MPC可信硬件

> Secure multi-party computation (MPC) enable a group to jointly perform a computation without disclosing any participant’s private inputs. The participants agree on a function to compute, and then can use an MPC protocol to jointly compute the output of that function on their secret inputs without revealing them.

MPC(Secure multi-party computation) 可以用于解决一组互不信任的参与方之间保护隐私的协同计算问题，SMPC 要确保输入的独立性，计算的正确性，同时不泄露各输入值给参与计算的其他成员。本项目实现了可信硬件下的安全多方计算通用框架。

项目优势

目前的MPC框架存在大量的问题 - 计算效率低下。 - 无法保证可证明安全或安全性较低

尽管近年来研究者努力进行实用化技术的研究，并取得一些成果，但是离真正的大规模产业化应用还有一段距离。

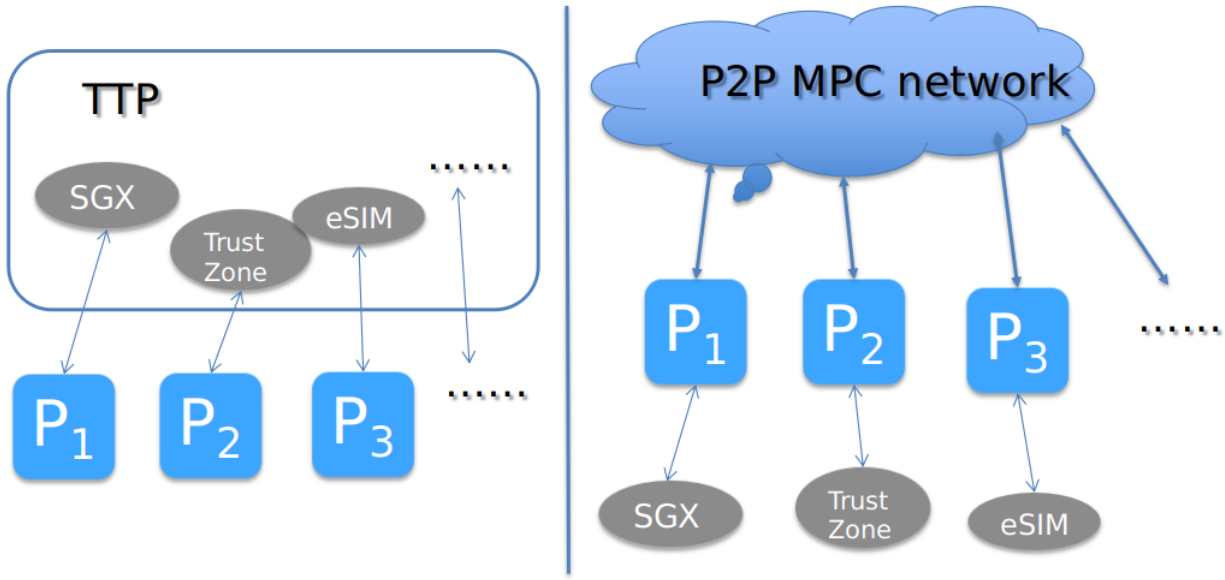
使用可信硬件的MPC

将可信硬件用于安全多方计算是一种有效提高计算速度的方法，并且在安全性假设下，其相较于传统交互式的MPC具有更高的安全性优势。但目前主流的可信硬件大多采用SGX，但由于英特尔的信任源问题，无法保证数据安全。本系统开发了通用的可信硬件支持框架，可以嵌入任何可信硬件。

与同类产品的比较

本项目设计开发了基于可信软硬件的通用安全多方计算平台。符合以下特点： * 异构性：该安全多方计算平台可以采用TPM、smartcard、TrustZone等各种实现，并非依赖于Intel SGX * 高效性：该框架可支持全部通用安全多方计算任务，性能(通信量与运行时间)超过目前最先进的非可信软硬件的通用安全多方计算系统2~3个数量级。 * 强鲁棒性以及高安全性：在任意n-1计算方恶意攻击的情况下，仍然保证输出计算结果的正确性。该框架超过现有已和安全多方计算方案的安全水平。该框架可以做到mobile下的malicious安全。

