心於至善

基于动态安全策略的Android应用程序隐私保护机制研究

宋睿

东南大学

学校代码: 10286
分 类 号: TP309.2
密 级: 公开
U D C: 004.49
学 号: 170815

東南大學
碩士学位论文

基于动态安全策略的 Android 应用程序隐私保护机制研究

研究生姓名: 宋睿
导 师 姓 名: 宋宇波

申请学位类别 工学硕士 学位授予单位 东南大学
一级学科名称 网络空间安全 论文答辩日期 2020 年 5 月 30 日
二级学科名称 学位授予日期 2020 年 6 月 20 日
答辩委员会主席 吴 蒙 评 阅 人 聂长海
秦中元
2020 年 6 月 4 日

東南大學

硕士学位论文

基于动态安全策略的 Android 应用程序隐私保护机制研究

专 业 名 称: 网络空间安全

研究生姓名: 宋 睿

导 师 姓 名: 宋 宇 波

RESEARCH ON PRIVACY PROTECTION MECHANISM OF ANDROID APPLICATION BASED ON DYNAMIC SECURITY STRATEGY

A Thesis Submitted to

Southeast University

For the Academic Degree of Master of Engineering

BY

SONG Rui

Supervised by

A. Prof. SONG Yu-bo

School of Cyber Science and Engineering

Southeast University

June 2020

东南大学学位论文独创性声明

本人声明所呈交的学位论文是我个人在导师指导下进行的研究工作及取得的研究成果。尽我所知，除了文中特别加以标注和致谢的地方外，论文中不包含其他人已经发表或撰写过的研究成果，也不包含为获得东南大学或其它教育机构的学位或证书而使用过的材料。与我一同工作的同志对本研究所做的任何贡献均已在论文中作了明确的说明并表示了谢意。

研究生签名：_____ 日 期：_____

东南大学学位论文使用授权声明

东南大学、中国科学技术信息研究所、国家图书馆、《中国学术期刊（光盘版）》电子杂志社有限公司、万方数据电子出版社、北京万方数据股份有限公司有权保留本人所送交学位论文的复印件和电子文档，可以采用影印、缩印或其他复制手段保存论文。本人电子文档的内容和纸质论文的内容相一致。除在保密期内的保密论文外，允许论文被查阅和借阅，可以公布（包括以电子信息形式刊登）论文的全部内容或中、英文摘要等部分内容。论文的公布（包括以电子信息形式刊登）授权东南大学研究生院办理。

研究生签名：_____ 导师签名：_____ 日 期：_____

摘　要

近年来，消费者在利用移动智能终端进行日常工作和生活娱乐等活动的同时，也面临着个人隐私泄露的安全风险。作为市场占有率最高的移动操作系统，Android 系统所受到的恶意攻击和安全威胁也最为剧烈。一些 Android 应用向用户索取的系统权限远大于其实现功能所必须的权限范围，而这种对权限机制的滥用不仅仅来自攻击者开发的恶意程序，也来自合法厂商开发的所谓良性程序，因而无法通过已有的恶意应用检测方法来解决这一问题。此外，用户对隐私数据的定义和保护强度往往视具体需求和使用场景而定，而 Android 操作系统提供的基于系统权限的安全模型无法满足用户多变的个人隐私防护需求。用户面临的另一个难题是当他们拒绝提供必要的系统权限或个人信息时，应用程序的正常功能将会受到很大影响。

针对上述问题，本文提出了一种 Android 系统应用程序隐私保护机制，该机制能够根据用户的安全需求和使用场景对应用程序的危险权限访问及敏感接口调用行为进行动态限制和主动防护。本文首先提出了一种基于虚拟机字节码注入技术的 Android 应用程序权限访问控制方法，该方法能够根据用户的安全需求和使用场景生成虚拟机字节码形式的安全策略，并将其注入到 Android 应用涉及到危险权限请求和敏感数据访问的代码单元中。该方法能够在应用程序运行时根据用户安全需求和使用场景的变化实时调整安全策略，以实现动态的应用行为控制和隐私保护。为了保证用户对系统权限和敏感接口的限制不影响到应用程序的正常功能，并在此基础上保护用户的个人隐私，本文提出了一种基于差分隐私的应用数据脱敏方法。该方法对 Android 应用程序所有和服务器共享的数据流进行脱敏处理，能够在不影响应用程序正常的数据采集分析功能的同时切断数据与个人信息的关联，从而在应用程序的可用性和安全性之间实现平衡。本文的主要工作和创新点如下：

1. 提出了一种基于 Android 虚拟机字节码的安全策略注入和权限控制方法。该方法能够基于用户的特定安全需求生成实例化安全策略，并将安全策略以虚拟机字节码的形式注入到 Android 应用程序的涉及危险权限请求和敏感数据访问的代码单元中，以实现对 Android 应用程序危险行为的有效控制和主动防护。
2. 针对现有的字节码注入技术无法根据安全需求的变动而动态调整控制级别和屏蔽粒度，且在安全策略更新时需要重新注入字节码的问题，提出了一种基于 Java 反射技术的动态安全策略调整机制。该机制能够在安全需求发生变更时动态切换权限控制级别，而无需重新注入字节码或对应用程序进行重打包。
3. 针对限制必要的系统权限或敏感调用会影响应用程序的正常功能的问题，提出了一种基于差分隐私控制的隐私保护和数据脱敏方法，在确保隐私数据安全的前提下保护应用程序的可用性。该方法基于差分隐私的可证明隐私保证，并通过隐蔽

方法将差分隐私和高斯过程相结合，能在保证应用宏观数据统计特性正确性的同时避免泄漏用户个体的隐私信息，在可用性和安全性之间实现平衡。

4. 在上述方案和技术的基础上，设计并实现了一个 Android 应用程序隐私保护原型系统，该系统能够根据用户的安全需求对应用程序的危险行为进行字节码级的实时动态控制，并对应用于服务器的共享数据流进行差分隐私脱敏处理。测试表明该系统对 Android 应用程序进行的安全策略注入在 85.27% 的情况下不会对应用程序的健壮性造成影响，在此基础上能够对 85.33% 的敏感 API 调用和危险权限请求进行有效限制。与此同时，注入的安全策略对应用的启动时间的影响至多只有平均 0.635s，在应用程序重打包后仅增加了平均约 0.372% 的大小。

关键词： Android，隐私保护，安全策略，虚拟机字节码，差分隐私

Abstract

In recent years, consumers have been exposed to the security risks of compromising their personal privacy while using mobile smartphones for daily work and entertainment activities. As the mobile operating system with the highest market share, Android is also subject to the most malicious attacks and security threats. Some Android applications demand far more system permissions than what are necessary for their functionality, and this abuse of the permission mechanism comes not only from malicious applications developed by attackers, but also from so-called benign applications developed by legitimate vendors, thus cannot be addressed by existing malicious detection methods. In addition, the definition and intensity of privacy protection often depends on specific requirements and usage scenarios, and the security model provided by the Android operating system based on system permissions cannot meet users' variable personal privacy protection needs. Another dilemma faced by users is that when they refuse to provide the necessary system permissions or personal information, the normal functioning of applications will be greatly deteriorated.

In response to the above issues, we propose an Android application privacy protection mechanism which can dynamically restrict and proactively protect applications' dangerous permission access and sensitive interface invocation behavior according to users' security requirements and usage scenarios. In this thesis, we first propose an Android application permission access control method based on virtual machine bytecode injection technique, which can generate security strategy in the form of virtual machine bytecode and inject it into the code unit of applications involving dangerous permission requests and sensitive data access. The approach enables dynamic application behavior control and privacy protection by adjusting security strategies in real time while the applications are running. In order to ensure that users' restrictions on system permissions and sensitive interfaces would not affect the normal functionality of the applications and to protect users' personal privacy, we propose a differential-privacy-based application data desensitization method. The method desensitizes all the data streams shared by Android applications and servers, and is able to sever the association between data and personal information without affecting the normal data collection and analysis functions of applications, thus balancing the availability and security of the applications. The main work and innovations of this thesis are as follows.

1. A security strategy injection and permission control method based on Android virtual machine bytecode is proposed. The method is capable of generating instantiated security strategies based on user-specific security requirements and injecting the security strate-

gies in the form of virtual machine bytecode into the code units of Android applications involving dangerous permission requests and sensitive data access to achieve effective control and proactive protection against dangerous behavior of Android applications.

2. To address the problem that existing bytecode injection techniques cannot dynamically adjust the control level and shield granularity according to changes in security requirements, and that bytecode needs to be reinjected when security strategies are updated, a dynamic security strategy adjustment mechanism based on Java reflection technique is proposed. This mechanism dynamically switches the level of permission control when security requirements change without having to reinject bytecode or repackage the applications.
3. To address the problem that limiting necessary system permissions and sensitive calls would affect the normal functionality of applications, a privacy protection and data desensitization method based on differential privacy that protects the availability of applications while ensuring the security of private data is proposed. This method is based on a provable privacy guarantee for differential privacy and combines differential privacy with Gaussian Process through a disguised approach that can balance availability and security by ensuring the correctness of the statistical characteristics of the applied macro data while avoiding leakage of individual user privacy information.
4. Based on the above-mentioned schemes and techniques, a prototype Android application privacy protection system is designed and implemented, which is capable of real-time dynamic control of applications' dangerous behavior at the bytecode level according to users' security requirements, and perform differential privacy desensitization of the shared data streams with servers. Evaluation shows that the security strategies injected into Android applications would not affect the application robustness in 85.27% of cases, on the basis of which it is able to effectively restrict 85.33% of sensitive API invocations and dangerous permission requests. Meanwhile, the impact of injected security strategies on the startup time of applications is at best only 0.635s on average, increasing the size by about 0.372% on average after applications are repackaged.

Keywords: Android, Privacy Protection, Security Strategy, Virtual Machine Bytecode, Differential Privacy

目　录

插图目录

表格目录

术语与符号列表

ABI	Application Binary Interface
ANN	Artificial Neural Network
AOT	Ahead-of-time Compilation
API	Application Programming Interface
APK	Android Package
AR	Activation Record
ART	Android Runtime
DEX	Dalvik Executable
DOM	Document Object Model
DP	Differential Privacy
DVM	Dalvik Virtual Machine
ECA	Event-condition-action
GP	Gaussian Process
GUI	Graphical User Interface
ID3	Iterative Dichotomiser 3
IP	Instruction Pointer
IPC	Inter-process Communication
JIT	Just-in-time Compilation
JVM	Java Virtual Machine
MLP	Multilayer Perceptron
OEM	Original Equipment Manufacturer

PC	Program Counter
PCA	Principal Component Analysis
PE	Policy Executor
SDC	Security Decision Center
SDK	Software Development Kit
SGP	Sparse Gaussian Process
SMS	Short Message Service
SSE	Sum Square Error
SSG	Security Strategy Generator
URL	Uniform Resource Locator

第一章 绪论

1.1 研究背景

随着集成电路技术的持续高速发展，计算设备的尺寸得以一再缩小。近些年，智能手机、平板电脑以及其他的一些可穿戴设备相继出现，这些设备的计算性能不断跃升，并被广泛使用。这些设备已经可以替代以往只有计算机才能实现的大部分功能。根据中国互联网中心于 2019 年发布的第 44 次《中国互联网发展状况统计报告》显示，截止到 2019 年 6 月，中国网民规模达到 8.54 亿人，互联网普及率达到 61.2%。其中值得关注的是，网民使用手机上网的比例达到 99.1%。而与 5 年前相比，移动网络平均下载速率提升了 6 倍，用户月均使用的移动流量高达 7.2GB，为全球平均水平的 1.2 倍[1]。图 1-1 和图 1-2 显示了近几年智能手机网民增长的具体态势。

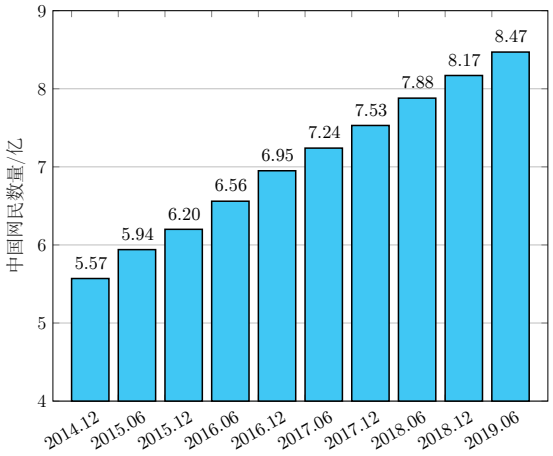


图 1-1 近几年中国手机网民数量

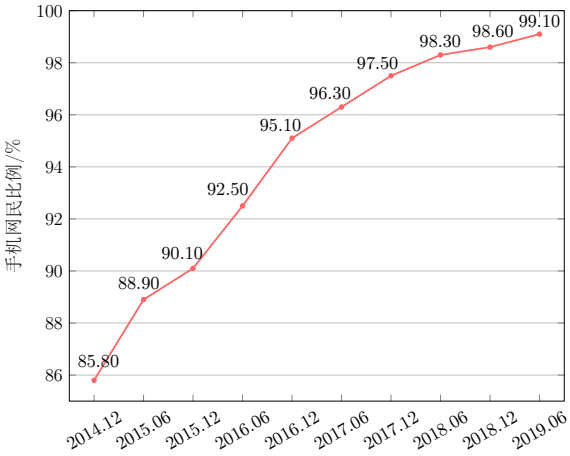


图 1-2 近几年中国手机网民比例变化趋势

人们的设备使用习惯正在以惊人的速度发生剧变。人们逐渐依赖智能手机来进行即时通信、社交活动、电子商务、网络购物和在线教学等活动。在这个过程中，手机不可避免的会越来越多地接触到用户的敏感信息和个人隐私，包括但不限于用户的支付密码、短信记录、移动轨迹和通讯记录等。这种和个人信息高度捆绑的特性开始让人们重新审视智能手机的安全性，特别是智能手机应用的安全性。恶意攻击者在近些年开始逐渐把目光投向智能设备，正是因为以智能手机为代表的智能设备中包含了海量的个人隐私，这些隐私在某种意义上处于不设防的状态。

在众多的智能设备中，Android 系统拥有最多的装机量，这主要是因为 Android 系统作为一个基于 Linux 的开源操作系统，拥有无与伦比的开放性和可定制性。截止到 2019 年第 4 季度，在所有上市的智能设备中 Android 的市场占有率高达 86.6%，并且仍在稳

定增长[2]。然而 Android 的繁荣也同样带来了安全隐患，Android 操作系统的诸多软件设计上和商业模式上的特性决定了这个操作系统同样将会收到攻击者的青睐。当我们考察 Android 操作系统的安全机制时，我们注意到了以下几个关键的薄弱环节。

首先是 Android 系统的碎片化问题给该系统的安全防护带来了巨大的障碍。考虑到系统漏洞是对现代计算机程序最严重的安全威胁之一，在理想情况下系统设计者应该在发现漏洞时尽全力修补它们。不幸的是，Android 操作系统的设计者和维护者 Google 公司采用的补丁策略无力解决这一重要问题。Google 官方发布的系统补丁只优先供给 Android 的最新发行版本，而通常并不承诺乃至拒绝前向兼容到旧版本的系统[3]。此外，由于系统的高度定制性，各个 OEM 厂商通常也会提供各自的定制 Android 发行版，这些发行版基于不同的 Android 版本，并且接受不同的更新策略，这给补丁的及时应用带来了另一重困难。因此，Android 生态迫切地需要对数量庞大的古旧 Android 版本设备提供补丁支持，允许设备提供商和用户主动地修复系统漏洞，而不是被动地等待各级系统提供商发布更新补丁。

其次，Android 采用了基于权限的安全模型来保护用户的敏感资源和个人隐私。然而这种通用的模型并不能满足某些特定的安全需求。一些个人和组织出于使用环境或场景的考虑，有强烈动机需要对特定应用程序执行定制的安全防护策略[4][5]。比如说，一些用户或者组织可能希望在某些场景下限制特定应用程序和网络服务器的通信行为，或者限制某些应用程序对设备定位信息的访问。这样的需求是广泛存在且完全合理的，但是它们超出了 Android 标准安全模型的能力范围，仅依靠 Android 的安全架构是完全无法满足这一需求的。为了满足不同用户和组织的高度多样化的定制需求，Android 系统迫切地需要一个更加灵活的安全模型。

第三，和另一个主要移动设备操作系统 iOS 不同，Android 并不强制要求统一的应用程序发布和审查机制。由 Apple 设计并维护的 iOS 操作系统及其衍生系统 iPad OS，其所有合法应用程序都必须来自其官方的应用市场 App Store。而一个应用程序想要在 App Store 上架，就必须接受严格的安全审查。Android 系统则与此不同，设备可以接受来自各个第三方应用市场乃至其他未审查来源的应用程序。这些来源迥异的应用程序大部分并没有经过充分有效的审核流程就被安装到了用户的设备上，这给 Android 系统用户带来了巨大的安全风险。有数据显示，Android 系统恶意软件占所有移动恶意软件中的 97%[6]。更糟糕的是，即使是 Google 官方应用市场 Google Play 中也存在相当高比例的恶意应用程序[7][8]。

针对上述安全漏洞和恶意攻击方法，已经有很多研究者提出了各种分析和检测 Android 恶意应用的方案，并且开发出了许多技术和工具[9][10][11]。研究者们通常采用静态代码分析或动态运行时检测等手段对恶意应用软件进行检测与甄别[7]。静态检测通常通过分析 Android 应用程序的 AndroidManifest.xml 文件来考察应用程序的权限调用情况，或者逆向得到应用程序的源代码，并对代码的执行逻辑进行分析。然而考虑到现代 Android 应用程序的软件规模与日俱增，同时代码混淆和加固技术被广泛应用于软件发

布和打包过程，静态源代码分析的甄别能力极其有限。相对地，动态分析方法能够从程序执行逻辑层面考察应用程序的行为，因而对甄别恶意程序有更强的能力。动态分析通常可以采用构建独立检测环境并将应用程序置于该环境中，通过污点分析、黑盒测试等方式对应用程序进行分析和测试[12]。然而，恶意攻击者的攻击手段也在与时俱进。一些攻击者设计的恶意软件拥有这样的能力，能够检测应用程序的运行时是否处于被监控环境或仿真器环境下。倘若恶意应用检测到了类似的环境，就会使其恶意逻辑失活从而逃脱动态运行时检测[13]。

考虑到以上问题，Android 操作系统及其应用程序迫切地需要一种用户自主可控的权限控制和安全策略实施方法，该方法能够有选择地限制侵犯用户个人隐私的行为，并能够根据场景需求动态地调整控安全策略。通过这种权限策略控制，用户和组织能够根据需求有选择地调整应用程序的权限授予和使用，并能够动态地给予和收回权限。这样，即使用户使用未经过严格审查的应用程序，也可以通过权限控制的方法规范应用程序的数据和隐私访问行为，从而保护用户和组织的隐私信息。

1.2 研究目的和意义

本文的目的是考察 Android 应用程序的权限控制方案，提出一种用户自主可控的权限控制和安全策略实施方法。该方法基于 Android 的虚拟机字节码注入技术，在目标应用程序涉及敏感操作和危险权限调用的方法单元中添加安全策略执行代码，并通过 Java 反射特性动态地调用安全功能。为了实现安全策略的动态生成和执行，添加的安全策略可执行代码可以根据动态策略触发与执行。

此外，考虑到一些针对特定权限调用的限制可能会影响程序的正常功能，然而这类数据又涉及到用户的个人隐私，因此需要对一些需要网络通信和向应用服务器传输的数据进行脱敏处理。这样能够保证在不影响应用程序正常功能的前提下保护用户的个人隐私。为了实现这个目标，本研究将考察基于差分隐私的应用程序数据脱敏方法，以及如何将该技术和上述安全策略执行机制相结合，从而在应用程序的可用性和用户隐私的安全性之间实现平衡。

基于上述系统理论和模型，本文还将实现一个 Android 应用程序的动态策略权限控制系统。该系统结合了 Android 虚拟机字节码注入技术和 Java 反射技术，通过安全策略动态生成权限控制方案，并且通过差分隐私技术对服务器通信数据进行脱敏处理，从而实现对 Android 应用程序在多场景多环境下的权限动态控制和数据保护，从而有效保护用户和组织的隐私信息。

1.3 国内外研究现状

Android 移动操作系统的安全性构建在以权限系统为基础的敏感资源访问模型上，这些敏感资源包括但不限于位置信息、相机传感器、麦克风、通讯录和 SMS 信息等。通

常，若一个 Android 应用程序企图获取对某个敏感资源的访问权限，就必须在 Android-Manifest.xml 清单中显式地声明所需的权限。而用户可以选择授予或者不授予该权限。但是，这样的权限系统正在被一些应用程序滥用，这些应用程序声明的权限远多于实现必要功能所必须的权限[14]。Felt 等人的研究考察了应用市场上的相当一部分应用程序，并发现大量应用程序都存在该问题[15]。更糟糕的是，每当操作系统或者应用程序更新时，这些恶意的应用程序都可以通过更新累积的方式进行特权升级，从而在静默状态下转换成恶意软件[16]。

从 Android 的第一个版本到 Android 6.0，Google 不断地为 Android 的权限系统添加新的权限和权限组[13]。这看似是对 Android 的安全性做了进一步增强，但事实上无助于解决安全问题。因为权限的增加也同时意味着恶意程序对 Android 权限系统的攻击面的扩大。此外，由于 Android 操作系统严重的碎片化现状，尽管 Android 操作系统已经更新至 Android 10.0 版本，截止到 2020 年 1 月，仍有 89.6% 的 Android 用户在使用早于 Android 9.0 的历史版本[17]。Android 发行版的碎片化和版本之间对权限系统的差异化兼容策略导致，当面对针对较新版本开发的应用程序时，使用古旧版本操作系统的设备无法预见并且决定如何回退并兼容这些应用程序的权限请求[18]。此外，权限和权限组的快速膨胀以及 Google 官方维护说明文档的滞后性使得用户很难理解每个权限请求的含义并陷入信息过载的困境中，从而轻易被恶意程序欺骗。

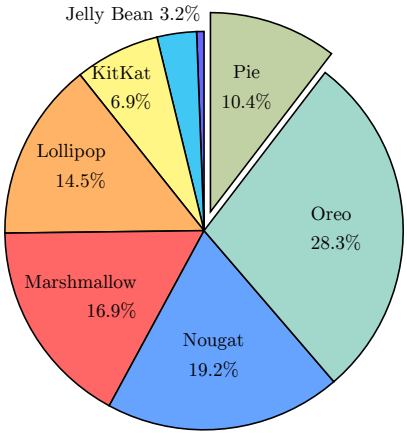


图 1-3 截止到 2020 年 1 月各 Android 系统版本市占率

针对上述困境，近年来相关领域研究者主要采用静态分析和动态分析两种思路来进行恶意软件检测与防护。静态分析方法主要利用静态代码分析的手段来检查应用程序是否包含异常的权限申请和信息流[19]。静态分析方法具有较好的可扩展性，可以从较大数量级的应用中快速筛选出可疑的应用程序。但是由于以下三点原因，静态分析方法始终无法准确甄别出恶意应用程序。

首先，作为一个面向普通消费者的商业级操作系统，Android 需要面对各种应用场景，因此必须使用事件驱动的设计思路，具体来说就是广泛利用生命周期钩子和 GUI 回

调处理来处理用户的实时请求。因此，运行时的权限申请控制和数据流并非总是可以通过静态代码分析检测出来，因为应用程序的执行流程取决于具体的运行时环境和用户行为[20]。

其次，代码和 AndroidManifest.xml 清单中存在某些权限声明并不总是意味着它们实际上在运行时真的使用了这些权限，也不能衡量在运行时这些应用程序如何使用所申请的权限。Enck 等人的研究表明，一些恶意应用程序通过对看似无害的权限或权限组进行不同频次的调用，同样也可以发动对用户隐私的攻击[21]。因此，简单地检查权限声明会导致静态分析的过程中出现大量误报或漏报。特别是，从 Android API 级别 23 开始，Android 操作系统新增了动态权限的支持，以便应用程序在运行时新增对权限的请求，并动态地获取和撤销权限。这使得静态分析在面对权限系统时几乎无计可施[22]。具体地，这种新的运行时权限机制意味着静态分析方法无法发现应用程序何时在运行时请求并被授予了异常的权限或权限组。并且如果用户在应用安装后手动赋予或撤销了敏感权限，则会进一步导致误报率和漏报率的大幅提升[23]。

第三，静态分析方法在检测通过动态代码构造（如通过 Java 反射机制调用敏感 API）所执行地恶意行为方面的能力极其有限。恶意攻击者通常能够轻易抓住这一缺陷构造攻击，采用代码混淆[24] 或代码异变[25] 的方法来误导静态检查机制。Suarez 等人的研究提出了以资源为中心的新的静态分析方法以克服上述的限制[26]，但是恶意程序仍然可以通过采用资源混淆的方法来规避以资源调用为中心的检测系统[27]。

相比之下，动态分析提供了检测和防护恶意应用程序的补充和完善方法。具体而言，常用的动态检测方法包括基于行为的恶意软件检测[28]、基于系统 API 调用追踪的程序行为建模[29][30] 以及基于资源利用的而已用程序行为分析[31]。在动态分析领域，机器学习技术越来越被广泛应用于区分恶意应用和良性应用。然而，当恶意应用对系统调用进行代码混淆时，基于系统调用分析的动态检测系统仍然有可能被规避[32]。此外，敏感的 API 调用和权限声明并不总是一定表明存在恶意意图，异常的资源使用也并不总是与异常的行为相对应。类似地，基于系统调用序列或者系统调用依赖的行为分析方法也很容易被恶意应用绕过，这些恶意应用可以通过模仿攻击[33] 或系统调用混淆[34] 来规避这些检测手段。

最近的一些研究对上述手段进行了扩展，利用了系统调用的统计数据[35] 和调用链[36] 来甄别恶意软件。Chen 等人的工作对 Android 应用程序的 API 调用进行了严格的区分并和 API 签名进行严格匹配，来防止系统调用混淆并区分良性应用和恶性应用。但是由于使用了底层的功能，尽管该方法能够防止系统调用混淆（即对系统调用签名进行重命名），却不能防止替换攻击。替换攻击这种攻击方式会使用语义上的等效变体替换系统调用中的依赖项[28]。此外，由于这种方法依赖于通过静态方式获取应用权限表，所以并不适用于那些采用了在新的 Android 版本中引入的动态权限机制的应用[37]。

综上所述，无论是静态检测还是动态检测方法，都存在固有的弊端，并不适用于全部的 Android 应用程序。此外，这些方法主要着眼于分类并甄别出恶意应用，而很少讨

论应该如何对甄别出的恶意应用的攻击行为进行防护。针对防护方面，一些已有的研究提出了基于模型的安全工具，这些安全工具能够根据需求生成安全策略，从而在程序运行时根据策略动态调整权限的授予与撤销。Neisse 等人提出了一种基于预先配置权限控制策略的动态权限控制方法，当 API 调用具有潜在的隐私威胁时，该策略可以设置提供的信息级别[18]。权限控制库始终处于活动状态并始终监视 API 调用，在存在相应调用时根据该策略调整提供的信息级别。他们设计的系统从程序结构、操作、威胁规则等方面设定策略。这种设计的弊端是无法检测程序在执行时处于哪一条执行流程，从而根据执行细节设定安全策略。

此外，也有一些研究在字节码级别上对需要提供安全性的方法单元位置添加安全功能。Lee 等人提出了一种使用应用程序包装技术的方案，该方案能在代码的方法上添加必要的安全功能[38]。这种方案在字节码级别添加了安全功能，而并不依赖 Android 的 Java 或 Kotlin 源代码。具体地，他们将所需的安全功能封装成字节码级别的 smali 代码，并在需要安全性的方法点上将这些代码注入到应用程序中。这种方法的缺点很明显，它是在静态策略的基础上运行的，这意味着每当更改安全策略时，都需要重新提取并注入安全代码，并重新包装应用程序。

1.4 研究主要工作

本文旨在研究 Android 操作系统中应用程序的隐私泄漏问题及其防护技术。如 1.3 节所述，本领域目前已有的研究主要着眼于通过静态和动态分析等手段对恶意应用程序进行甄别，并对甄别出的应用程序进行相应的封禁处理。这些已有的研究无法解决新的 Android 版本中增加的动态权限申请特性带来的识别困难问题，也无法解决利用反射特性等技术进行动态代码构造的恶意应用程序的识别问题。针对这些问题，本文提出了一种 Android 应用程序隐私保护机制，该机制能够根据用户的安全需求对应用程序的危险权限访问和敏感接口调用行为进行控制和防护，并对应用与网络服务器的通信数据流进行脱敏处理以保障应用程序基本功能的可用性。针对应用的敏感调用和权限访问行为，本文提出了一种基于虚拟机字节码的动态安全策略控制方法，该方法能够检测 Android 应用中涉及危险权限请求和敏感数据访问的代码单元，并将用户指定的安全策略以虚拟机字节码的形式注入到相应代码位置，依照用户的安全需求对应用程序的危险行为进行防护和控制，并根据用户所指定的安全策略的变更而动态的调整控制级别。针对应用程序与服务器的通信数据，本文提出了一种基于差分隐私的数据脱敏方法，该方法能够在保证应用程序功能的前提下对应用与服务器的通信数据进行脱敏处理，从而在应用程序的可用性和安全性之间实现平衡。和已有的一些研究相比，本文提出的机制能够根据用户需求动态地调整安全策略，以保证安全策略更新时无需重新注入安全代码并重新包装应用程序。此外，该机制的实现与 Android 系统版本和硬件条件完全无关，这意味着该机制能够作用于几乎所有 Android 设备搭载的应用程序，从而避免了 Android 碎片化带来的安全防护难题。具体地，本文完成的主要工作包括：

1. 研究了 Android 操作系统的基于权限控制的安全模型，对 Android 操作系统常见的涉及用户隐私的危险权限和敏感 API 进行了梳理。考察了 Android 的 Dalvik 虚拟机和 Android Runtime 运行时环境的代码编译与执行原理，以及由应用程序的 APK 包反编译得到虚拟机字节码和原始静态资源的相关技术。
2. 提出了一种基于 Android 虚拟机字节码的安全策略注入和权限控制方法。该方法能够基于用户的特定安全需求生成实例化安全策略，并将安全策略以虚拟机字节码的形式注入到 Android 应用程序的涉及危险权限请求和敏感数据访问的代码单元中，以实现对 Android 应用程序危险行为的有效控制和主动防护。
3. 针对现有的字节码注入技术无法根据安全需求的变动而动态调整控制级别和屏蔽粒度，且在安全策略更新时需要重新注入字节码的问题，提出了一种基于 Java 反射技术的动态安全策略调整机制。该机制能够在安全需求发生变更时动态切换权限控制级别，而无需重新注入字节码或对应用程序进行重打包。
4. 针对限制必要的系统权限或敏感调用会影响应用程序的正常功能的问题，提出了一种基于差分隐私控制的隐私保护和数据脱敏方法，在确保隐私数据安全的前提下保护应用程序的可用性。该方法基于差分隐私的可证明隐私保证，并通过隐蔽方法将差分隐私和高斯过程相结合，能在保证应用宏观数据统计特性正确性的同时避免泄漏用户个体的隐私信息，在可用性和安全性之间实现平衡。
5. 在上述方案和技术的基础上，设计并实现了一个 Android 应用程序隐私保护原型系统，该系统能够根据用户的安全需求对应用程序的危险行为进行字节码级的实时动态控制，并对应用于服务器的共享数据流进行差分隐私脱敏处理。测试表明该系统对 Android 应用程序进行的安全策略注入在 85.27% 的情况下不会对应用程序的健壮性造成影响，在此基础上能够对 85.33% 的敏感 API 调用和危险权限请求进行有效限制。与此同时，注入的安全策略对应用的启动时间的影响至多只有平均 0.635s，在应用程序重打包后仅增加了平均约 0.372% 的大小。

1.5 论文组织结构

本文共分为六章，从第二章开始各章的主要内容安排如下：

第二章介绍了与本文研究相关的一些理论与技术基础。本章首先介绍了 Android 操作系统的安全模型和以权限控制系统为核心的 Android 隐私保护机制，具体分析了 Android 操作系统的进程隔离机制和敏感权限控制策略。本章还介绍了 Android 操作系统中内嵌的 Dalvik 虚拟机和 Android Runtime 运行时环境，具体考察了 Android 应用程序的工程结构、编译方法、执行流程以及对 Android 应用程序进行反编译的技术基础，具体分析了以 smali 字节码为代表的虚拟机字节码的语法基础和运行逻辑，为接下来的字节码注入技术进行技术支持。

第三章介绍了基于权限控制和安全策略的动态权限控制方法和与之对应的权限控

制字节码注入技术。本章首先介绍了动态安全策略生成的理论基础和相关技术，并详述了生成安全策略的步骤与流程，阐述了动态安全策略的高覆盖率、动态性和健壮性等优点。接着，本章还介绍了基于虚拟机字节码的安全策略注入与执行机制，该机制能够使用字节码封装技术将安全功能添加到应用程序的权限敏感位置。通过与动态安全策略管理机制相结合，该系统能够在安全策略发生改变的情况下动态地切换权限控制级别，无需重新调整注入的代码或对应用程序进行重新打包。

第四章介绍了基于差分隐私的隐私保护和数据脱敏技术。本章首先介绍了差分隐私控制的数学原理，并详细阐述了基于随机化的差分隐私的技术基础及其可靠性。接着，本章介绍了基于高斯过程的差分隐私应用机制，该机制能够将差分隐私技术应用于移动应用数据交互场景中，并在该过程中有效保障数据的可用性和安全性。最后本章介绍了差分隐私技术如何作用于 Android 隐私保护系统，并展示差分隐私技术在对应用程序统计性数据进行可用性保护的前提下对隐私信息进行脱敏处理的具体方式方法。

