分类号_____ 学号 M201376154

学校代码_____ 密级

華中科技大學

硕士学位论文

加强安全的指纹认证系统的设计和实现

学位申请人 李祎

学 科 专 业： 软件工程

指 导 教 师： 肖来元 教授

答 辩 日 期： 2016 年 1 月

A Thesis Submitted in Partial Fulfillment of the Requirements

for the Degree for the Master of Engineering

Design and Implementation of a Fingerprint Authentication System with Enhanced Security

Candidate: Li Yi

Major : Software Engineering

Supervisor: Prof. Xiao

Huazhong University of Science & Technology

Wuhan 430074, P.R.China

January., 2016

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，除文中已经标明引用的内容外，本论文不包含任何其他个人或集体已经发表或撰写过的研究成果。对本文的研究做出贡献的个人和集体，均已在文中以明确方式标明。本人完全意识到本声明的法律结果由本人承担。

学位论文作者签名：

日期：　　年　　月　　日

学位论文版权使用授权书

本学位论文作者完全了解学校有关保留、使用学位论文的规定，即：学校有权保留并向国家有关部门或机构送交论文的复印件和电子版，允许论文被查阅和借阅。本人授权华中科技大学可以将本学位论文的全部或部分内容编入有关数据库进行检索，可以采用影印、缩印或扫描等复制手段保存和汇编本学位论文。

本论文属于　保密□，　　在_____年解密后适用本授权书。

　　不保密□。

（请在以上方框内打“ √ ”）

学位论文作者签名：　　指导教师签名：

日期：　　年　月　日　　日期：　　年　月　日

摘 要

在用户身份认证领域，尤其是基于网络的用户身份认证领域，指纹认证系统已经得到了越来越广泛的应用。指纹认证已经在电子支付，门禁系统等诸多领域发挥巨大作用。尽管如此，传统的指纹认证系统仍然存在着一些问题，由于传统的指纹识别系统需要采用特定的指纹机来读取指纹，这就导致了系统成本较高，易用性不好。为了解决这些问题，我们采用智能手机终端作为指纹采集设备结合移动互联网 Web 技术构建了一个新的指纹识别系统原型。在此基础上，我们使用数字水印技术加强系统的安全性。采用新的解决方案，系统可以有效防止信息窃取以及重放攻击。最终我们开发了一个原型系统并进行了相关实验和测试。实验结果表明，采用基于 Web 的指纹识别系统加强了系统的安全性和可用性，同时大幅度降低了系统的构建成本。

关键词：指纹认证系统；数字水印；加密；网络安全

ABSTRACT

Although use of fingerprint is highly effective in user authentication of networked services such as electronic payment, there are some problems in conventional systems, including high cost due to need for specialized fingerprint readers and limited usability. To resolve these problems, we propose a new system which incorporates a web-based fingerprint authentication using a smartphone as a fingerprint input device. Additionally, a watermark-based encryption solution is used to enhance system security. With this solution, the system can prevent information interception and replay attack. We demonstrate through prototype implementation and experiments that our solution enhances security of web-based system, and the cost and usability problems in conventional systems can also be resolved.

Keywords: fingerprint authentication; watermark; encryption; web security

目 录

1 绪论

1.1 基于指纹的用户认证系统

指纹认证系统是一种重要的用户认证系统。在未来社会，它的作用会变的越来越重要。一些传统的应用领域如安全系统、门禁系统都需要使用指纹识别技术。随着移动互联网的快速发展，越来越多的新兴应用领域如电子商务、电子支付都可能需要使用指纹识别技术。

传统的指纹认证系统遵循的技术架构如图 1.1 所示。

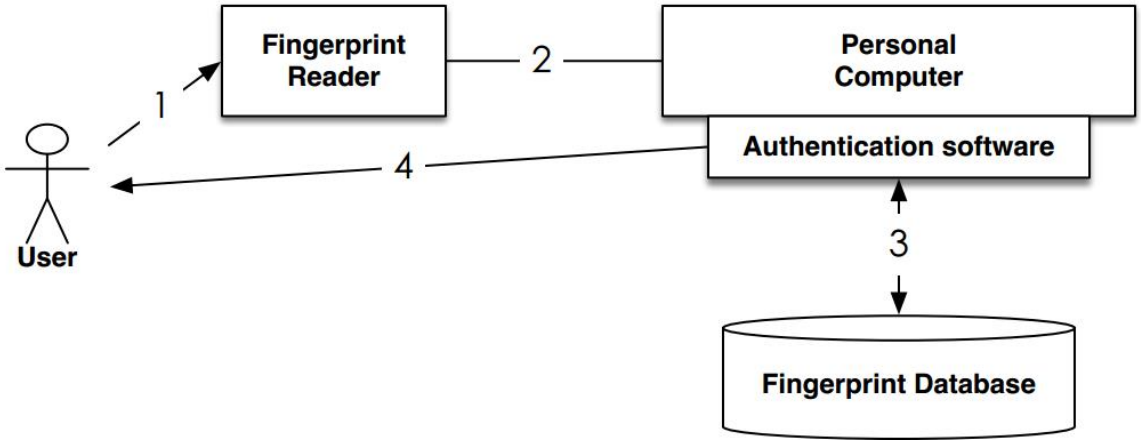

图 1.1 传统指纹认证系统解决方案

由上图可将传统指纹识别系统业务流程归结如下：

（1）使用指纹读取器读取用户指纹信息；

（2）指纹读取器连接计算机系统并将用户指纹信息发送给计算机；

（3）指纹识别软件将用户的指纹信息与数据库中的用户指纹信息进行对比；

（4）指纹识别系统计算指纹相似度，并判断用户是否认证成功，最终将指纹认证结果返回给用户。

然而，随着移动互联网的快速发展，这种传统的指纹认证系统的弊端也逐渐显现出来。首先是成本问题，每一个需要使用指纹识别系统的应用场景，都需要购买并部署相应的系统软硬件，这导致了成本的上升。其次是易用性问题，传统的指纹识别系统需要特定的指纹传感器来采集用户的指纹信息，每一个系统部署都需要安装单独的硬件和软件。当系统硬件或软件发生故障时，用户就无法使用指纹系统进行身份认证，易用性较差。