可证明安全下的异构



框架的逻辑结果和物理结构对比

不同的公司采用各自的底层可信硬件平台参与计算，信任源由单一的SGX到多信任源。在逻辑上可以规约到所有计算方直接与可信第三方交互。

同类型安全性比较

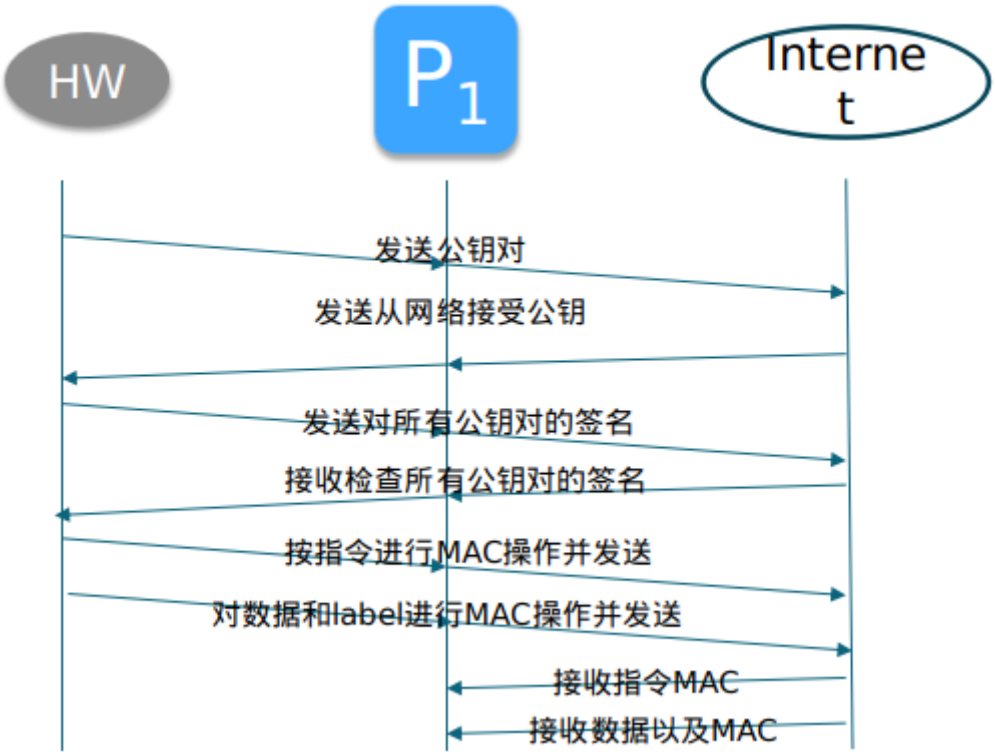
本框架与同类型相比在安全性上有较大优势，做到了mobile下的malicious安全，远远超过了目前市面上的主流框架。

安全等级(敌手模型)	可证明安全多方计算协议				
	Sharemind	Overdrive 2k	PrivPy	tf-encrypted	本项目
静态（static）半诚实	✓	✓	✓	✓	✓
静态（static）恶意	✗	✓	✗	✗	✓
自适应（adaptive）半诚实	✗	✗	✗	✗	✓
自适应（adaptive）恶意	✗	✗	✗	✗	✓
动态（mobile）半诚实	✗	✗	✗	✗	✓
动态（mobile）恶意	✗	✗	✗	✗	✓

与同类型框架安全性比较

协议说明

协议流程分为秘钥交换， 运行指令交换，数据交换，以及进行多方计算四个流程。 - 秘钥交换
协议需要两个非对称密钥对，以及一个对称秘钥对。在秘钥交换流程中，用户向网络广播其非对称密钥对：I.用户查询自己的可信硬件，可信硬件将在可信硬件中生成的公钥对中的公钥一起返回给用户，用户将该公钥分享到MPC网络上。同时用户监听MPC信道，收集所有收到的公钥对，并告知可信硬件。II.在I稳定后，可信硬件使用可信硬件中生成的签名私钥对所知的所有公钥进行签名。再由用户将该签名发送到MPC网络。同时检查收到的签名，检查正确后储存对应的公钥。 - 运行指令交换
运行指令交换目的是交换用户的运行协议以保证用户无法使用其他指令来进行作弊操作，向可信硬件发起签署请求，可信硬件对指令添加计数器，并使用生成的对称秘钥进行MAC，返回使用非对称秘钥对中用于加密的私钥对对称秘钥进行加密，并将该密文和MAC数据一起返回给用户。用户向MPC网络广播该密文。并监听MPC网络，储存接收的其他方发送的MAC数据。 - 数据交换
与指令交换类似的，需要进行数据交换来确保用户运行的指令中的数据的正确性。可信硬件同样使用非对称秘钥对中用于加密的私钥对对称秘钥进行加密，然后使用对称秘钥对数据进行加密，最后使用该秘钥对数据和指令中使用的标签整合内容进行MAC计算。用户广播并监听。