第五章介绍了基于安全策略注入技术的 Android 应用程序隐私保护系统的实现与评估。基于前面几章介绍的理论与系统，本章将实现一个使用动态安全策略的基于虚拟机字节码注入技术的 Android 应用程序隐私保护系统，该系统能够将封装的安全策略注入到应用程序的危险权限或敏感 API 相关的方法单元中。为了评估该系统功能的完成度与健壮性，我们将该系统应用于从各大应用市场上随机挑选的应用程序中，详细测试该系统对各种类型的权限调用的控制性能，并对安全策略的可用性和健壮性进行评估。

第六章归纳并总结了全文。本章反思了该研究的不足与缺陷，并梳理了实验与研究过程中涌现的新想法和新思路，展望了相关领域的研究前景和实用价值。

第二章　相关技术研究

本章将首先介绍 Android 操作系统的安全机制，具体包括 Android 操作系统的进程隔离安全模型和以权限控制系统为核心的 Android 隐私保护机制，并将具体分析 Android 操作系统的敏感权限控制策略。接着，本章还将介绍 Android 操作系统中内嵌的 Dalvik 虚拟机和 Android Runtime 运行时环境，具体分析了 Android 应用程序的工程结构以及应用程序包的编译、安装、执行和反编译流程以及涉及的相关技术。最后，本章将介绍 Android 应用程序的字节码反编译技术和以 smali 字节码为代表的虚拟机字节码的设计理论和执行流程。

2.1　Android 操作系统的安全机制

Android 操作系统基于 Linux 内核，基本继承了 Linux 操作系统的基本架构、应用程序接口和安全系统设计。在此基础上，Android 又针对移动设备的具体应用需求和应用场景，新增了一些新的安全系统和隐私保护系统。Android 操作系统的核心基于权限控制机制，通过引入权限、权限组以及权限等级等概念，操作系统可以引导并规范应用程序对硬件设备和敏感数据资源的访问，并防止恶意应用滥用软硬件资源。这套基于权限控制的安全模型在多年的应用中不断地被更新和强化，目前已经基本能够满足大部分普通用户的安全需求，防范一定等级的恶意攻击。在本节中，我们主要考察了 Android 操作系统的安全模型和权限系统，构建对 Android 操作系统权限控制的具体认识，并试图探讨该模型存在的内在问题，为接下来的研究提供理论支持。

2.1.1　Android 操作系统的安全模型

Linux 操作系统的安全模型以用户和用户组为核心，这样的安全模型不对用户之间的可信程度作任何假设，默认不同用户和用户组享有独占的资源控制权限。因此，Linux 操作系统的安全机制的设计思路是隔离不同用户和用户组对资源的访问，上述对资源的定义包括但不限于存储区域和磁盘文件、内存分区和活跃进程、硬件设备及其设备中断。基于 Linux 操作系统的安全模型，Android 系统在此基础上更进一步，对应用程序之间的互相访问做出了严格限制。Android 操作系统不对应用程序之间的可信程度作任何假设，默认不同应用程序独占一定的计算资源。因此，Android 将应用程序置于所谓沙箱的隔离环境内，对应用程序进行进程隔离和资源隔离。

Android 操作系统扩展了 Linux 系统的隔离机制，将多用户操作系统的用户隔离机制巧妙地移植为应用程序隔离。在 Linux 中，为了隔离不同用户的独占资源，操作系统采用用户标识 User ID 识别一个特定的用户，该 User ID 在系统全局中唯一且保持不变。

而在 Android 操作系统中，UID 则用于识别一个特定的应用程序，这个 UID 会在应用程序首次安装时由操作系统分配，且在应用程序在设备上存在期间该 UID 始终不变。为了衔接 Android 系统的安全机制和 Linux 内核的安全模型，Android 系统中的所有应用程序的 UID 和 Linux 内核中进程的用户 User ID 严格匹配，这样不同的 Android 应用程序在 Linux 内核中分属不同用户，各 Android 应用程序运行在独占的进程空间中，利用 Linux 的 User ID 隔离机制形成了天然的进程隔离和资源隔离，如图 2-1 所示。

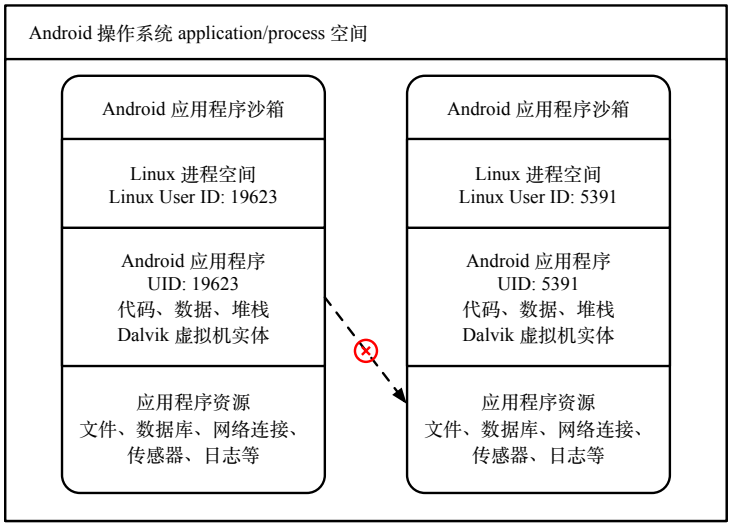


图 2-1 Android 操作系统的沙箱模型

上述安全模型就是 Android 操作系统中所谓的沙箱模型。在沙箱模型中，应用程序进程之间的安全隔离由 Android 操作系统以 UID 映射的形式授予 Linux 内核代理执行。与此同时，应用进程和系统进程之间的安全隔离由 Linux 内核的进程内存空间隔离机制实现。这样在默认情况下，应用进程互相无法得知对方的存在，更无法实现通信和数据共享。运行在进程沙箱中的应用进程在未经允许的情况下也无法陷入内核态，更无法访问操作系统的资源，这意味着对硬件设备的访问将被操作系统内核严格控制。这样，Android 操作系统就由操作系统应用层和 Linux 内核提供了两层隔离机制，运行在 Dalvik 虚拟机或 Android Runtime 运行时环境中的应用程序都将被妥善的隔离。

尽管 Android 操作系统不对应用程序之间的可信程度作任何假设，但由同一开发者或开发组织发行的应用程序之间客观存在着通信需求。考虑到这些应用程序间的信任关系由开发者提供背书，Android 操作系统也允许这些应用进程之间进行受控制的互相通信。Android 操作系统设计了一种共享 UID 的机制，在这个机制的指导下，由开发者提供信任背书的应用程序组能够在同一 Linux 进程空间内运行，如图 2-2 所示。为了指定这种互信关系，开发者或者开发组织需要以应用程序数字签名的形式显式地声明，并且在应用程序的 AndroidManifest.xml 清单中使用 `sharedUserId`[1] 属性，为他们指定相同的 UID。

[1]在本文中，凡涉及代码段及代码环境的文本均使用等宽字体，特此注明。

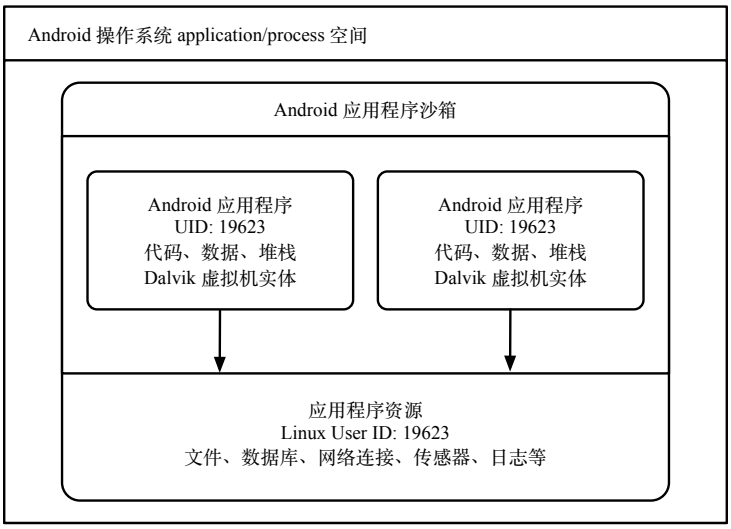


图 2-2 Android 应用程序的共享 UID 机制

2.1.2　Android 操作系统的权限控制机制

Android 操作系统以权限控制机制作为核心，其目的是限制恶意应用对敏感资源的访问，并保护用户的隐私。任何应用程序想要访问敏感的用户数据（比如通讯录、SMS）或者系统功能（比如调用照相机或向互联网发送数据），都必须申请相关权限。根据权限的等级和敏感程度，Android 可能会自动地授予应用操作权限，也可能会向用户请求获取权限。

Android 系统的安全体系结构设计的一个核心要点就是，在默认情况下禁止任何应用程序执行会对其他应用程序、操作系统或用户产生不良影响的操作。具体来说，这包括但不限于读取或写入用户的私人数据，如通讯录或即时通讯信息，读取或写入属于其他应用程序的文件，执行互联网访问，强行使设备保持在某种状态等等。

权限的授予

在 Android 操作系统中，应用程序必须通过在 AndroidManifest.xml 清单中通过 `<uses-permission>` 标签来声明其所需要的权限。举例来说，一个需要发送 SMS 消息的应用程序必须在 AndroidManifest.xml 清单中包含如下声明：

```
<user—permission android:name="android.permission.SEND_SMS"/>
```

对于应用程序在清单中列出的正常权限，系统会自动将其授予应用程序。正常权限指不会对用户的隐私或设备的操作造成太大风险的权限。相对地，对于应用程序在清单中列出的危险权限，系统会向用户提出警告并向用户显式请求授予这些权限。危险权限指有可能会影响用户隐私或设备正常运行的权限，比如上面介绍的发送 SMS 信息的权限。基于不同的 Android 操作系统版本和不同的应用程序的开发对象版本，操作系统发

出警告和请求的时机和方式也大相径庭：

- 对于 Android 5.1.1（API 级别 22）或更低级别的 Android 设备，如果安装的应用程序的 `targetSdkVersion` 小于 23，系统会在该应用程序安装时一次性地向用户警告并请求获取所有危险地权限，用户只能一揽子地接受或拒绝所有的权限授予，并且用户如果拒绝授予这些权限，操作系统将直接终止该应用程序的安装过程。
- 对于 Android 6.0 （API 级别 23）或更高版本的 Android 设备，如果安装的应用程序的 `targetSdkVersion` 大于或等于 23，其在安装时不会立即通知用户授予任何应用程序危险权限，而是在运行时才动态地请求并获取权限。

自动权限调整

应用程序的功能设计与操作系统的安全机制都在与时俱进，这意味着应用程序可能随着升级需要获取更多以前没有声索过的权限，而操作系统的权限级别和权限分类也有可能发生变化。在这样动态的过程中，Android 系统提供了稳健地前向兼容的手段。Android 系统的设计使得现有已安装应用可以自由获取这些 API，因此 Android 会把新的权限请求注入到这个应用的 AndroidManifest.xml 清单中，以避免在新的操作系统版本中遭遇应用程序的访问异常，从而逼迫开发者不得不重新修改代码并发布程序。Android 根据应用程序的 `targetSdkVersion` 字段作为判断依据，决定是否要在新的系统版本中向清单中添加新的权限请求。

举例来说，Android 操作系统从 API 等级 19 开始强制应用程序执行读取外部存储空间权限并限制对共享存储空间的访问请求。如果一个应用程序的 `targetSdkVersion` 小于或 19，这样的应用程序如果被安装在较新版本的 Android 系统中，系统会将这一权限的声明自动添加到 AndroidManifest.xml 清单中。这样的兼容操作带来的弊端是，即使应用程序实际上可能并不需要这些附加的权限，应用在权限申请时也可能会向用户请求这些权限。这样的请求可能会让用户误解应用程序的实际意图，甚至让用户对应用程序产生怀疑。一些根据申请权限作为判断依据的恶意应用程序静态分析系统也可能会受这样的机制的误导，从而得出错误的结论[29]。

权限的等级

根据权限所涉及的硬件资源和用户隐私，Android 操作系统将权限分成几个保护级别。其中两种权限保护级别会影响应用程序的运行，这两种权限级别分别是正常权限和危险权限。

正常权限，或称为普通权限，涵盖了应用程序通常涉及的一些低安全隐患的操作和资源，尽管这些资源或数据在应用程序的进程沙箱之外，但是操作系统判断对这些资源或数据的访问不会对其他应用程序和操作系统本身的运行造成影响。因此，如果应用程序在其 AndroidManifest.xml 清单中声明了对这些正常权限的需求，系统会在该应用程

序安装时自动授予它这些权限，而不会触发对用户的警告或通知，也不需要征求用户的同意。相应的，用户也无法手动撤销这些正常权限。

相对于正常权限，危险权限涉及到应用程序需要接触用户的隐私数据或敏感的系统资源的情况，或者执行可能影响其他应用程序的数据或操作系统的正常运行的操作。如果应用程序在 AndroidManifest.xml 清单中声明了对危险权限的请求，根据不同的 Android 版本，用户将会在应用程序安装时或者该权限被首次调用时收到警告和通知，并可以选择是否授予该应用程序这一项或若干项危险权限。在用户显式地授予危险权限之前，应用程序无法执行任何依赖与该危险权限的相关操作。

权限组

Android 操作系统将权限按照相关设备或逻辑功能分成若干个组，这些组被称为权限组。在这样的设计之下，应用程序的权限请求以权限组为处理的最小单位，而一个权限组往往对应于应用程序的 AndroidManifest.xml 清单中的多个权限请求声明，如图 2-3 所示。举例来说，`CONTACTS` 权限组就包含了 `READ_CONTACTS`、`WRITE_CONTACTS` 和 `GET_ACCOUNTS` 三个具体权限。通过对权限进行分组，操作系统可以给用户更直观和可读的权限解释，从而避免了用户因误解而导致的错误授予或撤销权限，也避免了权限请求的琐碎复杂影响到用户对应用的正常使用。

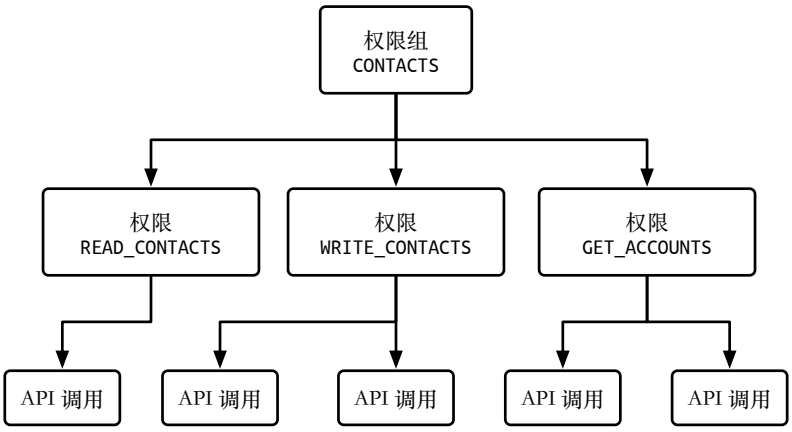


图 2-3 权限与权限组

在 Android 系统中，所有危险权限都从属于某个权限组。同时不管其他权限处于哪一个风险等级，都可以从属于某个权限组。在 Android 6.0（API 等级 23）之后的 Android 版本中，如果运行 `targetSdkVersion` 为 23 及以上的应用程序，当该应用程序在 AndroidManifest.xml 清单中申请危险权限时，Android 操作系统将做出如下反馈：

- 如果对于某个特定的权限组，该应用程序尚未获取任何一个从属于该权限组的危险权限授予许可，而该应用程序请求了该权限组中的一个或多个危险权限，则操作系统会显式地向用户弹出警告和权限申请提示，该提示会详细描述应用程序申

表 2.1 危险权限及其从属权限组

权限组	权限组说明	权限
CALENDAR	日历	READ_CALENDAR WRITE_CALENDAR
CAMERA	照相机	CAMERA
CONTACTS	通讯录	READ_CONTACTS WRITE_CONTACTS GET_ACCOUNTS
LOCATION	定位信息	ACCESS_FINE_LOCATION ACCESS_COARSE_LOCATION
MICROPHONE	麦克风	RECORD_AUDIO
PHONE	设备信息与通信功能	READ_PHONE_STATE CALL_PHONE READ_CALL_LOG ADD_VOICEMAIL USE_SIP PROCESS_OUTGOING_CALLS
SENSORS	运动传感器	BODY_SENSORS
SMS	即时短消息	SEND_SMS RECEIVE_SMS READ_SMS RECEIVE_WAP_PUSH RECEIVE_MMS
STORAGE	存储空间	READ_EXTERNAL_STORAGE WRITE_EXTERNAL_STORAGE

请的危险权限所属的权限组。需要注意的是，系统的警告和提示不会具体描述特定的危险权限，而只会刻画该权限分属的权限组的特征和功能。举例说明，如果某个应用程序向用户请求 `READ_CONTACTS` 权限，则系统弹出的警告只会知会用户该应用程序请求访问该设备的通讯录功能，而不会具体说明该应用程序究竟是需要读取通讯录、写入通讯录还是两者皆有。但是，当用户授权批准该权限请求时，系统只会赋予该应用程序其在应用 AndroidManifest.xml 清单中声明的权限，而不会将权限组的所有权限都赋予应用程序。举例说明，针对上面提到的应用程序，当用户授予其通讯录权限时，它得到的只是读取通讯录权限，而不是通讯录权限组下的所有权限。

- 如果对于某个特定的权限组，该应用程序已经获取了某一个从属于该权限组的危险权限，在该应用程序请求属于该权限组的其他危险权限时，系统会立刻自动授予该危险权限，而不需要向用户弹出警告或通知。举例说明，如果某个应用程序向用户请求了 `WRITE_CONTACTS` 权限，而在此之前该应用程序已经获取了 `READ_CONTACTS` 权限，系统会立刻自动地向该应用授予写入通讯录的权限，而不再需要用户的显式许可。

表 2.1 列出了截止到 Android 9.0 版本时 Android 操作系统的危险权限及其从属的权限组。

2.2 Dalvik 虚拟机及其执行原理

尽管早期的 Android 操作系统使用 Java 语言开发应用程序，但 Android 系统并不使用标准的 JVM 作为应用的执行环境。针对移动设备应用程序的运行特点以及移动设备的功耗问题，也为了规避和持有 Java 所有权的 Oracle 公司的知识产权纠纷，Google 为 Android 操作系统定制了其专有的 Dalvik 虚拟机。Android 4.4 之后，Google 又对 Android 的执行环境进行了大幅改进，引入了 Android Runtime 运行时环境。ART 在基于即时编译技术 (JIT) 的 Dalvik 虚拟机上新增了预先编译技术 (AOT)，极大地改善了 Android 应用程序的运行性能。为了实现前向兼容，ART 环境仍然完全支持 Dalvik 执行环境及其字节码规范[39]。

2.2.1 Dalvik 虚拟机

Google 于 2007 年发布的 Android SDK 使得开发者第一次接触到 Dalvik 虚拟机。Dalvik 虚拟机是 Android 操作系统的核心组成部分之一，其特点有如下几点：

- 采用 DEX 可执行文件格式，针对内存和处理器速度有限的设备进行了优化；
- 基于寄存器架构，拥有完整的指令集及其执行系统；

- 运行时体积小，占用内存空间小；
- 提供了对象的生命周期管理、垃圾回收、线程和并发优化、异常系统等重要机制；
- 每个 Android 应用程序都运行在独占的系统进程中，且每个进程都独占一个 Dalvik 虚拟机实例。

尽管同样采用 Java 作为开发语言，Dalvik 虚拟机和 JVM 却有着根本上的区别。首先，两者所编译出的字节码格式完全不同。JVM 将 Java 源代码编译为 JVM 字节码，而 Dalvik 虚拟机将 Java 或 Kotlin、Scala 源代码编译成 Dalvik 字节码。Java 源代码经过 JVM 的编译，生成的 JVM 字节码被保存在 class 文件中，随后 JVM 读取 class 文件中的字节码，并通过解释执行的方式运行 Java 程序。而 Dalvik 虚拟机将 Java 代码编译成 Dalvik 字节码，并将其字节码打包到 DEX 文件中，随后 Dalvik 通过 JIT、AOT 等手段执行应用程序。

在 Android 操作系统中，名为 dx 的程序负责将 Java 代码转换成 Dalvik 字节码。dx 能够重新排列 Java 类文件，并消除所有类文件中存在的冗余信息，以避免 Dalvik 虚拟机在初次运行应用程序时因反复加载并解析文件而损失性能。Java 类文件中通常包含多个方法签名，如果其他类文件引用了该类的方法，相应的方法签名也会被重复复制到其他类文件中。换言之，如果不同的类同时使用了相同的方法，则大量的方法签名会被多个类重复使用。这些冗余信息造成的程序体积膨胀严重影响了虚拟机对字节码的解释执行效率。针对该问题，dx 程序对所有 Java 类文件的常量池进行了分解并消除了其中的冗余信息，然后将其重新组合成一个新的常量池，并让所有类共享该常量池。图 2-4 展示了 dx 程序将 Java 类文件转换成 DEX 文件的过程。

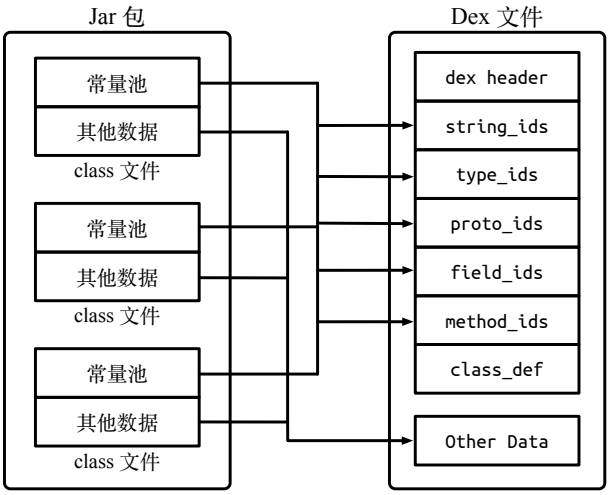


图 2-4 dx 程序对 Java 类文件的处理

Dalvik 虚拟机和 JVM 的另一个重要的区别就是它们的架构完全不同。众所周知，JVM 基于堆栈架构，当 Java 程序在 JVM 中运行时 JVM 会频繁地对堆栈进行读写。而对堆栈的频繁访问会导致多次处理器指令分派和内存访问，从而消耗大量的 CPU 时间。

此外，因为堆栈结构并未针对设备硬件进行优化，JVM 在执行字节码时还往往会频繁遭遇缓存未命中和内存缺页的问题。因此 JVM 及其不适合移动设备环境，因为这些硬件设备通常处理性能极其有限，内存资源紧张，且高度功耗敏感。相对地，Dalvik 虚拟机基于寄存器架构，数据的访问直接在寄存器之间传递，这样的访问方式避免了缓存命中和缺页中断的问题，极大地提高了应用程序的执行效率。

2.2.2　Dalvik 字节码的编译与执行

下面通过实例介绍 Java 代码被编译成 JVM 字节码和 Dalvik 字节码的过程，并分析 JVM 字节码和 Dalvik 字节码的区别。

```
public class Test {
    public int foo(int a, int b) {
        return (a + b) * (a - b);
    }
    public static void main(String[] argc) {
        Test test = new Test();
        System.out.println(test.foo(5, 3));
    }
}
```

使用 Javac 编译命令编译上述代码，可以得到如下的 JVM 字节码：

```
public int foo(int, int):
Code:
    0:      iload_1
    1:      iload_2
    2:      iadd
    3:      iload_1
    4:      iload_2
    5:      isub
    6:      imul
    7:      ireturn
```

上面的字节码节选部分是 Java 代码中 `Test` 类的 `foo` 方法的 JVM 字节码。JVM 虚拟机的指令集通常被称为零地址形式指令集。零地址形式指令的源参数和目标参数都是

隐式的，JVM 提供一种名为求值栈的数据结构来传递这些参数。当 JVM 执行 Java 应用程序时，每个线程拥有一个程序计数器 PC 和一个独占的 Java 栈。程序计数器以字节为单位，记录程序当前的执行位置和方法首地址之间的偏移量，其作用类似于 ARM 指令集中的指令寄存器指针 PC 和 x86_64 指令集中的指令指针寄存器 IP。和这两种寄存器不同的是，JVM 的程序计数器只针对当前方法，而不是整个 Java 程序。Java 栈用于记录 Java 方法调用的活动记录 AR，以帧为单位保存线程的运行状态。每当调用一个方法，就会为该方法分配一个新的栈帧并压入 Java 栈。相对地，每从一个方法返回，就从栈顶弹出相应的栈帧。每个栈帧都包括了一个局部变量区和操作数栈，其中局部变量区用于存储方法的参数和局部变量，参数按照 Java 源码中从左到右的顺序保存在局部变量区低地址开始的区域中。操作数栈则保存了求值的中间结果和调用其他方法的参数等。

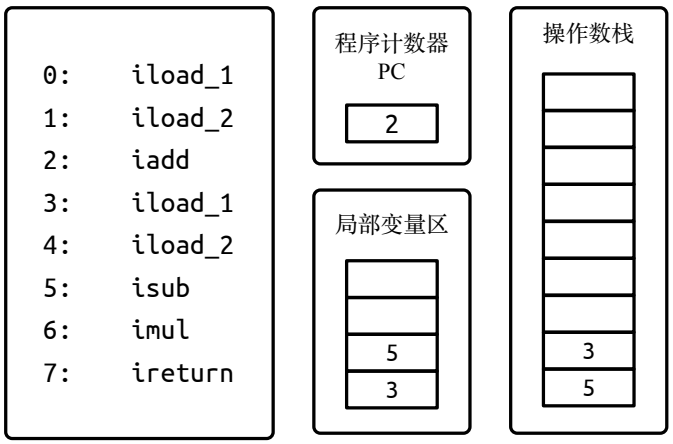


图 2-5 JVM 运行时操作数栈的状态

图 2-5 展示了当执行到上述代码第 2 行时，操作数栈的数据状态。第 0 条指令 `iload_1` 中的 `iload` 表示的是 JVM 的 `load` 指令，功能是将局部变量压入操作数栈。指令中的 `i` 是指令前缀，表示操作类型是 `int` 整型数。`iload_1` 中的数字表示操作的局部变量偏移量，如 `iload_1` 就代表压入第 1 个 `int` 型数据进入操作数栈。类似地，第 1 条指令 `iload_2` 用于取出第 2 个方法参数。当代码执行到第 2 行时，JVM 从栈顶连续弹出两个数据，计算这两个整型数的和，并将计算结果压入栈顶。后面几行的操作与此类似，第 3、4 两行重新向栈中压入方法参数，第 5 行将两个参数弹出并求差，接着压回堆栈。代码执行到这里时，栈内保存的两个元素分别是方法参数的和与差。接下来第 6 行计算栈内两个元素的积，第 7 行方法返回并结束。

Dalvik 虚拟机在运行时也为每个线程维护了一个程序计数器 PC 和一个寄存器列表。寄存器列表中的寄存器数量在方法元数据中的 `register` 字段声明，Dalvik 虚拟机根据该值来申请相应大小的寄存器列表。和 JVM 字节码相比，Dalvik 字节码简洁得多，下面的代码就是上述 Java 代码中 `Test` 类的 `foo` 方法对应的 Dalvik 字节码。

```
Virtual methods     -
  #0                : (in LTest;)
  name              : 'foo'
  type              : '(II)I'
  access            : 0x0001 (PUBLIC)
  code              -
  registers         : 5
  ins               : 3
  outs              : 0
  insns size        : 6 16-bit code units
    000198:                         |[000198] Test.foo:(II)I
    0001a8:    9000 0304            |0000: add-int           v0, v3, v4
    0001ac:    9101 0304            |0002: sub-int           v1, v3, v4
    0001b0:    b210                 |0004: mul-int/2addr     v0, v1
    0001b2:    0f00                 |0005: return            v0
  catches:          : (none)
  positions:        : 0x0000 line=3
  locals            : 0x0000 - 0x0006 reg=2 this LTest;
```

上述 Dalvik 字节码中的指令 `add-int` 用于将寄存器 `v3` 和 `v4` 中的值相加，并将结果保存到寄存器 `v0` 中。类似地，下面的指令 `sub-int` 用于将寄存器 `v3` 和 `v4` 中的值相减，并将结果保存到寄存器 `v1` 中。指令 `mul-int/2addr` 则用于将寄存器 `v0` 和 `v1` 中的值相乘，并将结果保存到寄存器 `v0` 中，最后的一条指令声明返回寄存器 `v0` 中的值。

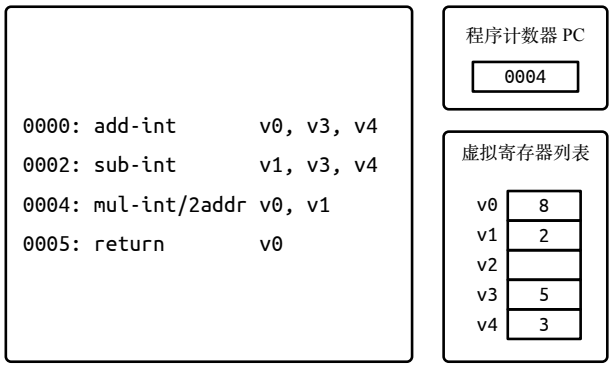


图 2-6 Dalvik 虚拟机运行时虚拟寄存器的状态

图 2-6 展示了当执行到命令 0004 时，虚拟寄存器列表的数据状态。上面的分析说明，和基于堆栈的 JVM 指令集相比，基于虚拟寄存器架构的 Dalvik 指令集生成的代码指令更少，更便于移动设备针对性的对缓存命中和虚拟内存空间进行优化，提升程序的运行性能和执行速度。

2.2.3 Dalvik 虚拟机的运行原理

Android 操作系统基于 Linux 内核，在 Linux Kernel 的上层添加了自己的核心库和运行时。Android 操作系统基于分层的思想，旨在减少系统各部分的耦合，便于模块式开发和软件工程化。Dalvik 虚拟机及其继承者 ART 都属于 Android 运行时部分，而运行时和核心库一起负责 Android 应用程序解释和执行的核心工作。

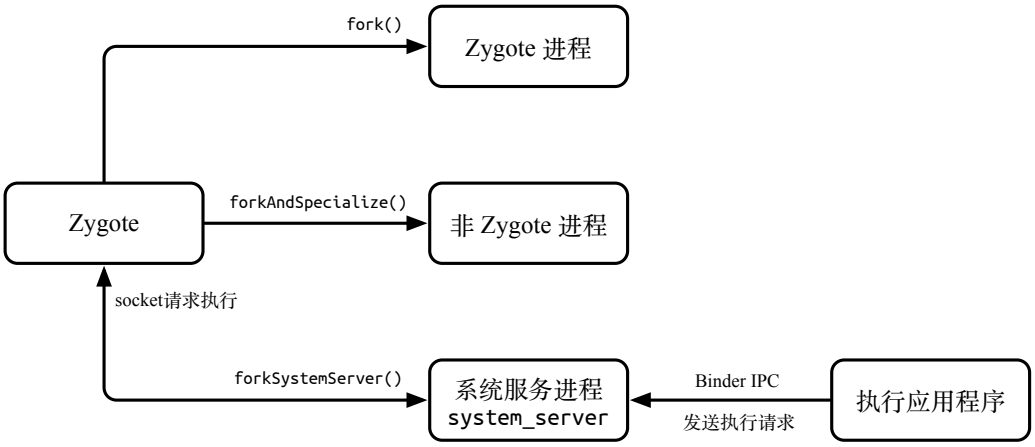


图 2-7 Android 应用程序的启动流程

图 2-7 揭示了 Android 系统启动应用程序的标准流程。Android 操作系统在启动并加载内核后，会立即执行一个 `init` 进程，该进程用于完成设备的初始化工作，并读取 init.rc 文件以启动系统中最重要的外部程序 Zygote。Zygote 是 Android 操作系统中所有进程的孵化器进程。Zygote 启动后会先初始化的 Dalvik 虚拟机，接着启动 `system_server` 进程并进入 Zygote 模式，通过套接字等候系统下达命令。当用户启动一个 Android 应用程序时，`system_server` 进程通过 Binder IPC 的方式将命令发送给 Zygote。Zygote 收到命令后通过 fork 自身进程的方式创建一个 Dalvik 虚拟机实例来执行相应应用程序的入口函数，从而完成应用程序的启动过程。

进程 fork 成功后，应用程序的执行工作将移交给 Dalvik 虚拟机来完成。Dalvik 虚拟机通过 `loadClassFromDex()` 函数进行类加载。每个 Java 类被成功解析后，都在运行时环境中生成一个对应的 `ClassObject` 类型的数据结构，而 Dalvik 虚拟机通过维护一个名为 `gDvm.loadedClasses` 的全局散列表来存储和访问所有已加载类。下面，字节码校验器通过 `dvmVerifyCodeFlow()` 函数对装入的类进行校验。随后虚拟机调用 `FindClass()` 函数寻找并装载含有 `main()` 方法的类。最后，Dalvik 虚拟机调用 `dvmInterpret()` 函数来初始化代码解释器并开始解释执行代码。

2.3 Dalvik 虚拟机字节码规范

Dalvik 虚拟机拥有专属的指令集、指令格式和调用规范。依照 Dalvik 指令集组成的代码被称为 Dalvik 汇编码。Dalvik 汇编语言拥有专属的机器模型和类似 C 语言的调用

规范，具体如下：

- 采用基于寄存器的设计思路。方法在内存中创建后拥有固定大小的栈帧，栈帧占用的空间取决于方法中指定的寄存器的个数；
- 如果整数和浮点数按位表示，可以使用 32 位寄存器存放。相邻的两个寄存器可以组合来表示 64 位数据以存放大数类型。寄存器不需要考虑内存对齐；
- 调用约定使用 N 个寄存器表示 N 个参数。对于非静态方法，参数列表的第 0 个参数传递的是实例的 `this` 指针；
- 指令流以 16 位无符号整型作为存储单元，某些特殊指令会将部分位置零或忽略。在使用指令操作寄存器时，不需要对指令具体说明其操作数的具体类型。

