Propose a novel extension to IKNP and KK OT protocol

  achieve an 1-out-of-n OT ， for arbitrarily large n
  - Batch，Related-key OPRF （BaRK-OPRF）
- 优点：运行时间快
  - 3.8s 2^20（百万）
  - 1m 2^24（1.6千万）
  - 最新PSI论文中比较的基准
- 缺点：
  - 内存占用大
  - 通信量大



BC22 PCG PSI介绍

- 基于sVOLE的BaRK-OPRF
- Generalized Cuckoo Hash
- Permutation-Based Hashing

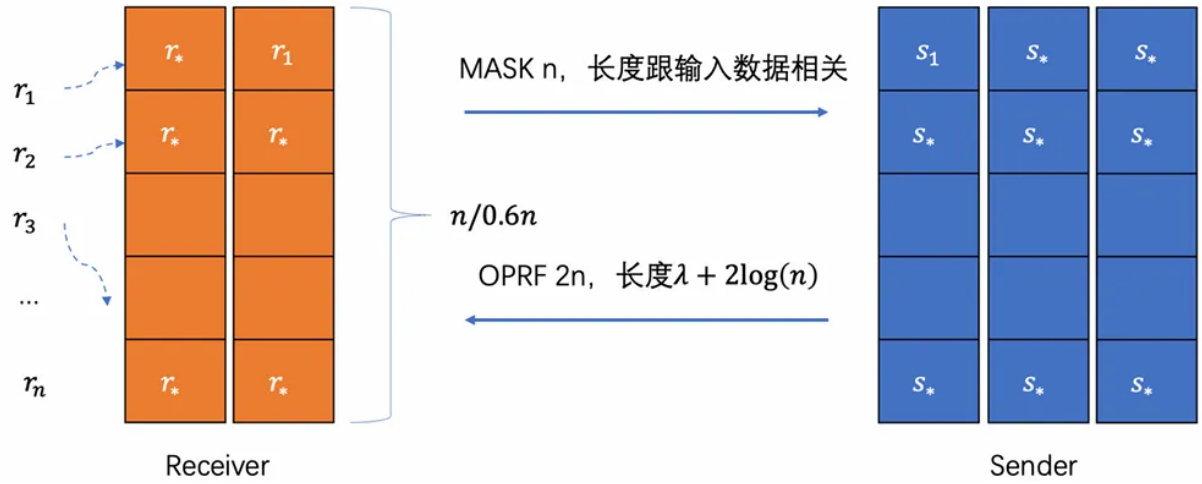
Generalized Cuckoo Hash(GCH)

参数: (d,k) (d,k)=(1,3), table_size=1.3n
(d,k)=(3,2), table_size=0.65n
(d,k)=(2,2), table_size=n

