基于 JSR 269 的安全多方计算编译器

刘 芹 汪鹏程 崔竞松 涂 航
(空天信息安全与可信计算教育部重点实验室武汉大学国家网络安全学院 湖北 武汉 430079)

摘 要 随着多种通用安全多方计算协议的提出,在这些协议上构建的框架、领域特定语言层出不穷,但都有着易用性差、现有编程语言难以交互等问题。因此,针对这些问题,设计一种基于 JSR 269 的安全多方计算编译器构建方案。该方案将经过了安全多方计算相关的注解标注的 Java 源代码,通过编译器插件编译为安全多方计算应用,并且可以和 Java 语言进行交互。通过实验及结果分析,该方案可保留 Java 语言的高级语言特性,以高度抽象的方式进行安全多方计算应用逻辑的编写。

关键词 JSR 269 安全多方计算 编译器

中图分类号 TP309 **文献标志码** A DOI:10.3969/j.issn.1000-386x.2024.07.043

SECURE MULTI-PARTY COMPUTATION COMPILER BASED ON JSR 269

Liu Qin Wang Pengcheng Cui Jingsong Tu Hang
(*Key Laboratory of Aerospace Information Security and Trusted Computing*, *Ministry of Education*, *School of Cyber Science and Engineering*, *Wuhan University*, *Wuhan 430079*, *Hubei*, *China*)

Abstract With the development of multiple general secure multi-party computation protocols, frameworks and domain-specific languages built on these protocols are emerging in endlessly, but they all have problems on usability and are difficult in interaction with existing programming languages. Therefore, to solve these problems, a secure multi-party computation compiler construction scheme based on JSR 269 is designed. In this scheme, Java compiler compiled the annotated Java source code into a secure multi-party computation application through a compiler plugin, and compiled code could interact with the Java language. According to experiments and analysis, this scheme can retain the high-level language features of the Java language, and programmers can write secure multi-party computation application in a highly abstract level.

Keywords JSR 269 Secure multi-party computation Compiler

0 引 言

安全多方计算(Secure Multi-party Computation, MPC)[1]是现代密码学领域的一个重要研究方向,在学术界、工业界和政府中具有广泛的应用。它提供了一种机制:一组私有数据所有者可以通过该机制来联合计算其私有数据的结果,而其中协议的输出并不会暴露输入数据的任何内容。例如 Yao[2]提出安全多方计算这个概念时引入的“百万富翁问题”:两个百万富翁想比较彼此之间谁更有钱,但又不想让对方知道自己拥有的金额。假设富翁 A 拥有的金额为 a,富翁 B 拥有的金额为 b,那么“百万富翁问题”就是如何在不泄露彼此金额的前提下,进行 $a > b$ 的这个比较操作的安全运算。由于这些强大的安全保证,MPC 在实际应用中具有广阔的潜力,从基本的安全统计分析,到更多特定领域的用途,例如财务监督、金融数据处理、生物医学计算、卫星碰撞检测等,都有 MPC 的身影。

尽管很多领域都需要 MPC 技术,但 MPC 的实际

收稿日期:2021-03-15。“十三五”国家重点研发计划项目(2016YFB0501800)。**刘芹**,副教授,主研领域:密码学。**汪鹏程**,硕士生。**崔竞松**,副教授。**涂航**,副教授。

应用却受到了很多限制,部分原因是底层协议的效率很难满足应用的需求。为了解决效率问题,密码学家针对各种使用场景,开发了高度优化的专用 MPC 协议,而对于通用 MPC 协议如混淆电路也有了很多针对性的优化工作。但其仍面临的另一个问题就是缺乏易用性,即使这些 MPC 协议在理论上足以满足实际使用的效率,对于使用者也有着极高的密码学背景要求,并且从底层开发一个满足要求的 MPC 应用需要耗费巨大的时间和人力。

通用 MPC 协议能够安全地计算任何函数功能,并在密码学界几十年的研究中,已经诞生了非常多的优秀协议。但直到最近,才出现了可以执行任意功能的通用 MPC 编程框架/编译器。这些项目迅速改善了 MPC 的现有技术水平和应用场景,大大减轻了设计多个专用协议的负担,使得非密码学专家也可以快速建立原型并部署 MPC 应用。通过编译器,通用 MPC 协议在易用性、安全性上的工程化工作可以获得最大的解放。

