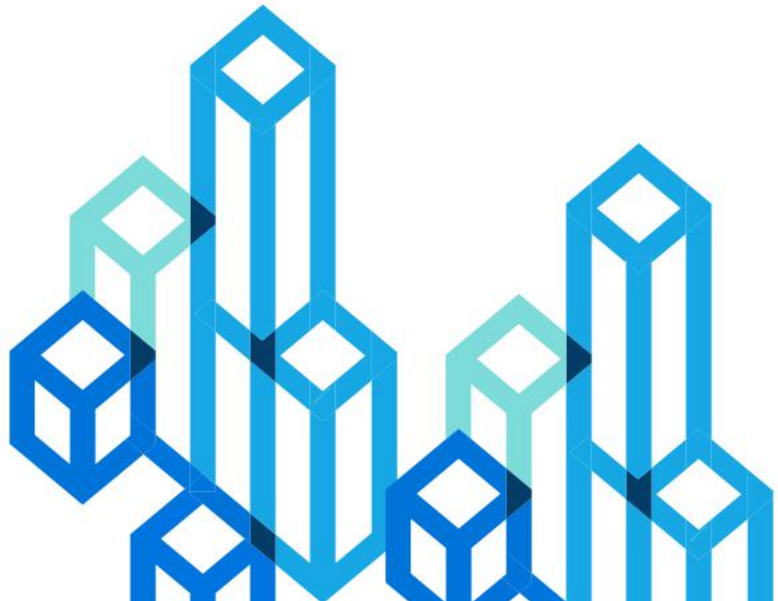
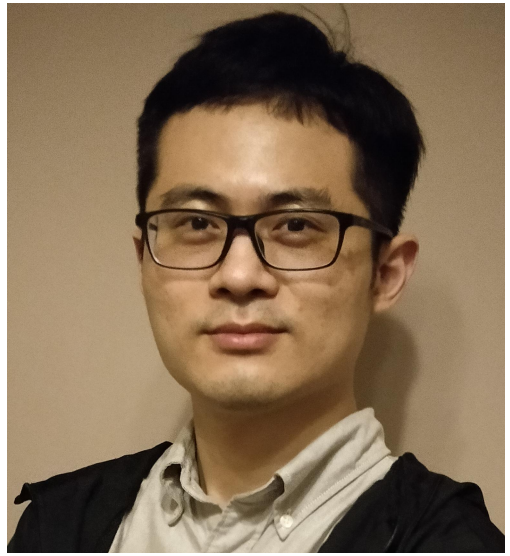
隐私计算系统架构设计的思考与实践

--以 开源框架 Rosetta 为例

2021.07.31



个人简介



矩阵元算法架构师
Rosetta 框架核心开发者、开源社区技术负责人；在安全多方计算(MPC)、零知识证明、同态计算和差分隐私等前沿隐私计算技术方面有丰富的落地实践经验。

曾在百度安全事业部、腾讯区块链业务中心从事超大规模分布式访问控制系统、AI安全、联盟链应用平台等项目开发.

南京大学计算机系硕士毕业，研究方向为分布式系统安全和应用密码学，在安全云计算、隐私保护数据挖掘等领域发表了多篇论文

目录

当我们在谈论隐私时，我们到底在谈什么？

当我们在谈论隐私时，我们到底在谈什么？

AI 给我们带来便利的同时，也对我们个人的隐私构成了潜在威胁

- AI 正在改变我们的世界，融入到我们的日常生活中；
- AI 模型的训练需要大量的数据，数据量对模型的效果具有决定性意义
  - ImageNet、自动驾驶

当我们在谈论隐私时，我们到底在谈什么？

AI 给我们带来便利的同时，也对我们个人的隐私构成了潜在威胁

- AI 正在改变我们的世界，融入到我们的日常生活中；
- AI 模型的训练需要大量的数据，数据量对模型的效果具有决定性意义；
- 矛盾：

数据的过度采集、数据的不透明使用
VS
更好的 AI 服务

当我们在谈论隐私时，我们到底在谈什么？

如何应对潜在侵犯隐私的挑战？

- 法律法规的跟进、监管的介入

当我们在谈论隐私时，我们到底在谈什么？

如何应对潜在侵犯隐私的挑战？

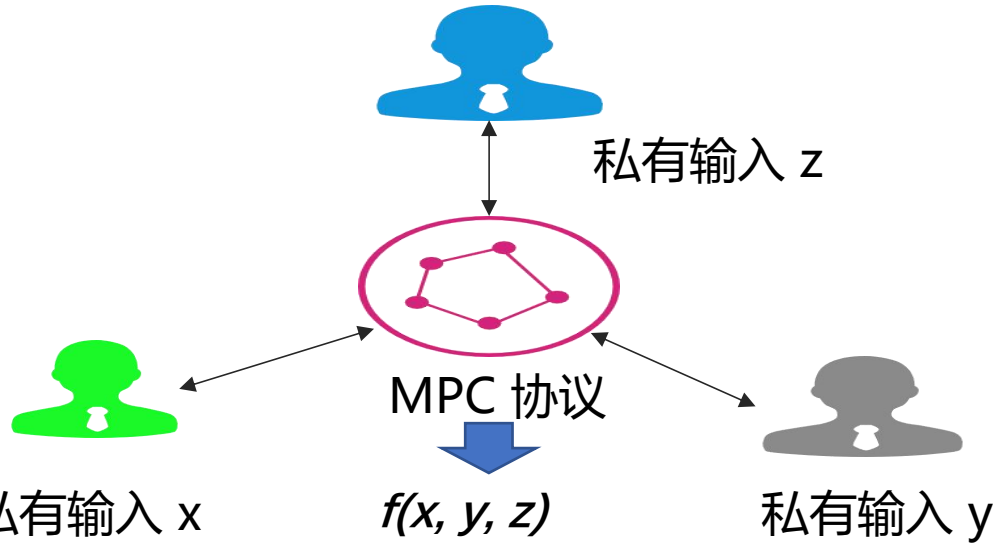
- 法律法规的跟进、监管的介入

技术发展带来的问题，技术本身可以做什么？

有技术解决方案吗?

有！密码学技术提供了强大可靠的工具，如安全多方计算 MPC

- 由图灵奖获得者、中国科学院院士姚期智先生于1982年首先提出
  - 多个参与方在不泄露隐私数据的前提下，对等的进行协同计算
  - 理论上可以计算任意的图灵可计算函数



- 随后，理论密码学界不断提升**通用方案**基础操作的效率
  - GMW87、BGW88、Beaver triple trick(92) 等等

- 2000s，Privacy-Preserving Data Mining 领域开始兴起
  - **实验性的应用**于决策树、SVM、NN 等经典机器学习算法

- 最近几年，和具体 AI 算法的**高效融合**
  - SecureML、SecureNN、Manticore 等协议聚焦于面向更上层算子功能的优化

其他途径的探索：

- *可信执行环境 TEE：2015 年 Intel 开始推出 SGX，硬件隔离为特定程序提供CPU、存储访问；*
- *联邦学习 FL: 2017 年 Google 等提出大规模分布式机器学习中保护用户原始数据*

如何开始落地？需要一个通用的软件框架

- 前沿理论到大规模落地一般都需要一个软件框架作为媒介
  - 智能化的需求 + 深度学习算法的逐渐完善 --> TensorFlow、PyTorch
  - 用户隐私保护需求 + MPC等技术的成熟 --> 隐私计算系统框架
- 实际痛点：

我是搞AI模型的算法工程师，
我想去隔壁部门的数据仓库里多关联些用户特征来优化模型，但他们担心让我直接访问会泄露数据；
我其实并不care他的明文数据，能不能想办法搞个安全的方式让我的模型在它的数据上跑起来啊？

我是一个密码学专业的学生，
我对Max算子有个提升性能的idea，
我不太懂AI，但我想看一下使用了这个算子之后，隐私保护版本的ResNet 能有多快？这样我好发paper毕业呀.