2个Hash H_i: $R \cup S \to [0, n-1]$

Hash个数减少，Sender端计算和发送的OPRF减少

3*len(oprf)*n→2 *len(oprf)* n



$GCH(3,2)$

N:table size

$\vec{w} = \Delta \cdot \vec{u} + \vec{v}$

$n = 2 \cdot N$

PCG Gen and Expand

插入CuckooTable

插入SimpleTable

BaRK-OPRF

Compute and Mask Polynomial Coefficients

Masked Bin Polynomial Coefficients

OPRF values

Compute and Shuffle OPRF values

Compare OPRF Get Intersection

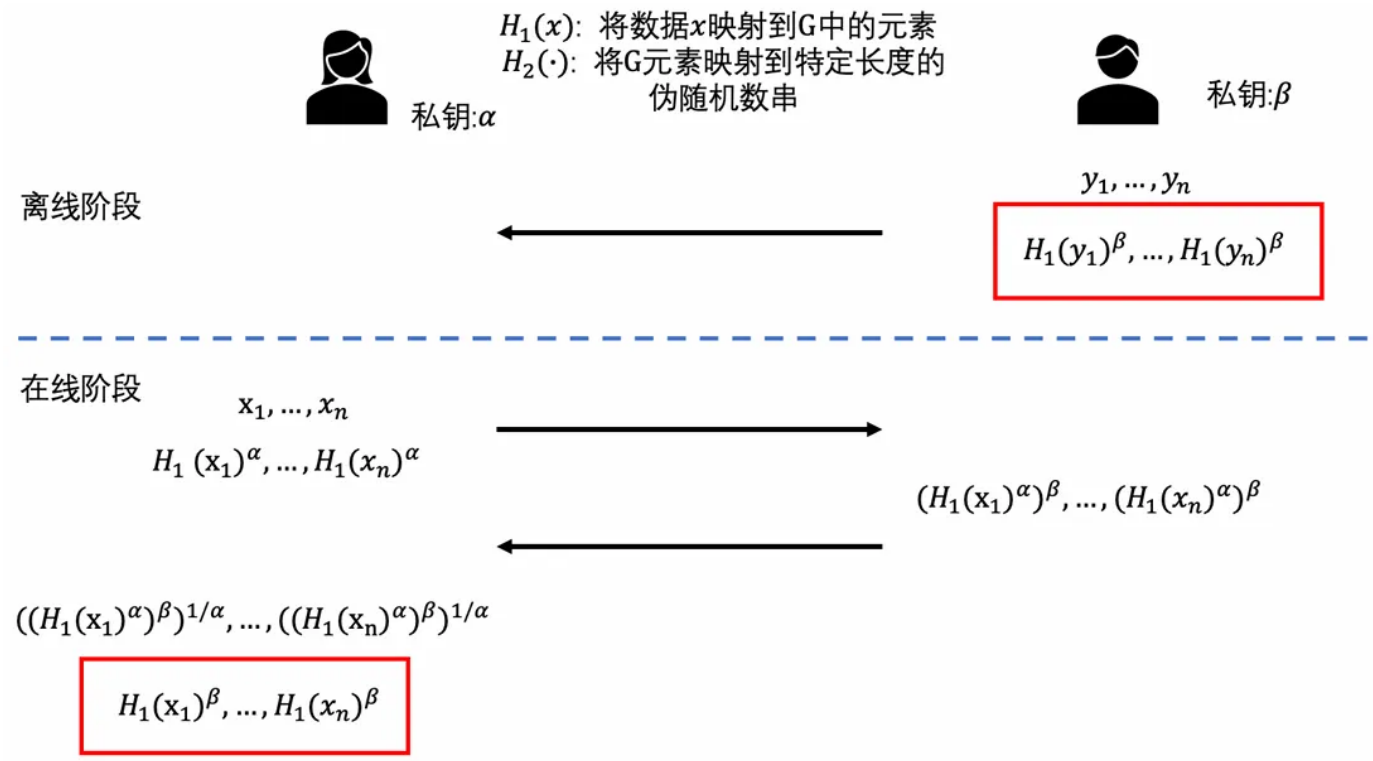
BC22 PCG PSI协议流程

- 实现选用的参数
  - Generalized CuckooHash（3,2)
  - VOLE 使用emp-zk中的WYKW21 Wolverine方案
- 与论文中的差异
  - 左端插值多项式，论文中给出了不做填充的方案，目前实现都做填充