秘钥、数据交换过程

- 计算
  用户需要进行计算时将标签、数据密文，指令，以及标签-数据密文的MAC、指令的MAC输入到可信硬件，可信硬件返回输出标签对应的密文结果和标签-密文MAC。

代码实现

项目结构

- compiler
  - mixedProtocolsAnalysis
  - Protocols_Analysis
- sootOutput
- HW
- *.cpp
  - main
  - network
  - protocol

HW hardware 中是模拟可信硬件的类文件。

compiler 负责将java code 编译成电路语句，输出到sootOutput文件夹，其中mixedProtocolsAnalysis是编译程序的编译运行文件，Protocols_Analysis是源文件。

protocol 程序负责读入电路语句，对电路语句进行展开、去除数据依赖，与HW交互对指令进行MAC操作等。

network 程序负责整个网络的通信，对数据进行序列化，提供缓冲区，储存交互的数据等。

main 程序中运行了整个程序的业务流程，包括多次的数据交互，秘钥交换，指令的运行等。

其他文件 包括了编译安装的脚本，一些中间函数等。

运行环境

系统环境：Linux

运行：

```
cd install_tool
#检查环境，或手动安装jsoncpp
bash ./install.sh
#编译java code 文件 bash ./compile.sh java_code项目路径  如
bash ./compile.sh ./java_code_demo/mexp
#运行程序
./main
```

Java code编译成电路语句

将java code 编译成电路语句，涉及到代码块的拼接，代码参数、宏定义的展开等。使用Protocols_Analysis程序对java文件进行编译得到电路程序，输出在sootOutput文件中。使用file_deal函数对该文件进行裁剪获取运行有效部分。

