南 京 工 程 学 院

毕业设计说明书(论文)

作 者：  **封光**  学 号： **202130414**

院 系： **计算机工程学院**

专 业：  **计算机科学与技术（嵌入式系统）**

题 目： **智能门禁系统开发**

指导者： **钱瑛 讲师**

评阅者： **讲师**

2017年5 月 南 京

**毕业设计说明书（论文）中文摘要**

|  |
| --- |
| 关键词 |

**毕业设计说明书（论文）外文摘要**

|  |
| --- |
| **Title**  **Abstract**  **Keywords** |

**目　　录**

[前 言 1](#_Toc482633150)

[第一章 绪论 2](#_Toc482633151)

[1.1 课题研究意义 2](#_Toc482633152)

[1.2 国内外研究现状 2](#_Toc482633153)

[1.3 课题研究内容 2](#_Toc482633154)

[第二章 技术概览 3](#_Toc482633155)

[2.1 Android things嵌入式系统 3](#_Toc482633156)

[2.2 NFC协议介绍 4](#_Toc482633157)

[2.3 Android Studio开发平台 5](#_Toc482633158)

[第三章 通讯模块设计实现 6](#_Toc482633159)

[3.1 基于socket封装的类http通讯格式 6](#_Toc482633160)

[3.2 基于AES的通讯信息加密算法 8](#_Toc482633161)

[3.3 串口通讯设计 10](#_Toc482633162)

[第四章 智能门禁系统软件设计 13](#_Toc482633163)

[4.1 系统软件总体设计 13](#_Toc482633164)

[4.2 Android开锁模块设计 14](#_Toc482633165)

[4.3后台数据分析管理模块设计 14](#_Toc482633166)

[4.4 Android things服务端模块设计 14](#_Toc482633167)

[第五章 智能门禁系统硬件设计 15](#_Toc482633168)

[5.1门禁控制-舵机模块 15](#_Toc482633169)

[5.2 Raspberry Pi 3 运行平台 15](#_Toc482633170)

[5.3 PN532 NFC模块设计 15](#_Toc482633171)

[第六章 系统测试 16](#_Toc482633172)

[6.1 android开锁功能测试 16](#_Toc482633173)

[6.2后台数据分析管理功能设计 16](#_Toc482633174)

[6.3 android things服务端稳定性测试 16](#_Toc482633175)

[第六章 结论 17](#_Toc482633176)

[参考文献 18](#_Toc482633177)

[致谢 19](#_Toc482633178)

[附录：英文技术资料翻译 20](#_Toc482633179)

前 言

第一章 绪论

## 课题研究意义

智能门禁系统是一种新型的现代安全管理系统，将计算机自动识别技术与现代安全管理措施整体结合起来，涉及电子，机械，光学，计算机技术，通信技术，生物技术等诸多新技术。它是用来解决重要部门的出入口安全管理的措施。适用于银行，酒店，机房，军械库，保密室，办公空间，智能小区，工厂等各类保密部门。在数字网络技术快速发展的今天，智能门禁技术得到了飞速的发展。门禁系统早已不再只是单纯的门控和钥匙管理，它已经逐渐发展成为一套成熟的出入管理系统。它在人事考勤管理，工作环境安全等行政工作中发挥了巨大的作用。

在原有门禁系统的基础上增加相应的辅助设备可以实现保安巡检管理、物业消防监控、电梯控制、安全检查管理等，真正实现智能管理。

## 1.2 国内外研究现状

智能门禁系统随着自动化识别技术的发展而迅速发展，国外门禁系统的知名品牌有美国的修斯（HID）、西坞（WSE）、罗泰克（NTK），以色列的DDS、英国的集保等。国内有销售门禁系统有深圳捷顺、门吉、红门、青云等公司。目前，国内外使用和研制的门禁系统主要分为生物辨识别门禁系统和感应门禁系统。在生物识别门禁系统中，又以指纹门禁系统最为广泛应用。随着模式识别理论、计算机图像处理技术、集成电路技术等的不断成熟与发展，指纹识别模块体积的不断缩小和算法的不断优化，其价格也不断降低。据国际生物认证组织（IBG）的预测：生物识别市场份额2002年将突破7亿美元，其中虹膜识别占8%，而指纹占将近50%。生物识别市场份额到2005年将达到19亿美元，其中虹膜识别占10%左右，指纹识别占40%左右。并且生物识别市场份额将在未来十几年内仍会保持高速增长的势头，纵观当下门禁市场，预估未来1－3年内中国的门禁市场增长比将达到30％。

目前国外制作的门禁系统又较高的安全性，但是价格确比较昂贵，感应式门禁控制器的价格从5000到4万元不等，如果包括门锁、指纹模块、控制器、以及软件在内的一整套系统价格基本上在万元以上。而国内生产制作的的指纹门禁系统和感应式门禁系统价格相比起国外同类产品价格大约低14%-55%，但是系统的可靠性和安全性相对来说较差。

## 1.3 课题研究内容

本课题采用Linux和 Android 技术来设计开发一个具有远程控制门禁、NFC身份识别以及后台数据分析及管理等功能的系统。本课题主要分为三个部分：服务器、android解锁客户端、android后台统计数据分析客户端。其中服务器采用Android things物联网系统(基于Linux)，用来控制门禁开关、接收Android解锁客户端的网络请求和后台统计系统的网络请求。解锁客户端只要负责实现远程解锁和刷NFC解锁两大功能模块。后台统计系统主要负责门禁开关事件的记录、用户身份控制，异常分析等等。