但现有 MPC 编译器通常采用自定义的领域特定语言(Domain Specific Language,DSL)作为前端语言,增加了学习成本并且很难和现有编程语言编写的程序进行交互。为了与现有编程语言交互、减少开发者学习新语言的心智负担,基于现有编程语言如 Java 等来开发 MPC 应用是非常直观的,但因为缺乏相关编译器的知识,以及通用编程语言和 MPC 协议之间存在鸿沟,这方面的相关工作一直是缺失的。现代编程语言的编译器一般都拥有插件化的扩展接口便于用户对编译流程的定制、优化,例如 Java 编译器的 JSR 269 规范[3]等,它们都可以对编译器流程进行修改。因此,可以基于这些编译器扩展将现有编程语言编译为 MPC 应用,提供更具易用性、功能性的编程语言、编译器等工具。

1　现状分析

当前很多国内外的学者已经投身到如何提升 MPC 易用性的研究当中。文献[4]提出了一种名为 Fairplay 的第一个通用 MPC 编译器实现,它可以将一种称为安全功能定义语言(SFDL)的自定义领域特定语言编译为姚氏混淆电路并执行,但其无法与现有编程语言交互;文献[5]提出了一种允许将符合 ASNI C 语法子集的程序编译为大小优化的布尔电路的 CBMC-GC 方案,但并没有提供可运行的安全计算执行引擎;文献[6]提出的 Frigate 可以将一种自定义的类 C 语言编译为用于任意数量输入的自定义布尔电路表示形式,但同样不提供安全计算执行引擎的实现;文献[7]提出的 ABY 是一种支持混合协议的两方计算框架,它可以在姚氏混淆电路、布尔电路、算术电路三种不同协议之间切换,但它以 C ++ 库的形式提供,并且提供的接口仍限于门电路的形式,无法提供更高层的抽象;文献[8]提出的 HyCC 利用上述工作中的 CBMC-GC 和 ABY,首次将符合 ASNI C 语法的程序编译为混合协议的电路形式并应用在机器学习中,但其前端和 CBMC-GC 一样,虽然采用了相同的语法,但无法与 ASNI C 交互;文献[9]提出的 PICCO 编译器可以将扩展的 C 语言源文件源到源编译为可以执行安全多方计算的 C ++ 源文件,但其只能将输入输出以文件的形式提供,很难与其他语言编写的程序进行交互;文献[10]提出的 ObliVM 将 ORAM 和混淆电路结合,可以将一种类似 Java 的高级程序语言 ObliVM-Lang 编译为 Java 源代码,然后通过它提供的运行时库进行执行,因为它的输出结果是 Java 源码,所以存在和 Java 原生代码交互的可能性,但需要用户自己编写复杂的包装代码。

本文设计一种基于 JSR 269 的安全多方计算编译器,实现了 Java 语言编写的 MPC 应用逻辑到底层 MPC 协议的编译,同时保留与原生 Java 代码的交互性。利用 Java 语言提供高层抽象能力,可以方便地设计 MPC 应用,并且将其嵌入到业务系统中,大大提升了易用性和功能性。

2　编译器简介

在本方案中,用户主要利用编译器提供给上层的 Java 注解对方法进行标注。在编译的过程中,当发现某个方法包含 MPC 注解,则将该方法编译为电路等表示形式作为类的静态字段。同时将方法体替换为安全计算执行引擎的执行及相关的数据类型转换,从而完成安全多方计算的功能。