电路语句展开实现

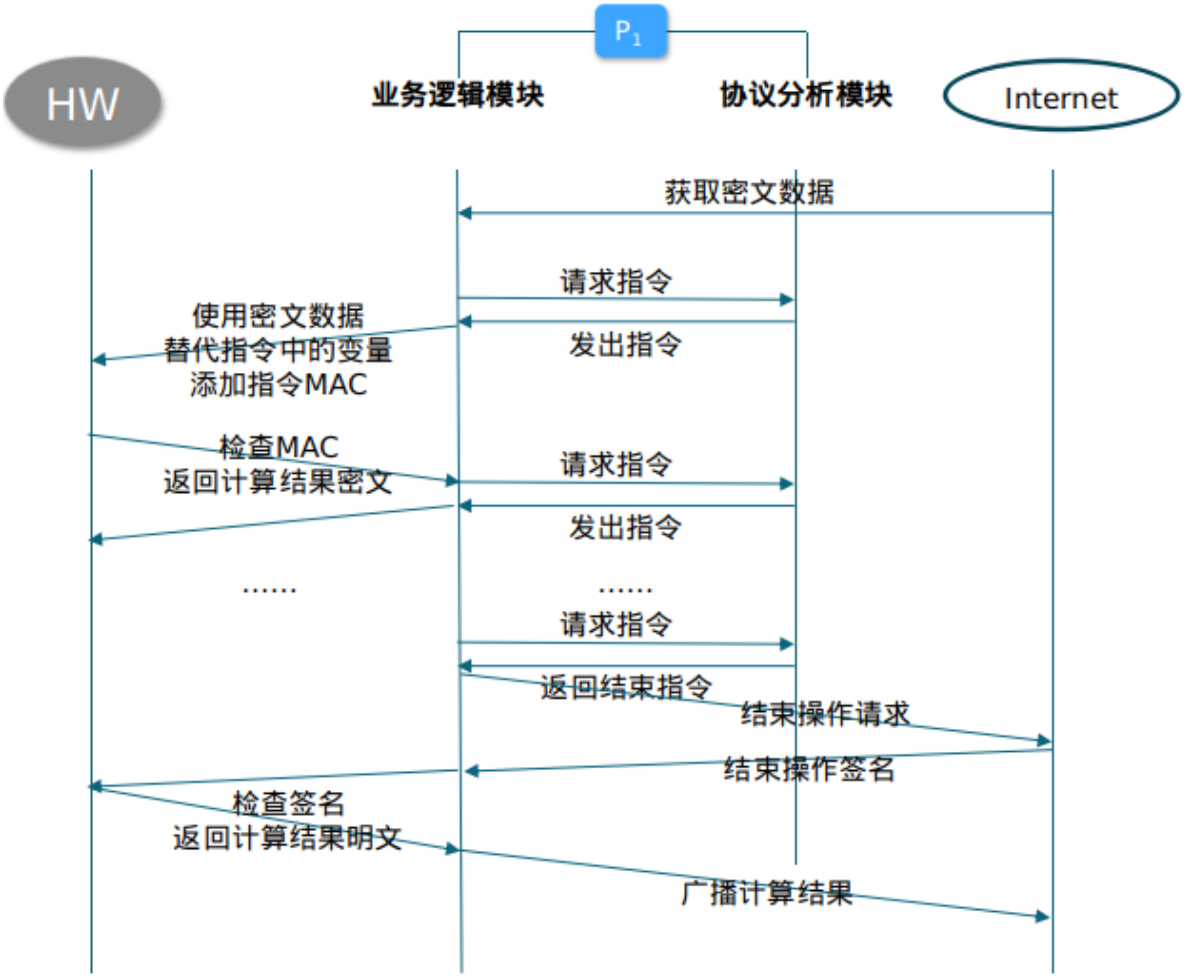
由于需要对指令进行数据有效性检查，需要添加MAC，在系统对指令进行签名时，需要对指令变量进行展开（针对数组变量等），此时要对程序段进行预运行，该预运行（展开）包括了 1. 对分支语句进行展开，以确保得到的指令可以添加计数器。 2. 对不合法指令进行检查，密文不能出现在分支语句中、密文不能出现在数组索引中等。 3. 对数组元素进行改写，如arr[i11], 先确保i11是明文数据，不然返回报错，在用密文改写数组，如i11在运行到此指令时value为100，则得到arr_100。 4. 对代码进行预展开能确保程序是数据独立的，只有通过了预展开，才能证明代码段在输入密文数据前数据流已经确定了。

协议运行实现

系统流程设计

- 代码编译成电路指令模块 在该模块，实现了对用户输入的java代码转换为电路指令的功能，使用算法消除了代码的数据相关性（牺牲一定的运算时间）。最后将转换的指令流读入内存供主程序查询。

- 网络层模块设计 网络实现了数据的接受和发送，以及对数据的序列化和对象化，主程序将数据对象交付给该模块后，该模块转换为二进制流发送到网络；该模块将在网络上监听到的二进制数据流转换为数据对象供主程序查询。