论文研究了传统指纹识别系统的几个主要缺陷，并对其进行了重新识别和改

进。我们提出了一个基于 Web 的指纹认证系统解决方案，使用这种新的解决方案，在传统指纹识别系统中存在的成本和易用性问题将会得到有效解决。

新的系统架构图如图 1.2 所示。

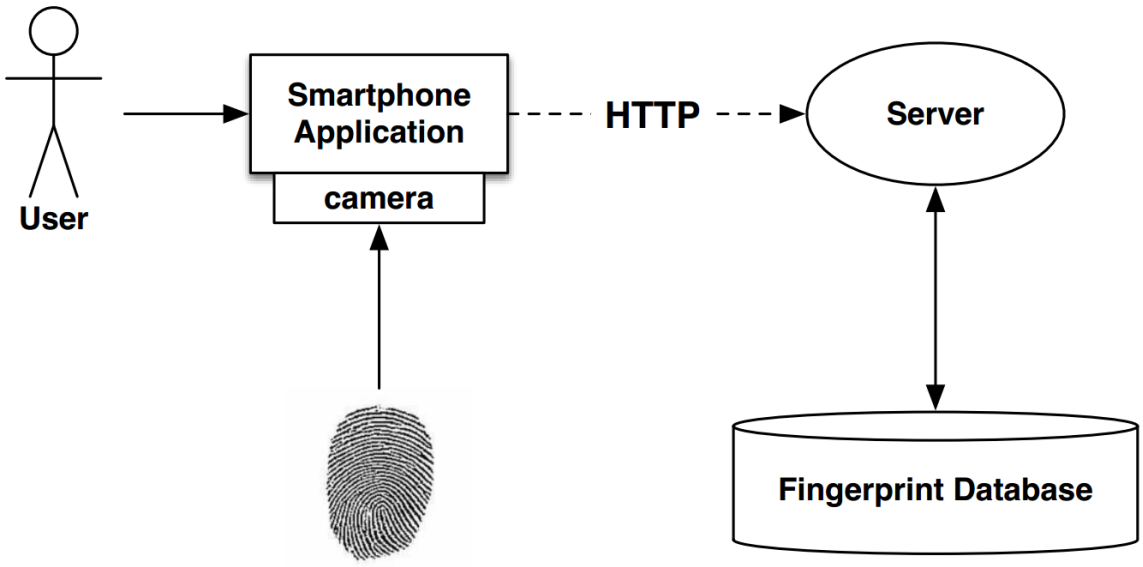

图 1.2 基于智能手机的指纹认证系统解决方案

该系统具有以下几个方面的创新：

（1）使用智能手机相机取代了传统的指纹传感器，节约系统成本并提高系统可用性。

（2）使用 Web Service 技术提供跨平台的服务以解决成本问题。

这种解决方案解决了系统成本与易用性问题，但与此同时也产生了系统安全性问题。由于用户的指纹信息通过 HTTP 协议进行传输，Web 攻击者可以监听网络请求并从中截获用户的指纹数据，诸如信息截取、重放攻击等网络安全问题也随之产生。由于系统安全事关重大，对我们的系统进行安全保障十分重要。因此，我将首先分析系统的安全性问题。

1.2 安全威胁与防御策略

在基于 Web 的指纹认证系统中将会产生以下几种安全问题：

（1）会话劫持。会话劫持的目的往往在于窃取信息内容本身。这种类型的安全攻击通常被认为是数据安全威胁。在指纹认证系统中，客户端需要通过网络向服务器端传输用户指纹数据，在传输过程中，一旦系统遭到攻击，客户端发送的指纹图像数据就有可能遭到窃取，用户的身份数据就会遭到泄漏。会话劫持的

基本原理如图 1.3 所示。

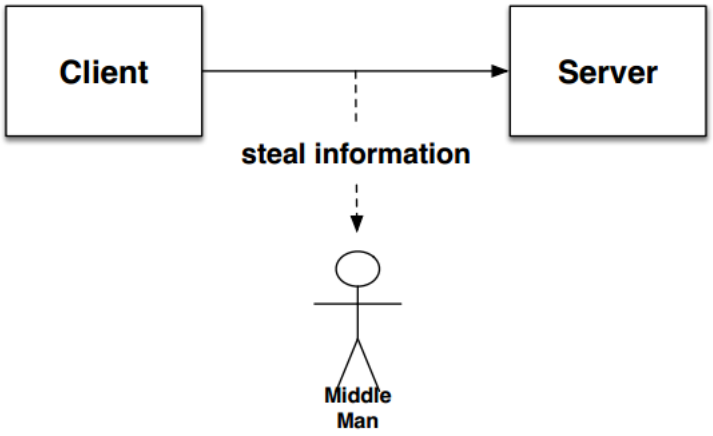

图 1.3 会话劫持原理

（2）重放攻击。重放攻击（Replay Attacks）又称重播攻击、回放攻击或新鲜性攻击（Freshness Attacks），是指攻击者发送一个目的主机已接收过的包，来达到欺骗系统的目的，主要用于身份认证过程，破坏认证的正确性。攻击者利用网络监听或者其他方式盗取认证凭据，之后再把它重新发给认证服务器。从这个解释上理解，加密可以有效防止会话劫持，但是却防止不了重放攻击。重放攻击任何网络通讯过程中都可能发生。一个经典的重放攻击如图 1.4 所示。