2.1　JSR 269 介绍

如图 1 所示,Java 语言的编译过程大致可以分为三个部分:1) 解析与填充符号表。2) 插件式注解处理器的注解处理。3) 分析与字节码生成。

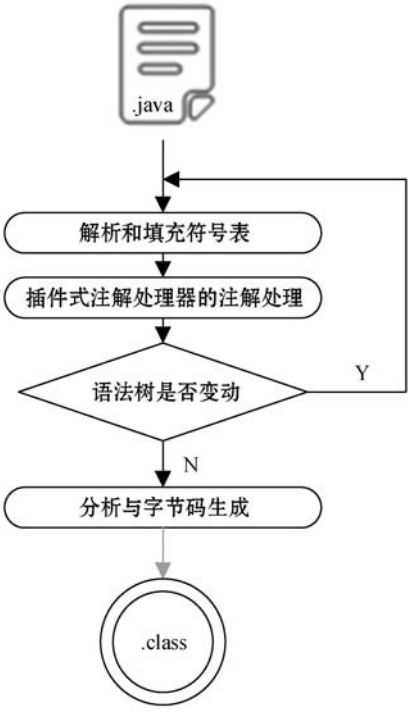

图 1 Java 语言的编译过程

JSR 269 是从 Java 6 版本开始纳入的一个 JSR(Java Specification Request)规范,全称是插件式注解处理器(Pluggable Annotation Processing API)。JSR 269 提供了一组插件式注解处理器的标准接口,可以在编译期间对注解进行处理。解析与填充符号表过程会将源码转换为一棵抽象语法树(Abstract Syntax Tree,AST)。每一个注解处理器可以理解为一个编译器插件,通过这个插件,可以读取、添加、修改、删除抽象语法树中的任意元素。而如果这些插件在注解处理期间,对语法树进行了修改,编译器将回到解析及填充符号表的过程重新处理,直到所有注解处理器都没有再对语法树进行修改后达到不动点。通过这组标准 API,我们可以干涉编译器的行为,让其编译结果为安全多方计算的执行代码。

2.2 安全多方计算编译器架构

将 Java 语言编译为 MPC 应用有两个难点:编译器如何区分 MPC 代码和非 MPC 代码以及如何保证 MPC 代码和非 MPC 代码的交互性。为了解决这些难点,在本文中,MPC 应用的编译单元是经过标注的类方法,并且为了保证与 Java 语言的交互性,编译的结果也必须是一个合法的 Java 类文件,所以在编译的过程中,保留原有类方法的签名信息。也就是说,MPC 应用的方法的类型系统满足:

$$\frac{\Gamma \vdash M : X \to Y}{\Gamma \vdash compile(M) : X \to Y}$$

即在执行环境 Γ 中方法签名为 $X \to Y$ 的方法 M 编译为 MPC 方法之后仍具有 $X \to Y$ 的方法签名。

将类方法作为编译单元可以有如下非常符合编程习惯的隐含信息:

1)方法的参数作为各方的隐私数据输入。

2)方法的参数个数为参与方个数。

3)方法的返回值为执行结果。

本文编译器基于 JSR 269 规范,对 Java 源代码编译的过程进行修改,将通过 MPC 注解标注的方法抽象语法树节点编译为电路等表示形式,并在类结构里添加网络、角色等配置信息用于安全计算执行引擎的实例化。总体架构如图 2 所示,其中:MPC 注解用于标注 Java 源码,表明其中被标注的类方法需要被编译为电路等表示形式,并提供配置信息;JSR 269 插件用于对标注的 Java 源码进行抽象语法树层面的变换和编译;经过标注的 Java 源码编译后生成一个 MPC 类,其中包含了被标注的方法编译成的电路等表示形式的字段和配置信息字段;电路等表示形式将由底层安全计算执行引擎运行,完成安全多方计算的功能;而原生的没有经过标注的 Java 类(例如应用的启动类)可以与 MPC 类进行交互,运行于 Java 虚拟机上。电路等表示形式可以采用常见的 Bristol 格式[11]、PCF 格式[12]等,或者采用底层安全计算执行引擎支持的格式。