$n < 2^{20}$，填充，$n \geq 2^{20}$，不做填充

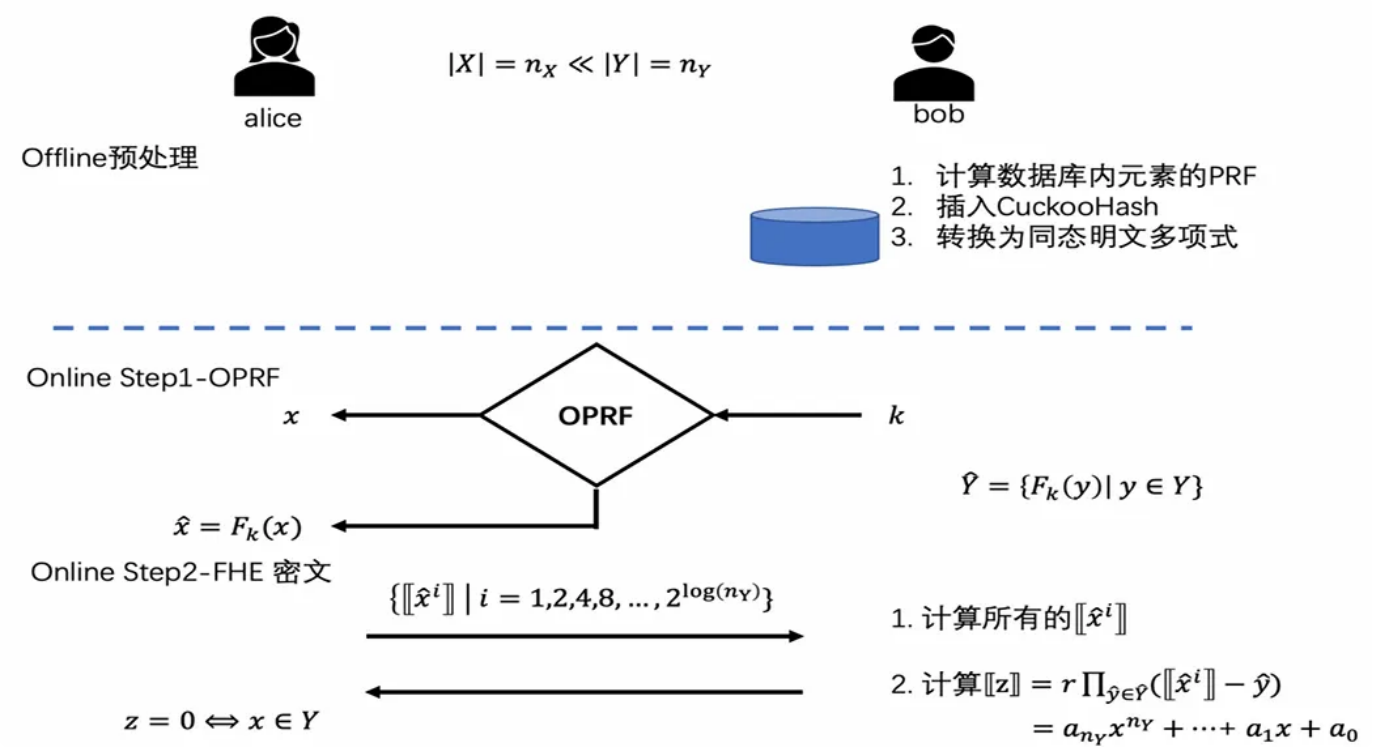
  - 优点：降低计算量
  - 缺点：需要额外的F_2 SubField VOLE，判别最高次x^3系数是否为0

Unbalanced PSI: ec-oprf based



计算复杂度比较		
	ecdh-psi	ec-oprf psi
alice	$n_X + n_Y$	$2 * n_X$
bob	$n_X + n_Y$	$n_X + n_Y$
总计	$2 * n_X + 2 * n_Y$	$3 * n_X + n_Y$

Unbalanced PSI: SHE-based



APSI与ec-oprf PSI比较:

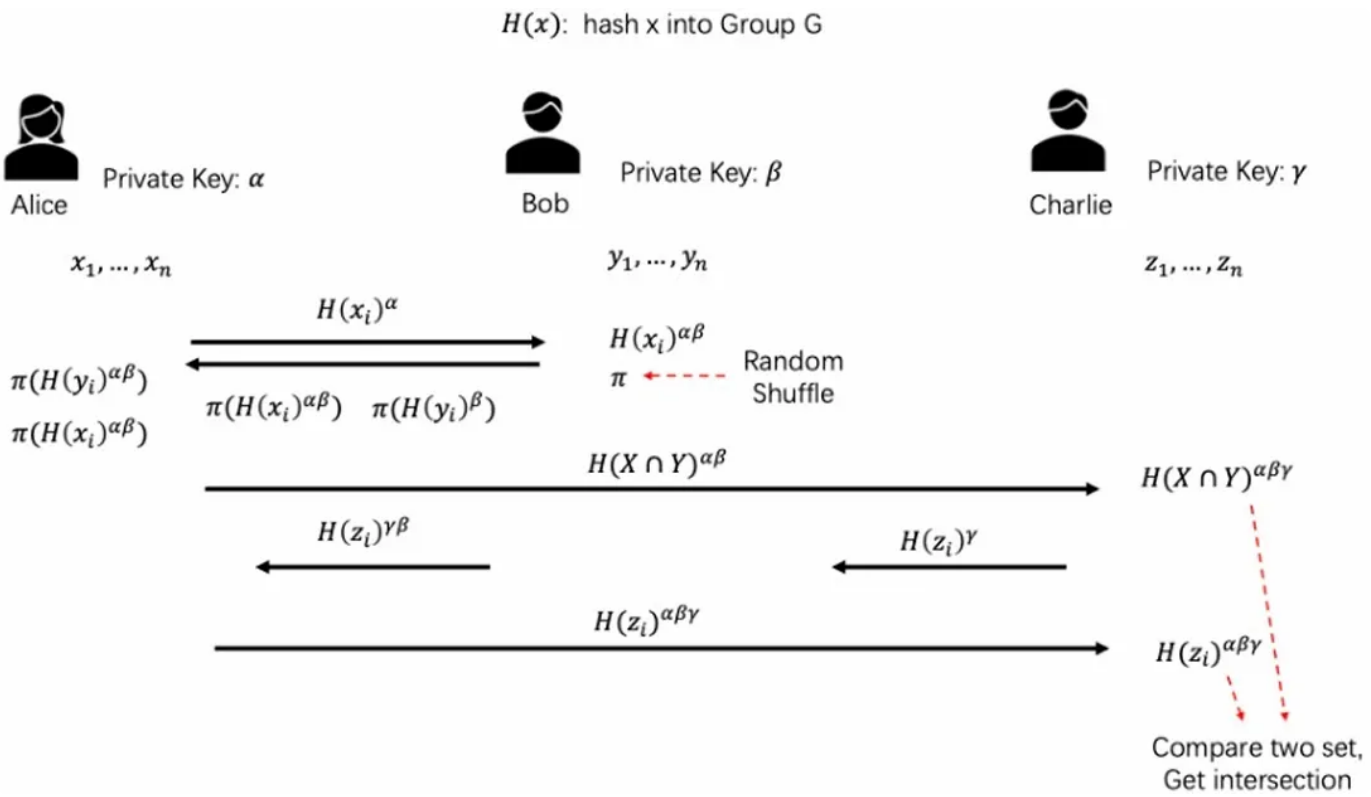
- 优点:

不需要将大数据方的数据传输到小数据方

- 缺点

计算量大，运行时间长

基于ecdh的三方PSI协议



- ◆ **Alice和Bob先进行交互，得到shuffle后的两方交集**
- ◆ **Alice将shuffle后两方交集，发给Charlie**
- ◆ **Charlie加密后的数据依次给Bob和Alice加密**
- ◆ **Charlie比较密态数据，得到交集**