第二章 技术概览

## 2.1 Android things嵌入式系统

Android Things是谷歌面向物联网设备市场推出的最新系统，这让物联网市场的竞争进一步升温。目前，微软正在向发烧友和原始设备制造商(OEM)推广其Windows 10 IOT系统。尽管亚马逊并未开发相应的设备系统，但是该公司已经把AWS Lambda计算服务开放给了物联网开发者。亚马逊云计算部门AWS正在与设备制造商合作，在设备中绑定Lambda Greengrass软件。该软件是AWS Lambda的本地版本，可以被安装到物联网设备和套件中。Android Things使用和Android开发一样的工具、Android框架和Google APIs，这些都使得开发物联网变得更简单。

Android things的平台架构如下：(图2-1-1)

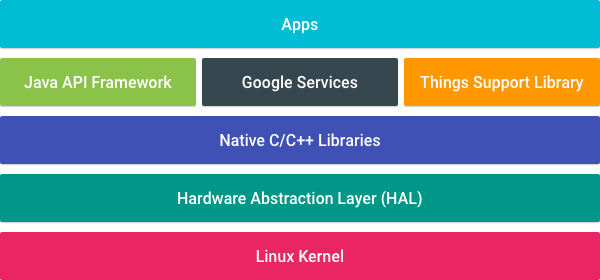


图2-1-1

Android Things继承了Android核心框架层并且通过支持包提供额外的APIs。这些APIs允许apps与新型的硬件之间交互，但是这些硬件并不存在于移动设备上。Android Things平台仍然为单一应用程序使用。系统app不存在，并且你的app将会自动启动来给你的用户带来沉浸式体验。

相比起之前Google发布的Brillo系统，Brillo把C++作为主要开发环境，而Android Things则面向所有Java开发者，不管开发者有没有移动开发经验。开发者可以针对物联网设备使用工具包。Android Studio是最常用的Android集成开发环境，它基于IntelliJ IDEA开源社区版本，支持通过Android Things进行物联网开发。应用开发的生命周期几乎和移动开发相同。Android Things系统支持原始Android SDK中的一个子集，不支持需要用户输入或者需要地图、搜索以及登录等验证凭证的应用程序接口(API)。和Android手机的OTA无线升级一样，开发者可以通过谷歌已经在其产品和服务中使用过的相同OTA基础设施，推送谷歌提供的系统更新和定制应用升级。谷歌推荐开发者在最初原型设计和开发时使用微型电脑“树莓派3”、英特尔微型可穿戴设备“爱迪生”(Edison)以及恩智浦半导体的Pico平台。谷歌会在Android Things正式发布前增加更多认证设备。包括云消息传递服务Firebase在内的谷歌云平台组件能够轻易与Android Things整合。开发者可以利用多个云服务进行存储、状态管理以及发送消息。Android Things整合了物联网设备通讯平台Weave。Weave Server是用来处理设备注册、命令传送、状态存储以及与谷歌助手等谷歌服务整合的云服务。Weave SDK将嵌入到设备中进行本地和远程通讯。

本系统使用树莓派3b+作为Android things系统的载体，在树莓派3b+中刷入最新的Android things系统。

## 2.2 NFC协议介绍

NFC，近场通信（英语：Near Field Communication，NFC），又称近距离无线通信，是一种短距离的高频无线通信技术，允许电子设备之间进行非接触式点对点数据传输，在十厘米（3.9英寸）内交换数据。

这个技术由非接触式射频识别（RFID）演变而来，由飞利浦半导体（现恩智浦半导体）、诺基亚和索尼共同研制开发，其基础是RFID及互连技术。近场通信是一种短距高频的无线电技术，在13.56MHz频率运行于20厘米距离内[1]。其传输速度有106 Kbit/秒、212 Kbit/秒或者424 Kbit/秒三种。目前近场通信已通过成为ISO/IEC IS 18092国际标准、EMCA-340标准与ETSI TS 102 190标准。NFC采用主动和被动两种读取模式。

NFC工作模式可分为三种：卡模拟模式、读卡器模式、点对点模式。卡模拟模式：这个模式其实就是相当于一张采用RFID技术的IC卡。可以替代现在大量的IC卡（包括信用卡）场合商场刷卡、IPASS、门禁管制、车票、门票等等。此种方式下，有一个极大的优点，那就是卡片通过非接触读卡器的RF域来供电，即便是寄主设备（如手机）没电也可以工作。NFC设备若要进行卡片模拟相关应用，则必须内置安全组件之NFC芯片。点对点模式（P2P mode）：这个模式和红外线差不多，可用于数据交换，只是传输距离较短，传输创建速度较快，传输速度也快些，功耗低（蓝牙也类似）。将两个具备NFC功能的设备链接，能实现数据点对点传输，如下载音乐、交换图片或者同步设备地址薄。因此通过NFC，多个设备如数位相机、PDA、计算机和手机之间都可以交换资料或者服务。读卡器模式（Reader/Writer mode）：作为非接触读卡器使用，比如从海报或者展览信息电子标签上读取相关信息。

NFC和蓝牙都是短距离通信技术，而且都被集成到移动电话。但NFC不需要复杂的设置程序。NFC也可以简化蓝牙连接。NFC略胜蓝牙的地方在于设置程序较短，但无法达到低功率蓝牙（Bluetooth Low Energy）的传输速率。在两台NFC设备相互连接的设备识别过程中，使用NFC来替代人工设置会使创建连接的速度大大加快：少于十分之一秒。NFC的最大数据传输量424 kbit/s远小于Bluetooth V2.1（2.1 Mbit/s）。虽然NFC在传输速度与距离比不上蓝牙（小于20 cm），但相应可以减少不必要的干扰。这让NFC特别适用于设备密集而传输变得困难的时候。