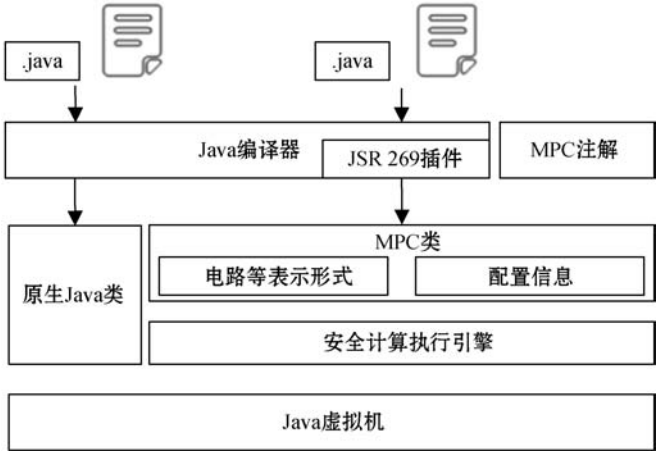

图 2 安全多方计算编译器架构

3 编译器详细设计

本文提出的安全多方计算编译器的主要组件包含 MPC 注解、基于 JSR 269 的注解处理插件、MPC 类结构设计和安全计算执行引擎实现几个部分。其中:MPC 注解用于标注需要转换为电路等表示形式的 Java 类方法,同时提供网络配置、角色等信息;基于 JSR 269 的注解处理器用于修改编译生成的抽象语法树,通过遍历类方法节点的形式,将方法体编译为电路等表示形式,同时提取注解里提供的网络配置、角色等信息作

为底层安全执行引擎的执行配置;MPC 类结构设计描述了编译之后类文件的结构,其中电路等表示形式作为类的静态字段,为保证交互性,保留原 Java 方法的签名,并将方法体替换为底层安全计算执行引擎的执行。

3.1　MPC 注解

MPC 注解包含三个部分:MPC 应用注解、MPC 类注解、MPC 方法注解。其中:MPC 应用注解和 MPC 类注解只能作用于类型声明;MPC 方法注解只能作用于方法声明。这些注解都只在编译期保留,而不保留在编译后的类文件中。

3.1.1　MPC 应用注解

MPC 应用注解(@ MPCApplication)用于标注该类型是一个 MPC 应用,并提供底层安全计算执行引擎所需的网络配置、角色等信息,用于在编译时自动生成配置提供给安全计算执行引擎,如表 1 所示。

表 1　@MPCApplication 注解字段

字段	含义
name	MPC 应用的名称
role	MPC 应用的角色,可以作为服务端(Server)或者客户端(Client)
serverInfo	服务端网络配置,包含网络地址和端口号;地址默认值为 localhost,端口默认值为 2107
clientInfo	客户端网络配置,包含网络地址和端口号;地址默认值为 localhost,端口默认值为 2108

3.1.2　MPC 类注解

MPC 类注解(@ MPCClass)用于标注该类型是一个 MPC 类,其类结构会经过编译器插件处理后进行变换。MPC 类注解作为注解处理器的注册入口,并且必须和 MPC 应用注解同时存在才会进行后续的 MPC 方法的编译流程。在 MPC 类注解中,可以设置安全计算执行引擎后端的类型,可提供多种后端让用户选择。

表 2　@MPCClass 注解字段

字段	含义
backend	安全计算执行引擎后端的类型

3.1.3　MPC 方法注解

MPC 方法注解(@ MPCMethod)用于标注该方法是一个 MPC 方法,其方法体会被编译为电路等表示形式。此注解主要用于将 MPC 方法和原生 Java 方法区分开来,没有字段信息。在一个 MPC 应用中,至少需要包含一个被 MPC 方法注解标注的方法。

3.2　MPC 注解处理器

MPC 注解处理器的注册入口是 MPC 类注解,如果类型定义上包含了该注解,则会在编译的过程中对抽象语法树进行变换,将被 MPC 方法注解标注的方法编译为电路等表示形式,并作为类的一个静态字段;同时提取 MPC 应用注解的配置信息;最后替换方法体为安全计算执行引擎的执行。

如图 3 所示,MPC 注解处理器主要流程为:

1)判断类节点是否包含 MPC 应用注解和 MPC 类注解,如果都包含则进行后续操作。

2)提取 MPC 应用注解中的配置信息,将其作为一个类字段。

3)遍历类的方法定义节点,找到包含 MPC 方法注解的方法节点:

(1)对方法节点进行树遍历,将方法体里包含的二元操作符、赋值、条件判断、循环等语句转换为电路等表示形式,在这个过程中,进行常量传播、公共子表达式消除等优化手段来优化生成的代码,最后将方法返回语句替换为返回安全计算执行引擎的执行结果(将执行结果转换为方法返回值的类型)。