隐私计算系统该如何构建?

隐私计算开源框架的探索

- 工程技术挑战
  - **易用性**：面向AI，不需要了解密码学等技术
  - **可扩展性：快速集成新的隐私计算协议和 AI 模型**
  - **高效性**：优化多方之间的通信轮数和计算开销

- 现有框架都还处于早期探索阶段

Rosetta 是什么？

- 基于 TensorFlow 开发
- 几行代码，“一键切换”

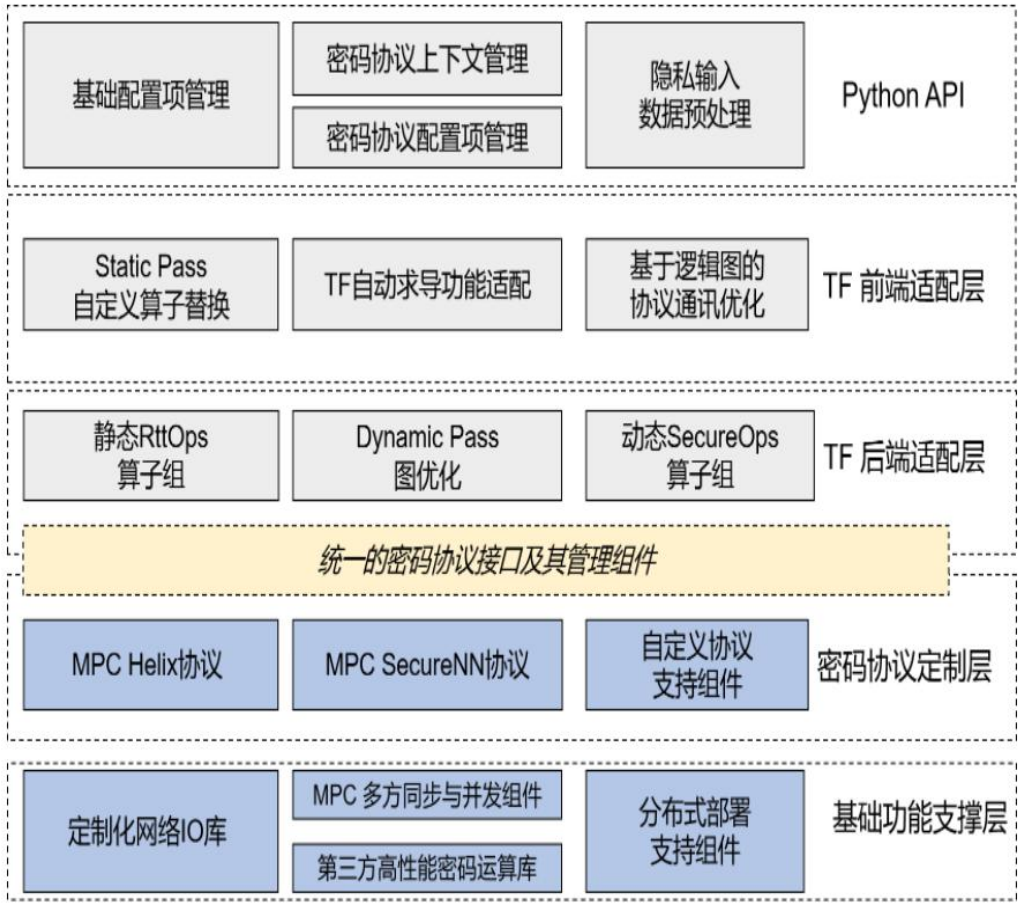
https://github.com/LatticeX-Foundation/Rosetta

```python
import latticex.rosetta as rtt
import tensorflow as tf

EPOCHES = 50
BATCH_SIZE = 64
learning_rate = 0.002

rtt.activate("SecureNN")

# input their private data
local_file_x = "feature_x.csv"
local_file_y = "label_y.csv"
collective_X, collective_Y = rtt.PrivateDataset(
    data_owner=(0, 1), label_owner=0).load_data(
    local_file_x, local_file_y, header=None)
FEATURE_NUM = collective_X.shape[1]

X = tf.placeholder(tf.float64, [None, FEATURE_NUM])
Y = tf.placeholder(tf.float64, [None, 1])
# initialize W & b
Weights = tf.Variable(tf.zeros([FEATURE_NUM, 1], dtype=tf
    .float64), name='w')
bias = tf.Variable(tf.zeros([1], dtype=tf.float64), name=
    'b')
# define model and loss function
logits = tf.matmul(X, Weights) + bias
loss = tf.nn.sigmoid_cross_entropy_with_logits(labels=Y,
    logits=logits)
loss = tf.reduce_mean(loss)
# TF built-in utils for initialize variables, auto-grad,
    and saving model result
init = tf.global_variables_initializer()
train = tf.train.GradientDescentOptimizer(learning_rate).
    minimize(loss)
saver = tf.train.Saver(var_list=None, name='v2')

with tf.Session() as sess:
    sess.run(init)
    BATCHES = int(len(collective_X) / BATCH_SIZE)
    for e in range(EPOCHES):
        for i in range(BATCHES):
            bX = collective_X[(i * BATCH_SIZE): (i + 1) *
     BATCH_SIZE]
            bY = collective_Y[(i * BATCH_SIZE): (i + 1) *
     BATCH_SIZE]
            sess.run(train, feed_dict={X: bX, Y: bY})
    saver.save(sess, './log/ckpt/model')
```

Rosetta 整体架构

- **基本原则**
  - **分层抽象**
  - **模块化解耦**

- **基本模式**
  - **以自定义算子(Operation) SecureOp 为核心抽象**
    - 连接起对于 TF 前后端的改造
    - 实现后端隐私协议开发和AI框架的解耦
  - **借鉴程序编译器领域中 Pass 的概念**
    - 将原生 TF API 构建的数据流图自动、动态的转换为多方协同运行的MPC程序