➢ 优点

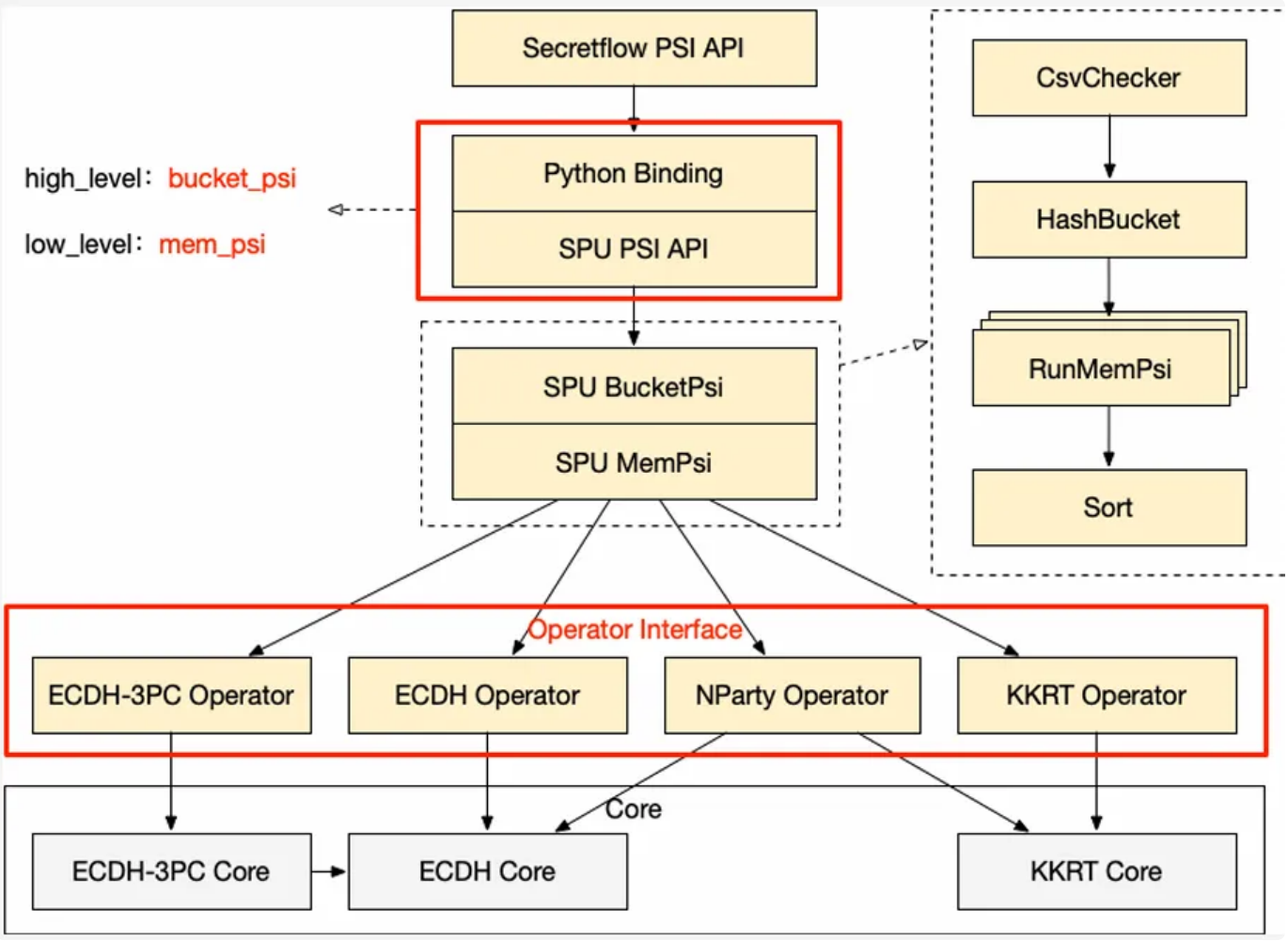
- ◆ 基于ecdh-psi，协议简单易于实现

➢ 缺点

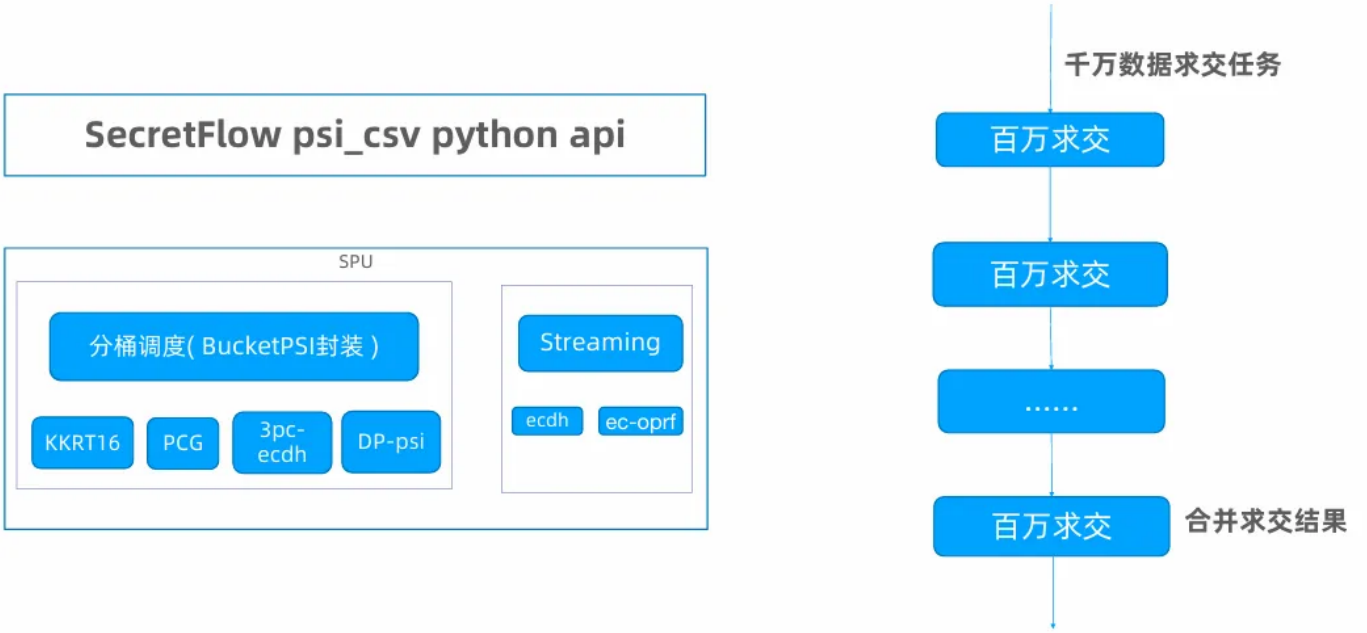
- ◆ 泄漏Alice和Bob两方交集数量

SPU PSI调度架构

- 统一入口
  - 入口函数：bucket_psi, mem_psi
- 支持分桶求交
  - 通过分桶支持大规模数据（10亿规模）
- 输入输出处理
  - 检查求交id列是否数据是否完整
  - 检查是否有重复项
- 输出处理
  - 支持按求交id列排序
  - 输出完整label列



- bucket_psi：高级API，通过Hash分桶支持海量数据，覆盖生产级全流程（数据查重、分桶求交、结果广播、结果排序）
- mem_psi：低级API，算法内核级的性能 + 统一易用的接口
- Operator：算法接入层，向上提供统一接口接入工程化封装；注册工厂模式，提升协议工程化效率



bucket_psi 接口

libspu/psi/psi.proto

```
message BucketPsiConfig {
  // The psi type.
  PsiType psi_type = 1;
  // Specified the receiver rank. Receiver can get psi result.
  uint32 receiver_rank = 2;
  // Whether to broadcast psi result to all parties.
  bool broadcast_result = 3;
  // The input parameters of psi.
  InputParams input_params = 4;
  // The output parameters of psi.
  OutputParams output_params = 5;
  // Optional, specified elliptic curve cryptography used in psi when needed.
  CurveType curve_type = 6;
  // Optional, specified the hash bucket size used in psi.
  uint32 bucket_size = 7;
}
```

```
  class BucketPsi {
  public:
      // ic_mode: 互联互通模式，对方可以是非隐语应用
      // Interconnection mode, the other side can be non-secretflow application
      explicit BucketPsi(BucketPsiConfig config,
             std::shared_ptr<yacl::link::Context> lctx,
              bool ic_mode = false);
```

memory_psi

libspu/psi/psi.proto

```
message MemoryPsiConfig {
  // The psi type.
  PsiType psi_type = 1;

  // Specified the receiver rank. Receiver can get psi result.
  uint32 receiver_rank = 2;

  // Whether to broadcast psi result to all parties.
  bool broadcast_result = 3;