2.3.1　Dalvik 寄存器

Dalvik 虚拟机是基于寄存器架构的，其代码中使用了大量的虚拟寄存器。Dalvik 虚拟机在设计之初主要面对 ARM 架构的处理器。ARM 架构的 CPU 本身集成了 37 个物理寄存器，包括 30 个通用寄存器、1 个程序计数器 PC 和 6 个状态寄存器，Dalvik 将部分虚拟寄存器映射到 ARM 的物理寄存器上，还有一部分虚拟寄存器通过调用栈进行模拟。Dalvik 虚拟机使用的寄存器都是 32 位的，并支持所有数据类型。对于 64 位的大数类型，可以用两个相邻的寄存器组合存储，组合的字节序可自定义，但一般默认为小端模式。

Dalvik 虚拟机为每个进程维护一个调用栈，这个调用栈的作用之一就是模拟寄存器。如前所述，每个方法都会在元数据部分用 `registers` 字段指定该方法使用的寄存器个数。当程序调用到该方法时，会根据寄存器个数分配适当的栈空间，这些栈空间就是用来存放寄存器的实际值的。Dalvik 虚拟机通过处理字节码对寄存器进行读写操作，实际上都是对栈空间进行写操作。Android SDK 中的 `dalvik.bytecode.Opcodes` 接口定义了 Dalvik 字节码列表，Dalvik 虚拟机使用宏 `HANDLE_OPCODE()` 来处理这些字节码。

2.3.2　寄存器命名法

Dalvik 虚拟机使用 v 命名法和 p 命名法作为寄存器的表示方法。如果一个方法在元数据部分的 `registers` 字段指定该方法使用了 M 个寄存器，而该方法有 N 个参数。根据 Dalvik 虚拟机对参数传递方式的规定，该方法参数使用高地址的 N 个寄存器存放，而局部变量使用从 `v0` 低地址开始的 $M-N$ 个寄存器。在 2.2.2 小节所展示的实例中，`foo(int,int)` 方法在元数据中声明了 5 个寄存器和 2 个显式的整型参数。考虑到该方法是类 `Test` 的非静态方法，其在被调用时会传入一个隐式的 `this` 指针，因此实际上传入了 3 个方法参数。因此根据上述参数传递规则，局部变量将使用低地址的 2 个寄存器，方法参数则讲使用高地址的 3 个寄存器。

v 命名法采用以小写字母 v 开头的形式表示方法所使用的局部变量和参数，寄存器命名从 `v0` 开始逐一递增。仍以 `Test.foo(int, int)` 方法为例，v 命名法会使用 `v0, v1, v2, v3, v4` 共 5 个寄存器，其中 `v0, v1` 两个寄存器用于表示函数的局部变量寄存器，而 `v2` 表示被传入的 `Test` 实例对象的引用，即对象的 `this` 指针，`v3, v4` 则分别用于表示连个传入的整型参数。

p 命名法对方法的局部变量寄存器仍然采用 v 命名法，而方法参数从 `p0` 开始逐一递增。因此，对于 `Test.foo(int, int)` 方法，p 命名法会使用 `v0, v1, p0, p1, p2` 这 5 个寄存器，其中 `v0, v1` 两个寄存器仍然用于表示函数的局部变量寄存器，而 `p0` 表示被传入的 `Test` 实例对象的引用，`p1, p2` 则分别用于表示连个传入的整型参数。通过观察可以发现，使用 p 命名法的 Dalvik 字节代码通过寄存器前缀名更容易判断寄存器中存放的是局部变量还是方法参数。在 Dalvik 汇编代码较长、使用寄存器较多的方法中，这种优势将更加明显。

2.3.3 Dalvik 字节码语法

Dalvik 字节码拥有一套完整的语言规范，可以将 Java 代码的相应元素完整映射到 Dalvik 字节码上。Dalvik 字节码和 Dalvik 虚拟机指令集一起构成了 Dalvik 汇编规范。本小节将简要介绍 Dalvik 字节码的类型系统以及对象中方法和字段的表示方法。

类型系统

Dalvik 字节码只有两种类型，分别是基本类型和引用类型，这在某种程度上和 Java 语言是一脉相承的。Dalvik 使用这两种类型来表示 Java 语言的所有类型。除了对象和数组属于引用类型，其他的 Java 类型都属于基本类型。Dalvik 字节码使用类型描述符来标记数据的类型，类型描述符如表 2.2 所示。

如前所述，每个 Dalvik 寄存器都是 32 位的，对于长度小于 32 位的数据类型来说，只用一个寄存器空间就足以存放该类型的值。而对于 `J, D` 等 64 位的数据类型来说，Dalvik 虚拟机需要使用两个寄存器来存储。`L` 类型可以表示任何 Java 类，这些类在 JVM 中以 `package.name.ObjectName` 这样的全限定名被索引和调用，而在 Dalvik 虚拟机中以 `Lpackage/name/ObjectName;` 的形式表示。其中 `L` 表示描述符后跟随着一个 Java 类的全限定名，`package/name` 表示对象所在的包，而 `ObjectName` 则表示类名称。和 Java 与 JVM 的实现类似，Dalvik 虚拟机字节码的内存模型并不强制要求内存对齐，因而虚拟机字节码的内存模型与虚拟机实现高度相关。尽管如此，从 Android 4.4 发布 Android Runtime 运行时环境以来，变量的内存模型并未发生显著变化。

`[`类型描述符可以描述所有基本类型的数组。其使用方法是在 `[`后跟随基本类型的类型描述符，如 `[I` 表示一个整型数组，相当于 Java 语言中的 `int[]`。多个 `[`可以嵌套用于表示多维数组，比如 `[[I` 一个二维整型数组，相当于 Java 中的 `int[][]`。多维数组的最大嵌套深度是 255。

表 2.2 Dalvik 字节码的类型描述符

类型描述符	对应的 Java 类型	含义
V	void	空类型（只用于返回值类型）
Z	boolean	布尔型
B	byte	字节
S	short	短整型
C	char	字符型
I	int	整型
J	long	长整型
F	float	浮点型
D	double	双精度浮点型
L	class	Java 类
[	array	数组

方法

Dalvik 方法的表现形式比类型系统稍显复杂。Dalvik 使用方法名、参数类型和返回值来详细描述一个方法。这有助于 Dalvik 虚拟机在运行时快速通过完整的方法全限定名索引到方法，也便于 Dalvik 对代码进行静态分析，便于事前优化。具体地，一个 Dalvik 方法的具体格式如下：

```
Lpackage/name/ObjectName;->methodName(III)Z
```

其中 `Lpackage/name/ObjectName;` 表示方法所属类的类名，而 `methodName` 是方法名。`(III)Z` 是方法签名的一部分，其中 `III` 表示该方法显式地接受 3 个整型参数，而 `Z` 表示该方法将返回一个布尔型值。一个更复杂的示例如下：

```
methodName(I[[IILjava/lang/String;[Ljava/lang/Object;)
    Ljava/lang/String;
```

其对应的 Java 方法签名如下：

```
String methodName(int, int[][], int, String, Object[])
```

字段

字段的表示方法和方法类似，和方法不同的是字段中没有方法的参数和返回值，但多了字段类型。字段的全限定名由类名、字段名称和字段类型组成，字段名称和字段类型之间用冒号分隔，下面是 Dalvik 字节码中字段的一个示例：

```
Lpackage/name/ObjectName;->fieldName:Ljava/lang/String;
```

上面的 Dalvik 字节码所表示的字段对应的 Java 代码如下：

```
String fieldName()
```

2.4 本章小结

本章介绍了本研究下面将要涉及和使用的相关技术理论，包括 Android 操作系统的安全机制和 Dalvik 虚拟机的相关理论。对于 Android 操作系统的安全理论，我们具体分析了 Android 操作系统的进程隔离和沙箱机制，并介绍了以权限控制机制为核心的 Android 系统用户隐私保护机制，详细地分析了 Android 操作系统的危险权限控制策略。对于 Dalvik 虚拟机的相关内容，我们介绍了 Dalvik 虚拟机的结构和设计原理，并分析了字节码在 Dalvik 虚拟机中编译和执行的过程，还阐述了 Dalvik 虚拟机在 Android 操作系统中的运行原理。为了便于后续基于字节码嵌入的系统设计，我们还讨论了 Dalvik 字节码的规范、寄存器分配理论和具体语法。

第三章　基于虚拟机字节码注入的应用隐私保护机制

本章提出了一种控制 Android 应用程序对用户敏感信息访问和危险权限调用的隐私保护系统，该系统基于安全策略生成的虚拟机字节码注入技术，通过在目标应用程序中注入控制机制以限制危险权限的访问并强制规范应用程序的行为。该系统基于动态安全策略生成机制，这意味着系统所应用的安全策略可以根据用户需求动态调整，该机制通过将安全约束规范和具体代码行为解耦合来实现动态性和灵活性。动态策略生成机制保证即使用户或组织频繁地修改安全策略，该系统也无需重新编译应用程序或对应用程序进行重打包。

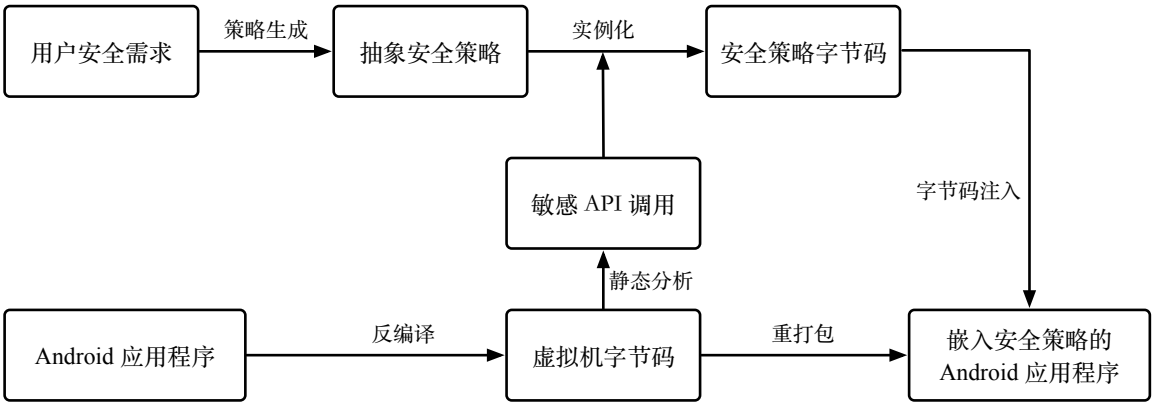


图 3-1 系统结构概览

图 3-1 是该系统的结构概览。该系统能够根据用户和组织的需求，通过数学和形式逻辑的相关理论抽象出特定的行为模型。该行为模型将数据流、事件时间、用户身份与角色、环境与场景、信任关系等因素列入考虑，并得到抽象的安全策略。与此同时，该系统提供了一个安全策略生成器 SSG，该生成器能够通过执行一组细粒度的安全规则将用户输入的抽象策略转换成具体的权限控制策略。上述安全规则具体考虑了应用程序的运行逻辑和具体模型，并在安全策略中部署安全决策中心。

实例化的安全策略将指导系统生成一个包含安全方法的字节码安全运行库，该运行库中包含了执行安全策略的代码。该安全运行库将随后被嵌入到应用程序字节码级的危险权限请求和敏感 API 调用位置，并和原始应用字节码一起被重新打包生成一个包含安全决策中心 SDC 的应用程序。这个包含安全决策中心的应用程序中嵌入了一系列的策略执行器 PE，这些执行器能够在应用触发权限请求行为时向安全决策中心发出事件通知，并向安全决策中心索取安全策略指定的授权操作以执行具体的策略。这就意味着，包含安全决策中心的应用程序将在危险权限被调用时向所有和该事件有关的执行器提供统一的权限控制接口。

接下来，本章将分别详细介绍基于安全策略的字节码注入系统的安全策略生成机制、安全策略优化机制和字节码嵌入与执行机制。

3.1 安全策略的生成

本节将介绍基于行为模型的安全策略生成和优化机制。该机制能够根据用户或组织所指定的安全需求生成抽象安全策略，并能根据安全策略的应用目标对其进行实例化。

3.1.1 安全策略的属性

首先，为了满足更广泛的用户需求和实现更强的敏感权限控制能力，我们所需要的安全策略应该具备以下属性：

- **感知用户所处动态环境**：安全策略机制必须能够针对用户或组织所处的不同环境给出相应的安全策略，以确保用户或组织所处的环境发生变化时安全策略也能够得到动态地调整。对安全策略的动态调整不仅需要满足环境所必须的安全和隐私要求，还要确保应用程序的可用性和执行效率；
- **支持信任机制**：对于用户或组织的特定需求，安全策略生成机制还需要拥有信任机制。该信任机制能够考虑在特定设备、特定用户、特定应用程序或者特定程序开发者签名等条件下的信任关系。为了清晰地界定信任的范围和语义并保证不滥用信任机制，需要对信任系统进行形式逻辑上的抽象；
- **考虑用户的认知层次**：普通的 Android 设备用户并不拥有对安全或隐私的知识水平和保护意识，且不同的用户或组织对安全策略和隐私保护的理解等级和需求层次也不尽相同。安全策略的生成机制应该照顾到不同用户与组织的不同粒度的要求，并能够对用户的抽象需求给出更具体的安全建议；
- **控制应用程序的数据流**：安全策略机制应该且必须能够向用户或组织提供对 Android 应用程序所生成、读取、修改、传输的数据的具体类型和规模的信息。根据用户制定的抽象安全策略，该机制也应该能够自动生成对数据流的控制策略，同时提供细粒度的数据控制机制；
- **数据匿名化**：安全策略中所支持的数据流控制还必须支持对所有应用程序所生成、读取、修改或传输的数据的脱敏和匿名化，这不仅是为了防止由于安全策略的引入给用户或组织带来隐私信息的次生灾害，也是为了进一步提升安全策略机制的安全水平。

在 Android 应用程序的使用过程中，不同的用户或组织可能针对场景、数据、控制级别和信任提出更细粒度的安全需求，而这些安全需求无法通过 Android 操作系统所提供的安全模型和权限系统得到满足。本系统能够针对上面开列的安全需求为 Android 用

户提供更细粒度的安全控制机制。为了实现该目标，本系统采用了基于行为模型的抽象策略生成机制，对行为模型进行了抽象和聚合，且能够对生成的具体安全策略进行进一步优化。针对数据匿名化需求所进行的工作，本文将在第四章中进行更为详细的介绍。

3.1.2　行为模型的抽象

想要实现能够满足细粒度控制需求的安全策略生成机制，我们首先需要对 Android 应用用户的行为模型进行抽象。行为模型抽象的目的是梳理用户实体和应用程序实体间的交互方式，并将实体间的详细交互流程进行抽象以便于工程实现。本系统所提出的行为模型能够提供在数据机密性、用户的感知与控制、访问权限控制和信任管理等多层次的安全功能。为了实现这样的需求，行为模型需要对数据流、事件时间、用户身份与角色、环境与场景、信任关系等因素进行逻辑学抽象。

从语义上定义，数据流指的是应用程序所生成、读取、修改、传输的所有数据，所有数据流都将被抽象为源宿之间的点到点的数据通信过程。事件时间指的是安全规则指定的具体事件发生的时间点、持续时长以及事件之间的时间间隔。用户身份与角色指的是用户自身的身份属性以及用户可能的在特定组织中所具有的安全权限和安全职责，以及用户实体的角色类型和所处的安全层次等级。环境和场景指的是用户所处的位置和应用环境，不同的位置和不同的场景可能提出对安全策略的不同需求。信任关系指的是用户与用户间、用户与组织间、用户和应用程序间的信任关系，该信任关系将直接影响到安全策略的制定。

数据流

对行为模型进行抽象，首先需要对应用程序涉及的数据流进行定义。本系统对涉及的所有数据类型都进行了定义，和面向对象的程序语言类似，本系统将所有数据类型划分为基本类型和复合类型，其中符合类型由基本类型或其他复合类型通过组合或继承方式构成。

由特定类型的实例化数据构成的数据集合称为数据流，数据流既可以由应用程序在运行时通过对资源的读取、修改、传输等操作生成，也可以在应用程序构建时参考代码定义或配置文件等静态信息创建。我们定义了完整的数据类型约束规范，因此对数据流的自动转型和默认派生都被严格禁止，所有对数据类型的操作或变换都必须依照我们对数据类型的定义并保证继承或组合关系的逻辑自洽。应用程序在运行期间可能会在应用内部 Activity 间、Intent 间进行数据交换，也会和其他应用程序或操作系统进行数据交换。此外，一些 Android 应用程序还会有与网络服务器进行数据通信的需求，应用程序在此过程中向服务器以一定模式传输特定数据流。

对于所有数据集和数据流，系统都需要进行类型和限定名的匹配，只有类型匹配的数据才能作为相应数据源的引用。系统在实现中采用正则表达式、XPath 表达式和静态

文本分析等手段保证限定名和数据类型的匹配。和身份与角色机制、环境与场景机制和信任机制相配合，本系统也能够规范化地将所有系统通信转化为数据流的形式。

身份与角色

对身份与角色的抽象也遵循着基于属性的方法，身份和角色的属性也属于上述数据实例化集合的一部分。用户的身份由其身份唯一全限定符指定，而其角色从属于特定角色集合。所有身份实例包括身份的唯一全限定符和角色构成。本系统使用匹配机制来进行身份和角色的权限认定。如果一个身份的全限定符与角色和特定权限的角色列表匹配，则证明该身份及其角色拥有访问相关权限的能力。

角色机制允许继承结构，子角色自动拥有父角色拥有的访问权限。为了防止角色层次结构出现循环依赖问题，本系统在将角色分配给每个以唯一全限定符标识的身份时都将从角色的继承树上溯，以确保继承结构中不出现有向环路。

环境与场景

环境和场景限定了获取数据、敏感信息或特定权限的特定时间、位置或者工作环境。对环境和场景的抽象是一件较为复杂的过程，因为系统需要考虑到特定时间点开始或结束的事件、环境的动态性以及场景的迁移，这都给行为的抽象带来了较大困难。本系统中对环境和场景的抽象除了和上述数据流机制相结合外，还对常见的应用使用场景和环境进行了枚举与建模。通过定义从特定场景到特定数据实例的映射，以及定义特定上下文情景到特定场景的映射，本系统能够以这种链式方式建立环境、场景到具体权限调用和数据获取的关联。尽管如此，系统对环境和场景的感知仍然不能实现完全覆盖。因此为了保证系统能够对多种应用场景以及环境的变化作出相应，我们将控制权部分交给了用户，这一方面保证了系统的可扩展性和可用性，另一方面也给用户更大的自主空间，能够保证用户在面对各种复杂环境和具体需求时能够制定出更合适的安全策略。

信任关系

为了支持不同的信任级别和信任关系，本系统对信任机制也进行了抽象和形式化。本系统既能够支持由用户或组织所显式声明的信任关系，也支持由已有的数据或场景默认推导的信任关系，当然这些由系统生成的信任关系最终仍然需要征求用户的许可与授权。举例来说，一个用户或组织决定信任某个开发者或开发机构所开发的某个应用程序对某些权限的访问，则系统也会倾向于默认用户或组织允许同样由该开发者或机构所开发的其他应用程序授予相同的权限，或相应权限集合的一个子集。

此外，信任机制也会和环境与场景机制发生关系。系统给出的信任建议视应用的具体使用场景而定，高风险的使用场景所作出的信任决策显然需要比低风险场景所作出的决策更加谨慎和稳健。此外，在组织中不同个体用户所拥有的角色也可以对其给出的信

任建议赋予不同的权重，组织的整体角色可以考虑不同个体的个性化需求，并根据最后的投票结果决定整体的信任决策。

3.1.3 安全策略的生成

基于上节中对行为模型的分析和定义，安全策略生成机制使用事件条件操作规则 ECA 来提取抽象策略和详细执行策略。算法 3.1 揭示了事件条件操作规则模板的抽象语法。安全策略生成器持续监视从模板实例化的 ECA 规则，每当观察到特定模式的事件使得特定组合的所有条件得到满足时，就会执行相应的授权操作。

算法 3.1 事件条件操作规则

输入: E: 事件模式集合
输入: C: 用户指定的条件
输入: D: 用户指定的时延
输入: T: 信任类型
输入: n: 应用程序控制单元个数
输出: R: 规则模板

$T \leftarrow T_{Indirect} \cup T_{Direct}$
$D \leftarrow D_{Amount} \times D_{Unit}$
for $i \in \{1, 2, \ldots, n\}$ **do**
  if $T_i == true$ **then**
    if $C_i \in \mathbb{P}E$ **then**
      $R_i \leftarrow C_i \dashv D_i$
    end if
  else
    $R_i \leftarrow deny$
  end if
end for
return R

上述活动模式指的是和临时或规律的行为或交互相匹配的模式，指定这种模式能够支持检测性和预防性的安全策略。上述的条件可能对应着各种复杂的执行流程和分支场景，如事件模式、外部行为、定时条件、执行条件、上下文、用户身份或角色、用户和行为可行度等等。上述权限操作的执行具体指的是对临时操作的授权或拒绝，也可以更细粒度地根据安全需求有选择地修改行为地属性或延迟行为的执行。规则模板甚至支持将行为进行模块化，这样就能将不同的行为按照一定规则进行组合、循环或分支判断，以支持额外的活动或动态地更新模型的信任等级，如提升或降低模型的信任度。举例来说，行为模块化支持我们实现这样的操作，在特定应用程序触发了某些指定的隐私敏感

行为时通过 SMS 或电子邮件通知用户。

考虑到上述安全策略规则条件针对持续的和临时的操作，应用程序访问敏感资源和权限的频率也要相应地考虑在内。以位置信息的访问为例，即使一个应用程序只是调用了 `getLastKnownLocation()` 接口获取了单一的位置点而不是连续位置轨迹，也需要列入考虑。针对类似的情况，我们考虑了各种具体情形并开列如下：

- **针对不同应用的伪设备 ID**：用户可以允许应用程序在目的尚不明确的情况下获取到一个伪设备 ID 而不是真实的设备信息。然而为了保护用户的隐私并且防止应用程序之间共谋以获取用户的行为模式或集体监视用户行为，该伪设备 ID 针对每个不同的应用程序也应该完全不同；

- **部分信息访问**：可以以更细粒度的形式管理信息和权限的授予，比如当授权应用程序访问 SMS 消息时只允许应用程序访问到消息内容但无法获取消息的发件人或收件人；或者当用户授权应用程序访问通讯录时，只允许应用程序得到联系人姓名的哈希及其电话号码；

- **访问时间与频次**：考虑到用户可能有细粒度的需求，应该允许应用程序在限定的时间点或者以限定的时间频率访问特定的权限或隐私信息，比如每天允许应用程序访问 5 次运动传感器以获取必要的信息维持应用程序的基本功能；

- **用量与统计**：统计并向用户汇报相关应用程序对特定权限和隐私信息的请求数据，包括访问时间和访问频率。举例来说，每周向用户生成报表，记录特定应用程序在本周发生网络通信的时间点、频次和通信数据量，以及访问了哪些服务器；或者应用程序在一个会话中访问了哪些文件描述符；哪些应用程序在频繁地访问用户的位置信息。

3.1.4 行为模型的聚合

使用安全策略对应用程序行为进行细粒度控制的最大难点在于控制必须以清晰、精细且没有歧义的形式界定和量化。因此，安全策略生成机制需要对操作系统的安全机制和权限系统拥有及其深刻的理解。然而正如我们在第二章中所描述的那样，Android 操作系统的权限系统及其复杂，这就使得普通的终端用户无法清晰地表明他们的需求，也无法在某些场合精准确定某个应用程序的某项授权是否应该同意。使情况更糟糕的是，在很多场景下特定的操作可能会以不同的方式实现，它们或许调用了不同的系统 API，访问了不同的类和程序资源。这使得即使针对同样行为的限制也需要指定相当数量的不同策略。为了解决该问题，本研究提供了一种可扩展的模型，该模型能够在应用程序和其他交换隐私信息和敏感数据的系统组件之间进行高层次的抽象。针对 Android 操作系统中信息访问不同的实现方式，本研究也对每种抽象交互进行了优化和聚合。

以定位信息的管理为例，应用程序使用 Android 操作系统提供的 API 可以通过调用 `getLastKnownLocation()` 方法和定位信息组件进行通信。或者，一个应用程序也可以通过订阅/发布机制来向定位组件注册自己的监听访问申请，这样当应用程序实现回调方法 `onLocationChanged()` 时，它就可以指定 Android 的定位组件以一定的频率或时间间隔向自身发布定位的更新信息。不管采用哪种方式，从用户或组织的角度来说，具体的技术实现细节是无关紧要的，因为不管应用程序采用哪种方式获取到定位信息，都意味着用户的位置和运动轨迹面临着泄漏的风险。

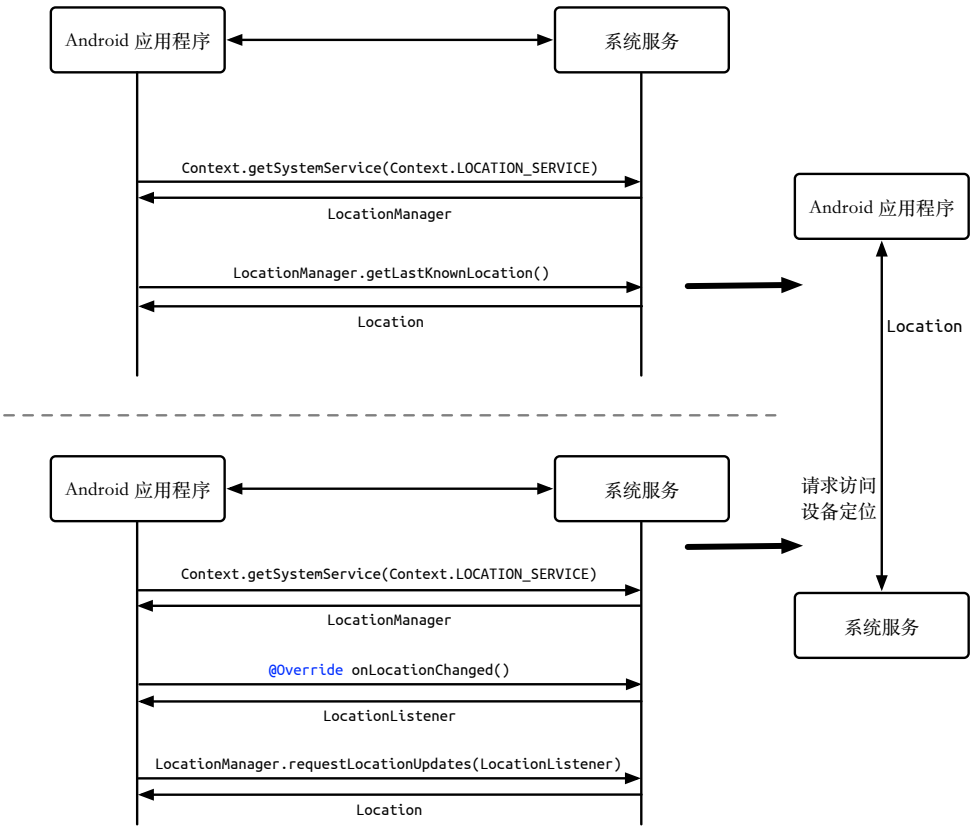


图 3-2 应用程序对定位信息访问的抽象

针对这样的情况，本研究提供了对用户需求进行上层抽象的解决方案。仍以定位信息为例，应用程序对定位信息的访问可以抽象为应用程序对 Android 系统的定位管理组件的间接或直接访问，并可能通过不同的路径读取设备的位置数据，如图 3-2 所示。本研究提出的模型中，对于一个抽象的交互模型集合 A，定义和该模型相关联的数据集实例为 D_A，定义行为实例对交互模型 A 所产生的影响程度为 R_A。这样的交互模型的最终结果使得声明的实例数据集能够用于所有参与该交互 A 的所有行为角色。

我们的策略生成规则考虑了两种类型的事件，分别是在发生具体交互之前触发的临时事件和在收到交互事件成功完成信号后触发的规律事件。当一个临时事件被触发时，策略规则假设所有的数据实例均处于可用状态，但尚未交换到交互参与者的手中。即尽管应用程序此时已经获取了相应数据集实例的访问能力，但尚未实际获得相应的数据

流。这样的设计允许系统进行预防性的授权代理操作，如强制允许、拒绝、修改或延迟授权。

对于一个抽象交互模型集合 A，可以进一步将其分解为一系列具体的交互行为 I_k。定义针对该具体交互的数据实例为 D_k^I。那么，该具体交互的数据实例 D_k^I 和原始模型数据集 D_A 之间存在的关系如式 3.1：

$$D_A \in \bigcup_{k=1}^{n} D_k^I \tag{3.1}$$

式 3.1 说明，具体交互行为集合必须至少建立起和抽象交互模型集合相同的数据实例集，而倘若具体的交互行为优化需要提升数据的粒度，还要在此之上包含其他数据实例。在很多情况下，指定的交互流程的起始行为和终止行为是已经被预先决定的，因此具体交互行为 I_k 还必须满足式 3.2 中的限制：

$$\begin{aligned} I_{start} &\in I_k \\ I_{end} &\in I_k \end{aligned} \tag{3.2}$$

举例来说，图 3-2 中给出的抽象定位信息访问行为可以实例化为两种具体的细粒度行为，但当这两种具体行为模型中不管哪种的交互最终完成时，抽象访问行为中定义的定位数据对特定应用程序来说都是可见的。

针对具体实现中使用的交互行为模式，本系统还将执行最终的行为优化和聚合。如之前所述，Android 操作系统中的交互行为可能被映射到订阅/发布机制，也可能被映射到调用/返回机制中，除此以外还有更多的数据通信机制。本系统针对上述机制的每一种都进行了方法调用级别的细粒度优化。举例来说，我们针对“获取设备最后定位位置”的数据访问需求分别定义了“主动获取设备最后定位数据”的方法调用和“订阅设备最后定位位置”的通知机制，这在图 3-2 中也得到了体现。

安全策略规则中通过 ECA 规则中的事件模式来代表交互行为。在 ECA 规则中，事件部分 E 只能包含 1 个交互模式，而条件部分 C 可能由 0 个或若干个交互模式使用策略运算符进行组合。如果操作部分 A 中包含权限授予或者权限撤销的操作，则事件部分 E 中定义的事件模式必须是一个临时事件，因为实际规律事件用于表示已经发生并且无法再进行干预的活动和事件。从策略执行的角度来说，考虑到行为模式容器正在执行安全策略，或者行为模式容器可能将行为的构成之一作为关注焦点，事件必须要拥有引用交互行为本身的能力。行为模式容器对行为组件的关注能力是必要的，因为事件的执行并不总是能够成功，有些情况下行为容器要求对特定抽象实体外部事件的控制权。考虑到在本文提出的方法并不对 Android 操作系统的源代码进行修改，因此执行的重点始终是和 Android 系统及其 API 直接通信的应用程序。这个因素使得本系统能够在应用程序中注入安全策略执行代码，而不去修改 Dalvik 虚拟机的执行策略。

3.1.5　安全策略的优化

为了将高级策略映射到较低级别的可执行策略，本研究参考了 Lee 等人的策略优化方法[38]。本文将细粒度的策略规范及其评估结果输入一个系统模型，该系统模型显式地包含了抽象活动及其各自的具体策略。在 Android 操作系统中，一个抽象行为总是可以被映射到特定的数据源和数据信宿。举例来说，一个应用程序需要访问用户的定位信息，该定位信息就是该访问行为的数据源。当获取到定位信息后，该应用程序或者通过网络将数据发往服务器，或者将数据写入本地文件或数据库，抑或将该数据进一步分发给其他应用程序组件，这些数据的使用方都被称为数据信宿。因此本系统可以实现这样的功能，在允许应用程序访问用户设备的定位信息的情况下精确的控制数据信宿的属性和其交互行为，限制信宿将数据向其他应用程序进行重新分配。为了实现这种功能，本系统需要对 Android 操作系统中信宿对数据源的所有分发操作进行梳理。这样，本系统就可以通过定义策略的方式限制数据的二次传输以及多次传输，封锁所有的数据重新分发通道，也可以针对性的限制某种特定的分发渠道，比如将数据写入外部存储或向互联网服务器传输。通过这种方式，本系统就能够预防恶意应用程序利用数据分发安全漏洞的攻击行为。

除了彻底拒绝某种权限敏感活动的运行之外，我们还必须要警惕对活动或行为的修改。我们的策略机制支持策略的嵌套和组合，这样的机制能够提供更计划的操作以保留抽象策略条件的所有原始语义。这样的优化操作没有在 3.1.4 小节的行为优化中实现而是在安全策略优化阶段实现，是因为本研究将所有活动都视作 Android 中的方法调用行为。因此，本研究针对具体敏感权限活动集合中的所有相关行为制定了细粒度的安全策略规则，采用更精细的策略来控制对活动或行为的修改。下面列出了本研究采用的具体的安全策略优化规则：

- 对于定义了拒绝授权的临时事件模式，在所有可能的初始具体交互行为集合 I_{start} 中，将其策略细化为一组事件模式 E 的引用。该规则能够保证所有可能的初始活动都将收到本系统的控制；
- 对于定义了允许授权或修改授权的临时事件模式，将其策略映射到一组在具体行为集合 I_k 中定义的行为中。其中数据实例 D_k^I 必须满足 $D_k^I \in D_{modify}$，其中 D_{modify} 指所有可能被修改的数据实例集合。该规则能够保证所有对授权的授权或修改都被限制在具体的活动中；
- 事件模式 E 或条件 C 中出现的所有事件都会被映射到具体终止交互行为 I_{end} 上。该规则保证了任何可能的抽象行为的完成信号都能成功地被策略规则所追踪与捕获。

本小节所使用的优化规则考虑到了交互行为可能传输的所有数据实例。对具体系统模型的分析和安全策略的制定必须在具体级别上予以落实。也就是说我们不可能在访问

定位权限的交互级别指定或限制应用程序的位置更新订阅行为的时间间隔，这是因为抽象交互并不能预先得知下层实现所具体采用的交互模式到底是订阅/发布模式还是调用/返回模式。