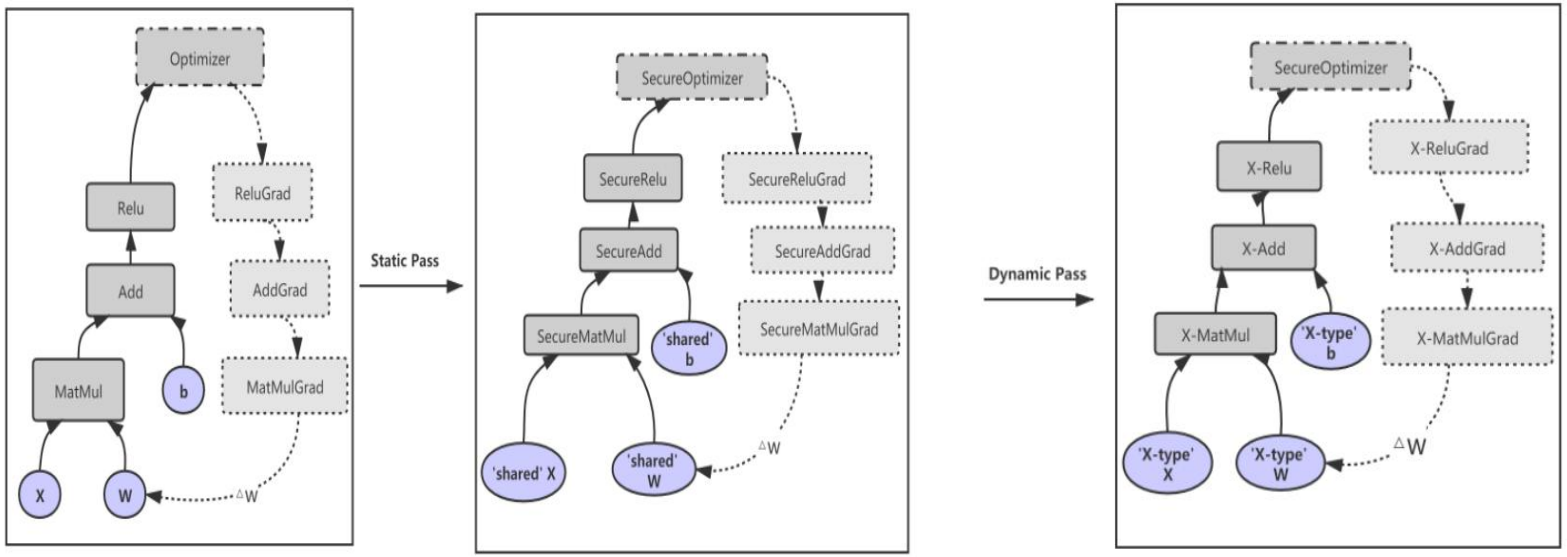


Rosetta 整体处理流程

TF Graph 的自动、分阶段映射

- TF 前向图静态替换
- TF 梯度图自动替换
- 算子执行时“多态”

```
class SecureSquareOp : public StrUnaryOp {
 private:
  /* data */
 public:
  SecureSquareOp(OpKernelConstruction* context) : StrUnaryOp(context) {}
  ~SecureSquareOp() {}

  int UnaryCompute(const vector<string>& input, vector<string>& output) {
    log_debug << "--> Square OpKernel compute.";
    ProtocolManager::Instance()->GetProtocol()->GetOps(msg_id().str())->Square(input, output, &attrs_);
    log_debug << "Square OpKernel compute ok. <--";
    return 0;
  }
};
```



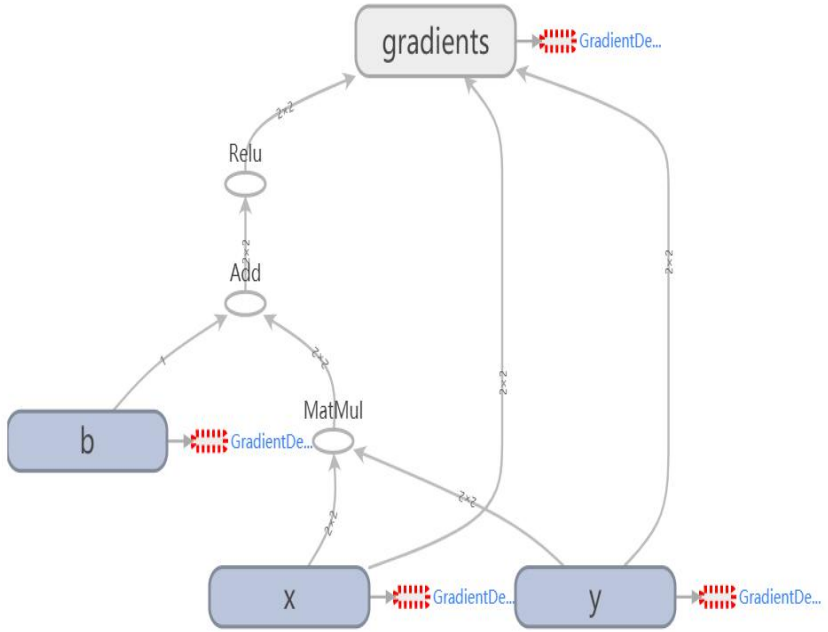
*X 表示用户选定的安全密码协议

Rosetta 整体处理详解(1): 基于 Static Pass 构建图

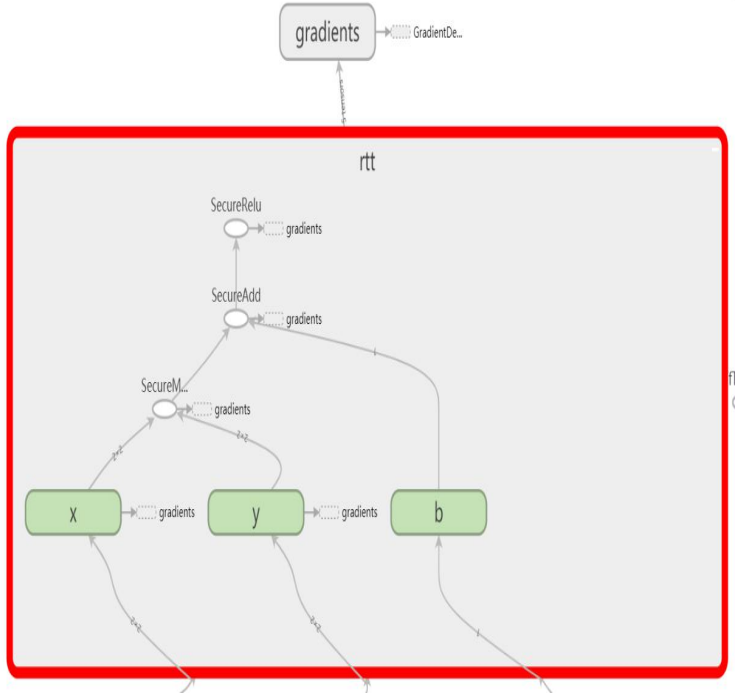
- **在调用 session.run 时**
  > 前向图：将 RttOp 进一步替换为对应的 SecureOp