(2)将编译结果(电路等表示形式)作为一个静态字段,添加到类节点的定义中。

(3)将方法的方法体替换为安全计算执行引擎的执行,执行引擎通过 MPC 应用注解中的配置信息与其他参与方连接,进行例如秘密共享、结果揭示等操作。

4)输出修改后的抽象语法树。

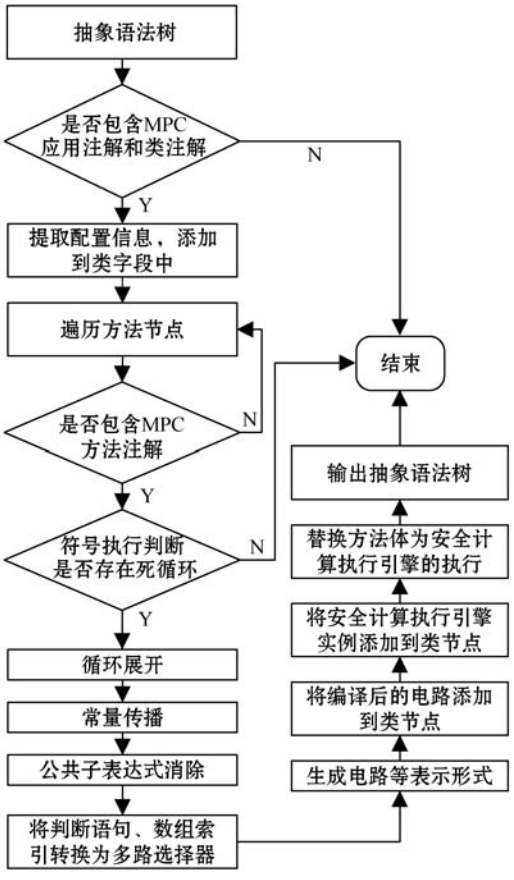

图 3　MPC 注解处理器流程

在对方法节点进行编译的过程中,受限于 MPC 应用程序必须可停机、电路表示形式无状态、需要防侧信道攻击等的限制条件,对于编译的程序有如下的限制:

1) 方法参数类型和返回类型必须是 Java 原始类型或者原始类型构成的组合类型、数组。

2) 程序不能有死循环。

3) 不可将隐私数据作为循环变量,也就是说循环的次数是确定性的。

在实践中,通过以下手段可以尽可能避免、解决这些限制条件,从而减少对上层语言的抽象能力的削弱:

1) 通过符号执行[13],可以判断程序中是否包含死循环,以此规避编译包含死循环的程序;如果检测到程序包含死循环,则输出错误信息。

2) 对于循环结构,在编译时进行循环展开,从而避免循环变量这个状态,将循环转化为确定性、扁平化的代码流程。此过程会增加编译后的代码体积。

3) 对于判断结构,先通过数据流分析得到两个分支的数据修改列表,分别执行两个分支,然后通过多路选择器得到数据的值 $d_{\text{ata}} = mux(c_{\text{ondition}}, d_{\text{ata}_{\text{true}}}, d_{\text{ata}_{\text{false}}})$,其中:$mux$ 为多路选择器;c_{ondition} 为条件值;d_{ata} 为执行后的数据值,$d_{\text{ata}_{\text{true}}}$ 为 true 分支的数据值,$d_{\text{ata}_{\text{false}}}$ 为 false 分支的数据值。

4) 对于使用私有数据作为索引的数组访问,同样使用多路选择器进行数据的选取。

3.3 MPC 类结构设计

为了保证和 Java 语言的交互性,对于编译后的 MPC 类需要保证其同时也是一个合法的 Java 类文件。一个 Java 类的对外接口是其中的公开方法,因此,为了不影响原有方法的语义,保留原方法的签名;对于配置信息、电路等表示形式、安全计算执行引擎,无须对外开放,因此作为 MPC 类的私有字段;其他信息如类名、未被标注的方法等保持不变。因此,经过 MPC 注解处理器处理之后,一个 MPC 类相较于原生 Java 类包含如下的结构变化:

1) 新增一个私有的配置信息字段,包含网络、角色等信息。

2) 新增一个私有的安全计算执行引擎实例字段。

3) 新增 MPC 方法对应的编译后的电路等表示形式作为类的私有字段。

4) 保留原方法签名,替换修改 MPC 方法的方法体为安全计算执行引擎的执行。

3.4 安全计算执行引擎

安全计算执行引擎的实现和电路的表示形式相关,本文提出的编译器设计并不限制电路表示形式和安全计算执行引擎的实现。编译器可以根据 MPC 类注解的配置信息将 Java 代码编译为不同后端。例如可以基于 SCAPI[14] 来执行算术电路;基于 ABY 框架来执行混合协议电路;基于 Fresco 框架[15] 执行 Bristol 格式的布尔电路。

安全执行引擎在执行前,需要根据 MPC 应用的角色处理隐私数据的输入,对本参与方输入数据进行例如线性秘密共享、OT 传输等操作,同时忽略在方法参数中传递的应该由其他参与方输入的数据。

安全计算执行引擎的执行可以分为三个阶段:

Setup 初始化阶段。根据的底层协议的不同,初始化阶段首先会生成可能需要用到的本地运算资源(例如乘法三元组),然后通过线性秘密共享或者 OT 将隐私数据分发给其他参与方。初始化阶段结束后,除了生成的本地运算资源,参与方 p_{arty_i} 会将子秘密 $s_{\text{hares}_i} = share(p_{\text{arams}_i})$ 分享给其他参与方,其中 p_{arams} 为方法参数列表。

Execute 执行阶段。在执行阶段,安全计算执行引擎解析编译之后的电路等表示形式,根据底层协议对电路进行执行。

Open 结果揭示阶段。当协议执行完毕,执行引擎会通过秘密重建等手段对结果进行揭示。揭示阶段结束之后,参与方 p_{arty_i} 将会获得具有类型 r_{type} 的结果,其中 r_{type} 为方法的返回类型。

基于上述的流程,我们对于安全计算执行引擎做了进一步的抽象,一个安全计算执行引擎如果要适配本文的方案,需要实现以下三个方法:

1) List<Share> init(MPCConfig config, Object ... params)初始化方法。根据输入的配置参数 c_{onfig},对用户输入的参数 p_{arams} 进行秘密共享等初始化操作,如果有本地计算资源,也在这部分生成;返回初始化后的秘密数据列表。

2) Share compute(byte[] code, List<Share> shares)安全计算方法。解析形式为字节数组的电路等表示形式 c_{ode},然后根据解析结果在初始化后的秘密数据 s_{hares} 上进行安全多方计算;返回计算结果(仍是秘密数据形态)。

3) <T> T open(Share result)结果揭示方法。对计算结果 r_{esult} 进行秘密重建等揭示操作,并将其转换为返回结果的类型。

其中秘密数据的表示形式 Share 需要根据安全计算执行引擎的不同实现如下的方法:

1) Share(Role role,Object data)秘密数据构造方法。根据输入的角色 r_{ole}对数据 d_{ata}进行秘密共享等操作;返回该数据的秘密形式。

2) byte[ ] open( )秘密数据揭示方法。对该秘密数据进行揭示,获得其明文;返回字节数组形式的结果,后续进行进一步的类型转换。

除此之外,提供一个 MPC 类调用的接口,以屏蔽、解耦底层安全计算执行引擎的实现,其实现大致如下:

```
<T> T execute(byte[ ] code, Object… params){
    List<Share> shares = init(this.config, params);
    Share result = compute(code, shares);
    return open(result);
}
```

基于上述的抽象,绝大多数的安全计算执行引擎均可适配到本文方案中,并且可以实现 MPC 方法的编译结果和底层安全计算执行引擎的解耦,如果两个底层安全计算执行引擎支持相同的电路等表示形式,还可以实现彼此的无缝切换而无须重新编译。

4　实验及结果分析

针对上文描述的编译器架构设计,基于 Java 8 的 javac 编译器实现了 MPC 相关的注解处理器插件,对本文提出的编译器进行了实现和实验,其中安全计算执行引擎基于 Fresco。因为 Fresco 本身提供对电路更高层的抽象,我们使用自定义字节码来表示 Fresco 中的操作。实验中,我们对“百万富翁问题”进行安全两方计算,MPC 方法包含两个输入参数,第一个参数为服务端输入,第二个参数为客户端输入。实验环境为 Ubuntu 20.04 LTS 操作系统,JDK 版本为 1.8.0_191,网络为本地无线局域网。