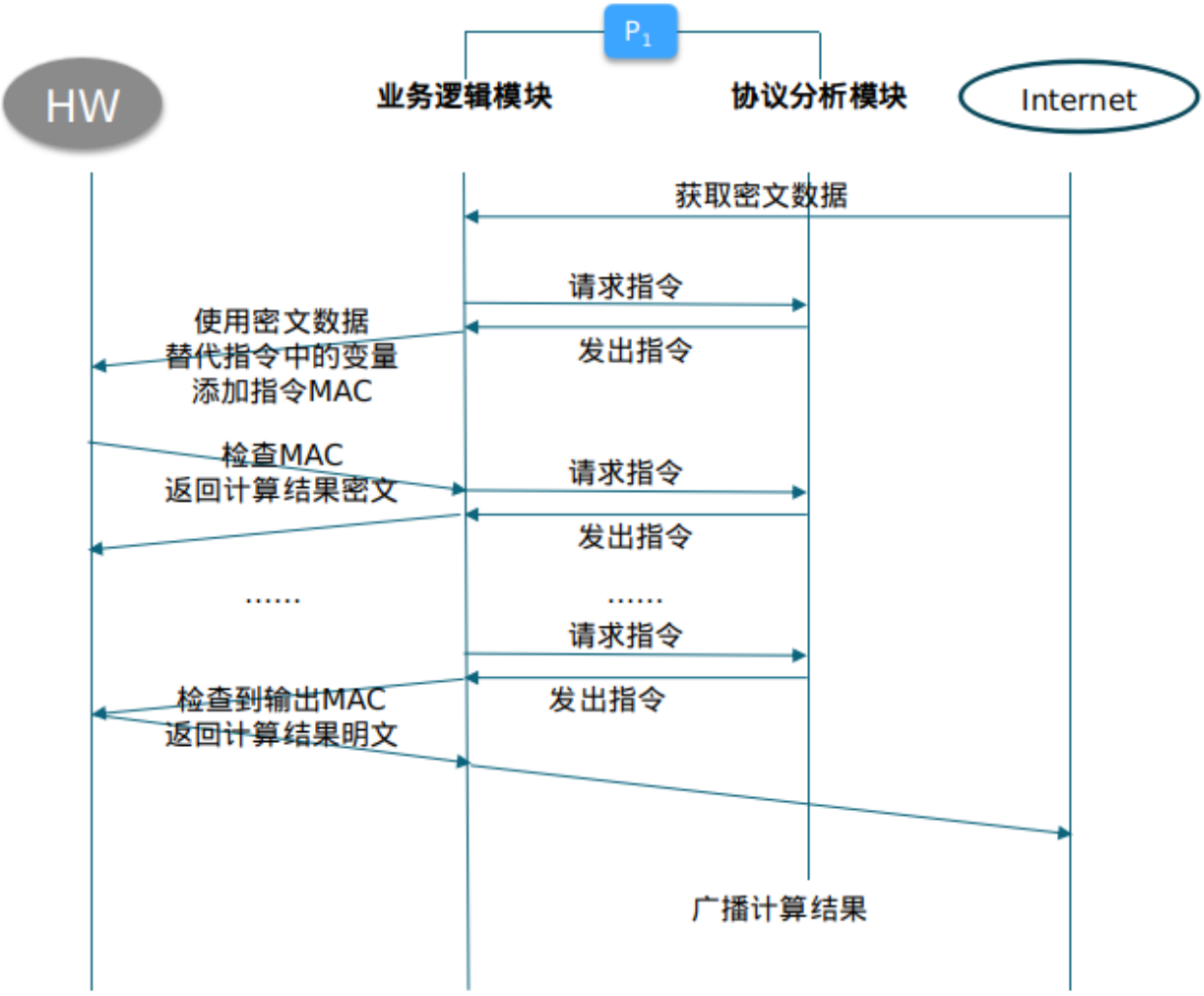


协议运行

- 应用层模块设计 在该模块实现了整个程序的调度逻辑，包括密钥交换，签名，指令流的提交和轮询，控制整个用户程序的业务流程，按上图协议运行流程运行。而在该模块的具体实现中为了降低代码的耦合度，将继续划分成多个功能模块和主模块。

- 可信硬件程序设计 在可信硬件程序的设计中，先根据用户交互设计固定了抽象类的所有接口，然后根据不同的平台开发相同接口的可信硬件程序。具体实现了

    - 所有的逻辑运算操作、数值计算操作，并返回操作结果的密文形式。
    - 公钥对，对称密钥对生产和对应的查询公钥接口。
    - 数据的加密查询接口。
    - 对不同流程中签名的检查逻辑，并返回对应的数据流。



协议运行优化

- 相关优化 对协议进行了优化，如数据交互中的数据流压缩，指令签名交互过程中对指令 MAC的重新设计等， 如上图对计算流程的优化，去除了输出结果前的签名交互流程，减少了网络开销。

Release

- Jan 2, 2020
  实现了基本逻辑，可以运行大部分指令，目前已经实现了array的所有功能，但java code累计从命令行输入变量到数组中的代码，编译成电路语句时由于需要用户交互输入，而框架设计为non-interactive，无法支持此类代码，需要在下一步进行改写。
- Jan 6, 2020
  修改了移位操作时计算错误的BUG，修改了验证指令完整性MAC的逻辑，改为使用链式计算MAC的方法，记录累计MAC值，进行指令MAC发送时只发送结束MAC，可信硬件只有在检查到结束MAC时才进行明文输出，该方案减少了网络通信量。