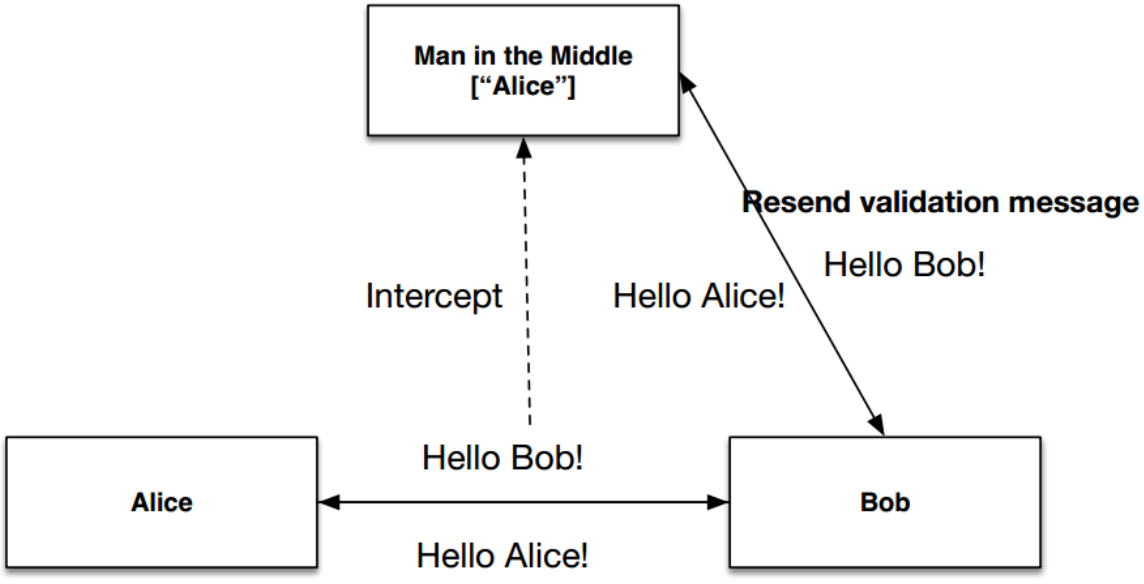

图 1.4 重放攻击示例

在上图中，Bob 充当了服务器端的角色，Alice 将其个人身份认证信息（用户名、密码）加密后发送给 Bob 进行认证并认证通过。于此同时，网络中间人

劫持了这次会话，中间人截获了 Alice 向 Bob 发送的认证数据，尽管该数据是加密的，但中间攻击人并不需要知道数据真实的内容，他只是伪装成 Alice 再次向 Bob 发送了同样了认证信息，Bob 接收到了同样的认证信息，再次通过了网络中间人的认证请求，并将网络攻击人误认为是 Alice.重放攻击的基本原理如图 1.5 所示。

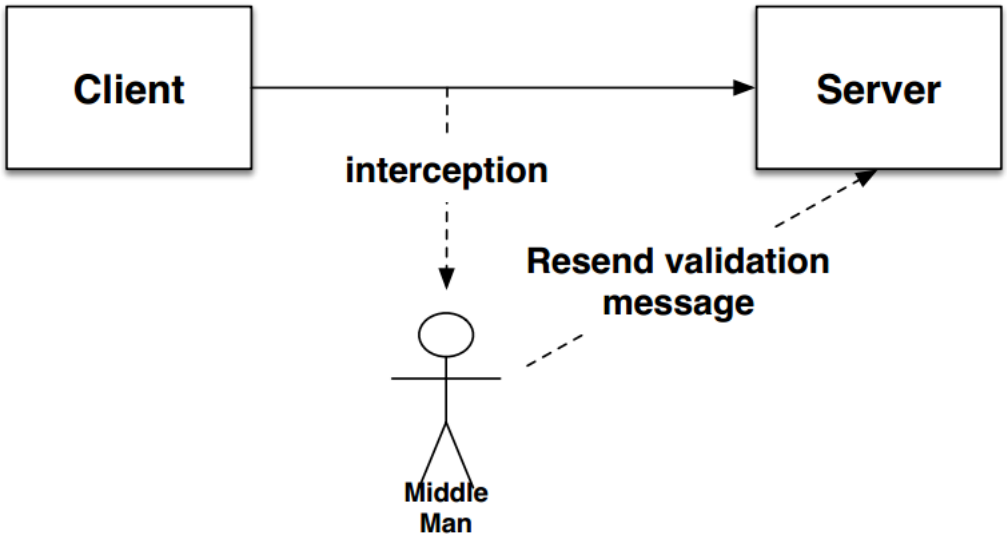

图 1.5 重放攻击原理

（3）数据篡改。这种类型的网络攻击可以认为是重放攻击的扩展。攻击者利用重放攻击破坏系统认证并进入系统，之后攻击者可以很容易地向服务器发送伪造的数据，甚至可以对服务器系统进行 SQL 注入攻击和系统数据篡改。其基本模式如图 1.6 所示。