对于每个引用了抽象事件的抽象策略，本系统都提供了将其映射到具体事件的功能，这归功于 Android 操作系统的字节码分析技术能够识别出特定 Android 应用程序所使用的具体实现方法。这样，特定的具体事件就能够通过使用本节所提出的策略机制进行拦截。使用优化策略规则所生成的具体策略都将部署在代码的安全决策中心 SDC 中，安全决策中心是运行时执行和评估安全策略的中心引擎。策略实施组件通过一系列的策略执行器 PE 来代理事件，并向安全决策中心索取策略指定的权限操作来执行具体的策略。安全决策中心和策略执行器将以字节码的形式被注入到 Android 应用程序中，在后面的章节中我们还将详细介绍相关的字节码注入技术。

3.2 虚拟机字节码注入

为了将 3.1 节中阐述的安全策略应用到给定的 Android 应用程序中，本研究提出了一种基于 Android 虚拟机字节码注入的隐私保护技术。该技术能够根据用户或组织的安全需求修改任何给定的 Android 应用程序，从而将用户或组织制定的安全策略以字节码的形式嵌入应用程序，并在特定敏感权限请求或执行时进行相应的安全操作。

正如本文在 2.2 节中叙述的那样，早期所有的 Android 应用程序都使用 Java 编写。尽管最近几年 Google 官方鼓励开发者使用 Kotlin 语言进行 Android 开发，但是不管是 Kotlin 还是 Java 编写的应用程序，在编译阶段都将统一编译成 Android 系统专有的基于寄存器的 Dalvik 字节码。这样的 Dalvik 字节码针对资源有限的移动设备设计，能够在 Android 4.4 之前的 Dalvik 虚拟机和 Android 5.0 之后的 Android 运行时环境中执行。

和对 Java 应用程序的逆向工程类似，有许多工具提供了针对 Android 应用程序的逆向工程能力。比如著名的 ApkTool 能够对 Android 应用程序的安装包 APK 进行解包，并将其中的资源文件和代码文件分离[40]。smali/baksmali 这套工具提供了将 Android 应用程序二进制代码逆向为字节码的能力[41]。此外还有其他的一些工具如 Androguard，DECAF 等，都提供了针对 Android 应用程序的逆向工程工具[42][43]。

使用上述逆向工具，可以将 Android 应用程序编译产生的二进制文件转换成字节码的中间代码，这种字节码的语法和执行原理本文在 2.3 节中已有详细的介绍，在此不再赘述。这种字节码的中间表示具备一种折衷特性，既体现了 Android 应用程序的机器级执行流程和寄存器的访问原理，又提供了一定程度的人类可读性。这就使得当我们想要对 Android 应用程序进行特定修改时，不需要想方设法将应用程序的二进制代码还原回原始的 Java 代码或 Kotlin 代码。

只利用中间层的 Dalvik 字节码而不使用反编译源代码的另一个重要原因是，成熟的 Android 应用程序开发者出于应用安全角度和知识产权保护角度，通常会对应用程序源代码进行代码混淆、加固或代码加壳等操作，这极大地增加了将二进制代码或字节码

还原到 Java 或 Kotlin 源代码的难度。而即使将二进制文件或字节码还原回原始代码，其因代码混淆带来的极低可读性也使得这样的努力得不偿失。

下面的代码展示了对 Android 应用程序进行反编译的其中一个实例，该实例将一个应用程序的二进制文件反编译为 Dalvik 字节码的 smali 表示。在这段代码中，该应用程序试图调用 `TelephonyManager` 类中的 `getDeviceId(String)` 方法，该方法能够得到用户设备的唯一标识符，该标识符通常被应用程序用来对用户进行追踪和识别，以对用户行为进行建模或分析用户的行为偏好。

```
invoke-virtual {v0}, TelephonyManager;->getDeviceId()Ljava/lang/String;
move-result-object v1
iput-object {v1, v0}, Lcom/crumblejon/MainActivity;->imei:Ljava/lang/String;
```

3.2.1　安全策略的实例化

为了能够在任何用户或组织指定的位置注入安全决策中心和策略控制器的相应字节码级表示并应用安全策略，本系统首先需要将应用程序进行反编译并执行静态分析，以定位处理用户特定敏感信息或权限行为的相应 API 方法调用。我们使用 ApkTool 进行 Android 应用程序的解包工作，ApkTool 中包含的 smali/baksmali 组件能够提供将二进制代码文件反编译成 Dalvik 字节码的 smali 表示的能力[40]。生成的反编译代码将通过一个静态分析组件进行分析，以识别字节码中所有与特定权限行为或敏感信息访问行为相关的代码，并根据用户和组织的需求将安全策略嵌入到代码中。为此，该系统中使用的静态分析器必须有能力提取所有 Android 应用程序中的特定方法和权限访问列表，并将其与相应的敏感 API 列表对比。表 3.1 列出了 Android 应用程序中涉及敏感信息访问和权限请求的部分 API 方法调用。

表 3.1 部分敏感的 API 方法调用列表

权限	相关的 API 调用
MANAGE_ACCOUNTS	invalidateAuthToken()
VIBRATE	NotificationManager.notify()
READ_CONTACTS	ContentResolver.openInputStream()
WAKE_LOCK	MediaPlayer.start()
	MediaPlayer.stop()
ACCESS_NETWORK_STATE	getActiveNetworkInfo()
	getNetworkInfo()
INTERNET	URL.openConnection()
	URL.connect()

本系统会根据用户或组织制定的安全策略向应用程序植入特定的安全控制代码。为了实现动态性，该系统所做的尚不止于此。在代码植入时，除了用户或组织指定的安全策略，本系统还会检测应用程序中尚未被用户指定的敏感 API 调用或危险权限请求。这样，在嵌入了安全策略的 Android 应用程序运行时，当到达敏感 API 调用方法时，安全决策中心将首先检查用户或组织是否已经为该 API 调用指定了安全策略。如果用户针对该 API 调用确实声明了安全策略，安全决策中心将会通知策略执行器执行相应的安全策略代码；而如果用户未定义相关策略，安全决策中心将会略过该 API 调用并继续检测其他的 API 调用。

需要说明的是，未定义策略时嵌入的安全代码并不会对应用程序的性能造成过大的影响，应用程序将以几乎可以忽略的延时继续运行。基于这样的设计，本系统提出的字节码嵌入技术能够在用户或组织想要更改安全策略时动态地响应用户的需求，而不需要重新嵌入安全策略字节码或对应用程序进行重打包，因为我们在初次进行代码静态分析和字节码植入时已经考虑了所有可能的安全控制需求。关于嵌入安全策略导致的性能影响的相关评估，请参看第五章的相关内容。

注入到应用程序的安全代码，更具体地说就是一系列地策略执行器，将会以代理模式被添加到应用程序的相应位置中。通过将策略执行器以安全代理的形式封装起来，本系统可以极大减少对原始应用程序代码的侵入性，并为修改后的应用程序提供更高的健壮性和可用性。在静态分析阶段，本系统将通过分析应用程序的 AndroidManifest.xml 文件和应用程序的反编译字节码确定在该应用程序中所涉及到的所有敏感操作和权限申请，并针对性地抽取相应的策略执行器组成用于注入程序的运行库。随后，系统将在原始应用程序的敏感操作和权限申请涉及到的所有方法前后注入对执行器的调用，这些调用将根据用户或组织的声明来决定是否调用相关的策略执行器，并在相关权限操作之前或之后进行相应的安全操作。

仍以设备定位信息的控制为例，以下的 Dalvik 字节码展示了一个应用程序获取定位数据的方法单元。篇幅所限，我们并未展示该方法的所有代码，仅列出了该方法的关键逻辑部分。可以看到，开发者预先在类中声明了一个 `android.location.LocationManager` 类实例 `Xl`，用于向操作系统的定位管理器申请定位数据。随后该方法通过 `LocationManager` 类的 `getLastKnownLocation()` 方法获取到设备的最后位置。值得关注的是，该应用在打包时使用了代码混淆技术，因此方法名 `A()` 和字段名 `Xl` 都是无意义的字符串。

```
.method private A(Ljava/lang/String;)Landroid/location/Location;
    iget-object v0, p0, Landroid/support/v7/app/i;->
        Xl:Landroid/location/LocationManager;
    invoke-virtual {v0, p1}, Landroid/location/LocationManager;->
        getLastKnownLocation(Ljava/lang/String;)Landroid/location/Location;
    move-result-object v0
```

上述私有方法封装了定位数据的获取方式，当应用程序在业务逻辑中需要请求设备定位时将会申请访问该类的公有方法。下面的 Dalvik 字节码展示了该类的一个关键公有方法，该方法通过调用上述私有方法向应用程序的其他业务逻辑部分暴露出一个定位数据申请接口。

```
.method public gn()Landroid/location/Location;
    invoke-direct {p0, v0}, Landroid/support/v7/app/i;->
        A(Ljava/lang/String;)Landroid/location/Location;
    move-result-object v0
    const-string/jumbo v3, "android.permission.ACCESS_FINE_LOCATION"
    invoke-static {v2, v3}, Landroid/support/v4/content/f;->
        checkSelfPermission(Landroid/content/Context;Ljava/lang/String;)I
    move-result v2
    if-nez v2, :cond_0
    const-string/jumbo v1, "gps"
    invoke-direct {p0, v1}, Landroid/support/v7/app/i;->
        A(Ljava/lang/String;)Landroid/location/Location;
    move-result-object v1
```

上述代码定义了一个名为 `gn()` 的公有方法。该方法首先检查了 AndroidManifest.xml 清单中是否声明了 `ACCESS_FINE_LOCATION` 权限，并确认用户已经向应用授予了该权限。随后该方法会调用上面定义的 `A(Ljava/lang/String;)` 方法以获取设备的定位数据。

显然，公有方法 `gn()Landroid/location/Location;` 是该应用程序访问定位数据的关键接口。为了对该应用程序的定位数据访问行为进行控制，我们需要在 `gn()` 方法中注入安全策略控制代码。为了保证用户能够根据具体需求对应用行为进行不同程度的限制，我们注入的安全策略代码不会将控制逻辑固化到 `gn()` 方法单元中，而是通过 Java 反射策略调用安全决策中心来实现动态控制。需要注意的是，对 `gn()` 方法的修改将影响虚拟机寄存器的分配。正如 2.3.2 小节所述，我们需要对寄存器名和调用顺序进行调整。

```
new-instance v1, Lcom/crumblejon/SecurityPolicy;
invoke-direct {v1}, Lcom/crumblejon/SecurityPolicy; -> <init>()V
invoke-static {v1, p0}, Lcom/crumblejon/SecurityPolicy; ->
    loadProxy(Landroid/context/Content;Landroid/app/Activity;)
const-string/jumbo v2, "gps"
invoke-direct {p1, v2}, Landroid/support/v7/app/i;->
    A(Ljava/lang/String;)Landroid/location/Location;
move-result-object v2
```

图 3-3 展示了在一个 Android 应用程序的敏感操作前后注入安全策略执行代码的流程。在这个实例中，当应用程序执行到这个敏感操作时，策略执行器将会向安全决策中心请求安全策略。倘若安全策略中包含了对该敏感操作的限制规则，安全决策中心将会指示策略执行器进行相应的权限控制和安全操作。举例来说，倘若安全策略对该敏感操作涉及到的危险权限做出了禁止声明，策略执行器就会直接忽略对该方法的调用；类似地，如果安全策略指示对该敏感操作进行延时或条件修改，策略执行器也会执行相应的安全操作。上述操作都是指在敏感操作之前进行的安全代理，对于在敏感操作之后进行的安全代理，只能够进行对操作的修改或延时，因为敏感操作已经执行，无法再对其进行禁止。上述安全操作和本文在 3.1 节中介绍的发生具体交互之前触发的临时事件和在收到交互事件成功完成信号后触发的规律事件相匹配。

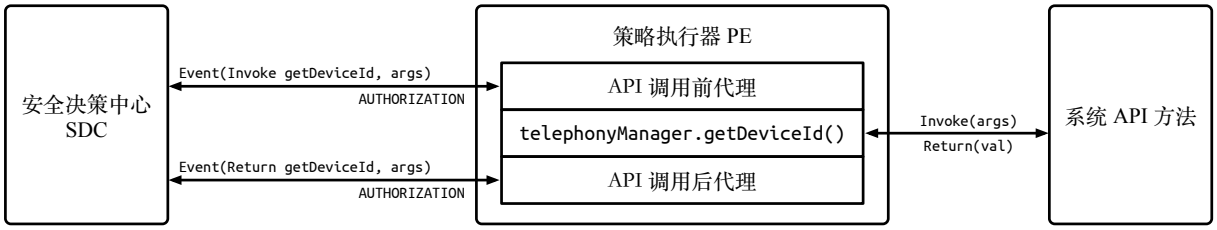


图 3-3 在敏感操作前后注入安全策略执行代码

当将安全策略注入到 Android 应用程序后，安全决策中心将在应用程序的主活动 (Main Activity) 中的 `onCreate()` 方法中初始化。尽管如此，在有些情况下仍然会因为设备、版本或其他原因而导致安全决策中心初始化失败而导致安全策略处于不可用的状态，这时被安全强化的应用程序将会询问用户是否重试初始化安全决策中心，倘若用户拒绝重试并试图在此情况下继续运行应用程序，当应用程序执行到敏感操作或危险权限调用时会对用户提出安全警告。此外在应用程序执行期间，由于软件资源、硬件问题或功耗等原因都可能导致安全策略中心变为不可用状态，这时为了保证安全性和健壮性，应用程序将在执行到安全敏感操作或权限申请时提示用户并让用户自行决定是否继续执行相关操作，并为应用程序接下来可能遭遇的安全决策征求用户的默认许可或拒绝。用户所进行的决策可以作为后续相关操作出现时的默认操作，系统将会缓存用户所给出的决策，并在下次出现类似决策时提供参考，以实现对于同一安全决策的一致性。

3.2.2 安全策略的动态调整

注入了安全策略的 Android 应用程序已经可以按照用户指定的安全需求完成相应的安全操作。但是随着场景需求、信任关系、用户身份与角色等因素发生变化，用户对安全策略的要求也在随时发生变化。此时就需要对安全策略进行动态地调整。如前所述，以往的研究在本阶段都束手无策，只能针对新的安全需求重新生成安全策略实例并将其重新注入应用程序中，最后将新的应用程序安装至用户的设备中。这种操作方式对安全策略的使用是灾难性的，考虑到个人或组织时常变动的安全需求，这种反复地重新编译

与打包所带来的时间成本和操作成本是无法容忍的。这也是之前的研究和设计难以投入使用的根本原因。

在本研究中，我们的系统在初次进行字节码注入时就会通过静态分析手段寻找应用程序中的所有敏感数据访问请求和危险权限操作，并在所有相关的 API 调用位置前后都嵌入了策略执行器。这就意味着只要安全决策中心给出特定的控制指令，所有敏感 API 调用都能够被本系统限制或直接屏蔽。这种动态控制的实现直接依赖于 Dalvik 虚拟机环境和 Java 语言的反射特性，正如 3.2.1 小节中代码所展示的那样。反射机制利用了虚拟机执行语言的自省能力，能够在程序运行时以数据流的方式提供对代码逻辑的控制。更具体地，基于 Dalvik 虚拟机的反射技术允许我们在程序运行时动态加载和使用类，并通过字符串形式的控制信息调整对方法、字段、类和接口的调用方式。反射技术可以将程序的以上逻辑单元变量化和参数化，使方法和类接口可以作为参数传输。因此我们只需预先定义所有可能的控制代码，应用程序在运行时就可以根据安全策略选取匹配的方法或接口。

为了帮助用户调整安全策略并向应用程序推送调整后的安全策略，本系统在安全决策中心中添加了网络通信模块，该网络通信模块能够以 HTTP 通信的形式向服务器请求安全策略，服务器也能够以 WebSocket 的方式向所有注入了安全策略的应用程序推送新的安全策略，使得所有程序都能够保持和最新安全策略的同步，如图 3-4 所示。

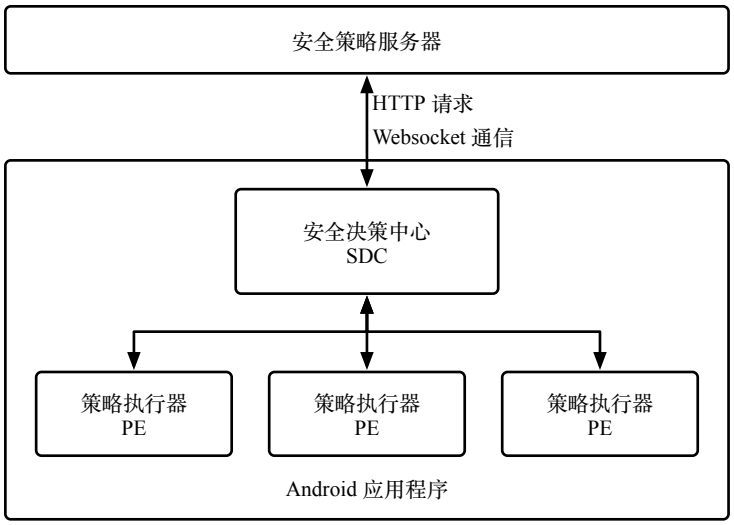


图 3-4 安全决策中心与服务器的通信

此外，安全决策中心可以控制所有策略执行器开启日志功能，并收集所有策略执行代理的执行日志。执行日志能够反映特定应用程序对敏感 API 或危险权限的调用时间、频率和时间间隔。通过分析应用程序的 API 调用规律也有助于识别并防护恶意应用程序和普通应用程序中滥用危险权限的行为。安全决策中心会以一定时间间隔将所收集到的日志发送到服务器中，这些日志将共享给相关用户或组织，帮助用户或组织更好地做出针对特定应用程序的安全决策。

3.3 本章小结

本章介绍了基于行为模型的安全策略动态生成机制和基于 Dalvik 虚拟机字节码注入的 Android 应用隐私保护机制。对于安全策略的生成机制，我们详细地从数据流、身份与角色、环境与场景、信任关系等方面对用户与应用程序的交互进行了行为抽象，并基于行为的抽象与聚合生成特定的安全策略。对于策略的应用和虚拟机字节码的注入，我们考察了 Android 应用程序的敏感操作和权限申请所涉及的 API 调用，通过静态分析的手段确定代码策略执行代理的注入位置，并通过安全决策中心和策略执行器进行交互。为了能够动态地调整或修改安全策略，我们还着重关注了安全决策中心与策略执行代理的动态性和灵活性，提出了通过解耦合的方式赋予安全决策中心动态更改策略执行代理行为的机制，避免了在用户或组织对安全策略进行更改时需要重新编译或打包应用程序的困境。

第四章　基于差分隐私的应用数据脱敏机制

本章提出了一种基于差分隐私技术的 Android 应用程序数据脱敏系统。本章首先会说明在第三章中介绍的 Android 应用隐私保护系统基础之上进行数据脱敏的必要性，并阐述现代 Android 应用程序所面临的另一个数据分析困境。接着，本章将详细介绍数据脱敏机制及其核心的差分隐私技术，重点探讨其数学原理以及可用性与健壮性。接下来本章还将详细介绍如何将差分隐私技术和数据脱敏机制应用于 Android 隐私保护系统中，并探讨差分隐私技术如何在加强 Android 应用安全性的同时保证应用程序的基本功能不受影响，并在此基础上保护基于数据挖掘和数据分析的现代 Android 应用程序的开发目的与商业模式。

4.1　Android 应用程序的数据挖掘与行为分析

和基于特定功能或功能集的传统应用程序不同，现代 Android 应用程序在向终端用户提供基础功能的基础上，更注重通过数据挖掘和行为分析对用户的使用习惯和行为模式进行分析。应用程序开发者关注用户的行为模式，一方面是为了能够向用户提供更有效的、高度定制化的内容或向用户推荐他们更有可能乐意接受的信息，以提升应用程序给终端用户带来的使用体验；另一方面是通过对用户与应用程序交互所产生的数据流进行分析，挖掘出用户在宏观上的使用偏好和兴趣位置，为应用程序及其背后的商业公司提供商业信息，构建更稳健的商业模式。

因此，大量的 Android 应用程序都在其代码中都包含了对用户行为习惯数据的收集和分析功能。举例来说，用户在不同的应用程序中所体现的电子消费记录、共享单车出行路径、网约车打车频率和常去位置、关注的微信公众号或微博账号、游玩特定游戏的时间和频率、个人资产的配置与理财习惯、贷款与保险信息等等，都将在量化后成为用户的行为标签，用于对用户的行为习惯进行刻画。在此基础上，通过用户在不同应用程序中所注册的姓名、居住地、手机号码、电子邮箱，或者设备唯一标识符等数据，不同的应用程序或商业公司甚至能够将用户在不同应用程序上的行为关联起来。这就使得用户在聊天软件中刚刚谈及对某种商品的需求，再打开网络购物应用时该应用就能够仿佛未卜先知一般向用户推荐这种特定的商品。

巨大的商业价值催动应用程序及其背后的商业公司不断提升数据挖掘能力和分析能力，机器学习和特征工程的技术进步也使得以往无法实现的精细化、定制化的大规模用户行为分析成为可能。此外，对应用程序的数据挖掘和分析还催生出了新的商业模式，一些应用程序在提供基础功能的同时还会将自己所收集的数据进行打包出售，供其他相关企业分析和使用。

商业公司和应用程序对用户数据的关注，其本意并非窃取用户的个人隐私或敏感数据。良性的应用程序也并不关心特定个体的数据特征，只是通过个体与数据的关联性进行更好的数据推荐和行为定制。尽管如此，应用程序的这种对用户行为数据敲骨吸髓式的分析不免让普通用户有个人隐私泄漏之虞。这种泄漏并不是因为单个应用程序的敏感数据暴露所带来的风险，而是在这种广泛的数据挖掘的商业模式下所暴露出的系统性风险。尽管数据掌握在良性商业公司手中一般不会导致数据被滥用，但是这些数据的丢失或泄漏仍将给普通用户带来安全风险。我们已经多次看到，商业公司或政府部门的数据库脱库导致大量数据被不法分子利用，恶意攻击者能够从数据集中进行另一种模式的挖掘，旨在得到具体用户及与其相关联的隐私信息。

上述系统性风险是根植于目前的应用程序设计和商业模式的。我们在第三章中所介绍的权限控制和隐私保护系统固然能够通过屏蔽所有用户敏感数据访问和危险权限请求来从根本上禁止应用程序对用户行为和信息的访问，从而从根本上斩断用户个体与特定行为集合的关联。然而这种粗粒度的反制对策也将使得大量良性应用程序的基本功能与可用性受到影响，甚至使得一些应用程序根本无法使用。因此我们需要寻找这样一种方法，它能对用户和应用程序交互所产生的所有数据流进行数据脱敏，切断应用程序数据流及其背后所隐藏的用户行为模式与特定用户个体的所有关联。

4.2 差分隐私技术及其数学原理

对于数据脱敏的技术目标，我们考察了已有的关于隐私保护和数据变形的相关研究。一些研究者提出对特定数据添加噪声或扰动，以这种方法干扰恶意攻击者对数据源的分析。以用户的定位信息或移动轨迹的保护为例，一些研究者提出在本地将虚假的定位数据掺入真实数据流中，应用程序和服务器通信时将本地的所有真伪数据发往服务器[44]。这种方法固然能够在很大程度上阻止恶意攻击者通过数据挖掘得到用户的真实隐私信息，但是正如我们在 4.1 节中叙述的那样，这种方法会对应用程序的基本功能造成不可逆的影响，甚至使得一些应用程序直接丧失功能。

在上述的数据扰动方法的基础之上，Dwork 等人提出了一种名为差分隐私的数据变形方法，该方法能够在数据可用性和机密性间达到较好的平衡[45]。差分隐私技术的核心目标在于，通过对数据添加扰动和噪声信息，使得数据分析者无法由数据集的任何子集通过任何手段推算出数据集和数据源的关联性，也即无法通过分析数据结果得到任何用户的准确信息。任何得到数据集的个体或组织，无论是善意的还是恶意的，都只能从数据集中获取到关于特定信息的统计结果，而不能得到关于任何单一个体的具体信息。为了实现这个目标，差分隐私技术通过在原始数据中添加扰动的方法保证当任何单个或多个个体的数据发生变化时，或者任何单个或多个实体加入或脱离数据集时，对数据集的特定宏观统计结果都不会发生显著变化。差分隐私技术不对数据使用方和恶意攻击者进行任何先验知识上的假设，即使攻击者掌握的先验知识发生变化，差分隐私技术仍然能够较好地保护数据集的隐私安全。

4.2.1　差分隐私

定义函数 K 为数据发布者在发布信息时使用的随机化算法，对于函数 K 来说，其输入是原始数据集 D，而其输出是经过扰动或随机化的待发布数据流。将数据集 D 视为数据行的集合，现有两个具体数据集 D_1 和 D_2，如果 D_1 和 D_2 之间最多只在一行上有所不同，更具体地，如果两个数据集中的一个是另一个的真子集，且更大的那个数据集只比另外一个数据集多一行数据，则称数据集 D_1 和 D_2 为**相邻数据集**。如表 4.1 和表 4.2 所示，两表之间只相差一行数据，且除该行外的其他数据均完全相同。由上述定义可知，表 4.1 和表 4.2 互为相邻数据集。

表 4.1 相邻数据集 D_1

ID	姓名	消费金额
d3820920	Lucia Smith	$0.99
23280677	Eugenia Thomas	$1.12
8ebfb2ae	Hayley Jones	$5.10
535b9313	Linette Brown	$0.99
479630ae	Jon Taylor	$2.25
71f15c4b	Julian Davis	$3.17
f0314dcd	Cynthia Wilson	$1.50
c82c1ae7	Kermit Evans	$3.41
36e03dad	Tobias Rodriguez	$4.65

表 4.2 相邻数据集 D_2

ID	姓名	消费金额
d3820920	Lucia Smith	$0.99
23280677	Eugenia Thomas	$1.12
8ebfb2ae	Hayley Jones	$5.10
535b9313	Linette Brown	$0.99
479630ae	Jon Taylor	$2.25
71f15c4b	Julian Davis	$3.17
c82c1ae7	Kermit Evans	$3.41
36e03dad	Tobias Rodriguez	$4.65

基于上述假设，我们可以这样定义**差分隐私**：对于在所有数据集中最多只有一个元素有所不同的相邻数据集 D_1 和 D_2，如果具体算法 S 和随机化算法集 K 之间满足 $S \subseteq Range(K)$，可以定义随机化函数 K 提供的 $\epsilon-$ 差分隐私满足：

$$\frac{\Pr[K(D_1) \in S]}{\Pr[K(D_2) \in S]} \leq e^{\epsilon} \tag{4.1}$$

在式 4.1 中，概率 $\Pr(X)$ 表示数据集 X 中某条或某些数据可经由对数据集进行数据变形、抽离、添加等方式推算得到的概率。满足式 4.1 定义的随机性算法解决了有关任何数据发布者或者恶意攻击者可能通过分析原始数据集从而得到和某个或某些具体用户相关信息的担忧，因为即使攻击者通过将某条或某些特定数据行抽离原始数据集，对数据集的特定统计结果也不会产生过大的影响，因此攻击者总是难以通过构造相邻数据集的方式来分析出特定个体与某些数据的相关性。在这种意义上，差分隐私机制提供了一种自治保证，它非常强有力的保证了数据分析者只能获取到针对数据集合的统计特性，而无法针对性地的得到具体数据和个体的相关性。这使得数据提供者和数据消费者能够各自独立的进行数据操作而无隐私泄漏之虞，并且数据的统计特性也与数据提供者

和数据消费者任何一方的算力与先验知识完全无关。

尽管如此，差分隐私机制也并不是对用户个体隐私的绝对保证。从理论上说，任何数据集只要表现出了统计特性上的某种规律或信息，其本身事实上已经损害了数据隐私的自然定义。这意味着对数据集的统计行为本身就涉嫌隐私的泄漏，只是程度差别而已。因此在工程实践中选用差分隐私机制作为数据隐私保护机制的意义在于，可以通过差分隐私系统保证关于特定信息的数据集不会产生于其使用目的无关的外部性影响，针对与其使用目的有关的信息，差分隐私机制能够控制其信息泄漏处在极其有限的范围内。

式 4.1 中的 ϵ 称为**差分隐私预算**，该参数用于衡量差分隐私系统对数据隐私保护的强度。考察式 4.1 我们可以发现，ϵ 的值越小，$\Pr[K(D_1) \in S]$ 和 $\Pr[K(D_2) \in S]$ 就越相似，差分隐私机制对数据隐私的控制能力就越强。然而对于隐私预算 ϵ 的取值并非越小越好，因为在提升隐私保护性能的同时，降低差分隐私预算意味着对原始数据添加的扰动增多。当差分隐私预算 ϵ 降低到一定程度时，经过变换的数据集将不再拥有可用性。因此在实际操作中需要经过对 ϵ 的多次尝试取值过程，以寻找到能够较好平衡隐私保护能力和数据基本功能的差分隐私预算。

需要说明的是，式 4.1 讨论的是随机性算法 K 的机制与行为，并且该行为是独立于良性数据统计者与消费者或者恶意攻击者对相关数据集的任何先验知识的。这意味着即使恶意攻击者已经了解了数据集中的某条或者多条数据，也无法从数据集中得到特定数据与特定个体的任何关系性信息，满足式 4.1 的任意随机性算法 K 都能够保证数据的隐私性和机密性。

此外，式 4.1 中定义的差分隐私算法也能够扩展到组隐私，或者某个个体对数据集共享了多条数据的情况。假设所有数据提供者中其中 c 个参与者担心属于他们这个较小集合的集体隐私被完整的数据集泄漏，即使这 c 个参与者中任何单个个体的信息都没有被泄漏，集体性的保密需求仍然是客观存在的。使用式 4.1 给出的定义，我们可以将概率限制扩展到 $e^{c\epsilon}$，这种差分隐私预算的扩展是能够容忍的，因为它对数据可用性的影响等同于单个数据情形时的原始差分隐私预算情况。当然，对大规模数据集的利用通常是基于统计信息的，而统计数据集的重点是只公开宏观的汇总信息，与此同时保护具体个体的隐私信息，因而隐私的范围会在可预见的情况下随着宏观数据的增加而渐次瓦解。

4.2.2 噪声机制

考虑工程实践上常见的对数据集的使用方式，数据消费者的所有针对宏观数据的处理都可以理解为对数据集的查询、统计、变形、筛选等行为的线性组合。对于这些交互映射 f，当其作用于数据集 D 时，可以将其作用的结果表示为 $f(D)$。在这样的场景中，式 4.1 中所提出的随机化算法 K 将适当选择随机噪声扰动并将其添加到真实数据中，这种对原始数据添加噪声的方法被称为**扰动响应**。通过扰动响应来保护隐私的方法并不是一个新鲜的思路，正如本节开头叙述的那样，已经有很多向原始数据中掺入噪声

的研究，但是这些方法都会极大地影响数据的可用性。举例来说，如果加入噪声的数学模型是关于均值 μ 以方差 σ 呈高斯分布的，那么当多次对数据集进行相同的查询时必须对扰动响应进行平均，从而消除噪声可能对结果产生的影响。

基于以上分析，对于 $f : D \to R^k$，其中 D 为数据集，R^k 为映射 f 所输出的 k 维向量，行为映射 f 的**全局敏感度**由公式 4.2 定义：

$$\Delta f = \max_{D_1, D_2} ||f(D_1) - f(D_2)||_1 \tag{4.2}$$

其中 D_1、D_2 为数据集中的任意子集，且 D_1 与 D_2 之间互为相邻数据集。特别地，当 $k = 1$ 时，映射 f 的敏感度是映射 f 在只有一个元素不同的两个数据集之间取值的最大差值。

对于很多对数据集的查询和统计行为来说，其全局敏感度 Δf 并不会非常大。举例来说，当统计数据集中的所有数据项中有多少行具有属性 A 时，这种简单计数查询的全局敏感度 Δf 就为 1。我们也可以很直观的想到，当全局敏感度较小时，差分隐私机制的效果最好，这是因为虽然此时引入的噪声扰动相对较小，但数据集已经获得了较好的隐私保护。需要再次提醒的是，全局敏感度 Δf 是行为映射 f 的固有属性，只与操作的类型和特征相关，而独立于数据集大小与数据特征。全局敏感度从本质上反映的是，针对特定操作，数据消费者通过向数据集中添加噪声能够在何种程度上消弭相邻数据集间的统计学差异。

在考察对数据集进行扰动的噪声机制之前，我们首先需要考察拉普拉斯概率分布函数。如果随机变量 ξ 的概率密度函数形如：

$$f(x) = \frac{1}{2\lambda} e^{-\frac{|x-\mu|}{\lambda}} \tag{4.3}$$

其中 λ, μ 为常数，$\lambda > 0$，则称随机变量 ξ 服从参数为 λ, μ 的拉普拉斯分布。可以证明：

$$\begin{aligned} \int_{-\infty}^{\infty} f(x)dx &= \frac{1}{2\lambda} \int_{-\infty}^{\infty} e^{-\frac{|x-\mu|}{\lambda}} dx \\ &= \frac{1}{2} \int_{-\infty}^{\infty} e^{-|t|} dt \\ &= \int_{0}^{\infty} e^{-t} dt \\ &= 1 \end{aligned} \tag{4.4}$$