4.1　编　译

基于上述的编译器架构与实现,我们通过非常简约的 Java 代码实现了“百万富翁问题”:

```
@MPCApplication(name = "Millionaire", role = SERVER)
@MPCClass(backend = FRESCO)
public class Millionaire {
    @MPCMethod
    public boolean compare(int a, int b) {
        return a > b;
    }
}
```

在编译时添加我们编写的注解处理器插件(也就是 mpc.jar 中的 MPCProcessor 类):

```
javac -cp ./mpc.jar -processor MPCProcessor Millionaire.java
```

经过 MPCProcessor 注解处理器的处理,会将抽象语法树中 compare 方法的方法体(即一个返回二元比较操作结果的返回语句节点)编译为自定义字节码,并添加 MPC 相关的配置信息、安全计算执行引擎实例等字段。编译后的等价代码为:

```
public class Millionaire {
    private MPCConfig config = new MPCConfig(SERVER, serverInfo, clientInfo);
    private MPCExecutor executor = new MPCExecutor(config);
    private static final byte[ ] CODE = new byte[ ]{0, 0, 0, 1, 23, 33};
    public boolean compare(int a, int b) {
        return this.executor.execute(CODE, a, b);
    }
}
```

其中,config 为从 MPC 应用注解中提取的配置信息;excutor 为安全计算执行引擎实例;CODE 为 compare 的方法体编译之后的字节码,等价于以下伪代码:

```
a = load(0);                          //加载第 0 个参数
b = load(1);                          //加载第 1 个参数
result = GreaterThan(a,b);            //比较两个参数的大小
return result;                        //返回执行结果
```

4.2　运　行

编译之后的结果为一个合法的 Java 类文件,我们可以将其当作普通的类进行实例化和方法调用。我们分别在不同的进程创建了服务端和客户端的 MPC 应用实例,然后输入数据进行运算。

在服务端代码中,我们输入富翁 A 的金额 a(也就是第一个参数)为 100,因为不属于本参与方输入的参数数据会被忽略,所以我们可以传递任意值给这个参数,这里我们设置第二个参数的值为 0。

```
Millionaire server = new Millionaire( );
boolean result = server.compare(100, 0);
```

同样地,在客户端代码中,我们输入富翁 B 的金额 b(也就是第二个参数)为 200,并且设置第一个参数的值为 0。

```
Millionaire client = new Millionaire( );
boolean result = client.compare(0, 200);
```

分别启动、运行服务端和客户端(没有第三方)之后,双方得到了比较 100>200 的结果(即 result 的值):false。在富翁 A 看来,除了输入的隐私数据 100 和结果 false,没有获得关于富翁 B 金额的任何信息;同样地,富翁 B 能获取的数据也只有自己的输入 200 和比较的结果 false。由此完成了安全多方计算下双方金额的比较,并且没有泄露任何信息。

4.3 对 比

参考文献[1]的评估方法,我们分别使用 ABY、Fresco、ObliVM 实现了计算“百万富翁问题”、两数乘积、向量内积、交叉表求平均的 MPC 应用,与本文的方案进行了比较。

图 4 为实现上述 MPC 应用所需的代码行数(包含逻辑实现、运行配置的代码),可以看出,本文提出的编译器框架可以更少的代码行数实现相同功能的 MPC 应用;而从表 3 可以看出,相比于其他框架,本文方案无须进行手动的配置,基于现有的 Java 语言而非电路或者 DSL 来编写 MPC 应用,且可与 Java 语言进行自动化的原生交互,在易用性、交互性上都比其他框架更具优势。