  // Optional, specified elliptic curve cryptography used in psi when needed.
  CurveType curve_type = 4;

}
```

```
class MemoryPsi {
public:
    explicit MemoryPsi(MemoryPsiConfig config,
                       std::shared_ptr<yacl::link::Context> lctx);

    std::vector<std::string> Run(const std::vector<std::string>& inputs);
}
```

operator

```
class PsiBaseOperator {
public:
    explicit PsiBaseOperator(std::shared_ptr<yacl::link::Context> link_ctx);

    // after call OnRun, it decides whether to broadcast result or not based on
    // param `broadcast_result`
    std::vector<std::string> Run(const std::vector<std::string>& inputs,
                                 bool broadcast_result);

    virtual std::vector<std::string> OnRun(
                const std::vector<std::string>& inputs) = 0;

protected:
    std::shared_ptr<yacl::link::Context> link_ctx_;
};
```

定义并实现KKRT PSI的operator

```
class KkrtPsiOperator : public PsiBaseOperator {
public:
    std::vector<std::string> OnRun(
                  const std::vector<std::string>& inputs) override;

}
```

注册KKRT PSI的operator

```
REGISTER_OPERATOR(KKRT_PSI_2PC, CreateOperator);
```

batch_provder

批量读取数据基类

```
class IBatchProvider {
public:

    // Read at most `batch_size` items and return them. An empty returned vector
    // is treated as the end of stream.
    virtual std::vector<std::string> ReadNextBatch(size_t batch_size) = 0;

};
```

对内存数据的批量读取实现

```
class MemoryBatchProvider : public IBatchProvider{
Public:
    MemoryBatchProvider(const std::vector<std::string>& items);
    std::vector<std::string> ReadNextBatch(size_t batch_size) override;

}
```