因此可以证明 $f(x)$ 定义了一个概率密度函数。图 4-1 给出了当 $\mu = 0$，λ 取不同值时的拉普拉斯分布概率密度曲线。

因此，对于标准差为 $\frac{\sqrt{2}\Delta f}{\epsilon}$，均值为 0 的拉普拉斯分布，其值在 x 处和 $e^{-|x|\frac{\epsilon}{\Delta f}}$ 成比例，定义：

$$Lap(\frac{\Delta f}{\epsilon}) = e^{-|x|\frac{\epsilon}{\Delta f}} \tag{4.5}$$

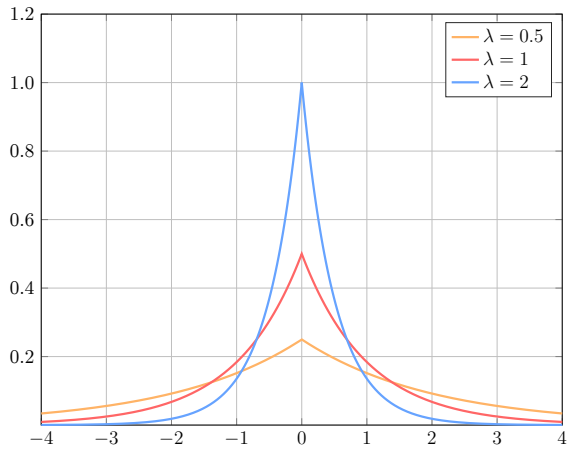


图 4-1 拉普拉斯概率分布曲线

如果令 $b = \frac{\Delta f}{\epsilon}$，则此时概率密度函数可以表达成：

$$p(x) = \frac{e^{\frac{-|x|}{b}}}{2b} \tag{4.6}$$

同时，其累积分布函数可以表达为：

$$D(x) = \frac{1}{2}(1 + sgn(x)(1 - e^{\frac{|x|}{b}})) \tag{4.7}$$

基于以上对拉普拉斯分布和拉普拉斯噪声机制的定义，对于 $f : D \to R^k$，其中 D 为数据集，R^k 为映射 f 所输出的 k 维向量，当映射 f 的全局敏感度为 Δf 时，如果映射 f 的响应结果满足：

$$R(x) = f(x) + (Lap(\frac{\Delta f}{\epsilon}))^k \tag{4.8}$$

则称响应 $R(x)$ 满足**拉普拉斯噪声机制**。拉普拉斯噪声机制的含义是，差分隐私技术机制对原始映射所得到的响应 $f(x)$ 的所有 k 个不同阶分量添加具有拉普拉斯分布 $Lap(\frac{\Delta f}{\epsilon})$ 的扰动。需要强调的是，由式 4.5 可知，当降低差分隐私预算 ϵ 时，拉普拉斯分布 $Lap(\frac{\Delta f}{\epsilon})$ 的曲线将会变得平坦，这将会产生更大的预期扰动响应。当差分隐私预算 ϵ 固定时，如果映射 f 全局敏感度 Δf 提升，其对应的拉普拉斯分布曲线也会变得更加平坦，从而产生类似的扰动响应增幅效果。

简单起见，我们现在说明当式 4.8 中的 $k = 1$ 的情况。关于随机化算法 K 在单个数据集映射 f 上能够产生 $\epsilon-$ 差分隐私的证明是显然的，不再赘述。考虑任何 K 的子集 $S \subseteq Range(K)$，并令相邻数据集 D_1 和 D_2 至多只在一行数据上有所不同。当选取的数据子集为 D_1 时，任意 $r \subseteq Range(K)$ 所对应的概率密度函数和 $e^{-|f(D_1)-r|\frac{\Delta f}{\epsilon}}$ 成正比，当选取的数据子集为 D_2 时情况也是如此。因此如果将三角不等式应用到指数中，我们可以得到最大的比率为 $e^{-|f(D_1)-f(D_2)|\frac{\Delta f}{\epsilon}}$。根据式 4.2 中关于全局敏感度的定义，必然有：

$$|f(D_1) - f(D_2)| \leq \Delta f \tag{4.9}$$

因此该比率的上限为 e^{ϵ}。这样我们就说明了拉普拉斯噪声机制在 $k = 1$ 的简单情况下必然能够提供 $\epsilon-$ 差分隐私。

容易看出，对于任何实现了随机化算法 K 的映射序列 $f_1, f_2, \ldots, f_d$，若上述序列中扰动响应均满足拉普拉斯分布 $Lap(\sum_i \frac{\Delta f_i}{\epsilon})$，则该序列必然能够提供 $\epsilon-$ 差分隐私。这也意味着，上述每个数据集访问行为映射 f_i 的质量都会随着映射的敏感度 Δf 的提升而劣化。对于上述性质的概括，我们有这样的定理：

定理　对于 $f : D \to R^k$，其中 D 为数据集，R^k 为映射 f 所输出的 k 维向量，将具有概率分布 $Lap(\frac{\Delta f}{\epsilon})$ 的各自独立产生的扰动添加到 k 个响应的任意随机化算法 K_f 都能够提供 $\epsilon-$ 差分隐私。

上述定理中的随机化算法 K 对于全局敏感度较低的映射 f 具有极高的准确度。需要特别说明的是，确保差异性所需的噪声机制必须仅取决于映射的全局敏感度 Δf 和差分隐私预算 ϵ，而这两者都是独立于数据集的规模和数据特征而存在的。因此，如果数据集的规模很大，由差分隐私机制引入的对数据集的行为映射，如统计、筛选、查询、变形等操作所带来的误差就相对较小。

也正因此，我们可以将随机化算法 K 视作数据集消费者和数据集本身的差分隐私保护接口。上述结论也给对用户数据进行宏观统计分析的数据集消费者提供了一种用于保护隐私的思路，就是将大的数据映射分解为操作较小、全局敏感度较高的查询算法。这样，即使数据集的规模相当庞大，或者所需要进行的映射或操作及其复杂，该机制也能够为许多标准的数据挖掘任务或统计任务提供精确且富有差异化的差分隐私。上述数据挖掘任务包括但不限于主成分分析，k 均值聚类分析、ID3 决策树、多层感知机以及人工神经网络等[46] [47]。

对于应用程序用户的数据收集和挖掘，除了上述数据挖掘任务外，商业机构和应用开发者还经常对数据集进行直方图分析。直方图分析是将数据集中的数据行任意划分为不同且不相交的单元，其任务是为每个上述的单元描述该单元中所有数据行关于某个属性或某组属性的统计计数。尽管可以将具有 k 个单元的直方图分析理解为 k 个单独的数据行映射行为，但是对单个数据行的添加或删除行为会影响最多 1 个位置中的整个 k 元组。由于该单元的统计特征最多收到的影响大小为 1，因此根据式 4.2 的定义，每个直方图分析中具体映射的全局敏感度均为 1。

4.3　基于高斯过程的差分隐私技术

本文在 4.2 节中详细介绍了差分隐私机制及其数学原理，着力证明了差分隐私技术能够提供数据集中数据的匿名化，并切断具体数据内容和对应数据源个体的关联。通过引入全局敏感度 Δf 和差分隐私预算 ϵ 两个衡量指标，对数据集进行脱敏的随机化算法

能够以科学而有效的方法被衡量。

尽管如此，传统的基于拉普拉斯扰动响应的差分隐私系统仍然存在问题。具体而言，有研究指出，基于拉普拉斯噪声机制的差分隐私机制容易受到少数异常值的影响从而极大劣化对数据集的脱敏效果。为了克服异常值的影响，必须向数据集中添加更强的扰动噪声来确保差分隐私的实现，然而这种行为会进一步地影响数据集的可用性，从而导致数据消费者陷入两难境地[48]。因此在本小节我们将引入一种基于高斯过程的差分隐私机制，该机制执行了一种超参数优化方法并将稀疏近似和分类结果相结合，能够在高维分类问题上发挥较大作用。

4.3.1 隐蔽方法

高斯过程是一组随机变量的集合，这组随机变量中的任何有限子集都满足高斯分布。通过假设数据集满足高斯分布特征，数据分析者可以使用回归分析方法对数据集进行数据拟合。事实上自然界中的大部分未经人为修改的数据往往自然满足高斯分布的特征。这样，以对子集中的一部分的观察作为条件，分析者就可以对数据的其余部分或者有相似特征或来源的数据进行预测。

在高斯过程的回归分析过程中，数据项之间的协方差 $\boldsymbol{y}_*$ 仅仅取决于训练数据 X 和测试数据 X_* 分别在训练集和测试集中的位置，除此之外，高斯过程的均值还取决于训练数据的输出值 $\boldsymbol{y}$。具体来说，$\boldsymbol{y}_*$ 和 $\boldsymbol{y}$ 之间有如下关系：

$$\boldsymbol{y}_* = K_{*f}K^{-1}\boldsymbol{y} \tag{4.10}$$

其中 K_{*f} 表示测试输入和训练输入的协方差，而 K 表示训练数据项之间的协方差。在接下来的分析中，我们将上述两者的乘积所产生的矩阵称为**隐蔽矩阵**，即：

$$C = K_{*f}K^{-1} \tag{4.11}$$

隐蔽矩阵描述了当训练数据发生变化时测试数据将会如何反应。

在以上描述中被忽略的一个方面是对特定数据的观察和分析行为如何链接到其可能存在的潜在信息。在上面的示例中，我们可能假设观测值是行为映射上的点，而其受到高斯观测噪声的干扰。这样的假设使得我们可以默认观测数据集是满足高斯特征的后验分布。然而这种假设并不能代表普适情况，如果输出表示的是观察的分类，此时随机变量之间的分布将不再是高斯分布。我们固然可以通过一个连接函数将行为映射进行压缩以使其满足高斯分布特征，然而这种强行的数据变换将会给数据的可用性带来较大影响。

本文在 4.2 节中介绍差分隐私机制时主要以标量数据作为说明重点，但是差分隐私机制同样也能够应用于特定向量集[49]。若定义 M 为一个描述高斯过程的协方差矩阵，我们将从其中采样归一化后的差分隐私扰动，假若结果向量描述了相邻数据集 D_1 和 D_2，对于给定的 $\boldsymbol{y}_*$ 和 $\boldsymbol{y}'_*$，可以得到下述公式定义的边界：

$$\sup_{D_1 \sim D_2} ||M^{-\frac{1}{2}}(\boldsymbol{y}_* - \boldsymbol{y}'_*)||_2 \leq \Delta B \tag{4.12}$$

上式中 ΔB 是输出变化的边界，是以马哈拉诺比斯距离衡量的增加的噪声协方差。该算法的输出 $\tilde{\boldsymbol{y}}_*$ 满足：

$$\tilde{\boldsymbol{y}}_* = \boldsymbol{y}_* + \frac{c(\sigma)\Delta B}{\epsilon} Z, \quad Z \sim N_d(0, M) \tag{4.13}$$

这样，当 $c(\sigma)$ 满足：

$$c(\sigma) \geq \sqrt{2\log\frac{2}{\sigma}} \tag{4.14}$$

时，该过程提供 $(\epsilon, \sigma)-$ 差分隐私。

基于上述推导，在实践中我们希望 M 在尽可能小的同时仍然在训练集数据项变化影响最大的那些方向上具有较大的协方差，因为在这样的情况下能使得 ΔB 尽可能小。对于式 4.12 所定义的边界 ΔB，可以在此基础上提供一种**隐蔽方法**。更具体地说，这种所谓的隐蔽方法是一种能够使得高斯过程回归过程所训练的模型满足差分隐私的方法。假设我们能够对数据集 $\boldsymbol{y}$ 中的第 i 条数据施加的最大扰动噪声为 d，显然有：

$$y'_i = y_i + d \tag{4.15}$$

预测的变化仅仅取决于式 4.11 所定义的隐蔽矩阵 C 的其中一列 $\boldsymbol{c}_i$，并且有：

$$\boldsymbol{y}'_* - \boldsymbol{y}_* = d\boldsymbol{c}_i \tag{4.16}$$

将式 4.16 代入式 4.12，使用矩阵 M 的对称性质，可以得到：

$$||dM^{-\frac{1}{2}}\boldsymbol{c}_i||_2 = d^2\boldsymbol{c}_i^T M^{-1}\boldsymbol{c}_i \tag{4.17}$$

因此在工程实践中，我们希望找到这样的一个协方差矩阵 M，它能在使得扰动噪声 Z 较小的同时让边界 ΔB 也尽可能小。对于扰动噪声，我们可以使用：

$$\frac{1}{2}\ln((2\pi e)^k|\Sigma|) \tag{4.18}$$

来描述。现有的研究[48] 已经给出了在此情况下 M 的最优解为：

$$M = \sum_i \lambda_i \boldsymbol{c}_i \boldsymbol{c}_i^T \tag{4.19}$$

其中的 $\boldsymbol{\lambda}$ 可以由梯度下降法找到，$\boldsymbol{\lambda}$ 满足：

$$\frac{\partial M^{-1}}{\partial \lambda_j} = -\mathrm{Tr}\left(M^{-1}\boldsymbol{c}_j\boldsymbol{c}_j^T\right) + 1 \tag{4.20}$$

4.3.2 非平稳核

将数据消费者对数据集的使用行为进行抽象，考虑现有一个由特定应用程序产生的关于特定信息的数据集，数据集中可以抽象为公共输入矩阵 X，针对数据集进行特定统计、筛选、变形和模型学习得到的私有输出向量为 $\boldsymbol{y}$。现在假设数据集消费者希望使用该数据集作为训练集来预测 $\boldsymbol{x}_*$ 处所对应的输出值，并且使得任意第三方无法从输出中获取有关 $\boldsymbol{y}$ 中任何单个数据项的详细信息。

仅仅使用标准的基于拉普拉斯噪声机制的差分隐私方法在面对这种问题时常常表现平庸，因为为了实现差分隐私而向数据集中添加的扰动噪声往往在相邻数据集的集中区域异常的大，这主要是因为数据集中对异常值产生了影响。正如我们在 4.2 节中所分析的那样，这种过大的扰动噪声往往会极度劣化数据集的可用性，而随着数据集维度的扩展，每个数据项成为异常值的概率都将成比例提升。在数据集的异常区域中，只有少量的数据项能够项训练模型提供有效帮助，而为了保护各个数据项，差分隐私系统将不得不添加更多的扰动噪声来保护各个数据项。

事实上，通过操纵由高斯过程执行的聚合能够在相当程度上减轻上述影响。考虑隐蔽矩阵 C 的结构，该矩阵简单描述了每个数据项对训练模型的有效贡献程度。假设我们在特定数据项的领域中由 n 个外围数据项集中在一起，那么显而易见的是这组数据项在隐蔽矩阵 C 中的关联项都比例近似为 $\frac{1}{n}$。这意味着他们将会有效地对训练模型进行平均，也意味着如果更多的数据项落在该集合内，则每个数据项提供的贡献都将成比例减少。虽然真实的数据项之间往往具有更复杂的协方差关系，然而这种类似的数学关系是客观存在的，即如果更多的数据项落在位于测试训练项的特定尺度范围内，则单个数据项对训练模型的贡献程度将成比例减弱。

基于这个结论，可以考虑采用一个非平稳的长度尺度来在高斯过程聚合中位置近似数量的数据项，从而有效地控制训练项位置附近的数据密度。然而这种粗粒度的方法无法适应过于复杂的数据集模型，因此作为该方法的一个升级方案，我们考虑使用输入数据集的稀疏近似，将其置于高数据项密度的区域来实现类似的目的，这种方案也能够有效减少异常离群值对模型的贡献程度。上述两种方法都可以视为非平稳核的一种形式，因为它们都通过变换数据项位置的方式来操纵数据项对模型所施加的具体影响。

因此，可以通过使用非平稳的长度尺度来调整训练数据的权重，该长度尺度可以根据数据项密度进行调整。那么问题的重点就在于如何选定这个特定的非平稳长度尺度。事实上，这里的长度尺度由训练数据集的输入位置确定，，有研究指出如果数据消费者已经对每个数据项位置的长度尺度有了先验知识，则可以据此产生一个正定的协方差矩阵[50][51]。对于隐蔽矩阵 C，可以由特定数据项 $\boldsymbol{x}_i$ 和 $\boldsymbol{x}_j$ 和第三点 $\boldsymbol{u}$ 的协方差进行积分决定：

$$C(\boldsymbol{x}_i, \boldsymbol{x}_j) = \int_{\mathbb{R}^d} k(\boldsymbol{x}_i, \boldsymbol{u}) k(\boldsymbol{x}_j, \boldsymbol{u}) d\boldsymbol{u} \tag{4.21}$$

其中 $k(\boldsymbol{x}, \boldsymbol{u})$ 是关于 a 的协方差函数。接下来的工作是选定长度尺度函数 $l(x)$，该函数

能够给出特定位置的长度尺度。如前所述，如果我们已经知道了特定位置 x 处的数据项密度 $\rho(x)$，那么我们大致希望长度尺度函数近似表现为:

$$l(x) = \frac{n}{\rho(x)} \tag{4.22}$$

其中 n 是我们所期望的训练集在 x 的单个邻域内的数据项个数，该邻域的尺度参考最大扰动噪声 d。为了防止长度尺度出现任意膨胀的情况，还需要对长度尺度进行归一化。可以向式 4.22 的长度尺度中添加一个修正参数 m^{-1}，其中 m 是长度尺度的上界，这样长度尺度可以表示为:

$$l(x) = [m^{-1} + \frac{\rho(x)}{n}]^{-1} \tag{4.23}$$

实际实验表明，这种设计的效果较差，主要原因正如上面我们所分析的，因为较长的长度尺度区域已经和高密度的较短长度尺度区域完全分离，这使得这种粗粒度的方法无法适应过于复杂的数据集模型。

在训练模型的输入空间上修改内核的另一种方法是使用稀疏高斯过程，该方法能够有效降低距离训练项较远的数据项的权重。因此通过简单的将训练项选择位置倾向数据项高密度区域，就可以显著降低数据集中离群异常值的影响，进而降低维护差分隐私所需的扰动噪声。除此之外，该方法还能够允许我们继续使用之前选定的协方差函数和对应的参数而不需要重新调试。尽管如此，这种方法还是需要审慎选择训练项的数量和位置，并且还会因为位置和数量的选择而引入对数据密度较高区域数据的倾向性偏差。但无论如何，高斯过程的稀疏近似确实能够有效降低异常值的影响并显著提升差分隐私扰动噪声对个体信息的隐蔽特性[52]。

基于上述分析，基于稀疏高斯过程的差分隐私系统中的预测值分布是一个正态分布，其均值和方差满足:

$$\mu_* = \boldsymbol{k}_*^T Q_{MM}^{-1} K_{MN} (\Lambda + \sigma^2 I)^{-1} \boldsymbol{y} \tag{4.24}$$

$$\sigma_*^2 = K_{**} - \boldsymbol{k}_*^T (K_{MM}^{-1} - Q_{MM}^{-1}) \boldsymbol{k}_* + \sigma^2 \tag{4.25}$$

其中:

$$Q_{MM} = K_{MM} + K_{MN} (\Lambda + \sigma^2 I)^{-1} K_{NM}, \quad \Lambda = \text{diag}(\boldsymbol{\lambda}) \tag{4.26}$$

$$\lambda_n = K_{nn} - \boldsymbol{k}_n^T K_{MM}^{-1} \boldsymbol{k}_n \tag{4.27}$$

式 4.24 - 4.27 中 K_{NM} 表示训练集和输入集之间的协方差，K_{MM} 表示输入集之间的协方差，$\boldsymbol{k}_{*m}$ 表示特定测试和输入向量间的协方差，$\boldsymbol{k}_n$ 表示训练集和特定训练数据项 n 间

的协方差，K_{nn} 指特定训练数据项 n 的方差。正如之前所述，所有方差项均只取决于输入数据，因此在这种情况下我们只需要关注输入数据的均值即可。基于上述条件，隐蔽矩阵 C 可以表示为：

$$C = \boldsymbol{k}_{*m}^{T} Q_{MM}^{-1} K_{MN} (\Lambda + \sigma^2 I)^{-1} \tag{4.28}$$

基于 4.3.1 小节中介绍的隐蔽方法，我们可以使用式 4.20 介绍的梯度下降法找到最佳的 M。

4.3.3 高斯过程与差分隐私

下面我们将讨论如何将将高斯过程和差分隐私机制结合，以解决数据挖掘过程中的隐私问题。在基于高斯过程的分类问题中，假设存在映射 $f(x)$ 通过压缩向我们提供了类似 $\pi(x)$ 特性的概率分布。尽管这种似然估计可能导致在解析上的不可积分，但我们仍然可以通过数值或者解析近似方法来对其进行近似，比如使用拉普拉斯近似方法。这恰好和差分隐私机制暗合，通过使用拉普拉斯近似法只需要较少的迭代次数就能快速收敛，并且此方法中训练输出和迭代更新方向之间存在清晰明确的关系。更为重要的是拉普拉斯近似法的形式非常适用于差分隐私机制。为了应用拉普拉斯近似法，可以使用下面的近似估计[53]：

$$q(\boldsymbol{f}|X, \boldsymbol{y}) = N(\boldsymbol{f}|\hat{\boldsymbol{f}}, A^{-1}) \tag{4.29}$$

式 4.29 使用近似值 $N(\boldsymbol{f}|\hat{\boldsymbol{f}}, A^{-1})$ 替换了精确的后验分布 $q(\boldsymbol{f}|X, \boldsymbol{y})$。式 4.29 中的 $\hat{\boldsymbol{f}}$ 表示 p 的估计值，而 A 是对数后验的黑塞矩阵。显然 $\hat{\boldsymbol{f}}$ 并不存在解析解，因此只能通过适当算法通过迭代法不断更新 $\hat{\boldsymbol{f}}$ 以找到合适的估计值：

$$\hat{\boldsymbol{f}}' = (K^{-1} + W)^{-1}(W\hat{\boldsymbol{f}} + \nabla \log p(\boldsymbol{y}|\hat{\boldsymbol{f}})) \tag{4.30}$$

式 4.30 中 K 表示训练数据项之间的协方差，W 是一个对角阵：

$$W = -\nabla\nabla \log p(\boldsymbol{y}|\hat{\boldsymbol{f}}) \tag{4.31}$$

W 中的元素为 $-\pi_i(1-\pi_i)$，且 π_i 满足：

$$\pi_i(\hat{\boldsymbol{f}}) = p(y_i = 1|f_i) = (1 + e^{-f_i})^{-1} \tag{4.32}$$

为了使得式 4.30 所表达的迭代更新满足差分隐私特性，需要讨论的是如何在迭代过程中使用训练输出向量 $\boldsymbol{y}$ 更新 $\hat{\boldsymbol{f}}'$。当模型训练使用 logistic 回归函数时，梯度项为：

$$\nabla \log p_i(\boldsymbol{y}|\hat{\boldsymbol{f}}) = t_i - \pi_i(\hat{\boldsymbol{f}}) \tag{4.33}$$

其中 $\boldsymbol{t} = \frac{\boldsymbol{y}}{2} + \frac{\mathbf{1}}{2}$。

对于隐蔽矩阵 C，依据式 4.30 可以定义：

$$C = \frac{1}{2}(K^{-1} + W)^{-1} \tag{4.34}$$

这样，基于式 4.33 和式 4.34 的定义，可以将式 4.30 改写为：

$$\begin{aligned} \hat{\boldsymbol{f}}' &= 2C(W\hat{\boldsymbol{f}} + \frac{\boldsymbol{y}}{2} + \frac{\mathbf{1}}{2} - \pi(\hat{\boldsymbol{f}})) \\ &= 2C(W\hat{\boldsymbol{f}} + \frac{\mathbf{1}}{2} - \pi(\hat{\boldsymbol{f}})) + C\boldsymbol{y} \end{aligned} \tag{4.35}$$

式 4.35 中的最后一项 $C\boldsymbol{y}$ 是唯一的和 $\boldsymbol{y}$ 相关的项，而 K 和 W 都与 $\boldsymbol{y}$ 无关。因此在 $\boldsymbol{y}$ 发生变化时只需要考虑其对 $C\boldsymbol{y}$ 项造成的影响。将 4.3.1 小节中介绍的隐蔽方法和式 4.34 中定义的隐蔽矩阵相结合，就可以得出在差分隐私条件下的 $\hat{\boldsymbol{f}}$。

为了基于上述近似估计建立数据分析模型，我们需要计算似然映射 $\hat{\boldsymbol{f}}$ 在 $\boldsymbol{x}_i$ 位置的均值和方差。为了完成计算，首先需要计算测试数据项和训练输入数据项之间的协方差 $\boldsymbol{k}_*$。基于协方差值，有研究表明[54] 映射 $\hat{\boldsymbol{f}}$ 在测试数据项 $\boldsymbol{x}_i$ 位置的均值可以表示为：

$$\mathbb{E}_q[f_*] = \boldsymbol{k}_*^T \nabla \log p(\boldsymbol{y}|\hat{\boldsymbol{f}}) \tag{4.36}$$

但是 4.36 中均值的表示形式仍然和 $\boldsymbol{y}$ 相关，这意味着如果依据该公式进行模型训练，仍有泄漏个体隐私的可能性。为了进一步避免模型对隐私的泄漏，可以利用这样的近似[54]：

$$\nabla \log p(\boldsymbol{y}|\hat{\boldsymbol{f}}) = K^{-1}\hat{\boldsymbol{f}} \tag{4.37}$$

这样我们就可以得到后验均值和方差：

$$\mathbb{E}_q[f_*] = \boldsymbol{k}_*^T K^{-1}\hat{\boldsymbol{f}} \tag{4.38}$$

$$\mathbb{V}_q[f_*] = k(\boldsymbol{x}_*, \boldsymbol{x}_*) - \boldsymbol{k}_*^T(K + W^{-1})^{-1}\boldsymbol{k}_* \tag{4.39}$$

考虑到 C 中协方差矩阵的结构，$\hat{\boldsymbol{f}}$ 中的数据项以及与其相关联的相邻训练数据项都将在取值上趋近。因此即使 K 的逆矩阵中存在异常大或异常小的离群值，也不会对最后得出的模型产生压倒性的影响，这也就客观上保证了在这样的模型中应用差分隐私的健壮性和可用性。我们的实现也正如 4.3.2 小节所叙述的那样，在特定数据项 i 的邻域内有越多的训练数据项，那么该元素的差分隐私隐蔽近似值 $C\boldsymbol{y}_i$ 就越能在存在异常离群值的情况下保持稳定。

4.4 基于差分隐私的数据脱敏机制

基于 4.2 节和 4.3 节的分析，本节将讨论将差分隐私技术应用于 Android 应用程序的隐私保护系统。本节首先讨论了算法的构建过程以及参数的选择方法，我们将看到基于高斯过程的差分隐私机制能够在保证数据集宏观可用性的前提下在绝大多数情况中有效保护个体数据项的隐私。本节还将讨论差分隐私机制下数据的应用机制，该机制直接决定了如何将差分隐私机制和现有的系统结合起来并在最大程度上发挥其效能。

4.4.1 参数的选择

在 4.3 节中，我们详细讨论了基于高斯过程的差分隐私以及其似然估计参数的选择与近似。正如之前在该节所述的，通常可以使用边缘估计的方法或者通过对超参数进行积分的方式来进行这种似然估计[55]。对于应用于差分隐私系统中的数学模型，在面对使用特定具体数据项的技术选择时需要格外小心，因为这种选择极有可能使得该数学模型在个体数据保护中失去一般性。为了将数据挖掘过程和差分隐私机制相结合，已有研究使用指数机制来辅助回归学习中对参数的选择和迭代调整[56]。这种方法能够证实在确保差分隐私的基础上快速准确地选择最佳的回归模型训练参数，但是如果将该算法应用于更广泛的分类模型中，还需要对全局敏感度及其上下界进行更进一步的探讨。

事实上在大量的应用场景中训练模型和数据集的维度都并不太高，因此在这种条件下使用简单的网格搜索通过循环遍历尝试所有可能性所需要付出的计算成本也并非无法承受。将网格搜索方法与指数机制相结合，模型能够快速得到收敛并选出能够应用于模型中的参数组合。指数机制是一种能基于差分隐私技术从一组可选参数集中快速选举出最佳选择的方法，并且在该过程中能够保证个体数据项的隐私性[57]。在应用该机制的过程中，需要注意的是具体使用场景中对全局敏感度的要求，即在模型训练过程中对原始数据进行修改的上下界。通过指定差分隐私预算特定随机化算法的全局敏感度，指数机制能够以特定的概率进行参数选择并得到效用最高的参数组合。尽管并不一定总是能够得到最优化的最高效用，但该机制能够在保证差分隐私的前提下表现最优。在使用指数机制前需要对数据集的效用函数 $u(\boldsymbol{y}_*, \boldsymbol{y}_t, \theta_i)$ 进行评估，具体来说在数据集中我们需要关注的是数据估计值和测试输出值 $(\boldsymbol{y}_*, \boldsymbol{y}_t)$。效用函数 $u(\boldsymbol{y}_*, \boldsymbol{y}_t, \theta_i)$ 中的 θ_i 来自我们希望比较所有潜在的参数组合的集合 Θ。此外还需要评估效用函数 u 的全局敏感度 Δu，这样选择 Θ 中特定参数组合 θ_i 的概率正比于：

$$p_{\theta_i} \propto e^{\frac{\epsilon u(\boldsymbol{y}_*, \boldsymbol{y}_t, \theta_i)}{2\Delta u}} \tag{4.40}$$

当计算效用函数 $u(\boldsymbol{y}_*, \boldsymbol{y}_t, \theta_i)$ 时我们可以考虑使用对数边际似然估计，并在数据拟合时使用其他噪声捕获差分隐私扰动噪声的影响。为了减轻模型过拟合可能带来的模型失效和简化对效用函数的全局敏感度的上下界估计，我们使用和方差 SSE 来衡量参数的误差，此外还需进一步考虑效用函数 u 的全局敏感度 Δu。一些研究也使用了类似地

方法，但是他们只是简单地将效用等同于误分类的次数[58]。通过使用 SSE，该方法还适用于其他并不提供对数似然性质的回归算法和分类算法。

如上所述，我们需要考察全局敏感度的边界。正如 4.3.1 小节所讨论的，在差分隐私机制中称对一个数据项所能够施加的最大扰动噪声为 d，我们需要考察这种扰动噪声对和方差的影响。在计算由训练数据的变化引起的和方差中的扰动时，我们可以注意到和方差实际上就是预测数据项与测试数据项之间的欧几里德距离的平方。这就相当于预测结果移动了 $d\boldsymbol{c}_{jk}$，它最大的影响出现在预测的移动方向与测试数据项相反的情况下，此时预测结果的移动可能会在预测位置与测试位置之间的欧氏距离上产生影响。对于训练模型时采用 k 折交叉验证的情况来说，测试数据集和预测结果之间的最大距离将会增加或减小 $d\boldsymbol{c}_{jk}$，这也就意味着对于每一折交叉验证来说和方差的最大变化值为 $d^2 max_j|\boldsymbol{c}_{jk}|_2^2$。

如果被扰动影响的数据项 j 位于测试集中，则和方差的变化量为：

$$(y_{*j}+d-y_{tj})^2-(y_{*j}-y_{tj})^2=d^2+2d(y_{*j}-y_{tj}) \tag{4.41}$$

为了对效用的全局敏感度进行限制，Δu 的范围还将被强制限制在一个 $\pm 4d$ 的范围内，式 4.41 计算中落在该范围外的敏感度将被强行阈值化。这也就意味着如果被扰动影响的数据在测试集中，可以计算出的扰动噪声对模型所产生的影响最大为 $9d^2$。对扰动进行限制所造成的可以预期的后果是在选择参数组合时我们可能会低估和方差的值，但也只会影响那些和方差异常高的参数组合，而这种情况在实践中出现的概率是极低的。

4.4.2　算法的构建

在使用多折交叉验证的训练场合，总体的和方差可以由每折的和方差累加得到。通过将最大的 $k-1$ 折的训练所得到的敏感度叠加并再加上 $9d^2$ 能够代表扰动数据项 j 在测试数据项中对单折的影响。扰动固然可能出现在对数据进行交叉验证的任何一折中，但我们假设它处于训练数据敏感性最小的那折中，以便使我们可以将该折的结果设为和方差对敏感度影响的上界。在这种假设条件下，任何其他折所得到的敏感度都将比该敏感度更低。综上所述，和方差在 k 折交叉验证中的敏感度为：

$$\Delta u^{\theta_i}=9d^2+\sum_{k=1}^{k-1}d^2\max_j|\boldsymbol{c}_{jk}|_2^2,\quad \theta_i\in\Theta \tag{4.42}$$

其中交叉验证的所有 k 折通过敏感度降序排序。在所有可能的参数组合中，指数机制的敏感度是最高的：

$$\Delta u\triangleq\max_{\theta_i\in\Theta}\Delta u^{\theta_i} \tag{4.43}$$

因此我们需要为考察的每个参数组合计算和方差及其敏感度，并从每个参数组合中选择最大的敏感度作为全局敏感度 Δu。接下来我们将使用计算出的与指数机制绑定的敏感

度来选择参数组合。对于固定的差分隐私预算 ϵ 总和，在选择参数组合时将会消费其中的一部分，剩余的差分隐私预算将会被实际的模型训练过程消费。

单纯地使用上述方法的一个重要问题是，全局敏感度 Δu 由具有最高敏感度的参数组合所决定，然而该参数组合却并不一定能够代表最高的数据可用性和健壮性。需要说明的是，敏感度的计算仅取决于 $\boldsymbol{c}_{jk}$ 和指定的最大扰动噪声 d 而并不依赖于特定输出数据项，因此我们可以有选择地忽略敏感度高于特定阈值的那些参数组合。然而和方差对单个训练数据项高度敏感的参数组合，也可能是预测本身敏感度极高的参数组合，在这种条件下需要差分隐私的扰动噪声更高更有破坏性，这也就带来了和方差对单个训练数据项高度敏感的事实。因此很可能这种参数组合不是最佳的配置。因此在估算每种参数组合的和方差时，我们会考虑差分隐私所需的扰动噪声的影响。更具体地说，我们从差分隐私扰动噪声分布中进行了多次采样，以确保将其包括在最终的和方差中。因为所有扰动噪声都独立于特定数据项，所以这种添加不会引入额外的差分隐私预算需求。伪代码 4.1 叙述了本节所考察的参数选择和算法构建的大致过程。