相对于蓝牙，NFC兼容于现有的被动RFID（13.56 MHz ISO/IEC 18000-3）设施。NFC的能量需求更低，与蓝牙V4.0低功耗协议类似。当NFC在一台无供电的设备（比如一台关机的手机，非接触式智能信用卡，或是智能海报）上工作时，NFC的能量消耗会要大于低功率蓝牙V4.0。

对于移动电话或是行动消费性电子产品来说，NFC的使用比较方便。NFC的短距离通信特性正是其优点，由于耗电量低、一次只和一台机器链接，拥有较高的保密性与安全性，NFC有利于信用卡交易时避免被盗用。NFC的目标并非是取代蓝牙等其他无线技术，而是在不同的场合、不同的领域起到相互补充的作用。

本系统中将使用NFC的点对点模式，采用PN532模块和树莓派上位机通过串口通信，来实现对NFC模块的控制和信息采集。

## 2.3 Android Studio开发平台

Android Studio是一个为Android平台开发程序的集成开发环境。2013年5月16日在Google I/O上发布，可供开发者免费使用。2013年5月发布早期预览版本，版本号为0.1。2014年6月发布0.8版本，至此进入beta阶段。第一个稳定版本1.0于2014年12月8日发布。Android Studio基于JetBrains IntelliJ IDEA，为Android开发特殊定制，并在Windows、OS X和Linux平台上均可运行。

本系统中使用的版本有以下功能可用：

* 可视化布局： WYSIWYG编辑器 - 实时编码 - 实时程序界面预览；
* 开发者控制台： 优化提示，协助翻译，来源跟踪，宣传和营销曲线图 - 使用率度量；
* Beta版本测试，并阶段性展示；
* 基于Gradle的构建支持；
* Android特定代码重构和快速修复；
* Lint提示工具更好地对程序性能、可用性、版本兼容和其他问题进行控制捕捉；
* 支持ProGuard和应用签名功能；
* 基于模板的向导来生成常用的Android应用设计和组件；
* 自带布局编辑器，可让开发者拖放UI组件，并预览在不同尺寸设备上的UI显示效果。
* 支持构建Android Wear应用
* 内置Google Cloud Platform支持，支持Google Cloud Messaging和App Engine的集成

第三章 通讯模块设计实现

## 3.1 基于socket封装的类http通讯格式

在计算机科学中，网络套接字（英语：Network socket），又译网络套接字、网络接口、网络插槽，是电脑网络中进程间数据流的端点。使用以网际协议（Internet Protocol）为通信基础的网络套接字，称为网际套接字（Internet socket）。因为网际协议的流行，现代绝大多数的网络套接字，都是属于网际套接字。

socket是一种操作系统提供的进程间通信机制。socket最初被翻译为把socket译为“媒介(字)”。不久，ARPANET的socket就被翻译为“套接字”，其理由是：由于每个主机系统都有各自命名进程的方法，而且常常是不兼容的，因此，要在全网范围内硬把进程名字统一起来是不现实的。所以，每个计算机网络中都要引入一种起媒介作用的、全网一致的标准名字空间。这种标准名字，在ARPA网中称作套接字，而在很多其他计算机网中称作信口。更确切地说，进程之间的连接是通过套接字或信口构成的

在操作系统中，通常会为应用程序提供一组应用程序接口（API），称为套接字接口（英语：socket API）。应用程序可以通过套接字接口，来使用网络套接字，以进行数据交换。最早的套接字接口来自于4.2 BSD，因此现代常见的套接字接口大多源自Berkeley套接字（Berkeley sockets）标准。在套接字接口中，以IP地址及通信端口组成套接字地址（socket address）。远程的套接字地址，以及本地的套接字地址完成连接后，再加上使用的协议（protocol），这个五元组（five-element tuple），作为套接字对（socket pairs），之后就可以彼此交换数据。例如，再同一台计算机上，TCP协议与UDP协议可以同时使用相同的port而互不干扰。 操作系统根据套接字地址，可以决定应该将数据送达特定的进程或线程。这就像是电话系统中，以电话号码加上分机号码，来决定通话对象一般。

手机能够使用联网功能是因为手机底层实现了TCP/IP协议，可以使手机终端通过无线网络建立TCP连接。TCP协议可以对上层网络提供接口，使上层网络数据的传输建立在“无差别”的网络之上。

HTTP协议即超文本传送协议(Hypertext Transfer Protocol )，是Web联网的基础，也是手机联网常用的协议之一，HTTP协议是建立在TCP协议之上的一种应用。

HTTP连接最显著的特点是客户端发送的每次请求都需要服务器回送响应，在请求结束后，会主动释放连接。从建立连接到关闭连接的过程称为“一次连接”。

由于HTTP在每次请求结束后都会主动释放连接，因此HTTP连接是一种“短连接”，要保持客户端程序的在线状态，需要不断地向服务器发起连接请求。通常的做法是即时不需要获得任何数据，客户端也保持每隔一段固定的时间向服务器发送一次“保持连接”的请求，服务器在收到该请求后对客户端进行回复，表明知道客户端“在线”。若服务器长时间无法收到客户端的请求，则认为客户端“下线”，若客户端长时间无法收到服务器的回复，则认为网络已经断开。

由于通常情况下Socket连接就是TCP连接，因此Socket连接一旦建立，通信双方即可开始相互发送数据内容，直到双方连接断开。但在实际网络应用中，客户端到服务器之间的通信往往需要穿越多个中间节点，例如路由器、网关、防火墙等，大部分防火墙默认会关闭长时间处于非活跃状态的连接而导致 Socket 连接断连，因此需要通过轮询告诉网络，该连接处于活跃状态。