替换规则:
只要 Op 输入的 Tensor 不全是本地 const 明文值时就进行替换

SecureOp:
完整的前后端算子库，具有对应的梯度算子；在内部实现中调用隐私协议层的抽象算子接口实现和 TF 的对接



Static pass



Rosetta 整体处理详解(1): 基于 Static Pass 构建图

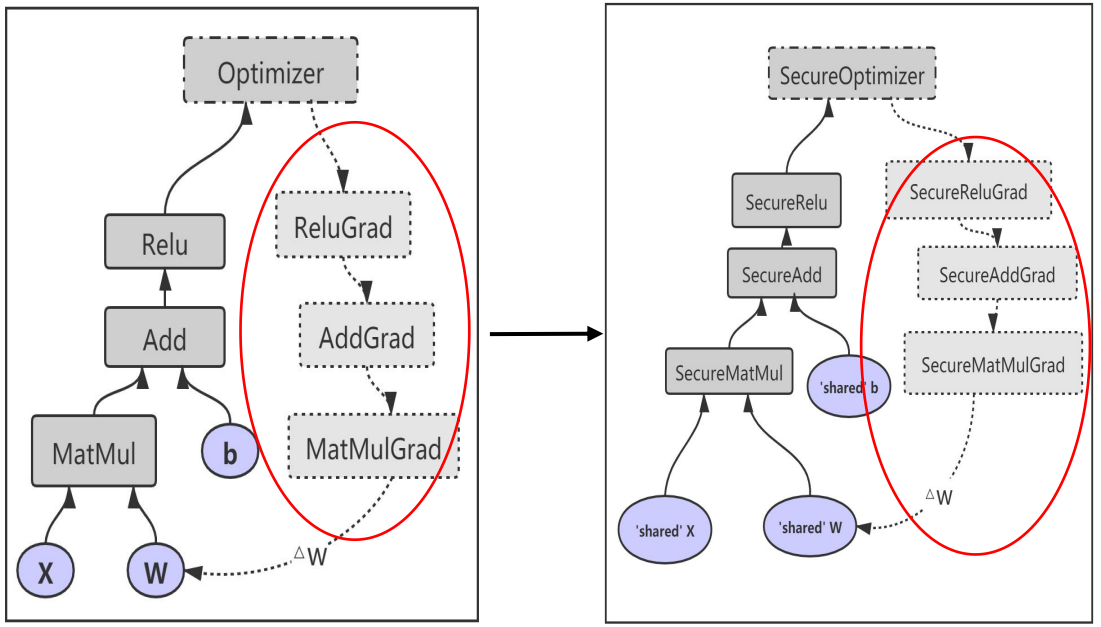
- **在调用 session.run 时**
  > 前向图：将 RttOp 进一步替换为对应的 SecureOp

  > 反向图：如果程序中调用了优化器 Optimizer，通过我们对 TF 中自动求导机制的改造，仍可自动得到梯度反向图

SecureOp:
完整的前后端算子库，具有对应的梯度算子；在内部实现中调用隐私协议层的抽象算子接口实现和 TF 的对接

自动求导改造方案:
在 tf.python.ops.gradients_util 等入口处 hook 原生函数，使得支持 tf.string 类型的自动求导

```python
def RttIsBackpropagatable(tensor):
    # NOTE(GeorgeShi): We make 'string' legal for backpropagating.
    if tensor.dtype == dtypes.string:
        return True                                    NEW!

    if gradients_util.IsTrainable(tensor):
        return True

    dtype = dtypes.as_dtype(tensor.dtype)
    return dtype.base_dtype == dtypes.bfloat16

# override tensorflow _IsBackpropagatable with RttIsBackpropagatable
gradients_util._IsBackpropagatable = RttIsBackpropagatable
```