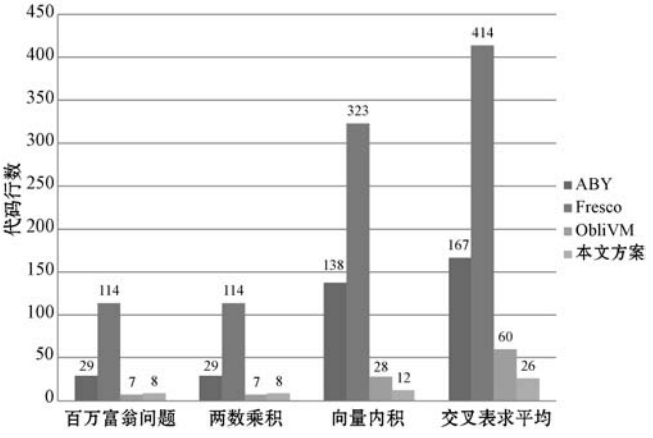

图 4 代码行数对比

表 3 ABY、Fresco、ObliVM 与本文方案的对比

指标	ABY	Fresco	ObliVM	本文方案
抽象程度	电路	电路	DSL	Java 语言
配置方式	手动配置	手动配置	手动配置	自动生成
交互性	手动类型转换	手动类型转换	无	自动类型转换
正确性	正确	正确	正确	正确

5 结 语

本文针对现有安全多方计算编译器易用性较差、难以与现有编程语言交互的难题,提出基于 JSR 269 的 Java 安全多方计算编译器,可以将 Java 方法编译为 MPC 应用,并且与原生 Java 程序保留良好的交互性,简化了用户开发 MPC 应用的流程,极大地降低了 MPC 应用开发的工程难度。

参 考 文 献

[1] Hastings M, Hemenway B, Noble D, et al. Sok: General purpose compilers for secure multi-party computation[C]// IEEE Symposium on Security and Privacy, 2019: 1220 – 1237.

[2] Yao A C. Protocols for secure computations[C]//23rd Annual Symposium on Foundations of Computer Science, 1982: 160 – 164.

[3] Pawlak R, Noguera C, Petitprez N. Spoon: Program analysis and transformation in java[EB/OL]. [2021 – 03 – 10]. https://inria.hal.science/inria-00071366v1/document.

[4] Malkhi D, Nisan N, Pinkas B, et al. Fairplay—A secure two-party computation system [C]//13th Conference on USENIX Security Symposium, 2004.

[5] Franz M, Holzer A, Katzenbeisser S, et al. CBMC-GC: An ANSI C compiler for secure two-party computations[C]//International Conference on Compiler Construction, 2014: 244 – 249.

[6] Mood B, Gupta D, Carter H. Frigate: A validated, extensible, and efficient compiler and interpreter for secure computation[C]//IEEE European Symposium on Security and Privacy, 2016: 112 – 127.

[7] Demmler D, Schneider T, Zohner M. ABY—A framework for efficient mixed-protocol secure two-party computation[C]// Network and Distributed System Security Symposium, 2015.

[8] Bucher N, Demmler D, Katzenbeisser S, et al. HyCC: Compilation of hybrid protocols for practical secure computation[C]//ACM SIGSAC Conference on Computer and Communications Security, 2018: 847 – 861.

[9] Zhang Y H, Steele A, Blanton M. PICCO: A general-purpose compiler for private distributed computation[C]//ACM SIGSAC Conference on Computer & Communications Security, 2013: 813 – 826.

[10] Liu C, Wang X S, Nayak K. ObliVM: A programming framework for secure computation[C]//IEEE Symposium on Security and Privacy, 2015: 359 – 376.

[11] David A, Steve L, Pieter M, et al. Bristol format[EB/OL]. [2021 – 03 – 10]. https://homes.esat.kuleuven.be/~nsmart/MPC/.

[12] Kreuter B, Mood B, Shelat A, et al. PCF: A portable circuit format for scalable two-party secure computation[C]// 22nd USENIX Conference on Security, 2013: 321 – 336.

[13] 李舟军,张俊贤,廖湘科,等. 软件安全漏洞检测技术[J]. 计算机学报, 2015, 38(4): 717 – 732.

[14] Ejgenberg Y, Farbstein M, Levy M, et al. SCAPI: The secure computation application programming interface [EB/OL]. [2021 – 03 – 10]. https://eprint.iacr.org/2012/629.pdf.

[15] Fresco: A framework for efficient secure computation[CP/OL]. [2021 – 03 – 10]. https://github.com/aicis/fresco.