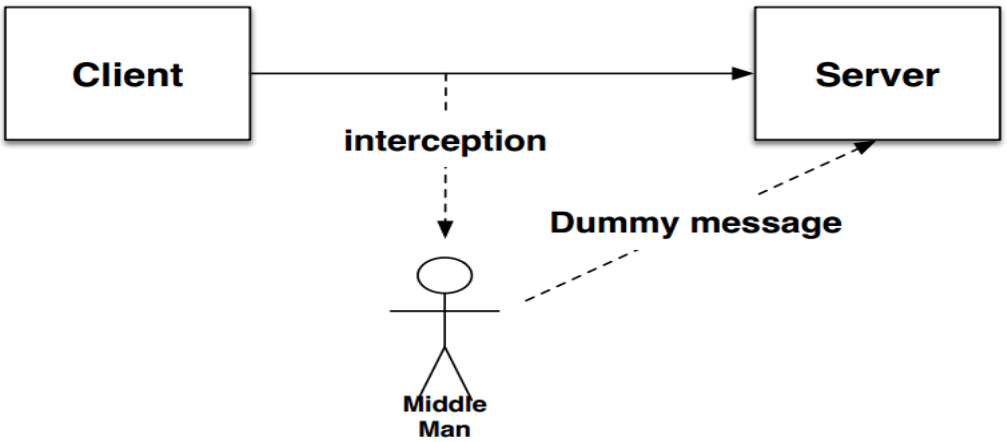

图 1.6 数据篡改原理

针对以上几类安全威胁，基本的防御措施如下：

（1）图像加密。针对上述第一种攻击类型，通常的解决方法是图像加密。在客户端向服务器发送图像文件之前，首先利用特定的加密算法对图像信息内容进行加密，再将加密后的图像发送给服务器。服务器接收到图像信息之后，利用对应的算法进行图像解密。这种情况下，即使攻击者窃取了图像信息，他仍然无法读取真正的信息内容，有效防止了信息泄漏。图像加密原理如图 1.7 所示。

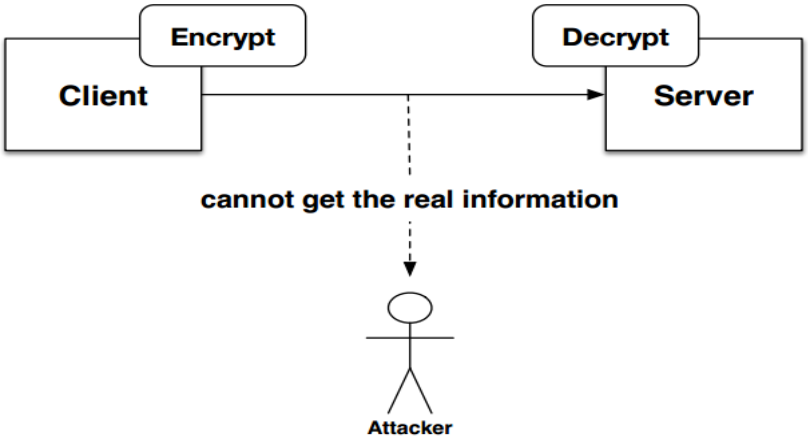

图 1.7 图像加密原理

（2）时间戳认证。针对重放攻击，一种行之有效的办法是采用时间戳认证。采用时间戳认证必须首先保证客户端与服务器端的同步，在此基础上，首先服务器端实时生成随机动态密钥，其次客户端发送请求获取该密钥，最后客户端向服务器端发送用户名和随机密钥以登录系统。在这种情况下，即使中间人截获了登陆信息，密钥的有效期也只有非常短暂的一小段时间。一旦服务器重新生成随机密码，中间人截获的信息就是无效的。时间戳认证的原理如图 1.8 所示。