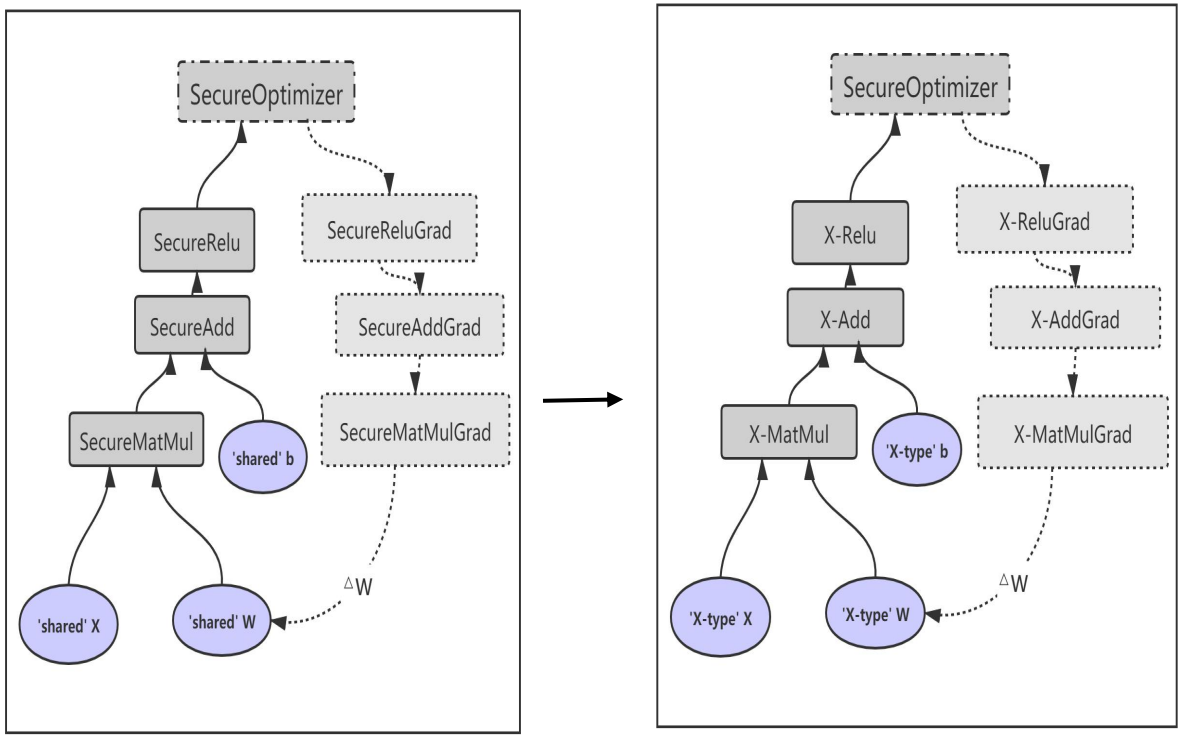


Rosetta 整体处理详解(2): 基于 Dynamic Pass 执行图

- **动态后端协议执行**
  执行图时，SecureOp 内部根据上下文调用此时的隐私协议的后端接口，实际的进行分布式多方运行

动态隐私协议绑定的实现方案：
SecureOp 的 kernel 实现中调用统一协议层的基类接口，不依赖具体协议内部实现

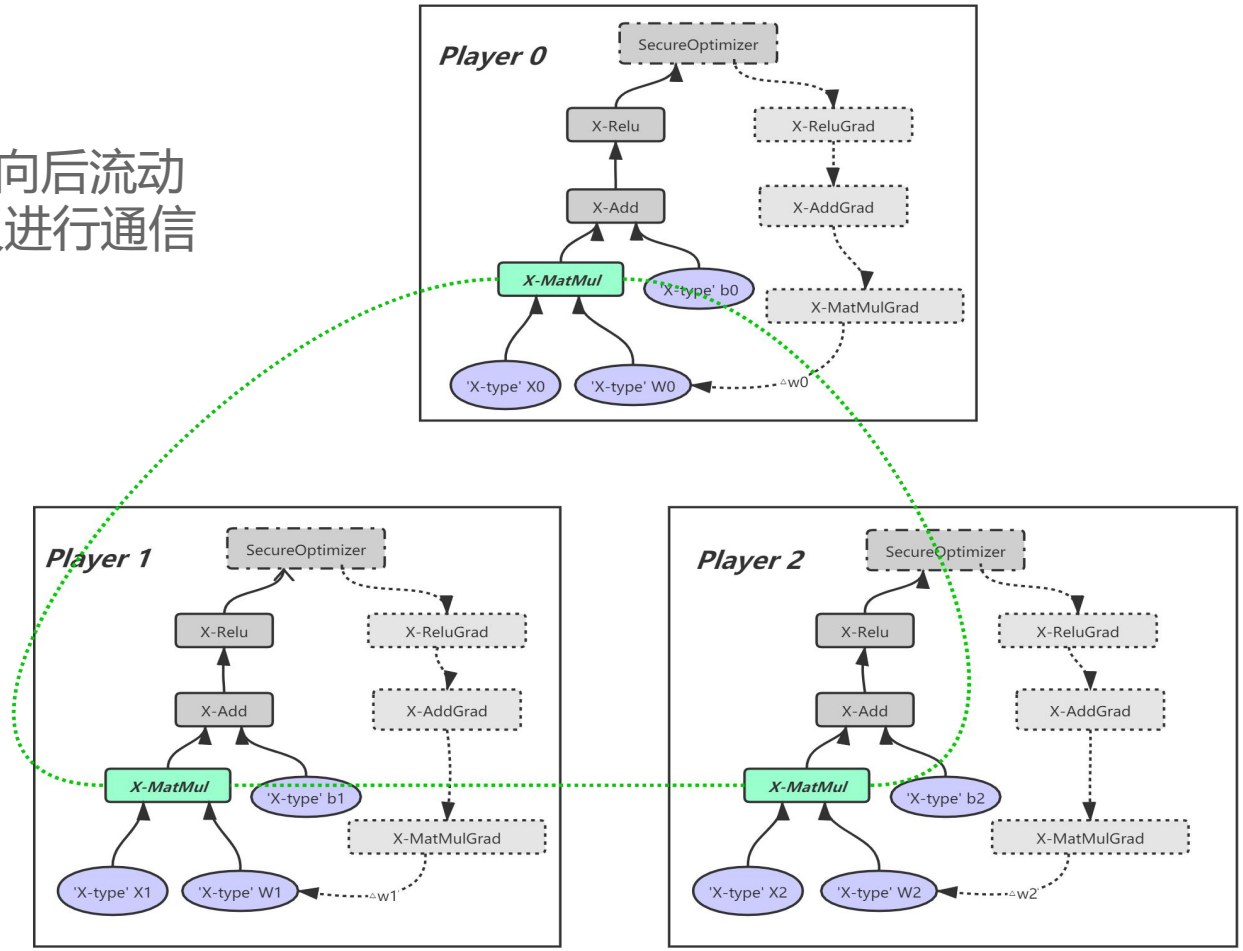
```
class SecureSquareOp : public StrUnaryOp {
 private:
  /* data */
 public:
  SecureSquareOp(OpKernelConstruction* context) : StrUnaryOp(context) {}
  ~SecureSquareOp() {}

  int UnaryCompute(const vector<string>& input, vector<string>& output) {
    log_debug << "--> Square OpKernel compute.";
    ProtocolManager::Instance()->GetProtocol()->GetOps(msg_id().str())->Square(input, output, &attrs_);
    log_debug << "Square OpKernel compute ok. <--";
    return 0;
  }
};
```



*X 表示用户选定的MPC协议

Rosetta 整体处理详解(2): 基于 Dynamic Pass 执行图

- **多方协同的分布式执行**
  从整体视角看，各方同步执行相同的图
  > 本地： “密文”数据在 DAG 图上从前向后流动
  > 协同： “密文”数据在多方之间按协议进行通信



*X 表示用户选定的MPC协议

在具体的实践中，架构设计怎么做?

隐私计算系统挑战--易用性

案例：如何做到和现有 TF 的模型 checkpoint 使用兼容?

- 训练 (training)：如何在多方数据集的协同计算，然后导出为某一方的明文模型?
- 预测 (Inference)：一方已训练好的明文模型，如何在新的数据样本上进行安全预测?

我是搞AI模型的算法工程师，
我想去隔壁部门的数据仓库里多关联些用户特征来优化模型，
但隔壁部门担心让我访问明文会泄露数据；
我其实并不care他的明文数据，能不能想办法搞个安全的方式让我的模型在它的数据上跑起来啊?

```
{ "MPC": {
    "P0": {
      "HOST": "192.168.0.1",
      "PORT": 11121
    },
    "P1": {
      "HOST": "192.168.0.2",
      "PORT": 12144
    },
    "P2": {
      "HOST": "192.168.0.3",
      "PORT": 13169
    },
    "FLOAT_PRECISION": 16,
    "SAVER_MODE": 1,
    "RESTORE_MODE": 1
  }
}
```

对 TF 前后端进行相关的适配改造，
通过配置文件配置后就达成用户无感的转换。

隐私计算系统挑战--易用性

案例：如何做到和现有 TF 的模型 checkpoint 使用兼容?

- TF后端对接和适配：自定义**隐私算子**，并根据用户配置对待保存的Tensor进行处理

```
REGISTER_OP("SecureSaveV2")
    .Input("prefix: string")
    .Input("tensor_names: string")
    .Input("shape_and_slices: string")
    .Input("tensors: dtypes")
    .Attr("dtypes: list(type)")
    .Doc(R"doc(
SecureSaveV2Op
)doc");

REGISTER_OP("SecureRestoreV2")
    .Input("prefix: string")
    .Input("tensor_names: string")
    .Input("shape_and_slices: string")
    .Output("tensors: dtypes")
    .Attr("dtypes: list(type)")
    .Doc(R"doc(
SecureRestoreV2Op
)doc");
```

隐私算子注册

```
class SecureSaveV2Op : public SecureOpKernel {
 public:
  explicit SecureSaveV2Op(OpKernelConstruction* context) : SecureOpKernel(context) {}

  void ComputeImpl(OpKernelContext* context) {
    const Tensor& prefix = context->input(0);
    const Tensor& tensor_names = context->input(1);
    const Tensor& shape_and_slices = context->input(2);
    ValidateInputs(true /* is save op */, context, prefix, tensor_names, shape_and_slices);

    const int kFixedInputs = 3; // Prefix, tensor names, shape_and_slices.
    const int num_tensors = static_cast<int>(tensor_names.NumElements());
    const string& prefix_string = prefix.scalar<string>()();
    const auto& tensor_names_flat = tensor_names.flat<string>();
    const auto& shape_and_slices_flat = shape_and_slices.flat<string>();
```

隐私算子实现

隐私计算系统挑战--易用性

案例：如何做到和现有 TF 的模型 checkpoint 使用兼容?