对csv文件的批量读取实现

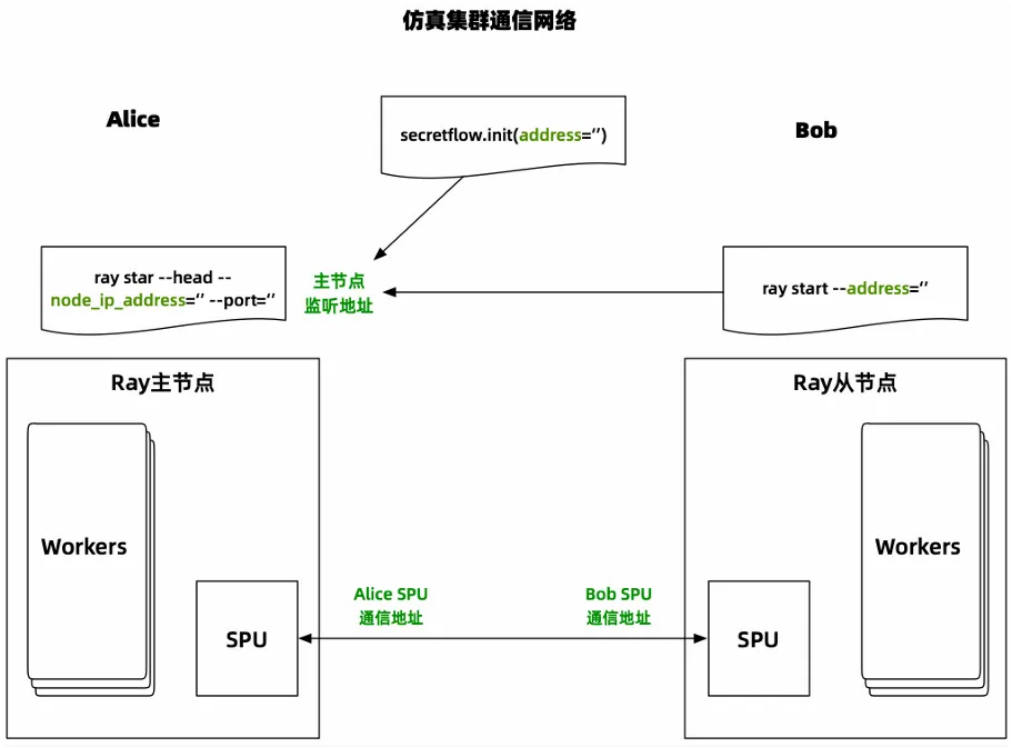
```
class CsvBatchProvider : public IBatchProvider {
public:
    explicit CsvBatchProvider(const std::string& path,
               const std::vector<std::string>& target_fields);

    std::vector<std::string> ReadNextBatch(size_t batch_size) override;
}
```