而HTTP连接使用的是“请求—响应”的方式，不仅在请求时需要先建立连接，而且需要客户端向服务器发出请求后，服务器端才能回复数据。

很多情况下，需要服务器端主动向客户端推送数据，保持客户端与服务器数据的实时与同步。此时若双方建立的是Socket连接，服务器就可以直接将数据传送给客户端；若双方建立的是HTTP连接，则服务器需要等到客户端发送一次请求后才能将数据传回给客户端，因此，客户端定时向服务器端发送连接请求，不仅可以保持在线，同时也是在“询问”服务器是否有新的数据，如果有就将数据传给客户端。

在本系统中，由于不需要完全实现http的所有格式支持，所以采用了socket模拟http的方式，根据业务需求写了部 分属性支持。比如一个完整http请求的设计、包头包尾的定义。结合多线程/线程池以及java回调的相关知识，延伸出一套可用的sdk，并集成到了项目当中。整个请求过程可以用下图表示。（图3-1-1）

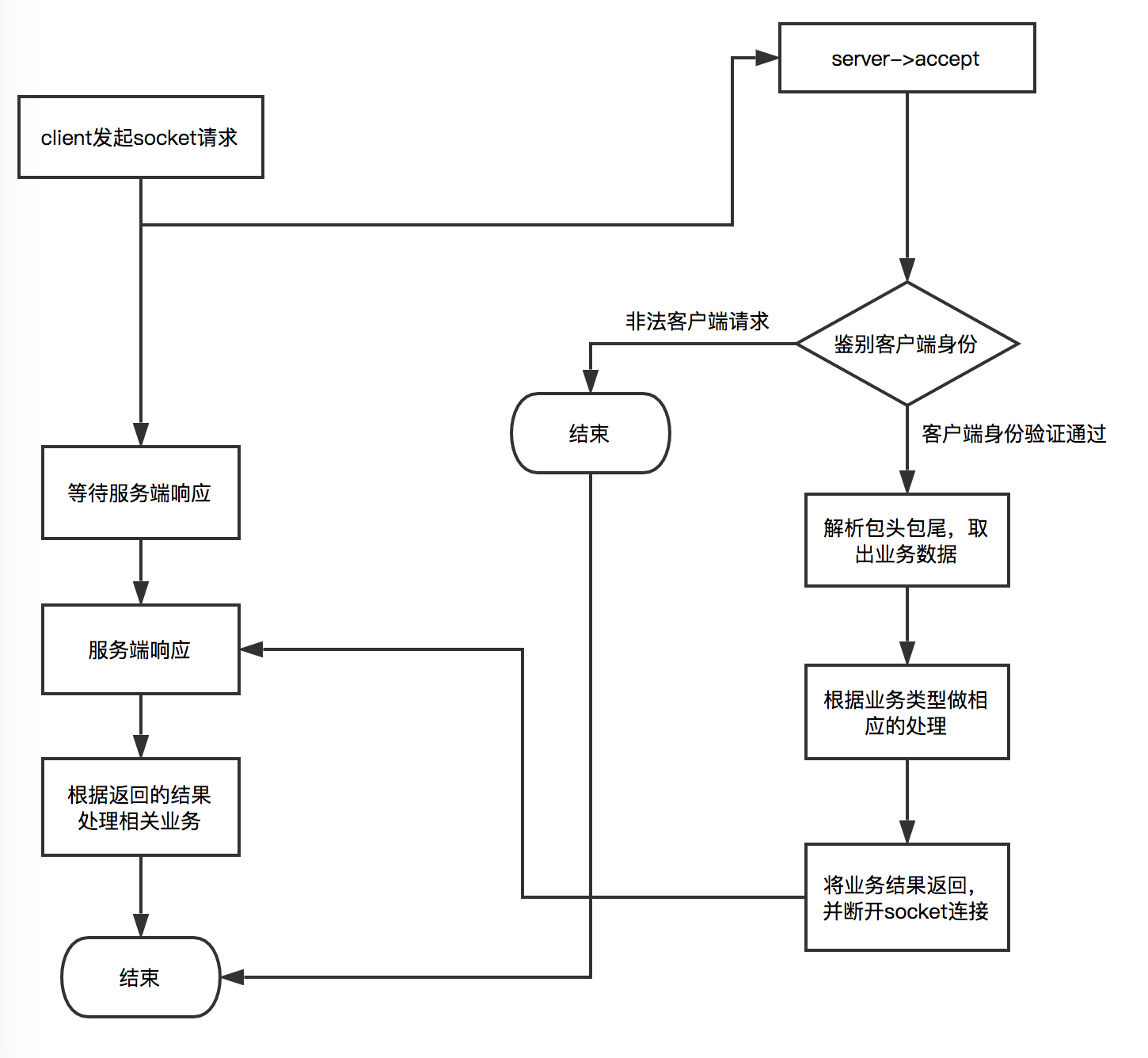


图3-1-1 socket通信流程图

在整个请求流程中，客户端先从线程池中抽出一个闲置的线程，通过tcp socket发起网络请求。服务端有个一直阻塞的accpet线程，用来接收客户端的请求。在来一个socket请求之后，可以得到一个客户端的套接字描述符。后期的身份鉴权以及通信将依赖这个套接字描述符展开。服务端在拿到套接字描述符之后，将用约定好的密文来进行客户端的身份验证，如果身份验证失败将直接断开连接。如果身份验证成功，则进行下一步的通信。身份验证之后，服务端将根据事先约定好的包头包尾部剥离出主要通信正文，注意这里的密文是加密过的，这段加密算法将在下一节提到。在进行通信正文的解密之后，就可以拿到客户端的头字段、type、action等字段。由于socket通信顺序的不确定性，这里设计了一个头字段的约定，头字段里面包含了一个根据时间戳生成的一个UID，这个UID将会原封不动的返回给客户端，客户端拿到服务端返回的字段之后，会解析出UID字段，并将这个通信通过java回调的方式分发给该请求的线程。然后这个线程将会处理服务端返回的字段，处理相关业务代码之后，完成一次完整的请求。其中type字段类似于http中的URL，标识了要访问的资源，在这里简化成了一个type字符串，用来标识业务请求，比如开锁、获取开锁事件等等请求。action字段则标识了这个请求的内容，可以携带一个需要的参数。然后服务端解析相关参数后完成相应的业务代码。