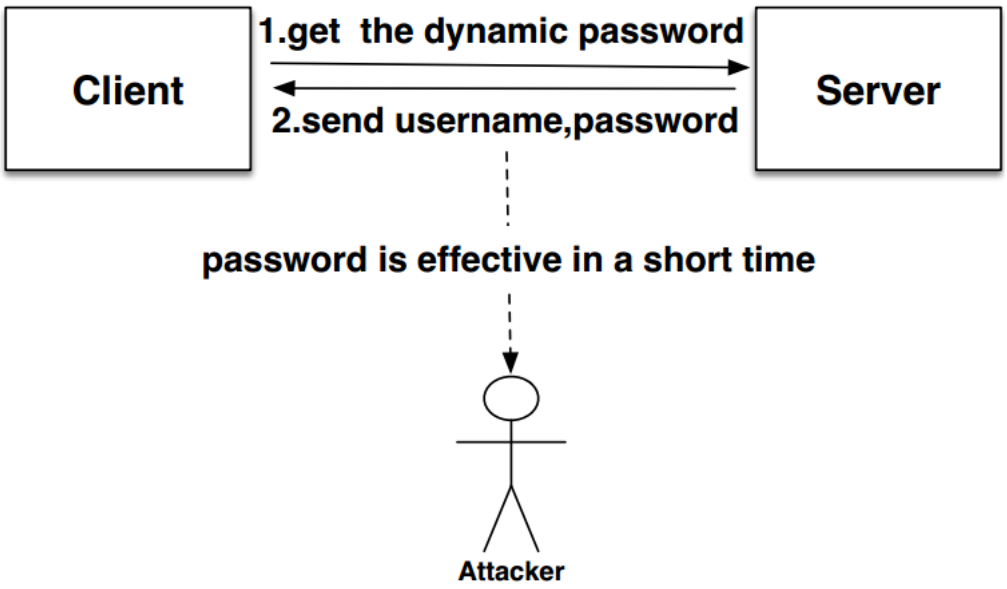

图 1.8 时间戳认证原理

（3）请求应答认证。请求应答认证是另一种防止重放攻击的有效方法。其基本模型如下所示：

第 1 步：客户端发起登陆请求。（可假设登陆请求是一个 Get 请求）

第 2 步：服务器端生成一个随机数 K＝random(num)，并将 K 返回给客户端。于此同时，服务器端缓存中保存该 K 值。

第 3 步：客户端计算 R＝Hmac(K,P)，在该公式中，K 代表密钥，是服务器返回的随机数，P 代表用户密码，Hmac 是一个用于加密的哈希函数。

第 4 步：服务器端从数据库读取用户密码，并进行与步骤 3 中相同的哈希函数运算 R’＝hmac(K,P’)，然后比较 R 和 R’，如果二者相等，用户登陆系统成功。

在这种模型下，中间人只能获取传输中的 K 值和 R 值，由于 K 值是一个随机数，R 值是一个哈希函数结果，因此这两个值都是无意义的。中间人无法通过这两个值获取用户密码，也无法发起重放攻击。因此提升了系统的安全性。请求应答认证的原理如图 1.9 所示。

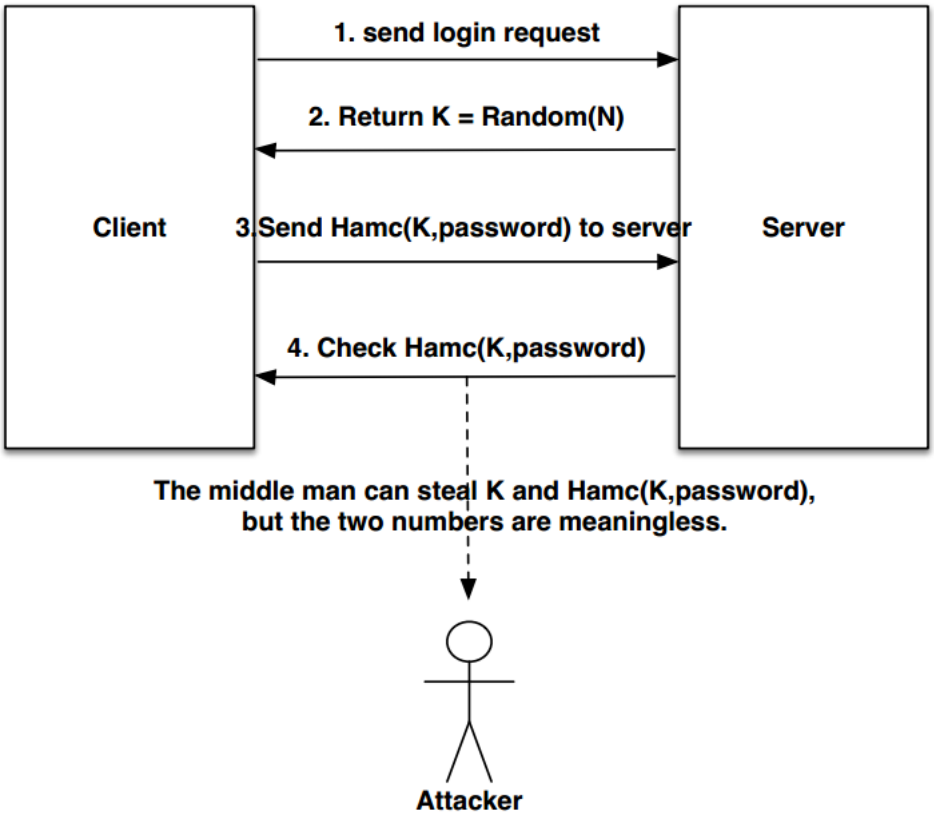

图 1.9 请求应答认证原理

1.3 基于数字水印的指纹认证系统

通过以上两部分分析，论文将主要探讨加强安全的指纹认证系统。首先，我们提出了一个新的解决方案取代传统的指纹认证系统。在新的解决方案中，我们没有采用传统的指纹读取器等硬件设备获取用户指纹信息，我们采用了智能手机中的相机模块获取相应的用户信息。这样就导致了系统更加便利，同时也降低了系统成本。其次，我们采用 Web Service 技术提供加强安全的 Web 网络认证服务。我们加入了一些安全策略以提高系统安全性并防止重放攻击。最终我们开发了一个系统原型，该系统原型解决了现有解决方案的高成本、低效率问题，具有低成本、便利、安全等优势。

1.4 论文章节结构安排

论文的章节结构安排如下：

论文第一章为绪论部分，讨论了现有指纹认证系统的问题，结合移动互联网、Web 安全等技术提出了一个新的基于 Web Service 的加强安全的指纹认证解决方案。

论文第二章为相关技术介绍，介绍了新解决方案所使用的几个关键技术，包括指纹识别技术，数字水印技术，Zxing 开源框架、SOA 技术与 Web Service 以及 Nancy Framework。