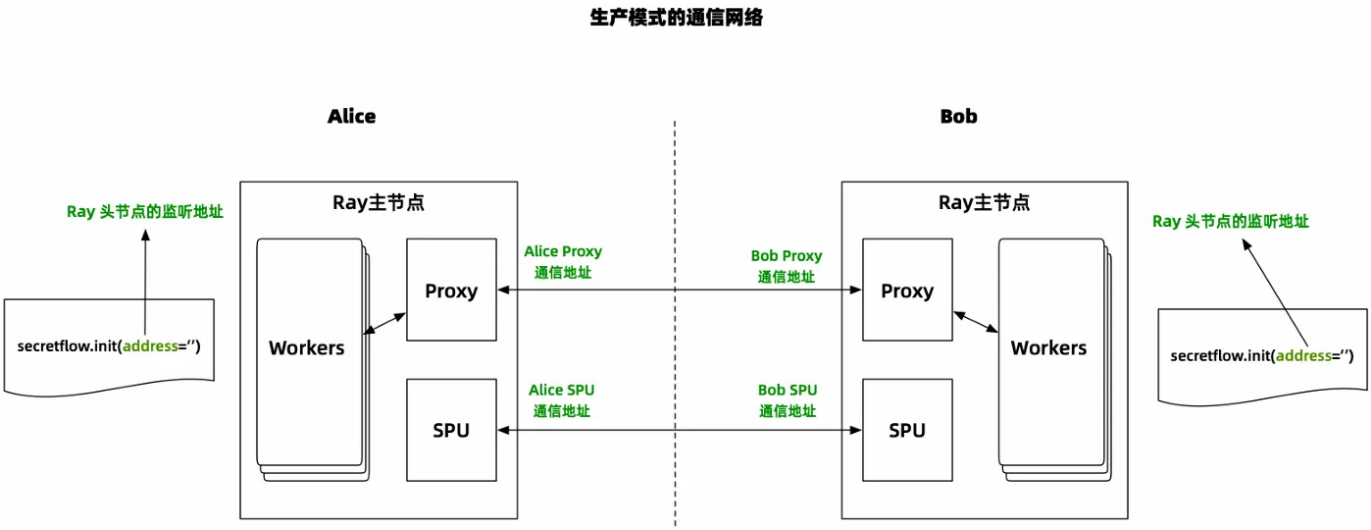
Secretflow PSI开发指南

Secretflow 部署模式

在隐语的集群仿真模式下，每个Ray节点模拟一个机构，具体做法是通过给每个Ray节点添加机构名称标记，从而保证机构的计算被调度到相应的Ray节点上



用于生产的SecretFlow由多个ray集群组成，每个参与方拥有各自的ray集群。与此同时，每一个参与方都要同时执行代码，才能完成任务的协 作



启动ray集群

alice首先启动ray集群。注意这里的命令是启动Ray的主节点。

```
ray start --head --node-ip-address="ip" --port="port" --include-dashboard=False --disable-usage-stats
```

bob首先启动ray集群

```
ray start --head --node-ip-address="ip" --port="port" --include-dashboard=False --disable-usage-stats
```

初始化 secetflow

```
sf_cluster_config ={
    'parties': {
        'alice': {
            # replace with alice's real address.
            'address': 'ip:port of alice',
            'listen_addr': '0.0.0.0:port'
        },
        'bob': {
            # replace with bob's real address.
            'address': 'ip:port of bob',
            'listen_addr': '0.0.0.0:port'
        },
    },
    'self_party': 'bob'
}
```

```
tls_config = {
    "ca_cert": "ca root cert of other parties ",
    "cert": "server cert of alice in pem",
    "key": "server key of alice in pem",
}
```

```
sf.init(address='alice ray head node address', cluster_config=sf_cluster_config, tls_config=tls_config)

sf.init(address='bob ray head node address', cluster_config=sf_cluster_config, tls_config=tls_config )
```

启动SPU设备

```
spu_cluster_def = {
        nodes': [
                # <<< !!! >>> replace <192.168.0.1:12945> to alice node's local ip & free port
                {'party': 'alice', 'address': '192.168.0.1:12945', 'listen_address': '0.0.0.0:12945'},
                # <<< !!! >>> replace <192.168.0.2:12946> to bob node's local ip & free port
                {'party': 'bob', 'address': '192.168.0.2:12946', 'listen_address': '0.0.0.0:12946'},
        ],

        'runtime_config': {
                'protocol': spu.spu_pb2.SEMI2K,
                'field': spu.spu_pb2.FM128,
        },
}


spu = sf.SPU(spu_cluster_def)
```

执行 PSI

```
reports = spu.psi_csv(

  key=select_keys,

  input_path=input_path,

  output_path=output_path,

  receiver='alice', # receiver get output file.

  # psi protocol  KKRT_PSI_2PC, BC22_PSI_2PC
  protocol='ECDH_PSI_2PC',

  curve_type='CURVE_25519',
# 'CURVE_FOURQ',  'CURVE_SM2'

  precheck_input=False,  # check inputfile duplicate entries

  sort=False,  # sort intersection by key ids

  broadcast_result=False, # true receiver send intersection to
other parties

)
```

运行结果

reports 结构

```
# 输入数据量总数
int64 original_count = 1;
# 交接结果
int64 intersection_count = 2;
```

PSI 交集输出

```
output_path = {
  alice: '/data/psi_output.csv',  # 节点alice端的输出
  bob: '/data/psi_output_bob.csv',  # 节点bob端的输出
}
```

隐语PSI后续计划

PSI协议开发

- •RS22 Blazing Fast vole+okvs
- •Circuit PSI
- •Multiparty PSI
- •Malicious PSI

PSI调用框架

- PSI独立代码库
- 优化入口函数和参数
- 优化协议封装架构

PSI产品化

- 轻量化部署
- 算法原理可视化