算法 4.1 参数选择与算法构建过程

输入: Θ: 待考察的参数组合的集合
X: 模型训练输入
$\boldsymbol{y}$: 模型训练输出
d: 敏感度阈值
k: 交叉验证的折数

for $\theta \in \Theta$ **do**
  $SSE^{(\theta)} \leftarrow 0$
  for $i \in \{1, 2, \ldots, k\}$ **do**
    $C_i \leftarrow C(X^{(i)}, X_*^{(\bar{i})}, \theta)$
    $\boldsymbol{y}_*^{(i)} \leftarrow C_k \boldsymbol{y}^{(i)}$
    $SSE^{(\theta)} \leftarrow SSE^{(\theta)} + \sum_{j=1}^{\frac{N}{k}} (y_{*j}^{(i)} - y_{tj}^{(i)})^2$
    $\alpha_i \leftarrow \max_j |\boldsymbol{c}_{ji}|_2^2$
  end for
  $\Delta u^{(\theta)} \leftarrow 9d^2 + \sum_{i=1}^{k-1} d^2 \alpha_i$
end for

4.5 本章小结

本章介绍了基于差分隐私技术的 Android 应用程序数据脱敏机制。考虑到直接使用第三章中提出的安全策略注入技术可能会影响 Android 应用程序可用性和健壮性，本研究使用差分隐私技术对安全策略应用机制进行了加强。通过引入差分隐私技术，本研究

能够在维护安全策略注入功能完备的前提下保证 Android 应用程序数据的可用性。具体地，本章考察了差分隐私技术的数学原理以及其在数据脱敏领域的应用，接着探讨了一种基于高斯过程的差分隐私应用机制，该机制能够将差分隐私技术应用于数据挖掘场景中，而数据挖掘正是绝大多数 Android 应用程序开发者和商业组织最常见的利用应用程序数据的方式。接下来，本章还具体说明了将差分隐私技术应用于 Android 应用程序数据挖掘过程中参数选择和算法构建的方法与详细步骤，并详细介绍了将差分隐私技术和应用程序隐私保护机制相结合的具体方式。事实证明差分隐私技术能在加强 Android 应用程序隐私保护的基础上保障应用程序的基本功能，并在此基础上保护基于数据挖掘和分析的现代 Android 应用商业模式，我们还将在第五章中通过一系列实验与测试再次证明这一点。

第五章 Android 应用程序隐私保护系统的实现与测试

基于第三章和第四章所提出的 Android 应用程序隐私保护机制和数据脱敏机制，本章将构建一个 Android 应用程序的隐私保护系统，该系统能够利用安全策略注入技术进行动态的用户隐私保护，并通过差分隐私机制对应用程序和网络服务器通信的数据进行脱敏化处理。对于安全策略注入机制，该系统利用了基于动态策略生成的字节码注入技术，能够在 Android 应用程序中植入对危险权限和敏感 API 调用的控制代码，并根据用户或组织的需求进行动态的调整。对于数据脱敏机制，该系统利用了基于高斯过程的差分隐私机制，在加强应用程序隐私保护功能的基础上维护应用程序的的数据分析需求及其相关基本功能。为了评估该系统各功能的可用性和健壮性，我们将该系统应用于从 Android 应用市场上随机挑选的应用程序中，并针对该系统对各种危险权限和敏感 API 调用行为的控制能力进行了细致地分析与研究，对植入安全策略后的应用程序的健壮性和可用性进行了调查与分析。

5.1 系统设计与实现

本节将详细介绍我们设计的 Android 应用程序隐私保护系统的整体架构以及各个部分的技术细节。对于整体架构，我们将主要关注系统各个部分的逻辑关联和系统数据在各部分中的流向和处理流程，并分析采用这种整体架构的原因和其中的关键技术重难点。对于各部分的具体设计分析，我们将主要关注各个部分所涉及的技术选型及其相关原理，旨在用较为简单的方式阐明各部分的构成与工作方式。

5.1.1 系统的整体架构

本章所实现的系统对第三章所提出的安全策略注入机制与第四章所提出的差分隐私机制进行了综合，旨在平衡隐私保护系统的安全性、可用性和健壮性。图 5-1 刻画了本章所介绍的 Android 应用程序隐私保护系统的整体架构。正如图 5-1 所示，该系统被分为了六个部分，分别是抽象安全策略的生成与实例化、字节码级安全策略注入、差分隐私控制与数据脱敏、安全策略服务器以及策略的调整与日志的反馈部分。

对于用户或组织所指定的使用场景及其安全需求，本系统首先需要生成特定的抽象安全策略。该部分需要根据用户或组织的具体安全需求，根据数学和逻辑学模型将具体需求抽象为特定的行为模型，这个行为模型需要涵盖数据流、事件发生的时间时长与频率、应用程序用户的身份与角色、环境与场景信息、信任关系等范畴，对应用程序运行过程中所有涉及到用户隐私和数据安全的环节列入考虑。随后，系统提供的安全策略生成机制将通过执行一组细粒度的安全规则将抽象策略转换为具体的权限控制策略。

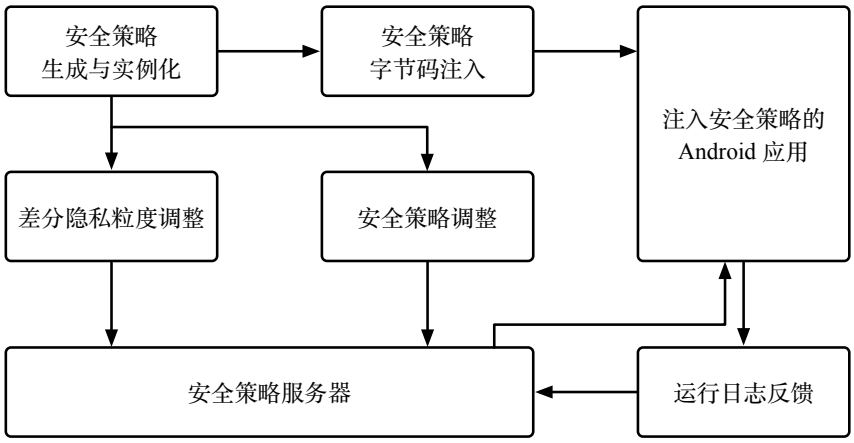


图 5-1 Android 应用程序隐私保护系统整体结构

策略生成模块所产生的安全策略实例将指导系统生成一个包含安全方法的字节码安全运行库，这个运行库中包含了执行安全策略所必须的所有功能代码。该安全运行库与通信、控制与日志功能一起打包构成一个针对特定应用程序的安全决策中心，安全决策中心除了负责处理各个安全方法的功能调用外，还要处理安全策略服务器可能提出的安全策略调整指令以及其他通信命令，还能够记录系统的调用状态和运行日志，并将运行日志实时返回给安全策略服务器。

对于用户或组织所指定的应用程序，首先需要对其进行字节码级别的静态分析，目的是找到危险权限和敏感 API 的调用位置。安全策略实例化所生成的具体控制策略将以虚拟机字节码的形式被注入到 Android 应用程序的相关调用位置，这些被注入的字节码被称为策略执行器。除此之外，上面介绍的安全决策中心也将被嵌入到应用程序中。在应用程序运行过程中，策略执行器将和安全决策中心进行实时通信，在出现权限调用事件时向安全决策中心发起通知，并执行安全决策中心所给出的权限控制指令。

尽管经过了字节码级别的安全策略控制的 Android 应用程序在理论上使得用户或组织能够全方位的控制应用程序的敏感操作和危险行为，但是对特定权限的封锁和数据访问请求的拒绝极有可能损害到良性应用程序的正常功能和应用程序开发者的数据收集需求。因此本系统中还加入了基于差分隐私技术的数据脱敏功能。通过使用基于高斯过程的差分隐私技术对所有敏感 API 调用或数据通信请求中所涉及的数据流进行脱敏，该系统能够保证应用程序所有与开发者或商业组织的网络通信数据都不会泄漏用户的个体信息。

除了对应用程序进行的相关操作外，本系统还提供了一个安全策略服务器，该服务器能够和所有嵌入了安全策略的应用程序进行实时通信。用户或组织能够经由该服务器对所有嵌入了安全策略的应用程序进行策略调整或者差分隐私的粒度控制。通过与应用程序中嵌入的安全决策中心进行通信，该服务器能够动态地调整应用程序中的安全策略而不需要重新生成字节码并将其打包到新的应用程序中。此外，该服务器还能够接收由应用安全决策中心返回的调用日志和错误信息，这些信息能够辅助用户和组织给出安全

策略实际实施的反馈信息，也能够指导用户或组织更好地调整安全策略以满足其具体需求。

5.1.2　应用程序采集与数据源构建

为了对本章所介绍的 Android 应用程序隐私保护系统进行评估和测试，我们需要将该系统应用于真实世界中的应用程序中。为此，我们从国内的几个主流 Android 应用市场中采集了应用程序，5.2 节中的所有分析都将基于这里采集的应用程序。为了使构建的数据集具有普适性，所采集的应用程序必须是 Android 用户在日常生活中最常使用的那些程序。此外，为了使得实验结果能够充分反映本系统功能的优势与不足，所采集的应用程序集的规模必须尽可能的大。

为了实现上述目标，我们首先考察了各中国 Android 应用商店的市场占有率，并选定了 4 个市场占有率最高的应用商店，它们分别是：

- 腾讯应用宝 (https://sj.qq.com/)
- 360 手机助手 (http://sj.360.cn/index.html)
- 华为应用市场 (https://appstore.huawei.com/)
- 小米应用商店 (http://app.mi.com/)

截止到 2019 年 12 月，上述 4 家应用商店的市场占有率之和接近 70% 且在呈持续增长趋势，而其他所有应用商店的市场占有率都未能超过 10%[59]。因此我们认为这几个应用市场的数据源已经可以充分代表大多数 Android 应用商店和应用程序的使用情况。

为了从应用市场中大规模批量采集应用程序，我们设计了一个基于 Scrappy 框架的 Python 爬虫。该爬虫从给定根域名出发，通过 HTML 的 DOM 元素采集界面中的所有相关域内 URL 并将其压入队列，随后以广度优先遍历方式递归搜索指定域名下的所有应用资源及其元数据。随后爬虫会将应用程序的 APK 安装包保存在指定目录，并将相关元数据保存在一个 MongoDB 数据库中以便在 5.2 节中对其进行分析。尽管如此，我们需要针对不同的应用商店设计不同的爬虫插件，因为各个应用商店的前端资源拓扑结构和页面 DOM 组织方式都各有不同。下页的示例代码展示了该脚本从腾讯应用宝中获取应用程序元数据的过程。对于所采集的应用程序元数据，我们主要关注应用程序的发行名称、应用大小、版本号、全限定名等信息，因为这些信息能够体现 Android 应用程序的基本特征。

我们从上述 4 个应用市场中各采集了 60 个应用程序，剔除来自不同应用市场的重复应用程序后一共剩余 129 个应用程序。篇幅所限，我们将不会在正文部分介绍这些应用程序的具体情况。想要了解实验所涉及的应用程序，读者可以到附录 A 中获取相关信息。我们所采集的应用程序包含购物、电子阅读、新闻、即时通讯、音乐等各种类型，能够基本囊括现实世界中 Android 应用程序的使用场景和常见需求。我们还将在 5.2.1 小节中对我们所采集的应用程序集进行详细的分析。

```
def get_application_metadata(soup_object):
    ## ......
    name_tag = soup_object.find(class_='det-name-int')
    size_tag = soup_object.find(class_='det-size')
    download_tag = soup_object.find(class_='det-down-btn')
    version_tag = soup_object.find(class_='det-othinfo-data')
    publish_tag = soup_object.find(id='J_ApkPublishTime')
    ## ......
    collection = database_settings('tencent')
    collection.insert_one({
        'app_name': name_tag.string,
        'app_size': size_tag.string,
        'apk_name': download_tag.get('apk'),
        'apk_url': download_tag.get('data-apkurl'),
        'apk_version': version_tag.string,
        'apk_publish_time': publish_tag.get('data-publishtime'),
        'insert_time': time.time()
    })
```

5.1.3 安全策略的生成与实例化

对于用户和组织的不同需求，本系统首先需要完成的工作是为特定的应用场景和安全需求生成抽象的安全策略，并根据特定的 Android 应用程序将安全策略实例化。本小节中叙述的安全策略生成方式和逻辑结构依据的是 3.1 节中叙述的理论模型。

图 5-2 是本系统中用户指定安全需求并生成抽象安全模型的界面。系统通过对用户或组织指定的 Android 应用程序进行静态分析得到该应用程序中涉及的所有敏感 API 调用和危险权限请求，并将所有这些策略点向用户或组织提供，用户或组织在界面中可以一览所有的危险权限请求以及更细粒度的 API 调用。

然而普通 Android 用户并非安全专家，考虑到用户或组织并不能够明确了解所有危险权限请求或者 API 调用的背后代表着什么，抑或类似的请求将会给用户在何种程度上带来风险，本系统有义务为用户或组织在抽象安全策略生成阶段提供安全建议。举例来说，访问设备定位信息这一抽象行为可能对应着多种具体 API 调用，应用程序或者要求在位置改变时执行回调，或者要求操作系统返回设备的最后位置，或者以一定时间间隔轮询定位传感器等等。对于这些具体的 API 请求，本系统能够针对性地提出抽象安全策略的建议与规划供用户或组织使用或作为模板。

除了特定 Android 应用程序所涉及的具体调用或权限申请，本系统还要考虑用户或组织自身的环境与需求。对于用户或组织所处的不同环境，本系统给出的抽象安全策略

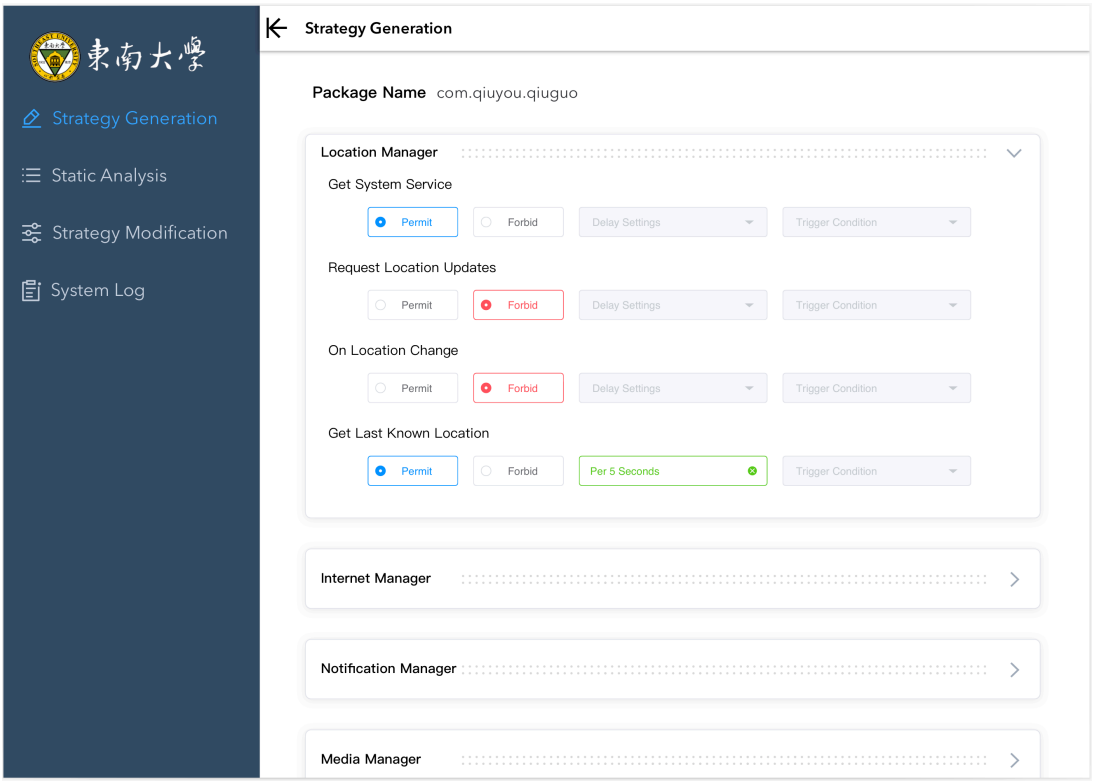


图 5-2 根据用户需求生成安全策略

模板必须能够体现出不同的安全等级和粒度。举例来说，用户在保密工作场景和会议过程中显然不希望应用程序拥有随时访问麦克风的权限，而用户在家庭场景中面对可能的来电需求则需要允许特定的通讯应用拥有访问麦克风的基本权限。

此外，对于一些常见开发者或开发组织所设计与开发的应用，本系统也支持不同级别的信任机制。用户或组织可以自由指定特定应用程序或者特定开发者所开发的一系列应用程序以不同的信任级别，这样在安全策略制定时可以参考信任级别对具体行为进行微调。举例来说，在长期使用某个开发者或开发组织所开发的应用程序并且从未面临任何安全问题时，用户多会倾向于信任该开发者或开发组织所开发的其他应用程序。而面对一些恶名昭著的开发者或商业组织所提供的应用程序时，用户必然会多加小心。尽管如此，信任机制更多的是为系统提供一种策略建议与可能性，本系统中安全策略的制定并非完全依赖信任机制。

除了对特定危险权限或者 API 进行访问外，应用程序的数据访问和数据操作行为也需要特别注意。本系统能够向用户或组织提供对 Android 应用程序所生成、读取、修改、传输的数据的具体类型和数量的信息。此外，根据用户制定的抽象安全策略，本系统也能自动生成对数据流的控制策略，同时提供细粒度的数据控制机制。在实际场景中，Android 应用程序之间常常存在互相唤起或数据通信的行为，这些行为也会被本系统所监控和限制。

5.1.4 安全策略的注入与调整

针对具体的 Android 应用程序，本系统将会对抽象的安全策略进行实例化，这种实例化将生成一个包含安全方法的字节码安全运行库。该运行库中包含了执行安全策略的具体代码。这些字节码将随后被嵌入到原始程序的字节码的安全敏感位置，并和原始应用字节码一起被重新打包生成一个包含安全决策中心的应用程序。包含安全策略的应用程序在敏感 API 的调用位置植入了一系列的策略执行器，这些执行器能够在权限调用事件中向安全决策中心发出事件通知，并向安全决策中心索取策略指定的授权操作以执行具体的策略。图 5-3 展示了将安全策略以虚拟机字节码的形式注入一个 Android 应用程序中，并实施特定安全策略的过程。

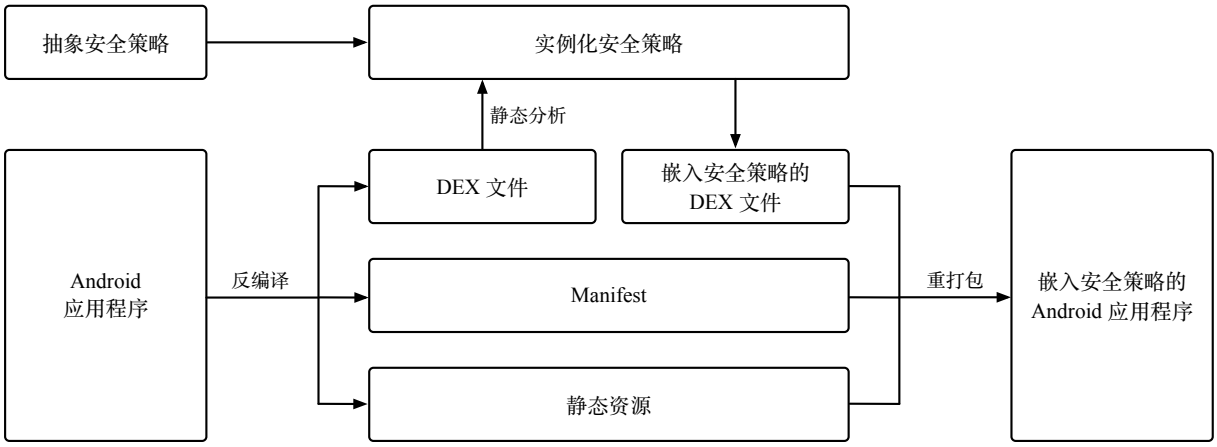


图 5-3 字节码的注入流程

实施流程的第一步是将应用程序的 APK 安装包进行解包。APK 是 Android 操作系统的软件包安装文件。和其他系统的软件安装包一样，APK 包拥有特定的组织结构。对一个完整的 APK 安装包进行解包，其中总是会包含如下文件：

- AndroidManifest.xml：以 AXML 二进制格式组织的清单文件，包含 Android 应用程序的配置信息和元数据；
- META-INF/：APK 安装包的开发者证书和签名信息，用于确认开发者的身份和确保 APK 未被篡改；
- classes.dex：Android 应用程序的所有可执行代码在编译后生成的 Dalvik 虚拟机二进制字节码；
- res/：用于保存所有 Android 应用程序中使用的静态资源文件，如图像、颜色配置、布局信息等；
- resources.arsc：编译好的二进制格式的资源配置，用于将 res/目录下保存的静态资源和程序执行代码相关联；
- assets/：Android 开发者可以选择使用 Asset 系统存放原始资源，在此情况下所有静态资源都将存放在 assets 目录下。

将 APK 安装包解包后得到的 classes.dex 文件中包含了该应用程序的所有执行代码的二进制编译结果。将该 DEX 文件进行反编译并进行静态分析，可以确认该应用程序中所涉及的所有敏感 API 调用和危险权限请求。为了识别所有可能的涉及危险权限的 API 调用，静态分析器将从应用程序中提取所有可能的 API 调用，并将这些 API 方法和危险权限 API 列表相比较，其中的交集就是该应用程序所涉及的所有敏感 API 调用。下面的代码清单中列出了部分敏感 API 调用的字节码格式。

```
## 发送 SMS 消息
invoke-virtual {v0, v1, v2, v3, v4, v5}, Landroid/telephony/SmsManager; ->
    sendTextMessage(
        Ljava/lang/String;
        Ljava/lang/String;
        Ljava/lang/String;
        Landroid/app/PendingIntent;
        Landroid/app/PendingIntent;
    )V
## 获取设备 ID
invoke-virtual {v0}, Landroid/telephony/TelephonyManager; ->
    getDeviceId()Ljava/lang/String;
## 获取 SIM 卡序列号
invoke-virtual {v0}, Landroid/telephony/TelephonyManager; ->
    getSimSerialNumber()Ljava/lang/String;
```

安全策略实施流程的下一步就是对于所有上述敏感 API 调用在调用前后注入安全策略代理，即策略执行器。此外，对于应用程序整体，本系统还会将安全决策中心和所有安全运行库以字节码的形式整体注入到应用程序字节码中，此后当应用程序执行到注入了策略执行器代理的 API 调用时，将会和安全决策中心通信并请求具体的安全策略。需要特别说明的是，对于反编译得到的虚拟机字节码，为了减轻对原始代码的侵入性和提升安全策略注入后应用程序的健壮性，我们采用了将安全决策中心和策略执行器分离的策略。如 2.3 节中所介绍的，虚拟机字节码采用基于寄存器的技术架构，其中每个方法都会在元数据的 `registers` 字段提前声明该方法所需要使用的虚拟寄存器的个数。该虚拟寄存器的个数和方法中声明的局部变量以及方法的入参个数相关。所有策略执行器 PE 的安全代理都将对方法新增 3 个局部变量，因此当向敏感 API 调用中注入策略执行器时，还需要注意对方法元数据的修改，以防止寄存器错误导致的编译失败或运行时错误。

考虑到上述安全策略的执行代码都是依据对应用程序执行代码进行静态分析得到的 API 调用表来进行字节码注入的，这样的字节码注入技术能够对所有的 API 调用进行安全代理，但是当安全策略发生更改时安全代理的行为需要得到更加灵活的变更。为

了进一步降低对原始代码的侵入性并提升安全策略执行器的动态性，本系统利用了反射技术来实现安全策略的动态调用。反射是一种允许程序在运行时动态地发现、加载和使用类的技术，该技术能够将类动态地加载到活动代码中。通过将行为参数化，反射技术能够将安全策略代理，即策略执行器作为参数传递到相关的 API 调用中。举例来说，该技术可以通过反射定义一个安全策略代理：

```
const-string v0, "ContentActivity"
.local v0, "onCreate":Ljava/lang/String;
new-instance v1, Lcom/crumblejon/SecurityProxy;
invoke-direct {v1}, Lcom/crumblejon/SecurityProxy; -> <init>()V
invoke-virtual {p0}, Lcom/crumblejon/SPActivity; ->
    getApplicationContext()Landroid/context/Context;
move-result-object v1
```

并通过如下方式在相关的 API 调用中触发 Java 反射技术定义的安全代理：

```
invoke-static {v0, v1, p0}, Lcom/crumblejon/SecurityProxy; ->
    loadProxy(
        Ljava/lang/String;
        Landriod/content/Context;
        Landroid/app/Activity;
    )V
```

在完成所有的字节码注入工作并测试完毕后，安全策略实施流程的最后一步是对封装了安全策略的应用程序字节码进行重编译和重打包。本系统使用 ApkTool 对应用程序进行重编译和重打包，并使用 signAPK 对得到的新 APK 文件进行签名。只有经过签名的应用程序安装包才能通过 Android 操作系统的检测并成功安装到用户的设备中。

5.1.5 差分隐私控制与数据脱敏

基于用户和组织所提供的安全需求生成的安全策略以字节码的形式注入到指定的 Android 应用程序中，这事实上被证明已经能够满足绝大多数用户和组织的安全需求。然而正如我们在 4.1 节中所论述的那样，绝大多数现代 Android 应用开发者和开发组织依赖应用程序所产生的数据来提供正常功能或者进行商业推广，比如移动定位应用的依赖于正确的定位信息，网络购物应用的推荐系统依赖于正确的浏览和购买历史等等。正因此，简单对敏感调用或权限申请进行屏蔽极有可能损害良性应用程序的正常功能。

为了平衡用户和组织的客观隐私保护需求和 Android 应用程序的数据利用需求，本系统还引入了基于差分隐私技术的数据脱敏功能，其具体原理和技术基础我们已经在 4.2 节中给出了详细的介绍。通过使用基于高斯过程的差分隐私技术，本系统能够对所

有敏感 API 调用和数据通信中所涉及的数据流进行脱敏，能够确保 Android 应用程序开发者和商业组织所获取的任何数据集都不会泄漏具体数据与特定用户的关联信息，从而切断隐私泄漏的渠道。

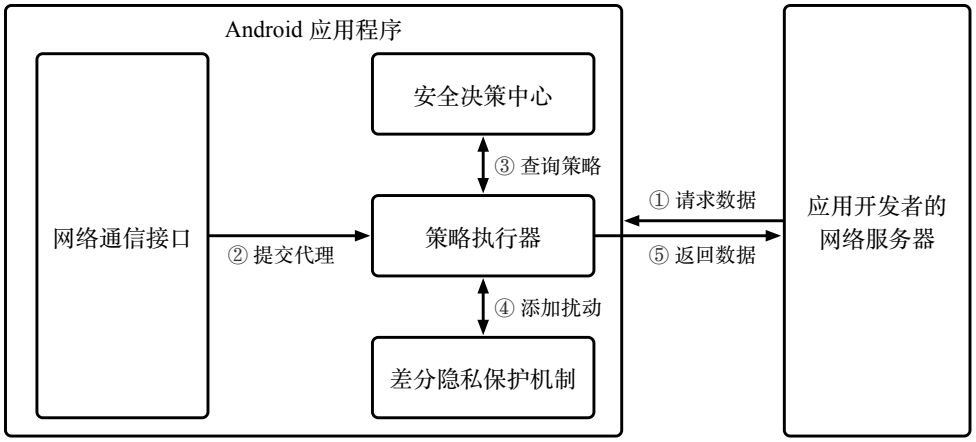


图 5-4 基于差分隐私技术的数据脱敏流程

图 5-4 展示了本系统中利用差分隐私技术进行数据脱敏的流程。基于 5.1.4 小节中介绍的字节码注入功能，本系统在注入应用程序的安全决策中心中也加入了差分隐私控制模块。这样，当任何的数据访问请求或数据通信请求出现时，相关的策略执行代理都将会通知安全决策中心，这样安全决策中心将指示策略执行器对数据流添加满足差分隐私配置的扰动噪声，从而对数据进行脱敏。而安全决策中心所给出的差分隐私配置也来自用户或组织的抽象安全策略，用户在基于安全需求设定安全策略时也将给出对特定数据流的差分隐私设置，包括差分隐私预算等配置信息。这些配置信息将和其他实体安全策略一道，作为安全策略中心的一部分被注入到特定的 Android 应用程序中。

具体地，基于差分隐私技术的数学原理以及其在数据脱敏领域的应用，本系统设计了一种基于高斯过程的差分隐私应用机制。使用这种机制的目的是为了将差分隐私技术应用于数据挖掘场景中，而数据挖掘正是绝大多数 Android 应用程序开发者和商业组织从应用程序中获取数据流的目的和归宿。基于高斯过程的差分隐私能够在保证数据挖掘的结果准确可用的基础上对具体数据项进行数据脱敏，以消除特定数据项与其数据源实体的关联。将差分隐私技术应用于 Android 应用程序数据挖掘过程中的具体参数选择和算法构建的详细步骤我们已经在 4.4 节中进行了详细的介绍，在此不再赘述。5.2 节还将对差分隐私机制进行评估和分析，考察本小节中引入的差分隐私机制对 Android 应用程序的隐私保护系统的贡献，以及相关参数的变化对数据流可用性和安全性的影响。

5.1.6　日志反馈与策略分析

将安全策略以字节码的形式注入到应用程序中后，本系统还需要持续追踪安全策略的执行情况和应用程序的运行状态，这不仅是为了考察在进行修改后应用程序是否还能够正常运行，也是为了反馈安全策略是否能够正确的达成用户的安全目标要求。

对应用程序和安全策略的追踪主要依靠运行日志来完成。运行日志能够辅助用户查看关键代码的运行流程，记录各种运行时异常的位置与上下文。本系统在安全策略注入时就在各个策略执行代理中嵌入了日志功能的相关代码，这些代码在应用程序的各个敏感 API 调用和权限申请位置埋入钩子函数，安全决策中心可以依据用户或组织设定的逻辑将运行日志功能挂载到这些钩子函数下面。这些钩子函数本身并不实现任何逻辑，而是依靠安全决策中心的指示动态地挂载相关执行函数，这一方面保证了代码的复用性和可扩展性，另一方面允许用户对运行日志的粒度和时间频率进行调整。下面的代码展示了应用程序安全策略执行日志的一个示例。

```
2020-02-11 09:12:33 INFO LaincrossActivity a.m.MediaRecorder.start()
2020-02-11 09:14:01 WARN ParActivity  a.d.s.SQLiteDatabase.insert()
2020-02-11 09:15:21 INFO AgentActivity  a.c.BroadcastReceiver.init()
2020-02-11 10:25:56 WARN LaincrossActivity j.n.URL.openConnection()
```

上面的日志记录了对敏感 API 的调用时间及其所属的应用程序活动，这些信息有助于定位具体行为的调用链与功能分支，从而帮助用户和组织分析应用程序的行为逻辑。通过统计日志中某些敏感 API 调用或者权限使用的时间及频率，用户可以直观地了解应用程序对隐私信息的访问模式及其威胁程度，同时这种信息也能够帮助用户在指定或后续调整安全策略时进行更好的决策。

为了辅助日志的自动化处理和灵活过滤不同粒度的日志，本系统的日志系统提供了不同的日志级别，如 `INFO`，`WARNING`，`DEBUG`，`ERROR` 等。对于普通 Android 设备用户来说，过于庞杂的日志信息丝毫不具备可读性，因此对特定级别日志的自动筛选和过滤功能尤为重要。这能将用户从过载的信息流中解救出来，并辅助他们做出更高质量的安全决策。此外，为了防止在多线程环境下的日志错误和缓冲区异常，本系统的日志系统能够保证在 Android 应用程序运行时始终提供线程安全的日志。这个功能由互斥锁提供，通过使用互斥锁日志系统能够防止多线程在同一时刻访问同一资源。

5.1.7 安全策略服务器

对于根据用户安全需求生成安全策略并将安全策略以字节码形式注入到特定应用程序中的需求，任何部署了本系统的计算机都可以实现。然而为了保证系统的动态性和灵活性，本系统还需要维护一个安全策略服务器。安全策略服务器的主要功能是和所有注入了安全策略的应用程序进行通信，更具体地说，是和这些应用程序中的安全决策中心进行通信，进而实现动态的安全策略调整功能。图 5-5 展示了安全策略服务器和注入了安全策略的应用程序的通信流程与结构。

在已经注入了安全策略的字节码级控制代码后，用户或组织可能有根据具体需求和使用场景动态地调整安全策略的需求。这种调整的需求包括但不限于添加对某些权限请求的限制、提升或降低对已限制权限的控制级别、提升或降低特定数据传输的差分隐私

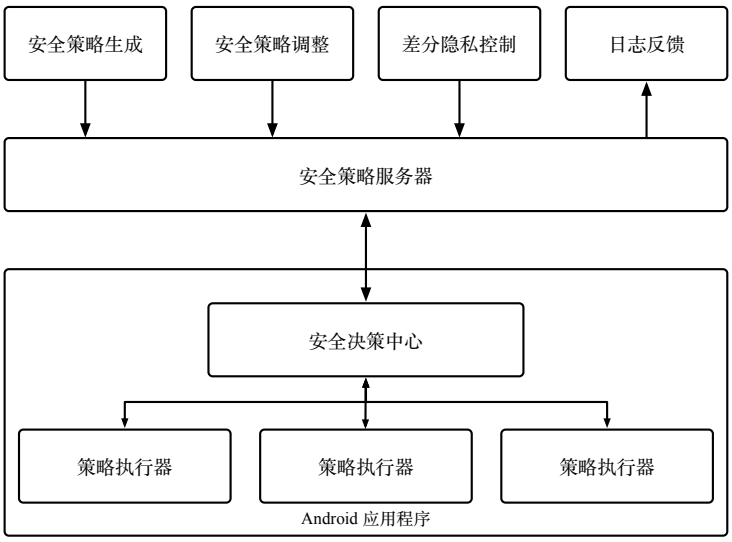


图 5-5 安全策略服务器与应用程序的通信

预算和差分隐私控制粒度等等。这些目标的实现都依赖于安全策略服务器，这是因为策略的调整控制与差分隐私的粒度控制都以该安全策略服务器为载体。除此之外，上小节中介绍的日志系统也需要依赖安全策略服务器，这是因为基于本系统的所有注入安全策略的应用程序的日志都将被传送回服务器，以便对日志进行自动化分析，并向用户提供同一的日志访问接口。