论文第三章为模型与需求，针对新的解决方案，建立了一些系统模型，并分析讨论了各个模型的问题，最终建立了一个比较安全可靠的系统模型，最后给出了需求建模。

论文第四章为系统架构设计部分，首先介绍了与系统密切相关的业务逻辑，然后给出了基于 REST 的系统架构设计方案，最终进行了数据库设计与建模。

论文第五章为系统实现部分，分模块实现了系统分析与设计中的功能，并详细探讨了系统中用到的关键算法。

论文第六章为系统实验结果部分，从成本、可用性、安全性等角度分析系统试验结果。

论文第七章为总结与展望部分，对基于 Web 的加强安全的指纹识别系统进行了总结，对系统开发中的核心问题和难点进行提炼，并对指纹认证系统的未来进行了展望。

2 关键技术分析

本章介绍与指纹认证系统相关的一些技术，由于基于 Web 的指纹认证是一个复杂的综合性系统，因此涉及到很多方面，这里只是对相关的重点技术进行简要介绍。

2.1 指纹识别技术

指纹识别技术是任何指纹识别系统的核心和关键技术。就指纹识别本身而言，大致可以分为以下 5 个步骤：

第一步，指纹数据采集。利用相应的图像采集设备获取指纹图像信息。

第二步，指纹预处理。由于采集到的指纹图像是未经加工的，初步的。因此，需要进行指纹图像的预处理，包括图像的增强处理，边缘检测，直方图均衡化等，经过处理的指纹图像应能达到下一步特征提取和匹配的要求。

第三步，特征提取。根据指纹的特点，对指纹图像进行特征提取，获取指纹的特征点信息。

第四步，指纹匹配。将两个指纹之间的特征点信息进行对比和匹配。

第五步，处理匹配结果。根据指纹对比匹配的结果，判断两个指纹之间的相似度，决定指纹是否匹配成功。在指纹识别领域，采用 FAR（认假率）和 FRR（拒真率）来衡量指纹识别算法的好坏。一般而言，一个好的指纹识别系统应该保证 FAR 和 FRR 尽可能的低。

目前指纹识别技术发展比较成熟。出现了一些著名的开源软件和一大批商业软件，比较有名的有美国国家标准与技术研究院 NBIS 课题组开源的 NBIS 指纹识别系统，基于 NBIS 的 SourceAFIS 系统，以及 CrossMatch 公司的 U.Are.U SDK. 许多指纹识别系统采用的指纹识别技术均基于此。这里简要介绍一下 NBIS 指纹识别软件。NBIS 指纹识别软件是由美国国家标准与技术研究院 NBIS 课题组开源的指纹识别系统。该系统可在类 Unix 环境下运行，提供了 Mindtct 特征提取工具，IMGTOOLS 指纹图像处理工具，BOZORTH3 指纹匹配工具，在 Unix 命令环境下可以方便地对指纹专有图像进行特征提取与处理。由于 NBIS 系统比较复杂，对系统环境配置要求较高，因此出现了基于 NBIS 的 SourceAFIS 开源库。SourceAFIS 对 NBIS 进行了简化，抽取了 Mindtct 与 BOZORTH3 的核心功能，并与.net 和 Java 技术相结合，提供了.net 版本和 Java 版本更便于实际使用。

Chris Stein, Claudia Nickel 和 Christoph Busch 使用上述相关技术给出了一

个基于智能手机相机的指纹识别系统解决方案，并使用 android 手机开发了原型系统。但是由于该系统并不是基于 Web Service 的，因此只实现了基本功能，许多安全性问题没有考虑。

2.2 数字水印技术

数字水印技术是保证信息来源真实可靠的重要的技术。客户端对原始信息加水印可以保证信息来源的准确可靠，服务器接收到客户端发送的水印信息之后，可以先分析水印信息，确保客户端的身份，之后在对水印信息进行去水印，对原始信息进行处理。如果客户端在加水印过程中融合了时间戳技术和挑战应答技术，就可以保证系统的安全性，加水印的数据既可以防止信息内容泄漏，又可以识别信息来源的真实性，防止重放攻击。因此，数字水印技术是加强系统安全的不二选择。

客户端加水印，服务器端去水印。这种数字水印技术被称为可逆数字水印技术，是水印技术发展的难点之一。实现数字水印的方法有很多，最基本的方法当属最低有效位算法。Waiton 最早提出了最低有效位算法来实现数字水印，Waiton 使用图像中的低位信息存储密钥信息，用图像中的其他位信息生成密钥信息，这样低位存储的密钥信息就成为数字水印。用该种算法生成的可逆水印简单容易使用。A.Z.Tirkel, R.G.van Schyndel, C.F.Osborne 基于最低有效位算法提出了二维数字水印技术，并讨论了该技术在 JPEG 图像转换的适用性。Zhang Ning, Zang Ya-Li, Tian Jie 结合生物识别技术和密码技术提出了一个新的身份认证解决方案，并讨论了指纹识别技术和安全密钥技术的结合。

2.3 ZXing 开源框架

ZXing Project 全称 zebra crossing，是由 Google 公司开源的条形码与二维码图像处理库，能简单快速地生成和解析多种条形码和二维码，支持 UPC，EAN，Codebar，QR Code，Data Matrix 多种条形码标准，该开源库采用 Java 实现，并提供了多种其他语言接口。ZXing 框架包含以下几大模块：code 模块，条形码编码解码核心模块；Javase 模块，Javse 客户端模块；Android 模块，Android 客户端模块；Android-test 模块，Android 客户端测试模块；Android-integration 模块，支持 Intent 集成条形码预览；Glass 模块，简单的 Google 眼睛应用实例等。使用 Zxing 框架，可以在系统中简单方便地集成条形码处理功能，大大简化了系统处理条形码和二维码的难度。Eui-Hyun Jung 和 Seong-Yun Cho 研究了条形码技术