## 3.2 基于AES的通讯信息加密算法

作为一个门禁系统，安全问题自然也是重中之重。在本系统中，主要的安全隐患存在于通信过程中。如果使用明文加密，通过反编译app或者中间人攻击，即可拿到通信协议，并且冒充客户端来欺诈服务器，逃过安全验证。所以，本系统将采用AES对称加密算法来实现通信过程的加密。

AES，高级加密标准（英语：Advanced Encryption Standard，缩写：AES），在密码学中又称Rijndael加密法，是美国联邦政府采用的一种区块加密标准。这个标准用来替代原先的DES，已经被多方分析且广为全世界所使用。经过五年的甄选流程，高级加密标准由美国国家标准与技术研究院（NIST）于2001年11月26日发布于FIPS PUB 197，并在2002年5月26日成为有效的标准。2006年，高级加密标准已然成为对称密钥加密中最流行的算法之一。

AES加密过程涉及4种操作：字节替代（SubBytes）、行移位（ShiftRows）、列混淆（MixColumns）和轮密钥加（AddRoundKey）。下图（图3-2-1）给出了AES加解密的流程，从图中可以看出：1）解密过程的每一步分别对应加密的逆操作，2）加解密所有操作的顺序正好是相反的。正是由于这几点（再加上加密与解密每步的操作互逆）保证了解密能够正确地恢复明文。加解密中每轮的密钥分别由种子密钥经过密钥扩展扩展得到。算法中16字节的明文、密文和轮子密钥都以一个4x4的矩阵表示。



图3-2-1 AES加解密流程

在AES加密中，需要key(私钥)、iv(偏移量)、加密原文，根据这三项输入可以得到密文的输出。这时候就会暴露一个问题，就是key(私钥)、iv(偏移量)的安全存储问题。由于这两个输入项是用户自定义的，而且必须是不能暴露的。所以，如何安全地存储这两个值是一个比较令人头痛的问题。

在本系统中，采用了安全图片存储机制。顾名思义，就是把这两个值存在图片中，然后这个图片可以通过http访问、本地存储等方式来获取，也可以用直接当成资源文件放在工程目录中。

在设计安全图片存储方案的时候，大致查阅到了以下几种思路：

* 在jpg中，可以通过写入Exif区，这块区域存入了相机拍摄信息：时间、拍摄地点等等，可以在这个区域存入加密后的信息，实现图片的无损加密。
* 在png中，可以在IEND(png文件尾部)区域后追加加密信息，在此区域追加的信息不会影响图像的正常读取。但是用png格式需要注意在编译的时候会被编译器自动压缩，从而失去加密的信息。可以考虑动态获取安全图片来规避这个问题，比如从网络中加载图片再写入本地。
* 从图片本身结构入手，比如通过你要加密的信息构造出RGB、alpha等信息，然后按照标准png/jpg/bmp等格式构造出一张全新的图片，也可以在中间按照某种规则穿插一些混淆数据。这样生成的图片加密程度较高，不容易破解，但相对来说开发成本较高。

最后在本项目中采用了第二种方案，即将key(私钥)、iv(偏移量)写入png图片尾部。这个png图片从工程里的任意一张图片获取，然后在app加载的时候读入这张图片，根据png的尾部信息来获取加密的信息，然后通过约定好的规则来算出key(私钥)、iv(偏移量)，并用私钥和偏移量来进行通信过程中的加解密。

在使用过程中还暴露出另外一个问题，就是防二次打包。因为如果盗窃者剥离出你的祥光代码和图片，并在自己的app中使用，就可以完全伪造客户端，所以在本系统中还加入了防止二次打包的验证，即在制作安全图片的时候也将app的签名摘要sha-1值写入图片中，在读取的过程中验证这个签名是否和图片中的是否相等。如果不等就直接杀死app进程，从而保证app的安全。整个解密流程如下图所示。（图3-2-2）