当面对组织用户时，可能存在大量注入安全策略的应用程序需要同时与安全策略服务器进行实时通信，这时服务器的并发性能就成为了设计的技术瓶颈。因此，我们在技术选型时并没有采用常见的 Java-Spring 框架或者 PHP-Laravel 框架，而是使用了基于 Node.js 的 Koa2 框架。这是因为 Node.js 技术基于 Google 的 V8 解释引擎，该引擎是一种基于事件循环的 I/O 密集型引擎，它能够解决大量前端请求时的资源并发访问问题。考虑到安全策略服务器并不需要过高的数据库访问能力，而是更注重 HTTP 连接的并发能力，采用 Node.js 是我们最好的选择。

考虑到在大多数情况下，用户或组织的设备条件及其运行环境和我们的开发环境不尽相同，而不同用户或组织所使用的操作系统版本级其他软硬件条件也千差万别，因此在不同的运行环境下部署我们的服务器程序是一个非常重要的挑战。尤其是服务器程序中涉及到不同的依赖环境在不同的系统环境中调试困难，甚至需要针对不同的环境作定制化的处理。为了解决这样的部署难题，同时也为了方便用户或组织快速部署自己的安全策略服务器，我们为系统设计了一个 Docker 容器。这个 Docker 容器中包含系统的安全策略服务器及其配套数据库，还嵌入了服务器运行所需的所有依赖环境。用户只需要在自己的设备上安装最基本的 Docker 环境，就可以轻松运行我们提供的 Docker 容器。这样，用户经过简单的配置就可以获得开箱即用的服务器环境，并且可以轻松地将服务器程序部署到自己的硬件设备中。我们对该 Docker 容器进行了广泛而全面的测试，它在以下主流计算机操作系统中都能够成功部署：

- **Microsoft Windows**: Windows 8, Windows 8.1, Windows 10
- **Apple MacOS**: MacOS 10.13 High Sierra, MacOS 10.14 Mojave, MacOS 10.15 Catalina
- **GNU/Linux**: Ubuntu 16.04 Xenial Xerus, Ubuntu 18.04 Bionic Beaver, Manjaro 19.0.2

5.2 实验评估与分析

为了对本文所介绍的 Android 应用程序隐私保护系统进行测试和评估，我们从国内各大应用市场中随机采集了一些应用程序，并以此为载体对系统进行了详细的分析。我们的评估主要考察注入安全策略后应用程序的可用性和健壮性，以及注入的安全策略对应用程序所产生的性能方面的影响。此外，针对本系统所利用的差分隐私技术，我们还对其数据脱敏能力进行了考察。

5.2.1 数据源的特征分析

正如我们在 5.1.2 中介绍的那样，为了使得所选定的应用程序载体能够反映用户日常生活中的常规使用情况，我们考察了国内 Android 应用商店的市场占有率，并选定了市场份额最高的 4 家应用商店。接着我们从这 4 个应用市场中各爬取了 60 个应用程序，在将这 240 个应用程序中的重复项剔除后得到了 129 个应用程序，它们构成了数据源的全部内容。需要强调的是，我们的爬虫在采集应用程序时并未对应用程序作任何限定或要求，而是采用完全随机的方式进行爬取。这也意味着我们并没有对应用的类型、下载量、大小等做出任何假设，这是为了保证数据集的随机性和普适性。

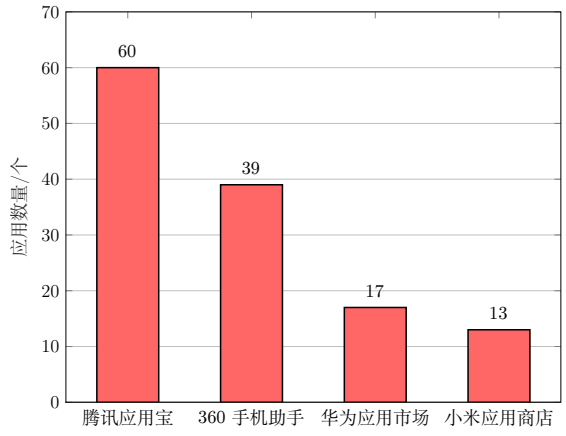


图 5-6 各应用商店对数据源的贡献度

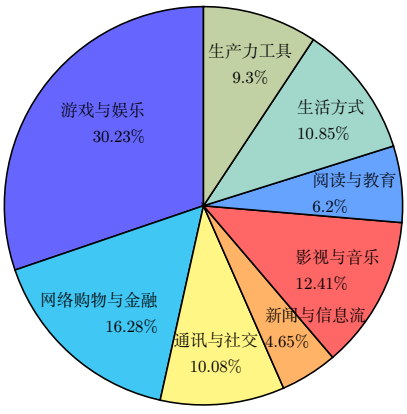


图 5-7 数据源中应用程序的类型分布

图 5-6 展示了在剔除重复的应用程序后数据源中来自各个应用商店的应用数量。需要说明的是我们在剔除重复应用时以应用商店的市场份额作为优先级。换言之，我们首先以从腾讯应用宝中爬取的应用集合作为基础，并剔除从其它 3 个应用市场中爬取的应

用程序集合中的重复项。接着对从 360 手机助手中获取的应用程序进行类似地操作，考察剩余 2 个应用商店相对的重复项并予以剔除，以此类推。正因此，图 5-6 中各商店所提供的应用数量必然是渐次减小的。

为了分析安全策略对不同类型的应用程序的影响，我们还对数据源中所有应用的类型进行了详细的考察。图 5-7 对数据源中 129 个应用程序的类型进行了统计和分析。需要说明的是，现代 Android 应用程序往往庞大而复杂，这也意味着单个应用程序也有可能包含多个复杂功能。举例来说，国内常用的某即时通讯应用同时也包含了在线结算的功能，而某些网络购物应用中也集成了信息流与新闻功能。这就给我们对应用程序的分类带来了困难。因为面对应用程序的复杂功能，想要统计数据源中所有应用的全部类型元素就只能依赖人工甄别。而由于主观评价标准的差异和较大的数据集带来的巨大工作量，都会导致人工甄别结果较真实情况有所偏差。因此我们在对应用程序的类型进行统计时只考虑了应用程序开发者所主张的应用类型。从图 5-7 可以看到，我们采集的应用中游戏与娱乐类程序占比最高，为 30.23%，此外网络购物、通信社交、影视娱乐等类型的应用占比也较高，分别为 16.28%，10.08% 和 12.41%。上述统计结果也和我们日常生活中的直观体验相符，说明我们的应用集合能够基本符合日常生活中应用分布的大致情况。

5.2.2　安全策略的健壮性

在对安全策略的功能和可用性进行详细测试之前，我们首先需要评价我们的系统和安全策略对程序的健壮性影响。所谓健壮性指的是在对应用程序进行修改和重新打包后，应用程序是否还能够正确运行。对于安全策略的健壮性，我们从两个维度对其进行评价，一个维度是该系统在执行特定安全策略后对应用正常运行的影响；另一个维度是对于不同的操作系统版本和不同的硬件平台，注入了安全策略的应用程序是否还能够正常工作。

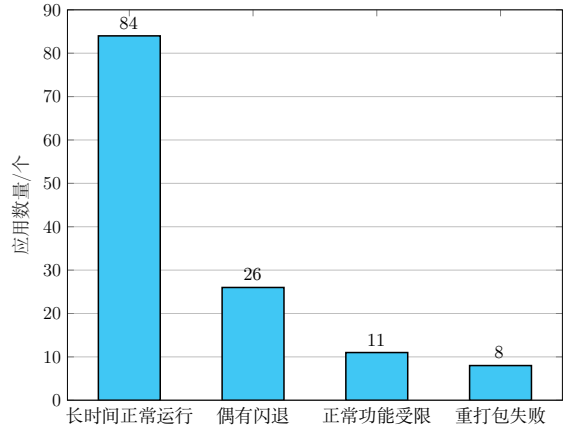


图 5-8 单一环境中应用的健壮性

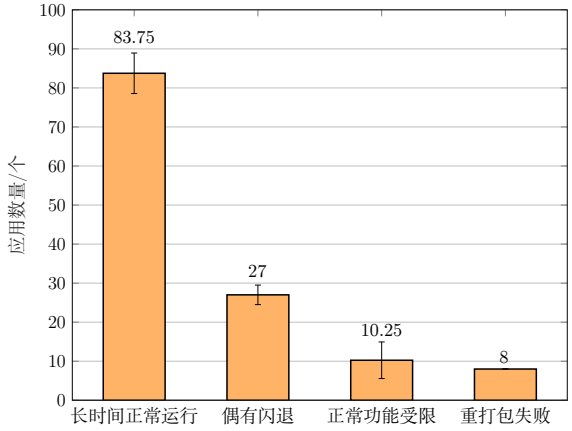


图 5-9 不同环境中应用的健壮性

图 5-8 显示了当运行环境限制在同一系统和同一硬件环境时，本系统对 Android 应

用程序健壮性的影响。在该实验中，我们使用了一台搭载 Android 10 系统的华为 P30 Pro 手机对 5.2.1 小节中介绍的数据集合中的所有应用程序进行了测试。由图 5-8 的数据可以看出，在限制了软硬件环境的情况下，85.27% 的应用程序都基本能够正常运行，其中大部分应用程序能够保持长时间的运行，而有 20.16% 的应用程序在执行特定安全策略时偶有闪退情况发生。除此之外，8.53% 的应用程序在测试时遭遇了无法打开或者正常功能收到影响的情况，还有 6.2% 的应用程序在嵌入了安全策略字节码后的重编译过程中发生异常而导致无法重打包。

考虑到 Android 操作系统的发布与使用存在严重的碎片化情况，我们有必要在不同软硬件条件下对应用健壮性进行多次测试。图 5-9 揭示了不同软硬件条件下安全策略对应用健壮性的影响。除了上个实验所使用的搭载 Android 10 版本系统的华为 P30 Pro 手机，在本实验中我们还使用了一台搭载了 Android 10 版本系统的小米 MI 9 手机，一台搭载了 Android 9 版本系统的华为 Mate 20 手机，以及一台搭载了 Android 9 版本系统的小米 Redmi K20 Pro 手机。实验结果表明我们的系统在不同的软硬件环境下并未表现出太大的差别。在多次测试中仍有 85.8% 的应用程序能够正常的启动与运行，而多次测试得到的数据的方差最大值也只有 5.1875。由于我们在开始本系统的开发时 Android 操作系统的最新版本尚在 Android 9，因此本系统的大部分功能都基于该版本开发。尽管如此，在 Android 10 上运行的应用程序的健壮性相比 Android 9 也未见明显差别。但有必要强调的是，由于时间和经济原因，我们这部分的实验所能够调动的设备数量非常有限，并不能够反映 Android 设备在社会中的版本分布，因此本实验只能在一定范围内体现本系统对应用健壮性的影响。对于其他的硬件设备或者更古旧的 Android 版本的设备，本系统尚需要更多的测试与评估。

5.2.3 安全策略的可用性

在确保本系统及其提供的安全策略的健壮性之后，我们接下来需要考察安全策略的可用性。在 5.2.2 小节所进行的几次测试中，有 75 个应用程序通过了所有健壮性测试，并且能够保持长时间运行而不出现异常。在本小节中我们以这 75 个应用程序为基础，对它们进行最细粒度、最高强度的安全控制。这意味着我们所采用的安全策略试图对这些应用程序所涉及的所有敏感操作和隐私访问行为进行限制。针对向应用程序中注入的安全策略，我们首先需要考察安全策略在应用运行过程中的可用性。更具体地说，是系统所注入的字节码级安全控制代码是否能够对应用程序的敏感 API 访问危险权限调用行为进行有效控制。

可用性一般也可称为有效性，在本系统中可用性用于反映安全策略对应用程序的保护能力。具体地，我们通过调查字节码级安全策略对所有应用程序的敏感 API 调用和权限控制能力来对安全策略的可用性进行量化，图 5-10 反映了对系统可用性的评估结果。针对数据源中的每个应用程序，我们都对其进行了静态分析以得到该应用程序所需要进行安全控制的策略执行代理嵌入位置。随后，我们对应用程序进行安全策略注入，

并针对每个策略执行器进行单元测试。通过对每个策略执行器进行单元触发，我们可以考察策略执行器对特定权限或 API 调用的控制能力。

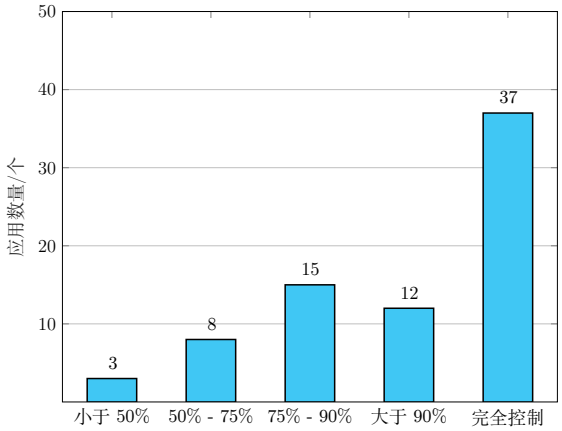


图 5-10 安全策略的可用性

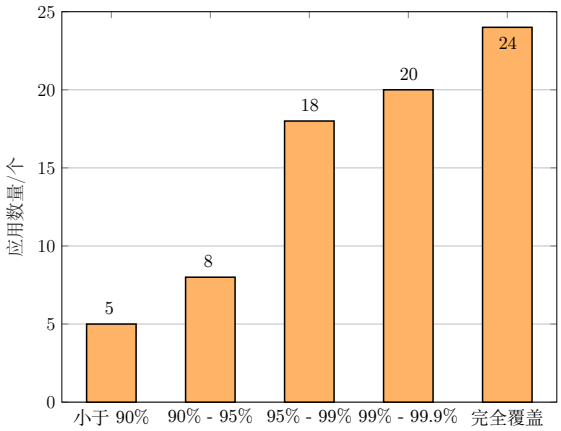


图 5-11 测试覆盖率

图 5-10 揭示了对本系统及其安全策略机制可用性的测试结果。实验数据表明，本系统能够对 49.3% 的应用程序所产生的所有危险权限请求和敏感 API 调用进行识别和控制，安全决策中心和各个策略执行器能够完全代理并控制所有的涉及隐私的访问请求。尽管如此，在 36% 的应用程序中我们的系统只能识别并控制一部分危险权限请求和 API 调用，还有少数访问请求无法得到有效地限制或封锁。除此之外，本系统在 4% 的应用程序中只能控制少量的危险权限请求，甚至无法通过静态分析方法检测出任何危险权限请求，抑或无法对检测出的敏感 API 调用行为做出任何有效控制。考虑到现代 Android 应用程序开发不仅支持基于 Java 或 Kotlin 的传统原生开发方式，还支持一些其他的 hybrid 应用开发形式，这可能导致我们基于原生 Android 虚拟机字节码的静态分析工具和字节码注入技术失去作用。举例来说，一些应用程序通过嵌入 WebView 使得 Android 应用程序支持 Web 前端的工具组件，进而使得应用可以通过 JavaScript 代码实现业务逻辑；而一部分游戏应用可能会使用如 Unity 3D 等游戏引擎辅助开发，这些工具都并非基于 Java 和 Android 虚拟机工作，从而无法被我们的系统有效控制。通过对相关应用程序的日志的分析也佐证了我们的判断。

仅仅说明对单独的操作和请求进行控制的能力还不足以全面反映安全策略的可用性。在测试中我们还需要关注的是测试覆盖率，即对应用程序的测试覆盖了多大比例的分支流程和代码执行范围。考虑到 Android 6.0 之后系统对应用程序提供的动态性支持，仅仅对策略执行器进行单元测试是远远不够的。图 5-11 反映了我们对数据源中所有应用程序测试的覆盖率，从图中可以看出，对于 82.67% 的应用程序我们的测试能够覆盖 95% 以上的代码逻辑分支，而测试覆盖率小于 90% 的应用程序占比仅有 6.7%。结合图 5-10 中体现的对单个策略执行代理的可用性测试，可以认为我们的系统可以有效限制较大比例的 Android 应用程序的敏感操作并对用户的隐私进行保护。

5.2.4 对应用性能的影响

对隐私保护系统和安全策略注入机制另一非常重要的评估是考察字节码注入对应用程序性能的影响，因为这也将直接影响应用程序的可用性。我们在这里所指的性能是一个较为广义的概念，它不仅指代应用的启动和运行效率，还包含了对应用大小和功耗的考察。

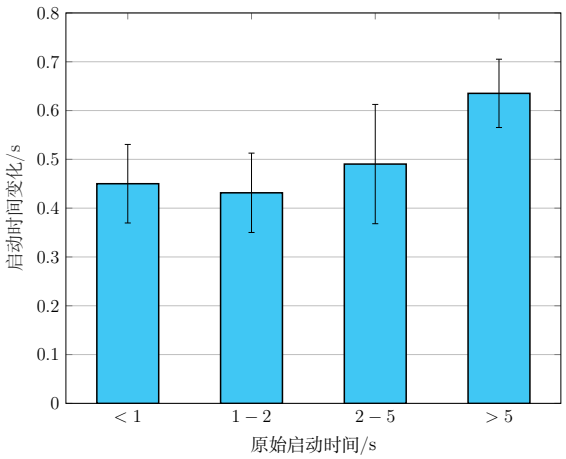


图 5-12 安全策略对应用启动时间的影响

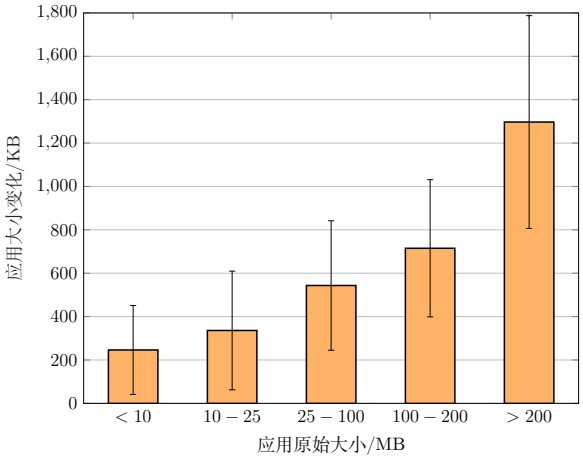
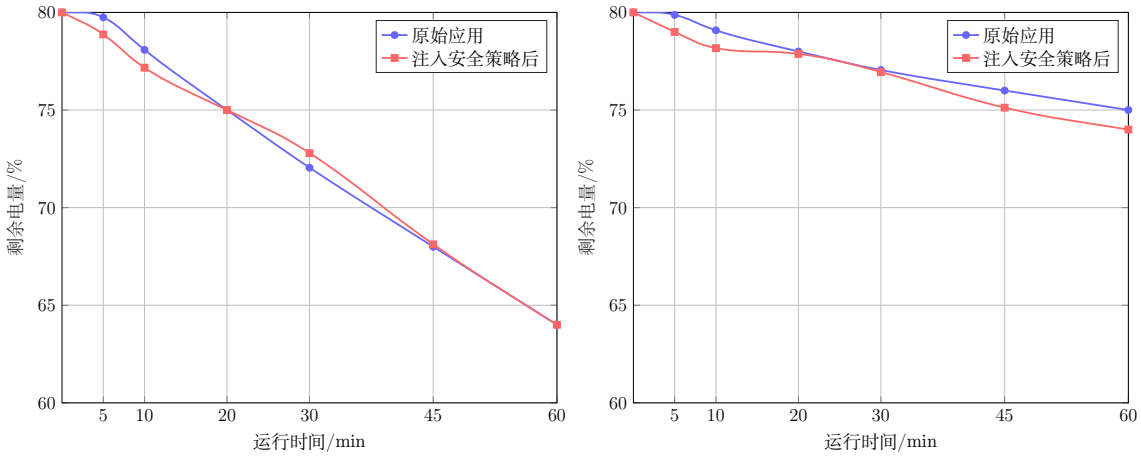


图 5-13 安全策略对应用大小的影响

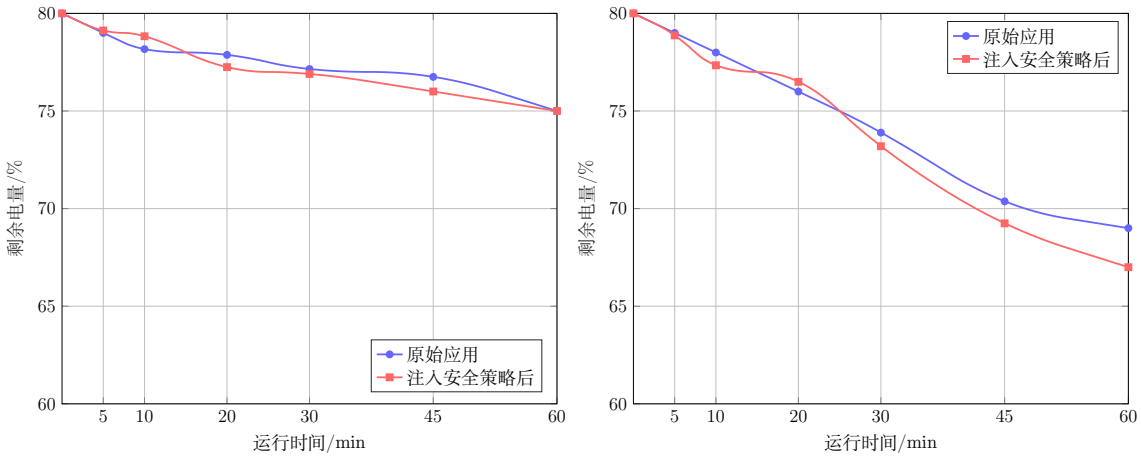
图 5-12 反映了本系统注入的安全策略对 Android 应用程序启动速度的影响。实验结果表明，尽管不同应用程序的启动时间分布广泛，但注入安全策略对启动时间的影响与应用原本的启动时间并无明显关联。对于原始启动时间小于 1s 的应用程序，安全策略对启动时间的平均影响约为 0.411s，实验数据的标准差为 0.0804s；而对于原始启动时间大于 5s 的那些大型应用，安全策略的平均影响也仅为 0.635s，而此时的标准差仅为 0.0699s。尽管如此，需要特别注意的是，对于那些启动所需时间本就较短的应用程序来说，注入的安全策略所造成的用户直观影响更大。这也很好理解，因为在这种情况下安全策略带来的开销所占的比例更高，而这种对比会给用户明显的程序变慢的感觉。事实上，注入的安全策略并非在程序启动同时被立即调用，而是在程序执行到特定位置或进行敏感操作时才会被唤起。因此从理论上说，程序启动时会对启动时间产生额外开销的只是安全决策中心的初始化过程，而这个过程本身对程序资源的耗费相对程序自身的初始化过程所产生的开销也要小得多。实验的结果也能很好地符合我们的理论分析。

安全策略对应用程序大小的影响和对启动时间的影响模式也很类似。图 5-13 揭示了注入了安全策略前后应用程序大小的变化。从实验结果可以很直观地看到，注入的安全策略给应用程序的大小带来的影响相比于应用程序自身的大小几乎可以忽略不计。对于原始体积小于 10MB 的应用程序，注入的安全策略带来的平均体积影响仅为 246.13KB，实验标准差约为 204.949KB；而对于原始体积大于 200MB 的大型应用，注入的安全策略所带来的平均体积影响也仅为 1296.95KB，此时标准差为 490.526KB。但是需要说明的是，安全策略对体积的影响和应用程序本身的大小呈现出明显的正相关，这与对应用

启动时间的影响模式完全不同。从图 5-13 可以看到，尽管不同应用程序本身的大小分布广泛，但是安全策略注入所带来的额外空间开销占重打包后应用程序总大小的比例始终保持在一定范围内。换言之，随着应用程序规模的增大，注入的安全策略字节码规模好像也增多了。从理论上来说注入的字节码分为两个部分，即安全决策中心和各个策略执行代理。而安全决策中心的大小一般是相对固定的，只有策略执行代理与具体的应用程序有关。更具体地，当应用程序中涉及到更多的敏感操作和权限调用时，所需的策略执行器就相对增多。因此，与其说安全策略带来的空间开销和应用程序本身的大小有关，不如说是和应用程序中包含的敏感操作和权限调用数量有关。而从统计学上来说，体积较大的应用程序恰好有更大概率包含这些操作或调用，这就解释了安全策略的体积影响与应用程序原始大小之间的这种统计关系。



(a) 游戏应用　　(b) 新闻应用



(c) 社交应用　　(d) 影音应用

图 5-14 安全策略对应用功耗的影响

对于安全策略对应用程序功耗的影响，我们以数据源中的 4 款应用程序作为基础进行了详细的测试。我们邀请了 2 位同学辅助了我们的测试，他们被要求使用我们提供的一台搭载 Android 10 系统的华为 P30 Pro 手机进行相关操作。在每次测试开始之前，我

们将手机充电至 80%，随后要求受试者使用手机打开注入了安全策略的特定应用程序，并依照程序的正常功能连续使用 1 个小时。随后，我们在使用过程的第 5、10、20、30、45 和 60 分钟分别记录此时手机的剩余电量，并依照记录数据绘制功耗曲线。为了和原始情况进行对比，对于每个应用程序，我们还要求受试者按照上述流程使用未经安全策略注入的原始程序，并记录对照数据。我们提供给受试者的应用程序分别是一个游戏应用、一个新闻应用、一个社交应用和一个视频应用，图 5-14 揭示了这 4 组实验的实验结果。从实验数据可以看出，本系统所注入的安全策略对应用程序所产生的功耗影响较小，在长达 1 个小时的使用过程中最多给功耗造成 2% 的差距，而在某些场合由于偶然因素甚至出现了注入了安全策略的应用功耗更小的情况。这从侧面说明了本系统对应用程序的性能并不会产生用户可感知级别的影响。需要说明的是，由于功耗测试极其耗费时间和精力，因此我们没有对更多的应用程序进行功耗测试。因此本小节所进行的测试只能部分反映本系统对功耗的影响，更全面的分析仍然需要依赖对更多应用程序进行的测试。

5.2.5 隐私权限的分析

本系统的日志系统也提供了一些额外的信息。正如 5.1.6 小节所述，本系统中包含的日志系统能够记录应用程序的操作和调用信息。我们对数据集中所有的应用程序的调用日志进行了定量分析，从中得到了应用程序调用权限的大致分布情况。已有很多研究通过 Android 应用程序的静态分析和概率学对 Android 操作系统中的各权限的安全威胁进行了详细的分析[60][61]。将运行日志所得到的权限分布和这些研究相结合，我们可以甄别出数据源中应用程序高频率使用的具有安全风险的权限，如表 5.1 所示。

表 5.1 高频使用的有安全风险的权限

权限名称	说明
RECORD_AUDIO	记录音频信息
READ_CONTACTS	读取通讯录
WRITE_CONTACTS	写入通讯录
READ_LOGS	读取系统日志
ACCESS_NETWORK_STATE	访问网络状态
READ_PHONE_STATE	读取设备信息
READ_SMS	读取 SMS 信息
SEND_SMS	发送 SMS 信息
ACCESS_COARSE_LOCATION	访问粗粒度定位
ACCESS_FINE_LOCATION	访问细粒度定位
RECEIVE_WAP_PUSH	接收 WAP 推送

需要特别注意的是，表 5.1 中提到的一些权限并没有在第二章中给出的 Android 危险权限表 2.1 中出现，比如 `READ_LOGS` 和 `ACCESS_NETWORK_STATE` 等。尽管 Android 官方并未将这些权限调用列为危险权限，但研究表明对这些权限的调用也常常会威胁到用户的隐私和安全，这就需要系统对这些调用给予特别的注意[61]。

5.2.6　差分隐私参数的选择

对于差分隐私机制对数据的保护能力，我们也进行了系统性地评估。显然，对于原始数据的任何噪声扰动行为都会不可避免地对应用开发者地数据挖掘和数据分析带来影响，因此差分隐私扰动噪声的添加始终需要考察数据安全性和可用性的平衡。因此，为了考察差分隐私机制对应用数据的影响，我们针对数据集中的 4 个不同应用的特定数据流进行了详细的分析。

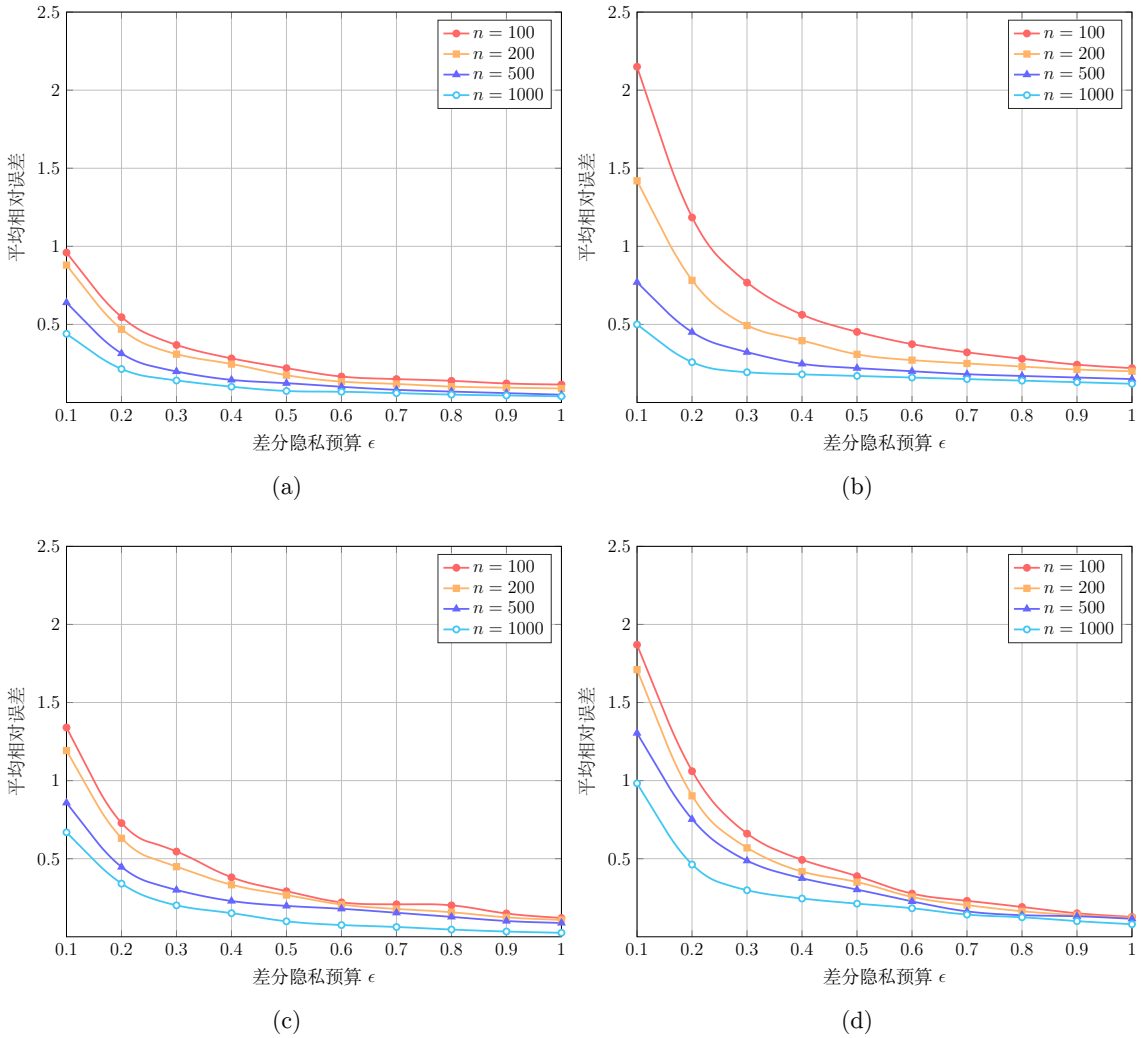


图 5-15 差分隐私的数据脱敏能力

图 5-15 揭示了差分隐私对不同数据流的影响。我们从应用程序集中随机挑选了 4 个应用程序，并拦截这 4 个应用程序中随机接口的服务器通信数据来进行此项分析，图中显示了不同的差分隐私预算 ϵ 对数据的平均相对误差的影响。从这 4 张图中我们可以发现一个共同的趋势，即当差分隐私预算 ϵ 增加时，数据的平均相对误差迅速减小并最终收敛到一个稳定值，该收敛值取决于数据流的固有属性。而当隐私预算 ϵ 较小时差分隐私的保护强度提高，此时差分隐私所带来的数据平均误差显著提高，因而此时数据的可用性极低。实验证明了实现更强的差分隐私保护需要以牺牲数据流的有效性和可用性为代价。

对于基于高斯过程的差分隐私机制，我们还需要考察尺度变换参数 n 对可用性的影响。图 5-15 反映了基于高斯过程的差分隐私机制相比普通的基于拉普拉斯扰动噪声的差分隐私机制能够更好的保护原始数据的可用性。由实验结果可以发现当 n 增加时，数据流的平均相对误差能够被有效降低，这意味着通过提升变换参数能够在保障差分隐私机制的数据脱敏能力的前提下提升数据的可用性。尽管如此，我们在 4.3 节中也说明了这种对 n 的提升并不是没有限制的。当对数据变换的调整超出一定限度时，经过扰动的数据将失去普适性，进而无法适应更复杂的数据分析和机器学习算法。此时尽管数据流的平均相对误差仍然不高，但是事实上这些数据已经从另一种意义上丧失了可用性。