- TF后端对接和适配：自定义**隐私算子**，并根据用户配置对待保存的Tensor进行处理
- TF 前端适配：继承并重载 `tf.train.Saver` 类的方法，然后静态替换

```
def restore_op(self, filename_tensor, saveable, preferred_shard):
  """Create ops to restore 'saveable'.

  This is intended to be overridden by subclasses that want to generate
  different Ops.

  Args:
    filename_tensor: String Tensor.
    saveable: A BaseSaverBuilder.SaveableObject object.
    preferred_shard: Int.  Shard to open first when loading a sharded file.

  Returns:
    A list of Tensors resulting from reading 'saveable' from
      'filename'.
  """
  # pylint: disable=protected-access
  tensors = []
  for spec in saveable.specs:
    tensors.append(
        io_ops.restore_v2(filename_tensor, [spec.name], [spec.slice_spec],
                          [spec.dtype])[0])
```

```
class SecureBaseSaverBuilder(saver.BaseSaverBuilder):
    # pylint: disable=unused-argument
    def restore_op(self, filename_tensor, saveable, preferred_shard):
        """Create ops to restore 'saveable'.

        This is intended to be overridden by subclasses that want to generate
        different Ops.

        Args:
        filename_tensor: String Tensor.
        saveable: A BaseSaverBuilder.SaveableObject object.
        preferred_shard: Int.  Shard to open first when loading a sharded file.

        Returns:
        A list of Tensors resulting from reading 'saveable' from
            'filename'.
        """
        # pylint: disable=protected-access
        tensors = []
        for spec in saveable.specs:
            tensors.append(
                SecureRestoreV2(filename_tensor, [spec.name], [spec.slice_spec],
                                [spec.dtype])[0])

        return tensors
```

隐私计算系统挑战 -- 可扩展性

系统可扩展性

- 要支持安全算法协议的扩展
- 要支持 AI 模型的快速适配

我是一个密码学专业的学生，
我对矩阵乘法有个提升性能的idea，
我不太懂AI，但我想看一下使用了这个算子之后，隐私保护版本的ResNet 能有多快？这样我好发paper毕业呢。

抽象解耦；
模块化组合；
可插拔式切换

隐私计算系统挑战 -- 可扩展性

案例：零知识证明(ZKP) 协议 Wolverine 的集成

- **Wolverine(2021) 是一个 SOTA 的 ZKP 协议**
- 密码学专家想要验证这个协议可以用于 ResNet 下的预测
  - ResNet 中包含 Convolution、ReLU、Pooling、BatchNorm（Sqrt、Div）等等TF算子

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
		3×3 max pool, stride 2				
conv2_x	56×56	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

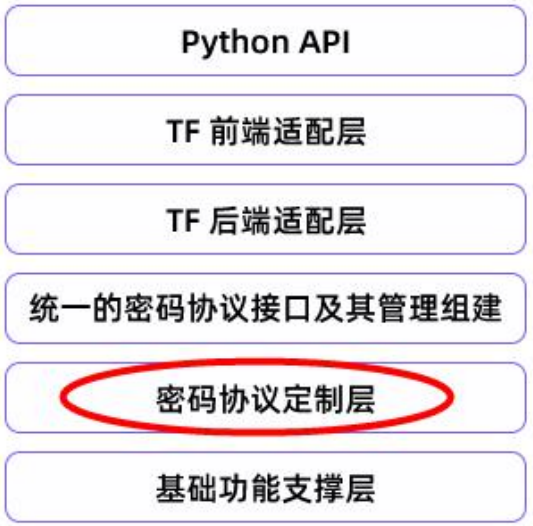
** Wolverine: Fast, Scalable, and Communication-Efficient Zero-Knowledge Proofs for Boolean and Arithmetic Circuits,Chenkai Weng, Kang Yang, Jonathan Katz, and Xiao Wang In IEEE Symposium on Security and Privacy (S&P), 2021*

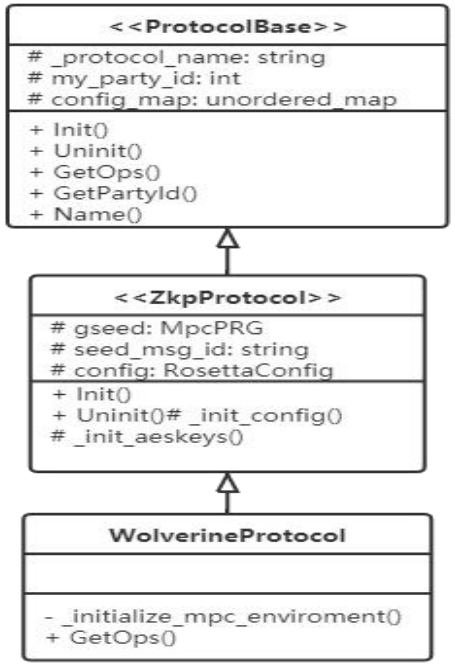
** Mystique: Efficient Conversions for Zero-Knowledge Proofs with Applications to Machine Learning, Chenkai Weng and Kang Yang and Xiang Xie and Jonathan Katz and Xiao Wang, In USENIX Security, 2021*