和数字水印技术，然后提出了采用二维条形码的数字水印解决方案。

2.4 SOA 与 Web Service

SOA 与 Web Service 技术是服务器端实现的重点与核心技术。这里将进行重点探讨。

2.4.1 SOA

SOA 即是面向服务的架构（service-oriented architecture）.简单的说，SOA 是开发应用系统的一种新架构，在基于 SOA 的系统中，应用程序的构建过程类似于服务组件的组合过程。这些组件都是松耦合的并具有明确的接口定义，我们通常将这些组件称为服务。SOA 技术架构的优点是实现系统的跨平台性和优良的可扩展性。

举例而言，在我们的指纹认证系统中，我们需要实现一个 Android 客户端和一个服务器端，Android 客户端与服务器端通信，调用服务器端提供的 API 接口服务以实现录入指纹和指纹认证的功能。在传统的软件体系结构下，客户端与服务器端的开发必须置于同一技术架构之下，如果客户端采用 Android 技术实现，相应的，服务端也必须采用 Java EE 技术实现；如果要开发 Windows Phone 的客户端，也必须开发对应的.net 框架下的服务器程序。这种传统的软件架构模型是一种紧凑的架构体系，其优势在于服务器与客户端紧密协作，共同依赖相同类型的核心组件。然而，这种架构体系的缺点也是很明显的，最突出的问题的扩展性较差。如果采用.net 技术进行客户端的开发，那么也意味着服务器端也必须采用同样的技术，这种传统的软件架构模型不适应移动互联网时代的业务发展需求。

在移动互联网时代，一个服务往往需要支持多种客户端共同调用。举例来说，社交软件"Line"就需要为多种客户端提供多种服务，比如登录服务既需要支持 PC 客户端进行登录，同时也要提供对 Android 手机客户端和 iOS 手机客户端的登录支持，甚至还要提供对 Web 浏览器的登录支持。如果我们采用传统的系统架构，我们需要为每一种平台编写服务端代码，然而许多服务代码都是类似甚至是重复的，这样就不符合软件工程中的基本原则——DRY 原则（Don't Repeat Yourself）.在这种移动互联网的背景下，SOA 架构迅猛发展。

在 SOA 架构体系下，每一个应用功能被封装成服务，这些服务都是与具体平台无关的。客户端与服务器端的通信通过消息传递进行。通信的数据应符合数据传输的标准和规范，如 XML 和 JSON.当客户端需要调用服务 API 时，客户端向对应的服务接口发一条消息，服务器端解析消息并响应客户端消息请求，向客

户端返回标准化数据，客户端解析标准化数据获取最终结果。这样就实现了从面向具体技术的架构向面向服务的架构的迁移。无论客户端采用 android 技术还是 iOS 技术，抑或是 Web 技术，甚至是桌面应用程序，他们都可以调用同一个服务接口，并且不需要修改服务器端代码。SOA 架构显著增强了系统的跨平台性和可扩展性，现在 SOA 架构已然成为了移动互联网领域的主流架构模型，而 Web Service 是实现 SOA 体系的一种具体技术。

典型的 SOA 模型如图 2.1 所示。

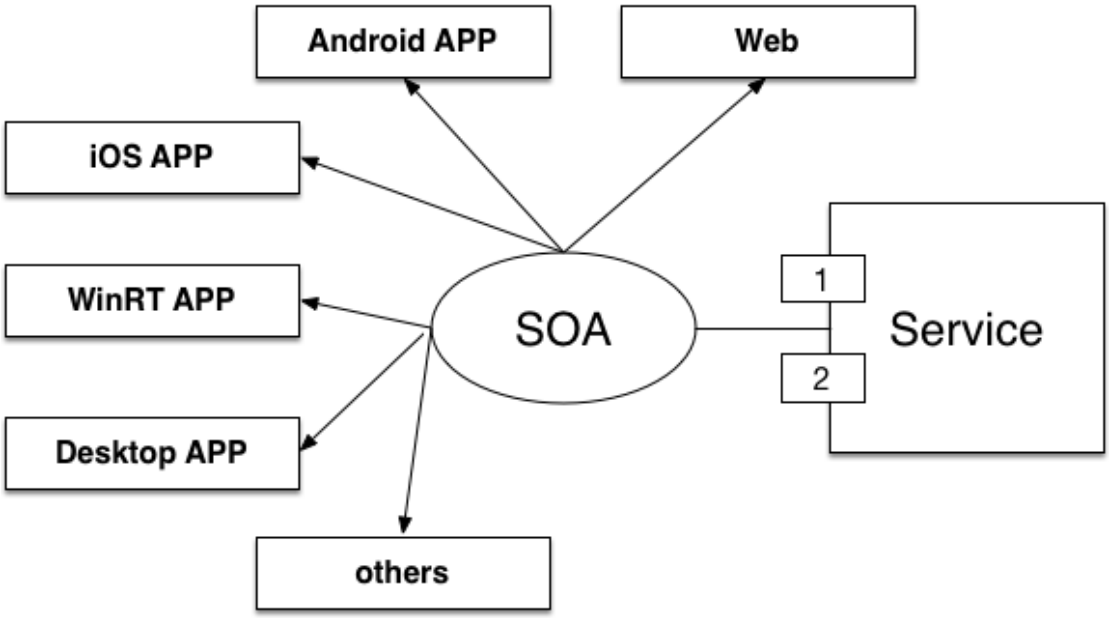

图 2.1 SOA 架构模型

2.4.2 Web Service

Web Service 技术是一种实现 SOA 最为通用和主流的技术。它通过 Web 标准与协议提供 Web 服务。Web Service 的目的在于实现异构系统间的相互协作。