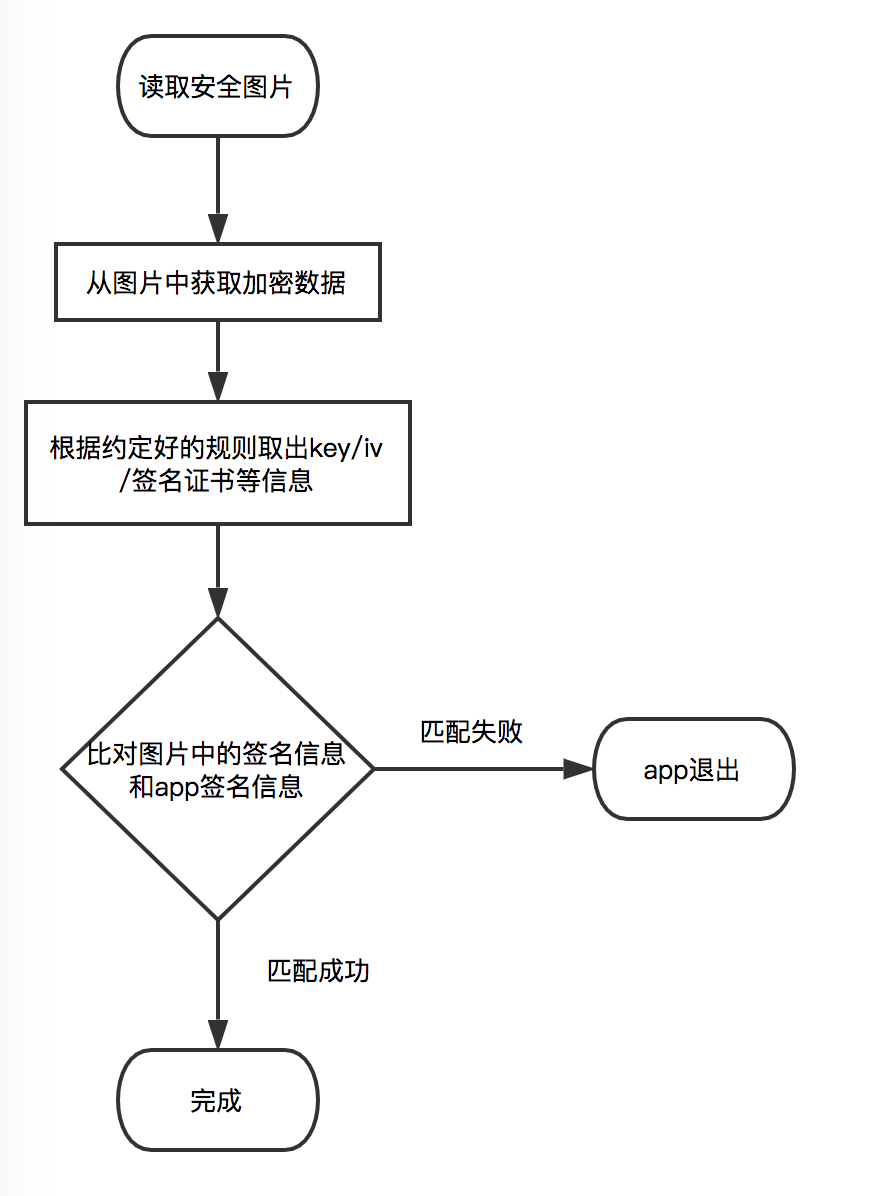


图3-2-2 安全图片验证流程

## 3.3 串口通讯设计

串口通信(Serial Communication)， 是指外设和计算机间，通过数据信号线 、地线、控制线等，按位进行传输数据的一种通讯方式。这种通信方式使用的数据线少，在远距离通信中可以节约通信成本，但其传输速度比并行传输低。其通信接线如图所示（图3-3-1）



图3-3-1 串口通信接线图

在本系统中，需要PN532和Android things系统进行通信，所以采用了波特率115200的串口通信。使用android things提供的UART通信API来完成模块之间的通信。

为打开指定端口号的UART串口连接，需要知道串口的名称。可以在初始化的时候，使用PeripheralManagerService的getUartDeviceList()函数来获取系统识别到的UART名称。

一旦知道目标名称，就可以使用PeripheralManagerService连接到该设备。与外围设备通信后，关闭连接以释放资源。 此外，在现有连接关闭之前，无法打开与设备的新连接。要关闭连接，请使用设备的close()方法。

在onstart()中打开device设备:

mUartDevice = manager.openUartDevice(UART\_DEVICE\_NAME);

在onstop()中关闭device设备:

if(mUartDevice != null){

mUartDevice.close();

mUartDevice = null;

}

数据帧

  通过UART发送的每个字符都包裹在一个数据帧中，该数据帧包含以下组件：（图3-3-2）



图3-3-2 串口通信帧格式

* 启动位 - 在发送数据之前，该行保持活动状态，持续1位持续时间的固定时间间隔，以指示新字符的开始。
* 数据位 - 表示数据字符的各位。 UART可以配置为在5-9个数据位之间发送以表示字符。 较少的位减少了数据的范围，但可以增加有效的数据传输速率。
* 奇偶校验位 - 可选错误校验值。 如果UART配置为偶校验或奇校验，则会在帧中添加一个额外的位，以指示数据位的内容是否为偶数或奇数值。 将此设置为none会从帧中删除该位。
* 停止位 - 在传输所有数据之后，线路被重置一个可配置的时间间隔，以指示该字符的结束。 这可以配置为保持空闲1或2位持续时间。

UART上的数据传输速率称为波特率。 它表示接收和发送的速度（以位/秒为单位）。 由于在通过UART连接的两个器件之间没有共享时钟，因此必须提前配置这两个器件，才能使用相同的波特率正确解码数据。

常用波特率包括9600,19200,38400,57600,115200和921600.该速率包括数据帧（开始，停止和奇偶校验位）的开销，因此有效数据传输速率将略低，并且根据 您配置的帧位数。

以下代码将UART连接配置为以115200波特，8个数据位，无奇偶校验和1个停止位（8N1）操作：

public void configureUartFrame(UartDevice uart) throws IOException {

uart.setBaudrate(115200);

uart.setDataSize(8);

uart.setParity(UartDevice.PARITY\_NONE);

uart.setStopBits(1);

}

传输输出数据:

public void writeUartData(UartDevice uart) throws IOException {

byte[] buffer = {...};

int count = uart.write(buffer, buffer.length);

Log.d(TAG, "Wrote " + count + " bytes to peripheral");

}

监听输入数据:

使用read()方法将输入数据从UART FIFO缓冲区拉入应用程序。 此方法接受空缓冲区以填充传入数据和要读取的最大字节数。 UART读取是非阻塞的，如果FIFO中没有可用的数据，则会立即返回。