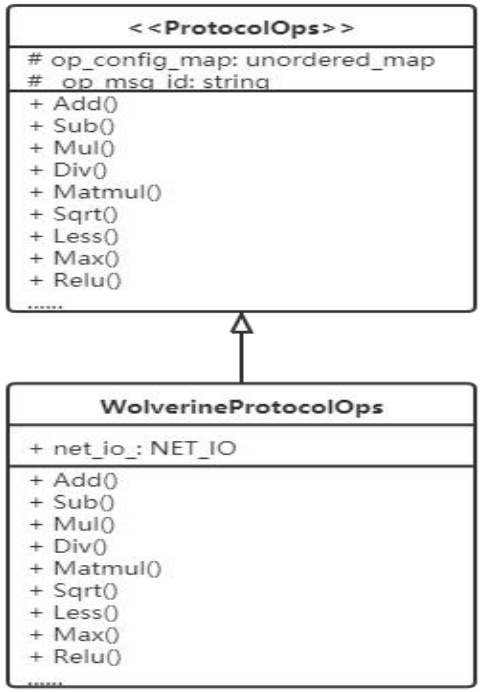
隐私计算系统挑战 -- 可扩展性

案例：零知识证明(ZKP) 协议 Wolverine 的集成

1：协议算子封装：基于 Wolverine 基础库继承实现 Rosetta 后端协议的接口类





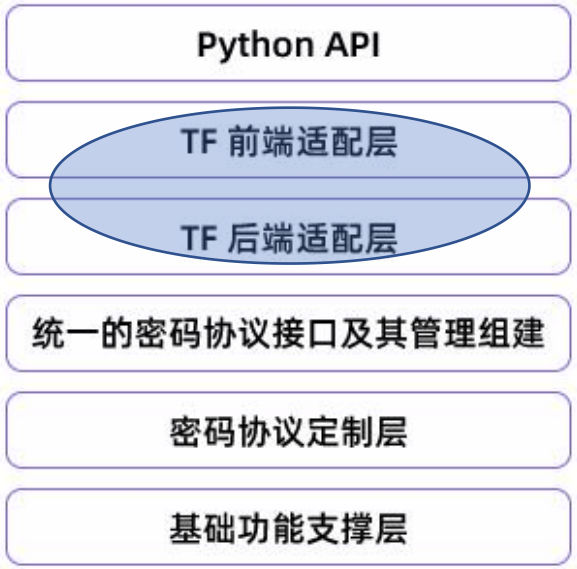


** 详细协议集成教程文档：https://www.infoq.cn/article/V3wm8NUGllvTEYsaw3Hj*

隐私计算系统挑战 -- 可扩展性

案例：零知识证明(ZKP) 协议 Wolverine 的集成

1：协议算子封装：基于 Wolverine 基础库继承实现 Rosetta 后端协议的接口类
2：TF 前后端对接集成：通用部分，可以进行相关优化




```
class SecureFusedBatchNormOp : public SecureOpKernel {
 private:
  float epsilon_;
  TensorFormat tensor_format_;
  bool is_training_;

 public:
  SecureFusedBatchNormOp(OpKernelConstruction* context) : SecureOpKernel(context) {
    float epsilon;
    OP_REQUIRES_OK(context, context->GetAttr("epsilon", &epsilon));
    epsilon_ = float(epsilon);
    string tensor_format;
    OP_REQUIRES_OK(context, context->GetAttr("data_format", &tensor_format));
    OP_REQUIRES(
      context, FormatFromString(tensor_format, &tensor_format_),
      errors::InvalidArgument("Invalid data format"));
    OP_REQUIRES_OK(context, context->GetAttr("is_training", &is_training_));
  }
```

** 详细协议集成教程文档：https://www.infoq.cn/article/V3wm8NUGllvTEYsaw3Hj*

隐私计算系统挑战 -- 可扩展性

案例：零知识证明(ZKP) 协议 Wolverine 的集成

1：协议算子封装：基于 Wolverine 基础库继承实现 Rosetta 后端协议的接口类
2：TF 前后端对接集成：通用部分，**可以进行隐私计算下相关优化**

```
class SecureFusedBatchNormOp : public SecureOpKernel {
 private:
  float epsilon_;
  TensorFormat tensor_format_;
  bool is_training_;

 public:
  SecureFusedBatchNormOp(OpKernelConstruction* context) : SecureOpKernel(context) {
    float epsilon;
    OP_REQUIRES_OK(context, context->GetAttr("epsilon", &epsilon));
    epsilon_ = float(epsilon);
    string tensor_format;
    OP_REQUIRES_OK(context, context->GetAttr("data_format", &tensor_format));
    OP_REQUIRES(
      context, FormatFromString(tensor_format, &tensor_format_),
      errors::InvalidArgument("Invalid data format"));
    OP_REQUIRES_OK(context, context->GetAttr("is_training", &is_training_));
  }
```

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β
Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma^2_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma\widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

预测

Algorithm 1: Batch Normalizing Transform, applied to activation x over a mini-batch.

通过简单的调整计算的顺序（先单独计算 rsqrt）就可以获得数倍的性能提升

隐私计算系统挑战 -- 可扩展性

案例：零知识证明(ZKP) 协议 Wolverine 的集成

- **成果：**首个支持在超过100层的大型深度学习模型上进行准确推断的 **ZKP** 协议

基础算子性能:

	50 Mbps	200 Mbps	500 Mbps	1 Gbps
Machine Learning (ML) Functions				
Sigmoid	2.1 ms	1.6 ms	1.6 ms	1.6 ms
Max Pooling	1.6 ms	0.5 ms	0.4 ms	0.4 ms
ReLU	908 μs	262 μs	185 μs	188 μs
SoftMax-10	209 ms	157 ms	161 ms	171 ms
Batch Norm	415 ms	261 ms	257 ms	269 ms

CIFAR-10数据集上深度模型性能:

Model	Image	LeNet-5	ResNet-50	ResNet-101
Communication				
Private	Private	16.5 MB	1.27 GB	1.98 GB
Private	Public	16.5 MB	1.27 GB	1.98 GB
Public	Private	16.4 MB	0.53 GB	0.99 GB
Execution time (seconds) in a 50 Mbps network				
Private	Private	7.3	465	736
Private	Public	7.5	463	735
Public	Private	6.5	210	369
Execution time (seconds) in a 200 Mbps network				
Private	Private	5.9	333	535
Private	Public	5.5	336	541
Public	Private	4.9	158	262

** Mystique: Efficient Conversions for Zero-Knowledge Proofs with Applications to Machine Learning, Chenkai Weng and Kang Yang and Xiang Xie and Jonathan Katz and Xiao Wang, In USENIX Security, 2021*

隐私计算系统挑战 -- 高效性

系统性能仍是隐私计算能否大规模应用落地的关键

- 函数参数值向量化，batch化的调用一次OP 完成多个独立的计算
- 尽量使用“低消耗”的基础算子来实现衍生算子

案例：Equal 算子的实现

假设我们目前有 LessEqual 、Mul 等基础算子，如何构造 Equal 算子?

- a == b 等价于 a - b == 0
  - 方案一：MPC 下基于 (([a]-[b]) <=[0]) && (([b]-[a]) <= [0])
  - 方案二：
    1). MPC 下各方本地生成随机数 [r]
    2). MPC 下各方计算 [a] - [b] + [r] 并交互得到明文 r' = (a - b) + r
    3). MPC 下判断每一位上XOR的乘积是否为 0： $\wedge_{i=0}^{i=l-1}[r_i] \oplus [r'_i]$

隐私计算系统挑战 -- 高效性

系统性能是隐私计算能否大规模应用落地的关键

- 函数参数值向量化，batch化的调用一次OP 完成多个独立的计算
- 尽量使用“低消耗”的基础算子来实现衍生算子

Algorithm 20 FastEqual($\{(P_0, P_1), P_2\}, [\![X]\!]_i^L, [\![Y]\!]_i^L$)

Input: $[\![X]\!]_i^L, [\![Y]\!]_i^L$ held by Party P_i for $i \in \{0,1\}$.

Output: $[\![X = Y]\!]_i^2$, i.e., $[\![1]\!]_i^2$ if equal, otherwise $[\![0]\!]_i^2$.

1: P_i sets $[\![Z]\!]_i^L = [\![X]\!]_i^L - [\![Y]\!]_i^L$ for $i \in \{0,1\}$ locally.