5.3 本章小结

本章介绍了我们基于第三章和第四章所提出的 Android 应用程序隐私保护机制和数据脱敏机制所构建的 Android 应用程序隐私保护系统。该系统能够利用安全策略注入技术对 Android 应用程序的敏感操作和权限调用进行细粒度的限制，进而保护用户的隐私。此外，系统所提供的差分隐私机制也能够对应用程序与服务器通信中涉及的所有数据流进行脱敏处理，在保护数据隐私的同时保障应用程序的可用性。具体地，该系统的安全策略注入机制利用了基于字节码的安全策略生成与注入技术，该技术能够在 Android 应用程序中植入对危险权限和敏感 API 调用的控制策略，并根据用户或组织的需求进行动态的调整；该系统的数据脱敏机制利用了基于高斯过程的差分隐私技术，该技术能够在基于拉普拉斯噪声的差分隐私机制的基础上进一步提升相同差分隐私预算情况下的数据可用性，进而在加强应用程序隐私保护功能的基础上维护应用程序的的数据分析需求及其基本功能。对系统的评估和分析的结果表明对 Android 应用程序进行的安全策略注入在 85.27% 的情况下不会对应用程序的健壮性造成影响，并在此基础上对 85.33% 的敏感 API 调用和危险权限请求进行有效限制。与此同时，注入的安全策略对应用的启动时间的影响至多只有平均 0.635s，平均占应用原启动时长的约 6.71%；在应用程序重打包后仅增加了平均约 0.372% 的大小；此外安全策略对功耗的影响几乎可以忽略不计。对于差分隐私的相关参数对数据可用性的影响，我们也进行了系统性的分析。实验结果表明，基于高斯过程的差分隐私技术相比普通的基于拉普拉斯扰动噪声的差分隐私机制，能够在获取相同的隐私保护效果的基础上进一步增强数据的可用性。

第六章　总结与展望

6.1　全文总结

Android 操作系统提供了基于权限控制的安全机制，但是该机制的粗粒度控制方式不能满足相当一部分用户或组织的安全需求，而且存在可以被恶意攻击者利用的系统性漏洞。为了防御这些漏洞带来的恶意攻击，已有研究通过静态分析方法来针对特定 Android 应用程序进行分析和防护。然而考虑到这种分析只能将特定 Android 应用程序作为一个整体来判断其是否具有恶意，这种分析仍然难以防止所谓良性应用程序滥用危险权限或者用户的敏感信息[10][12]。在此基础上，本文提出了通过逆向工程实现在字节码层面的安全策略注入方法，该方法能够在代码层面对程序的逻辑进行审计和调整。通过对应用程序的敏感 API 访问和权限请求进行代码层面的代理与限制，本文所提出的字节码注入技术能够根据用户的需求动态地调整对应用程序的敏感行为的限制与封锁。

为了实现这一目的，本文首先研究了 Android 操作系统的基于权限控制的安全模型，对 Android 操作系统常见的涉及用户隐私的敏感权限进行了梳理。本文在第二章中考察了 Android 的 Dalvik 虚拟机和 Android Runtime 运行时环境的代码编译与执行原理，以及由应用程序的 APK 包反编译得到虚拟机字节码和原始静态资源的相关技术。

针对现有研究无法检测程序的具体执行流程，以及无法根据用户需求的变化动态地调整应用程序地权限控制策略的弊端，本文在第三章中提出了一种基于权限控制策略的动态权限控制方法。该方法能够根据用户的需求生成具体的安全策略和权限控制列表，并能够在程序的运行期根据策略动态地调整敏感权限的的授予和撤销。该方法从程序的逻辑流程、执行分支、敏感操作与信息处理和威胁等级等方面对应用程序进行分析，并生成相应的安全策略。

此外，针对现有的代码注入技术只能静态地将代码注入权限敏感方法中，且无法根据安全策略地调整动态地调整权限控制级别和策略，本文在安全策略更新时需要重新注入安全代码并重新包装应用的问题，提出了一种新的字节码注入技术。本文在第三章提出了一个动态应用程序权限控制机制，该机制可以使用封装技术将安全功能添加到缺乏安全审计的应用程序的权限敏感位置。该机制可以在安全策略发生改变时动态地切换权限控制级别，无需重新注入代码并对应用程序进行重打包。

针对现有的代码注入技术可能会影响良性应用程序的正常功能的问题，本文在第四章提出了一种基于差分隐私控制的信息保护和脱敏方法。该方法能够提供一种在确保隐私数据安全的前提下对数据执行计算的方法。本研究基于差分隐私的可证明隐私保证，通过公开的伪装方法将隐私数据和高斯过程相结合，在保证应用宏观数据统计功能的正确性的同时保护用户个人的具体隐私信息，在可用性和安全性之间实现平衡。

在上述技术和方案的基础上，本文在第五章设计并实现了一个基于字节码注入的Android 应用程序隐私防护系统。该系统由策略设计和生成系统、虚拟机字节码注入系统和差分隐私数据脱敏系统组成。该系统能够以工具包的形式提供给应用程序开发者和安全敏感的组织与个人使用，在字节码级别调整并限制应用程序的权限获取与利用行为。对系统的评估和分析的结果表明本系统确实能够实现其设计目标。对 Android 应用程序进行的安全策略注入在大多数情况下不会对应用程序的可用性造成影响，并在此基础上对 85.33% 的敏感 API 调用和危险权限请求进行有效限制。与此同时，注入的安全策略对应用的启动时间的影响仅为约 6.71%，在应用程序重打包后仅增加了平均约 0.37% 的大小。对差分隐私机制的实验结果表明，基于高斯过程的差分隐私技术相比普通的基于拉普拉斯扰动噪声的差分隐私机制，能够在获取相同的隐私保护效果的基础上进一步增强数据的可用性。

6.2 问题讨论

我们在系统设计和实验评估的过程中对本研究的工作进行了深刻的反思，其中暴露出的一些问题有必要得到仔细地讨论与分析。本节列举了我们认为较为重要并且有希望在未来得到改进或解决的若干问题。

6.2.1 实验条件的限制

本文在第五章中从国内的几大应用市场中采集了上百个应用程序作为数据源参与对系统的测试，然而并非所有的测试项目都对所有的 129 个应用程序进行了完整的测试与分析。由于时间和精力有限，我们在应用功耗和差分隐私参数的测试中都只随机挑选了少数几个应用程序进行测试，因此这部分的测试并不能严格反映所有应用程序的特征。此外由于经济原因，我们在测试应用程序时只使用了 4 台 Android 硬件设备。这些设备的软件版本分布和硬件性能情况都只能反映个例，无法反映应用在更广泛的 Android 设备中运行的实际情况。因此在时间和经济条件允许的情况下，本系统尚需更全面的测试以反映其在更多应用和设备中的运行情况。

6.2.2 安全策略自身的安全隐患

本文中我们主要讨论了通过注入安全策略及其字节码级表示对 Android 应用程序进行权限和 API 调用的控制与封锁。虽然我们的系统能够通过安全策略在一定程度保障 Android 应用的隐私安全，但是注入的安全策略自身也有可能给用户或组织带来新的隐患。尽管我们已经尽最大努力完善了我们的系统和安全机制，我们仍然无法保证我们注入的安全策略及其字节码不会给 Android 应用程序引入其他的安全漏洞。此外当应用自身升级或虚拟机标准发生变化时，我们的系统可能需要同步升级以防止出现因应用程序二进制接口变化而导致的兼容性问题。

6.2.3　法律问题

由于我们的系统对 Android 应用程序的代码进行了修改，而这种行为有可能违反了相关应用程序的条约与政策，这是需要讨论的另一个问题。我们查阅了 Google Play 的分发政策，其中规定了禁止任何第三方对平台上分发的 Android 应用程序源代码进行任何形式的修改和再发布[62]。然而我们的系统所进行的修改并非在源代码级，而是在字节码级，因此这种行为是否违反了 Google Play 的政策尚存在争议。而其他 Android 应用平台和具体应用程序的条约和规范大多并未规定针对字节码级的修改所涉及的法律问题，因此我们可以认为本系统目前不会涉及到相关的法律纠纷中。

致　谢

在论文即将付梓之际，衷心感谢所有对本研究内容和论文撰写提供帮助的老师和同学，这篇论文的完成离不开他们的无私奉献和热心帮助。

首先需要感谢我的导师宋宇波老师。自我本科四年级进入实验室以来，宋老师一直在给予我无私的指导和教育。宋老师身据深厚的科研功底和丰富的工程经验。这近四年间，宋老师将我从一个对科学研究和工程实践一无所知的普通学生培养成为一个具有初步科研能力和工程能力的研究者。他对我的帮助还不仅限于此，宋老师风趣幽默、平易近人的性格和人格魅力也给我短暂的研究生求学生涯带来了深刻的影响。在此衷心感谢宋老师地教育和指导。

其次需要感谢的是我的同届同门石伟和李轩。作为同届进入实验室的同门，我们三人共同完成了很多的科研和工程项目，在此过程中我们的情谊愈发深厚。离别在即，祝愿他们都能拥有美好的未来，在新的天地里也能像在求学时一样快乐。

实验室和同师门的师兄和师姐们为我的研究生生涯带来了另一重的快乐和幸福。已经毕业的董启宏、张克落、黄强、武威、魏一鸣师兄和杨慧文、罗平师姐都是我科研路上的引路人，他们帮助我适应了硕士研究生新的工作氛围和生活方式，也在生活上无微不至的关照我。我们共处一间实验室的时间虽然短暂，但是同门之谊却将长久地延续下去。祝愿师兄和师姐们都能收获美好的前程，在各自的工作岗位上创造新的价值。

也要感谢师弟杨俊杰、樊明、张仕奇、赵灵奇、马小松、徐前川、蒋心造和师妹祁欣妤、耿益瑾、金星妤、陈琪，他们的到来为实验室新增了源源不断的生机和活力，他们的蓬勃朝气和奋发精神都深深感染着我、鞭策着我。祝愿他们能在科研一途不断精进，勇攀高峰。

我的父母在我近 20 年的求学生涯中，一直给予我爱护、理解和支持。他们的支持是我努力奋斗、不懈进取的动力，支撑着我迎接困难和挑战，实现人生的梦想。感谢他们对我的包容和爱护，真心祝愿他们永远健康、幸福美满。

这篇论文成稿正值国内爆发新型冠状病毒肺炎疫情期间，此次疫情使我的研究进度和论文撰写过程受到了较大影响。衷心感谢为战胜疫情长时间奋战在抗疫一线的医护工作者，没有他们的无私奉献和舍己为人，就没有我们安逸的生存环境。此外也要感谢所有湖北省和武汉市居民，他们为此次抗疫作出了难以想象的巨大牺牲，中国人民永远不会忘记他们。

最后感谢母校东南大学为我提供了宝贵的学习和科研环境，感谢所有教育、指导过我的老师、同学和朋友，是你们成就了现在的我。也衷心感谢为审阅本论文而付出宝贵时间和精力的各位老师，你们的宝贵意见和批评，是对我的科研生涯和人生之路的鞭策和鼓励，我将永远铭记在心。

参考文献

[1] 中国互联网中心. 第 44 次中国互联网发展状况统计报告[EB/OL]. http://www.cnnic.net.cn/hlwfzyj/hlwxzbg/hlwtjbg/201908/t20190830_70800.htm, 2019.

[2] IDC. Device market trends and smartphone market share[EB/OL]. https://www.idc.com/promo/smartphone-/market-share/os, 2020.

[3] EXTREMETECH. Google throws nearly a billion android users under the bus, refuses to patch os vulnerability[EB/OL]. https://www.extremetech.com/mobile/197346-google-/throws-nearly-a-billion-android-users-under-the-bus-refuses-to-patch-os-vulnerability, 2015.

[4] NAUMAN M, KHAN S, ZHANG X. Apex: Extending android permission model and enforcement with user-defined runtime constraints[C]. Proceedings of the 5th ACM Symposium on Information, Computer and Communications Security. ACM, 2010. 328-332.

[5] MAHINDRU A, SANGAL A. Deepdroid: Feature selection approach to detect android malware using deep learning[C]. 2019 IEEE 10th International Conference on Software Engineering and Service Science (ICSESS). IEEE, 2019. 16-19.

[6] FORBES. Report: 97% of mobile malware is on android. this is the easy way you stay safe [EB/OL]. https://www.forbes.com/sites/gordonkelly/2014/03/24/report-97-of-mobile-/malware-is-on-android-this-is-the-easy-way-you-stay-safe, 2014.

[7] CHEN K, WANG P, LEE Y, et al. Finding unknown malice in 10 seconds: Mass vetting for new threats at the google-play scale[C]. 24th USENIX Security Symposium (USENIX Security 15). Washinton D.C: USENIX, 2015. 659-674.

[8] ZHOU Y, JIANG X. Dissecting android malware: Characterization and evolution[C]. 2012 IEEE Symposium on Security and Privacy. IEEE, 2012. 95-109.

[9] BLÄSING T, BATYUK L, SCHMIDT A D, et al. An android application sandbox system for suspicious software detection[C]. 2010 5th International Conference on Malicious and Unwanted Software. IEEE, 2010. 55-62.

[10] GIBLER C, CRUSSELL J, ERICKSON J, et al. Androidleaks: Automatically detecting potential privacy leaks in android applications on a large scale[C]. International Conference on Trust and Trustworthy Computing. Springer, 2012. 291-307.

[11] SCHÜTTE J, TITZE D, DE FUENTES J M. Appcaulk: Data leak prevention by injecting targeted taint tracking into android apps[C]. 2014 IEEE 13th International Conference on Trust, Security and Privacy in Computing and Communications. IEEE, 2014. 370-379.

[12] VIDAS T, CHRISTIN N. Evading android runtime analysis via sandbox detection[C]. Proceedings of the 9th ACM Symposium on Information, Computer and Communications Security. ACM, 2014. 447-458.

[13] ANDROIDDEVELOPERS. Permissions overview[EB/OL]. https://developer.android.com/guide/topics/permissions/overview, 2019.

[14] FARUKI P, BHARMAL A, LAXMI V, et al. Android security: A survey of issues, malware penetration, and defenses[J]. IEEE Communications Surveys and Tutorials, 2019, 17(2):998-1022.

[15] FELT A P, CHIN E, HANNA S, et al. Android permissions demystified[C]. Proceedings of the 18th ACM conference on Computer and Communications Security. ACM, 2011. 627-638.

[16] LIU X, DU X, ZHANG X, et al. Adversarial samples on android malware detection systems for iot systems[J]. Sensors, 2019, 19(4):974.

[17] ANDROIDDEVELOPERS. Distribution dashboard[EB/OL]. https://developer.android.com/about/dashboards/, 2020.

[18] NEISSE R, STERI G, GENEIATAKIS D, et al. A privacy enforcing framework for android applications[J]. Computers & Security, 2016, 62:257-277.

[19] KANG H, JANG J W, MOHAISEN A, et al. Detecting and classifying android malware using static analysis along with creator information[J]. International Journal of Distributed Sensor Networks, 2015, 11(6):474-479.

[20] PENG H, GATES C, SARMA B, et al. Using probabilistic generative models for ranking risks of android apps[C]. Proceedings of the 2012 ACM Conference on Computer and Communications Security. ACM, 2012. 241-252.

[21] ENCK W, ONGTANG M, MCDANIEL P. On lightweight mobile phone application certification[C]. Proceedings of the 16th ACM Conference on Computer and Communications Security. ACM, 2009. 235-245.

[22] SARMA B P, LI N, GATES C, et al. Android permissions: a perspective combining risks and benefits[C]. Proceedings of the 17th ACM Symposium on Access Control Models and Technologies. ACM, 2012. 13-22.

[23] XIE N, WANG X, WANG W, et al. Fingerprinting android malware families[J]. Frontiers of Computer Science, 2019, 13(3):120-134.

[24] CHRISTODORESCU M, JHA S, SESHIA S A, et al. Semantics-aware malware detection [C]. 2005 IEEE Symposium on Security and Privacy (S&P'05). IEEE, 2005. 32-46.

[25] LEE J, JEONG K, LEE H. Detecting metamorphic malwares using code graphs[C]. Proceedings of the 2010 ACM Symposium on Applied Computing. ACM, 2010. 1970-1977.

[26] SUAREZ-TANGIL G, DASH S K, AHMADI M, et al. Droidsieve: Fast and accurate classification of obfuscated android malware[C]. Proceedings of the Seventh ACM on Conference on Data and Application Security and Privacy. ACM, 2017. 309-320.

[27] MAIORCA D, ARIU D, CORONA I, et al. Stealth attacks: An extended insight into the obfuscation effects on android malware[J]. Computers & Security, 2015, 51:16-31.

[28] MING J, XIN Z, LAN P, et al. Impeding behavior-based malware analysis via replacement attacks to malware specifications[J]. Journal of Computer Virology and Hacking Techniques, 2017, 13(3):193-207.

[29] SARACINO A, SGANDURRA D, DINI G, et al. Madam: Effective and efficient behavior-based android malware detection and prevention[J]. IEEE Transactions on Dependable and Secure Computing, 2016, 15(1):83-97.

[30] VINOD P, SHOJAFAR M, KUMAR N, et al. Identification of android malware using refined system calls[J]. Concurrency, Computation Practice and Experience, 2019(2):1-30.

[31] SHABTAI A, KANONOV U, ELOVICI Y, et al. Andromaly: a behavioral malware detection framework for android devices[J]. Journal of Intelligent Information Systems, 2012, 38(1):161-190.

[32] MA W, DUAN P, LIU S, et al. Shadow attacks: Automatically evading system-call-behavior based malware detection[J]. Journal in Computer Virology, 2012, 8(1-2):1-13.

[33] FORREST S, HOFMEYR S, SOMAYAJI A. The evolution of system-call monitoring [C]. 2008 Annual Computer Security Applications Conference (ACSAC). IEEE, 2008. 418-430.

[34] SRIVASTAVA A, LANZI A, GIFFIN J, et al. Operating system interface obfuscation and the revealing of hidden operations[C]. International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment. Springer, 2011. 214-233.

[35] XU L, ZHANG D, ALVAREZ M A, et al. Dynamic android malware classification using graph-based representations[C]. 2016 IEEE 3rd International Conference on Cyber Security and Cloud Computing (CSCloud). IEEE, 2016. 220-231.

[36] DASH S K, SUAREZ-TANGIL G, KHAN S, et al. Droidscribe: Classifying android malware based on runtime behavior[C]. 2016 IEEE Security and Privacy Workshops (SPW). IEEE, 2016. 252-261.

[37] CHEN S, XUE M, TANG Z, et al. Stormdroid: A streaminglized machine learning-based system for detecting android malware[C]. Proceedings of the 11th ACM on Asia Conference on Computer and Communications Security. ACM, 2016. 377-388.

[38] LEE S H, KIM S H, KIM S, et al. Appwrapping providing fine-grained security policy enforcement per method unit in android[C]. 2017 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW). IEEE, 2017. 36-39.

[39] 丰生强. Android 软件安全权威指南[M]. 第 2 版. 电子工业出版社, 2019. 23-25.

[40] IBOTPEACHES. Apktool: A tool for reverse engineering android apk files[EB/OL]. https://github.com/iBotPeaches/Apktool, 2020.

[41] JESUSFREKE. Smali and baksmali[EB/OL]. https://github.com/JesusFreke/smali, 2020.

[42] ANDROGUARD. Androguard reverse engineering, malware analysis of android applications[EB/OL]. https://github.com/androguard/androguard, 2020.

[43] DECAF PROJECT. Dynamic executable code analysis framework: a binary analysis platform based on qemu[EB/OL]. https://github.com/decaf-project/DECAF, 2020.

[44] FENG Y, LIU P, ZHANG J. A mobile terminal based trajectory preserving strategy for continuous querying lbs users[C]. 2012 IEEE 8th International Conference on Distributed Computing in Sensor Systems. IEEE, 2012. 92-98.

[45] DWORK C. Differential privacy: A survey of results[C]. International Conference on Theory and Applications of Models of Computation. Springer, 2008. 1-19.

[46] BLUM A, DWORK C, MCSHERRY F, et al. Practical privacy: the sulq framework [C]. Proceedings of the twenty-fourth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems. ACM, 2005. 128-138.

[47] BLUM A, LIGETT K, ROTH A. A learning theory approach to noninteractive database privacy[J]. Journal of the ACM (JACM), 2013, 60(2):1-25.

[48] SMITH M, ÁLVAREZ M, ZWIESSELE M, et al. Differentially private regression with gaussian processes[C]. International Conference on Artificial Intelligence and Statistics. IEEE, 2018. 1195-1203.

[49] HALL R, RINALDO A, WASSERMAN L. Differential privacy for functions and functional data[J]. Journal of Machine Learning Research, 2013, 14(Feb):703-727.

[50] GIBBS M N. Bayesian gaussian processes for regression and classification[D]. Citeseer, 1998.

[51] PACIOREK C J, SCHERVISH M J. Nonstationary covariance functions for gaussian process regression[C]. Advances in Neural Information Processing Systems. MIT, 2004. 273-280.

[52] SNELSON E, GHAHRAMANI Z. Sparse gaussian processes using pseudo-inputs[C]. Advances in Neural Information Processing Systems. MIT, 2006. 1257-1264.

[53] DEISENROTH M P, FOX D, RASMUSSEN C E. Gaussian processes for data-efficient learning in robotics and control[J]. IEEE Transactions on Pattern Analysis and Machine Intelligence, 2013, 37(2):408-423.

[54] WILLIAMS C K, RASMUSSEN C E. Gaussian processes for machine learning: volume 2 [M]. MA: MIT press Cambridge, 2006.

[55] KUSNER M, GARDNER J, GARNETT R, et al. Differentially private bayesian optimization[C]. International Conference on Machine Learning. ICML, 2015. 918-927.

[56] CHAUDHURI K, VINTERBO S A. A stability-based validation procedure for differentially private machine learning[C]. Advances in Neural Information Processing Systems. MIT, 2013. 2652-2660.

[57] DWORK C, ROTH A. The algorithmic foundations of differential privacy[J]. Foundations and Trends® in Theoretical Computer Science, 2014, 9(3–4):211-407.

[58] CHAUDHURI K, MONTELEONI C, SARWATE A D. Differentially private empirical risk minimization[J]. Journal of Machine Learning Research, 2011, 12(Mar):1069-1109.

[59] 艾媒报告. 2019 年中国移动应用商店行业的现状与发展趋势[EB/OL]. https://www.iimedia.cn/c1020/65302.html, 2019.

[60] JANG J W, KANG H, WOO J, et al. Andro-dumpsys: Anti-malware system based on the similarity of malware creator and malware centric information[J]. Computers & Security, 2016, 58:125-138.

[61] WIJESEKERA P, BAOKAR A, HOSSEINI A, et al. Android permissions remystified: A field study on contextual integrity[C]. 24th USENIX Security Symposium (USENIX Security 15). Washinton D.C: USENIX, 2015. 499-514.

[62] GOOGLE. Google play developer distribution agreement[EB/OL]. https://play.google.com/intl/ALL_us/about/developer-distribution-agreement.html, 2019.

附录 A 系统测试阶段所采集的应用程序

表 A.1 应用程序列表

应用名	大小	包名	版本号
逃出实验室	105.48M	com.tencent.tmgp.rod	V1.2.5
全民飞机大战	651.48M	com.tencent.feiji	V1.0.100
七猫精品小说	14.85M	com.book2345.reader	V5.11
Wifi 分析器	7.91M	com.farproc.wifi.analyzer	V20083
悟空理财	32.21M	com.wukonglicai.app	V4.2.0
违章查询	25.11M	com.sohu.auto.helper	V8.1.0
鲸鸣	54.01M	com.sl.whale	V0.11.35
跳跃吧球球	23.02M	com.wepie.jumpball.tencent	V1.3.1
MC 音乐库	4.44M	com.blueocean.extensionthree	V1.0.7
玖富钱包	56.42M	com.bank9f.weilicai	V5.1.4
龙支付商家版	49.46M	com.ccb.imerchant	V1.0.6
饭局狼人	96.63M	com.mewe.wolf	V3.6.0
作业精灵	30.9M	com.pcncn.jj	V3.6.29
商品条形码查询	1.85M	cn.yn.zyl.goodsbarcode	V1.8
滴答节拍器	38.09M	com.vtmi.gbpr8.h4l6n	V1.0.0
两面	51.12M	com.ilun.secret	V3.0.55
城市飞车	46.42M	com.racergame.cityracing3d	V6.9.8
微记账	22.99M	com.kunxun.wjz	V5.9.3
在线农场	18.23M	com.teamlava.farmstory	V1.9.6.4
亿贝卡	23.11M	com.jfbank.primecard	V1.1.1
我的汉克狗	121.06M	com.outfit7.mytalkinghank	V1.8.11
喵星大作战	70.75M	com.tencent.tmgp.cats	V1.11.1
马甲线	15.78M	cn.pocdoc.majiaxian	V3.3.7
建融慧家	83.81M	com.ccb.scchome	V1.1.7
风水罗盘实景	48.47M	com.mmc.fengshui.pass	V4.1.0
铲屎官的日常	34.31M	com.xiaoqs.petalarm	V1.1.2
小目标	46.57M	cn.nineton.starget	V3.3.0
365 日历	13.98M	com.when.coco	V7.3.3

（续表）

龙纹三国	131.02M	com.tencent.tmgp.sanguo	V1.0.28
同城热恋	7.7M	com.huizheng.tcyyhz	V5.6.4
遇见	36.67M	net.iaround	V7.8.1
挖财信用卡管家	12.26M	com.caimi.creditcard	V6.5.14
米尔军事	18.69M	com.miercnnew.app	V2.7.9
真实模拟驾驶	53.87M	com.xgame.rdriver	V3.02
卡点视频制作	49.39M	com.mobile.kadian	V3.4.0
全民突击	649.5M	com.tencent.WeFire	V4.13.0
数字尾巴	43.36M	com.funinput.digit	V4.2
嘀嗒出行	109.95M	com.didapinche.booking	V8.8.5
钱站	17.67M	aiqianjin.jiea	V3.3.5
橙感	12.28M	com.walimai.client	V4.0.0
私密相册	16.8M	cn.bluecrane.private_album	V3.9.7
极品小说	11.61M	com.qukan.jiping	V1.1.3
滑雪大冒险 2	99.22M	com.yodo1.skisafari2	V1.6.2.1
仙风道骨	179.57M	com.game.xianfengdg	V2.0.2.3
嘟嘟	25.33M	com.yunxin123.dudu	V1.2.5
网易云阅读	24.64M	com.netease.pris	V6.3.8
猫耳 FM	60.79M	cn.missevan	V5.3.5
木头人大建造	95.1M	com.noodlecake.blockheads	V1.7.6
爱康体检宝	17.4M	com.ikang.web	V3.12.3
玖富万卡	26.45M	com.jfbank.wanka	V3.4.11
狗语翻译	19.43M	com.leholady.dog	V1.2.5
全民斩仙	207.6M	com.tencent.tmgp.zhanxian	V2.05.00
视否	49.51M	com.za.youth	V3.0.3
节奏大师	60.08M	com.tencent.game.rhythmmaster	V2.5.12.1
约咖	53.44M	com.wanka.android	V2.9
贪吃蛇大作战	138.16M	com.wepie.snake.tencent	V4.3.26.1
任买	12.07M	cn.com.bestbuy	V3.6.3
傲世西游	159.08M	com.tencent.JWX	V1.5.6.1
球果	41.46M	com.qiuyou.qiuguo	V1.4.6
美木	21.24M	com.meimu.cstong	V3.4.2
姜饼	55.51M	com.iqiyi.pizza	V2.7.1
长途拼车	11.33M	com.example.pinche	V2.1.2

（续表）

LOFTER	45.73M	com.lofter.android	V6.8.4
微贷网	36.65M	com.renrun.aphone.app	V6.9.7
美团	89.46M	com.sankuai.meituan	V10.8.402
三国之刃	450.14M	com.tencent.tmgp.sgzr	V18.0.0
全民小视频	40.73M	com.baidu.minivideo	V2.3.5.13
连尚万能上网	10.51M	com.lantern.mastersim	V3.10.2
奶块	198.93M	com.next.netcraft	V4.6.1.0
人人美剧	7.92M	com.lj.amj	V2.0.1
熊出没 4 丛林冒险	70.14M	com.joym.xiongchumo4	V1.3.0
每日瑜伽	24.7M	com.dailyyoga.cn	V7.16.1.2
TED 演讲	17.5M	com.ted.android	V4.4.0
录音鸡	26.4M	com.zhangju.chickenrecorder	V1.0.1
全民小镇	352.47M	com.tencent.Mtown	V2.14.2
梨花直播	41.15M	com.tiange.multiwater	V4.0.0
省钱小助手	8.89M	com.app.business	V1.0.6
豆瓣阅读	34.53M	com.douban.book.reader	V5.11.0.1
趣约会	46.36M	com.quyue.android	V1.22.1
神俑降临	227.22M	com.tencent.tmgp.yh.syjl	V5.2.65
云听	26.75M	com.shinyv.cnr	V6.8.0
地狱之剑	44.73M	com.cwa.mojian	V1.00.01
哩咔	48.64M	com.ourydc.yuebaobao	V3.7.1
想看	27.39M	com.xiangkan.android	V4.9.28
生意助理	8.24M	com.ccb.cerp	V1.0.3
陌陌	80.89M	com.immomo.momo	V8.23
挖财记账	29.85M	com.wacai365	V12.1.1
电喵直播	35.53M	com.kwai.android.gzone	V1.4.1.32
珍爱优恋空间	52.5M	com.zhenai.ulian	V1.4.6
小寻	61.1M	com.xiaoxun.xun	V1.1.62
WiFi 万能钥匙	37.43M	com.snda.wifilocating	V4.5.63
大掌门	137.72M	com.playcrab.ares.qqandroid	V2.0.9
LOOK 直播	52.44M	com.netease.play	V1.9.6
京东金融	67.67M	com.jd.jrapp	V5.3.70
韩剧 TV	33.4M	com.babycloud.hanju	V5.0.2
我查查	15.9M	com.wochacha	V9.6.1

（续表）

油惠通	54.62M	com.sunbox.integralshop	V1.5.5
先锋影音	20.61M	com.xfyy.yax	V1.5
汽车违章查询	39.41M	cn.mucang.kaka.android	V7.8.6
触电新闻	34.87M	com.touchtv.touchtv	V3.0.2
美团拍店	33.52M	com.sankuai.meituan.pai	V4.6.3
保卫萝卜 2	94.92M	com.tencent.tmgp.carrot2	V4.4.0
3D 极限摩托	41M	com.sty.jixianmotuo.qq	V2.3.4
三国帮	16.01M	com.tencent.tmgp.rysgb	V1.0
氧秀	56.96M	com.syoogame.yangba	V3.1.5
全民水浒	180.92M	com.tencent.q108	V2.0.246
大姨妈月经期助手	31.16M	com.yoloho.dayima	V8.2.4
建行惠懂你	42.54M	com.ccb.leye	V1.11.0
机战王	138.19M	com.centurysoft.roboking	V4.7
模拟岛屿	40.16M	com.sparklingsociety.cias.qq	V6.2.2
苏宁易购	93.59M	com.suning.mobile.ebuy	V8.6.2
探探	72.71M	com.p1.mobile.putong	V3.9.5.1
拉手团购	14.25M	com.lashou.groupurchasing	V7.50
手机加速神器	16.78M	cn.am321.android.am321	V6.2.0
苍穹之剑 2	540.57M	com.tencent.tmgp.cq2	V1.1.1.0
金装裁决	781.7M	com.tencent.tmgp.csfgjh	V1.7.2.0
亚马逊	29.84M	cn.amazon.mShop.android	V18.21.4
世界征服者 3	79.5M	com.tencent.tmgp.wc3	V1.2.4
饥饿鲨世界	204.82M	com.tencent.tmgp.hsw	V3.8.0
英雄之剑	70.39M	com.tencent.tmgp.yxzj	V1.3
豪门足球风云	339.72M	com.tencent.tmgp.hmzqfy	V1.0.570
会玩	73.12M	com.wepie.weplay	V5.9.15
弹窝	30.6M	com.rent_app	V1.9
撩星球	15.43M	com.box.liaoxingqiu.play	V1.0.4
网易公开课	25.9M	com.netease.vopen	V7.5.1
Kindle 阅读	52.49M	com.amazon.kindlefc	V8.29.1.0
奥特曼传奇英雄	388.96M	com.joym.legendhero.yyb	V1.6.8
切水果达人	8.79M	com.tencent.tmgp.qmbydz.ninjia	V1.0.9
宠酱	17.33M	com.yz.mocha	V1.2.1

作者简介

宋睿（1994.11 -），男，江苏宿迁人，现为东南大学网络空间安全学院硕士研究生，主要研究方向为移动设备传感器安全、区块链共识机制安全和 Android 逆向分析。

作者攻读硕士学位期间发表的论文

[1]. **SONG R**, SONG Y, DONG Q, et al. WebLogger: Stealing Your Personal PINs via Mobile Web Application[C]. 2017 9th International Conference on Wireless Communications and Signal Processing (WCSP). Nanjing, 2017. 1-6. (EI Indexed)

[2]. **SONG R**, SONG Y, GAO S, et al. I Know What You Type: Leaking User Privacy via Novel Frequency-Based Side-Channel Attacks[C]. 2018 IEEE Global Communications Conference (GLOBECOM). Abu Dhabi, United Arab Emirates, 2018. 1-6. (EI Indexed)

[3]. **SONG R**, SONG Y, LIU Z, et al. GaiaWorld: A Novel Blockchain System Based on Competitive PoS Consensus Mechanism[J]. Computers, Materials and Continua, 2019, 60(3): 973-987. (SCI Indexed)

[4]. SHI C, **SONG R**, QI X, et al. ClickGuard: Exposing Hidden Click Fraud via Mobile Sensor Side-channel Analysis[C]. 2020 IEEE International Conference on Communications (ICC). Dublin, Ireland, 2020. 1-6. (EI Indexed)

[5]. 王伟康, **宋睿**, 胡爱群. 基于区块链的保密文档管理系统设计与实现 [C]. 第 11 届信息安全漏洞分析与风险评估大会 (VARA), 江苏无锡, 2018. 1-8. (EI Indexed)

[6]. 宋宇波, 张仕奇, **宋睿**. 一种基于投票权竞争的区块链共识机制 [J]. 山东大学学报 (理学版). (中文核心期刊)