  UartDevice将在读取时返回FIFO中的可用字节数，并把读取的内容写入Buffer中。读取之前要确保buffer中所有数据都重置，循环读取UART，直到它返回值为-1(没有更多的数据)

public void readUartBuffer(UartDevice uart) throws IOException {

// 一次读取的最大数据量

final int maxCount = ...;

byte[] buffer = new byte[maxCount];

int count;

while ((count = uart.read(buffer, buffer.length)) > 0) {

Log.d(TAG, "Read " + count + " bytes from peripheral");

}

}

  为了避免在缓冲区为空时不必要地轮询UART，用UartDevice注册一个UartDeviceCallback。 当有可用于读取的数据时，此回调将调用onUartDeviceDataAvailable()方法。当应用程序不再需要监听输入数据时，应该取消注册回调，以防止内存泄漏。

// 注册中断事件

mDevice.registerUartDeviceCallback(mUartCallback);

// 取消注册中断事件

mDevice.unregisterUartDeviceCallback(mUartCallback);

第四章 智能门禁系统软件设计

## 4.1 系统软件总体设计

本系统软件的总体设计如图所示（图4-1-1），主要分为三个模块，服务端控制模块、android开锁模块、android后台管理模块。

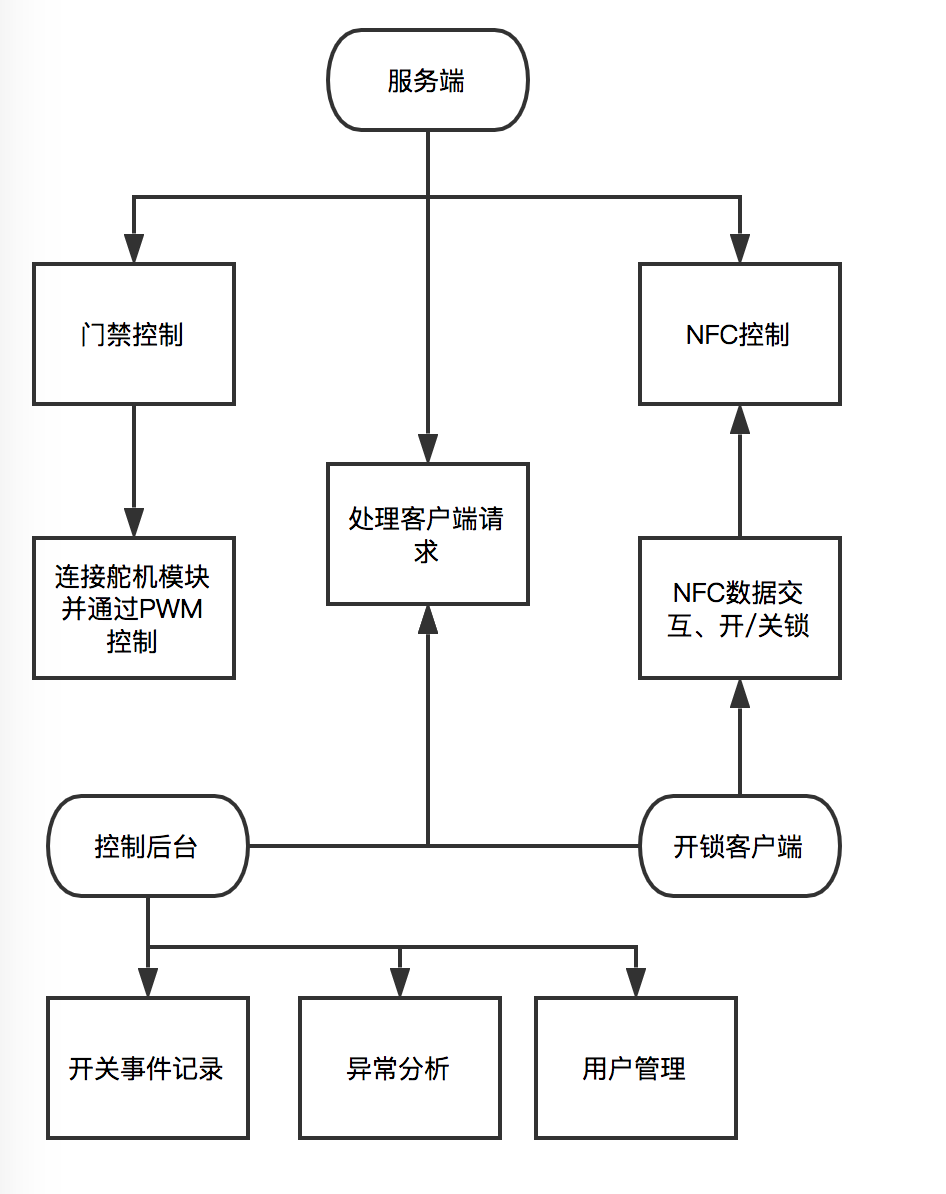


图4-4-1系统设计流程图

## 4.2 Android开锁模块设计

android开锁模块主要包含了开锁、关锁、NFC开锁、身份验证等功能。第一次进入app的时候需要注册账号，这个账号以后会作为开关锁的时候记录的凭据，方便后台管理。在实现注册之后，后台控制系统将会收到一条注册申请。

在后台通过注册申请之后，客户端需要设置一下解锁密码。解锁密码用九个点的手势密码来实现。在设置完密码之后，后台将记录下当前的密码，并存入android提供的sharedPreferences中。以后每次进入app都会有一个身份认证的界面。只有输入正确的密码才能进入开锁界面。