2: P_2 and P_0 generate random $r_0 = PRF(K_{02}||Common||count)$ to get an arithmetic share of r for P_0.
P_2 and P_1 generate random $r_1 = PRF(K_{12}||Common||count)$ to get an arithmetic share of r for P_1.
P_2 then sets $r = r_0 + r_1$.

3: P_2 and P_0 generate $r_0' = PRF(K_{02}||Common||count)$, and decompose r_0' into bits $\{r[j]_0\}_{j=0}^{l}$. Note that this is a bit-share of $\{r[j]\}_{j=0}^{l}$ for P_0.

4: P_2 gets $r_1' = r \oplus r_0'$ (i.e. $\{r[j]_1 = r[j] \oplus r_0[j]\}_{j=0}^{l}$), and then **send** r_1' to P_1. Note that this value is the bit-share of $\{r[j]\}_{j=0}^{l}$ for P_1.

5: P_i locally computes $[\![Z']\!]_i^L = [\![Z]\!]_i^L + [\![r]\!]_i^L$ for $i \in \{0,1\}$ locally, and then **exchanges** $[\![Z']\!]_i^L$ with each other to reveal $Z' = Z + r$ for $i \in \{0,1\}$.

6: P_0 sets $b_0[j] = (1 - r_0'[j] \oplus Z'[j])$, and P_1 sets $b_1[j] = r_1'[j]$. Note that $b[j] = b_0[j] \oplus b_1[j] = \overline{Z'[j] \oplus r[j]} = \overline{(Z + r)[j] \oplus r[j]} = \overline{Z[j]}$, and should all be 1 if and only if $Z = 0$.

7: P_i together perform $[\![B]\!]_i^2 = \wedge_{j=0}^{j=l-1}\{b_i[j]\}$, of which $\wedge$ (Add) is done by calling **Mul** on boolean value. And we can use binary-tree strategy so that the l fan-in Add can be done in parallel.

8: **return** $[\![B]\!]_i^2$.

方案二相比方案一有三倍性能提升.

代码参见：https://github.com/LatticeX-Foundation/Rosetta/pull/39

```
partyid:0 ppid:82842 pid:82843 beg
run SecureNN performance test
Notes: (partyid:0)
    OP: operator name. eg. 0.1.OpName: 0 means variable, 1 means constant.
    loops: how many loops to execute.
    elapsed(s): the total time spent executing the OP (loops).
    avg-elapse(ms): (elapsed(s)*1000.0)/loops.
    shape size: k = r * c = m * K = K * n.
    sent data/recv data: IO total bytes. (per operator; including message id)
    sent msgs/recv msgs: IO interface invocation times. (per operator)
+----------------------+-----------+-------------+-------------+-------------------+-----------+-----------+-----------+-----------+
|                   OP |     loops |   elapsed(s) | avg-elapse(ms) |        shape size | sent data | recv data | sent msgs | recv msgs |
+----------------------+-----------+-------------+-------------+-------------------+-----------+-----------+-----------+-----------+
|            0.0.Equal |        71 |  3.391608781 |  47.769137761 |  k=1452,k=1452 |    464840 |     93048 |        10 |         6 |
|         0.0.NotEqual |        73 |  3.308109725 |  45.316571575 |  k=1452,k=1452 |    464840 |     93048 |        10 |         6 |
|            1.0.Equal |        71 |  3.120501799 |  43.950729563 |  k=1452,k=1452 |    464840 |     93048 |        10 |         6 |
|         1.0.NotEqual |        73 |  3.311111973 |   45.35769826 |  k=1452,k=1452 |    464840 |     93048 |        10 |         6 |
|            0.1.Equal |        71 |  2.962096077 |  41.719663056 |  k=1452,k=1452 |    464840 |     93048 |        10 |         6 |
|         0.1.NotEqual |        73 |  3.131190565 |  42.893021438 |  k=1452,k=1452 |    464840 |     93048 |        10 |         6 |
+----------------------+-----------+-------------+-------------+-------------------+-----------+-----------+-----------+-----------+
Total elapsed(s): 19.2251
partyid:0 ppid:82842 pid:82843 end
```

```
run SecureNN performance test
Notes: (partyid:0)
    OP: operator name. eg. 0.1.OpName: 0 means variable, 1 means constant.
    loops: how many loops to execute.
    elapsed(s): the total time spent executing the OP (loops).
    avg-elapse(ms): (elapsed(s)*1000.0)/loops.
    shape size: k = r * c = m * K = K * n.
    sent data/recv data: IO total bytes. (per operator; including message id)
    sent msgs/recv msgs: IO interface invocation times. (per operator)
+----------------------+-----------+-------------+-------------+-------------------+-----------+-----------+-----------+-----------+
|                   OP |     loops |   elapsed(s) | avg-elapse(ms) |        shape size | sent data | recv data | sent msgs | recv msgs |
+----------------------+-----------+-------------+-------------+-------------------+-----------+-----------+-----------+-----------+
|            0.0.Equal |        71 |  1.141062137 |  16.071297704 |  k=1452,k=1452 |     34827 |     34827 |         8 |         8 |
|         0.0.NotEqual |        73 |  1.111960092 |  15.232330027 |  k=1452,k=1452 |     34827 |     34827 |         8 |         8 |
|            1.0.Equal |        71 |  1.049211396 |  14.777625296 |  k=1452,k=1452 |     34827 |     34827 |         8 |         8 |
|         1.0.NotEqual |        73 |  1.010779923 |  13.846300315 |  k=1452,k=1452 |     34827 |     34827 |         8 |         8 |
|            0.1.Equal |        71 |  1.001946166 |  14.111917831 |  k=1452,k=1452 |     34827 |     34827 |         8 |         8 |
|         0.1.NotEqual |        73 |  1.026755136 |  14.065138849 |  k=1452,k=1452 |     34827 |     34827 |         8 |         8 |
+----------------------+-----------+-------------+-------------+-------------------+-----------+-----------+-----------+-----------+
Total elapsed(s): 6.34245
```

总结

- **隐私计算是什么?**
  - 现实的隐私保护需求 + 密码学等相关基础技术上的成熟

- **如何构建隐私计算框架? 思考与实践**
  - **可用性**：尽可能复用 TF 的API
  - **可扩展性**：基于 TF后端内部的多态化实现和密码协议的解耦
  - **高效性**：面向特定功能算子的定制化优化

麦思博(msup)有限公司是一家面向技术型企业的培训咨询机构，携手2000余位中外客座导师，服务于技术团队的能力提升、软件工程效能和产品创新迭代，超过3000余家企业续约学习，是科技领域占有率第1的客座导师品牌，msup以整合全球领先经验实践为己任，为中国产业快速发展提供智库。

高可用架构主要关注互联网架构及高可用、可扩展及高性能领域的知识传播。订阅用户覆盖主流互联网及软件领域系统架构技术从业人员。 高可用架构系列社群是一个社区组织，其精神是“分享+交流”，提倡社区的人人参与，同时从社区获得高质量的内容。