进入开锁界面后，可以选择远程开锁和关锁两个选项。因为是用的舵机模拟的门禁，所以这里的关锁只是为了将舵机复位来方便模拟。在实际生活中，只需要一个开锁选项，而关锁是由人工手动关锁。也可以贴近NFC天线模块，在贴近的时候将会跳出一个对话框来让用户确认是否解锁，如果选择了解锁即会向Android things后台服务器发送开锁请求，在服务器验证通过后即可让舵机旋转90度，来实现开锁的模拟。

4.2.1用户注册界面设计

在用户注册界面中，用户需要填写账号、密码、申请理由三个选项。账号可以自行填写，但是在发送登录请求的时候会验证一下账号，如果账号是数据库中已经存在的账号则会注册失败，不通过。密码使用一个type为passwd属性的edittext，并使用hint属性来提示用户输入密码，使得界面更加友好。申请理由可以填写100字以内的申请理由，在注册的时候将会将这个理由发送到Android things后台服务器中，在后台管理界面收到请求后将会显示申请理由，并可以选择通过或者拒绝。

在通过注册申请后，将会有一次手势密码的录入。采用九点手势。在很多安全性比较高的应用程序中，每次打开程序，都会有让用户输入密码，这些安全性比较高的程序，密码也会相对复杂，在手机上输入起来，就会不大方便，至少要切换一次输入法，对用户体验造成不好的影响，在移动互联网时代，用户体验至高无上，在这种情况下，手势密码不失为一个很好解决方案，既安全，又快捷，用户很容易记住，一个手势即可解锁，不用输入繁琐的密码。

1. 正上方的提示区域，用一个类(LockIndicator.Java)来实现，自定义view来绘制9个提示图标；

2. 手势密码绘制区域，用一个类(GestureContentView.java)来实现，它继承自ViewGroup里面, 添加9个ImageView来表示图标, 在onLayout()方法中设置它们的位置；

3. 手势路径绘制， 用一个类(GestureDrawline.java)来实现，复写onTouchEvent()方法，在这个方法里面监听TouchEvent事件: ACTION\_DOWN、ACTION\_MOVE、ACTION\_UP事件，来绘制手势连接不同点之间的路径；

4. 9个点的对象，用一个类(GesturePoint.java)来实现，保存它的位置、状态、背景图片等相关信息；

5. 手势密码的获取，判断手指当前的位置，根据滑动路径经过的点，按顺序保存绘制的点的顺序，不能有重复的点。

4.2.2 NFC开锁设计

Android提供了NFCAdapter类来管理android的NFC功能。可以通过以下代码获取手机附带的NFC管理器。

this.nfcAdapter = NfcAdapter.getDefaultAdapter(context);

在onstart中判断获取到的nfcadapter是否为空则能判断该手机是否支持NFC。

if(nfcadapter != null){

if(nfcadapter.isEnabled()){

…

}

}

通过PendingIntent包裹一个activity对象，来设置当系统发现一张nfc卡的时候要跳转的activity，相当于给系统创建了一个快捷方式。

pendingIntent = PendingIntent.getActivity(activity, 0,new Intent(activity.

getApplicationContext(),getClass()).addFlags(Intent.FLAG\_ACTIVITY\_SINGLE\_TOP), 0);

intentFilters = new IntentFilter[]{new IntentFilter(NfcAdapter.ACTION\_NDEF\_DISCOVERED)};

techList = new String[][]{new String[]{}};

setReaderMode();

然后给NFCadapter设置一个卡读取模式的监听，当有卡识别的时候将会有相应的回调。

nfcAdapter.enableReaderMode(activity, new NfcAdapter.ReaderCallback(),NfcAdapter.FLAG\_READER\_NFC\_A, Bundle.EMPTY);

在这个函数里会有一个tag对象的回调参数。

public void onTagDiscovered(Tag tag){}

之后拿到Tag对象就可以作为整个通信过程中的上下文。从TAG对象中可以获取java输入输出流，只要按照标准的输入输出流处理方式就可以完成整个通信过程。

## 4.3后台数据分析管理模块设计

4.3.1 开锁事件记录

每次客户端发送开锁请求都会被服务端记录下来，记录的字段有开锁的时间戳，开锁用户，开锁方式(远程开锁/NFC开锁)，然后会生成一个时间线，在后台展示。

整个开锁记录事件用RecyclerView + Cardview来展示，从后台拉取的有三个字段：开锁用户名、开锁时间戳、和开锁方式。

从服务器获取开锁事件list之后，将按照时间戳进行排序，最新事件在最上面，并通过SimpleDateFormat进行时间戳的格式化：

SimpleDateFormat dataFormater= new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");

这样就可以以年-月-日 小时-分-秒的方式来对时间戳进行格式化，并在列表上展示。

4.3.2 用户管理

## 4.4 Android things服务端模块设计

开始的时候，系统会有一个默认的管理员账号root，之后管理员账号可以自行更改密码，还可以给普通用户提权升级到管理员账号，管理员有看开锁事件的权限、看异常分析的权限和对用户管理的权限。

管理员账号和密码会加密并放到本地数据库中，在进入后台管理界面的同时要输入管理员手势密码。这个手势密码也可以更改。创建账号管理的语句如下:

create table user (username long primary key, passwd varchar(50) not null)

passwd中记录的是手势密码的走向的md5值，仅用来做登录时验证比对。

第五章 智能门禁系统硬件设计

## 5.1门禁控制-舵机模块

## 5.2 Raspberry Pi 3 运行平台

## 5.3 PN532 NFC模块设计

第六章 系统测试

## 6.1 android开锁功能测试

## 6.2后台数据分析管理功能设计

## 6.3 android things服务端稳定性测试

第六章 结论

参考文献

致谢

附录：英文技术资料翻译

**英文原文**

**中文译文：**