经过数年的发展，目前 Web Service 技术已经发展为以下两大主要类型：

（1）基于 SOAP 的 Web Service

（2）基于 REST 的 Web Service

基于 SOAP 的 Web Service：SOAP 是简单对象访问协议的缩写，被用来描述信息传输的格式，WSDL 是网络服务描述语言的缩写，用于描述 Web 服务的公共接口，UDDI 是统一描述、发现和集成的缩写，用于管理、发布和查询 Web Service. SOAP 使用两大应用广泛的协议——HTTP 和 XML. HTTP 被用来实现 RPC 风格的 SOAP 传输，XML 是其编码形式。通过 SOAP，Web Service 具有了

优秀的扩展性，与特定技术、编程语言、平台的完全独立性。

基于 REST 的 Web Service: RT Fielding 博士在其博士论文"Architectural Styles and the Design of Network-based Software Architectures"中首次提出了 REST 风格的 Web Service 的基本概念。REST 是表述性状态转移的缩写，REST 风格具有以下几个特点：

（1）首先，REST 是一种架构风格，不是一个具体的标准；

（2）REST 风格是基于资源的架构风格；

（3）REST 架构风格的目的是实现一个组织良好的 Web 应用程序；

（4）REST 架构风格是对 HTTP 协议的真正充分利用：

（a）它逻辑上通过 URI 来定位资源；

（b）它通过 HTTP 请求头信息来判断客户端需要获取何种资源或是对资源进行何种操作；

（c）在 REST 风格体系中，使用不同 HTTP 的请求方法来实现 CRUD 操作（增删改查操作），具体而言，GET 操作可以获取资源，POST 操作可以添加资源，PUT 操作可以更新资源，DELETE 操作可以删除资源，等等。REST 风格的 CRUD 操作如图 2.2 所示。

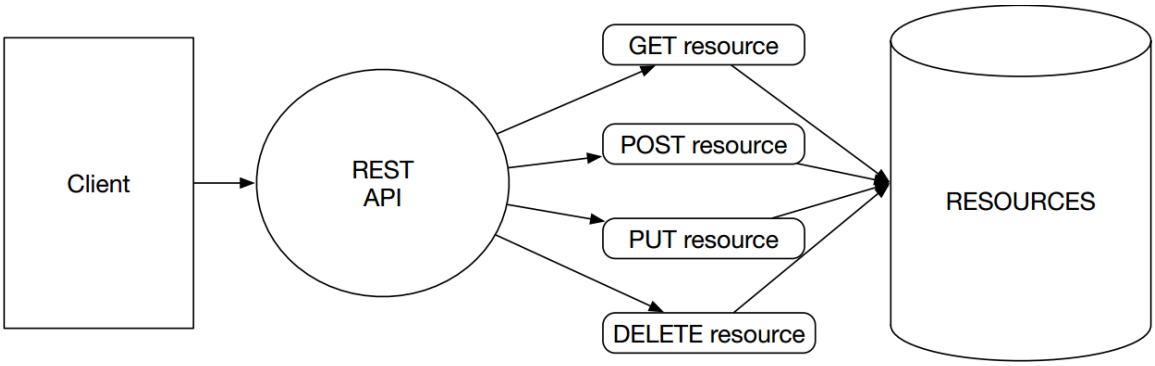

图 2.2 REST 风格的 CRUD 操作

REST 架构风格的以上特点使它比基于 SOAP 的 Web Service 更简单，更轻量，更能够完全地利用 HTTP 协议，REST 风格实际上代表了 HTTP 设计之初的真正意图，因此，近年来 REST 风格的 Web Service 已经完全改变了 Web Service 的状况，极大地适应了移动互联网的发展趋势，现在，主流的移动客户端开发都已经从 SOAP 向 REST 架构风格进行迁移。在我们的指纹认证系统中，为了配合 android 客户端的开发，服务器端同样也将采用 REST 风格的 Web Service 实现。

2.5 Nancy Framework

我们使用 Nancy Framework 来实现轻量级的 REST 风格 Web Service. Nancy Framework 是一个基于.net 和 mono 平台的轻量级的 Web 框架，它遵循 MVC 模式并被设计用来支持 REST 风格的 Web Service. Nancy 官方网站对 Nancy 框架的特征介绍如下：

（1）Nancy 是一个轻量级用于构建基于 HTTP 的 Web 服务，基于 .NET 和 Mono 平台，框架的目标是保持尽可能多的方式，并提供一个 super-duper-happy-path 所有交互。

（2）Nancy 设计用于处理 DELETE, GET, HEAD, OPTIONS, POST, PUT 和 PATCH 等请求方法，并提供简单优雅的 DSL 以返回响应。

（3）Nancy 和 Asp.net MVC 原理相似，但有自己的一套路由机制，在使用上更加易用，可以用 Nancy 快速开发一些网站。

（4）Nancy 并不依赖任何现有的框架，所以他可以运行在任何平台上面。

下面介绍 Nancy 框架的核心思想。

2.5.1 MVC

Nancy 框架的设计灵感来源于 Ruby 中的 Sinatra 框架，他们的基本思想都是 MVC 模式，MVC 模式是模型－视图－控制器模式的缩写。

模型是应用系统用于描述数据逻辑的部分，通常用于封装数据对象以存储数据。视图是应用系统中描述显示逻辑的部分，视图负责展示给用户的界面逻辑，视图中的数据通常通过模型层获取。控制器是应用系统中业务逻辑处理的部分，负责从视图中读取数据，处理用户输入跳转，将数据发送给模型等业务逻辑处理，是系统的控制中枢，也是 MVC 模式的核心部分。

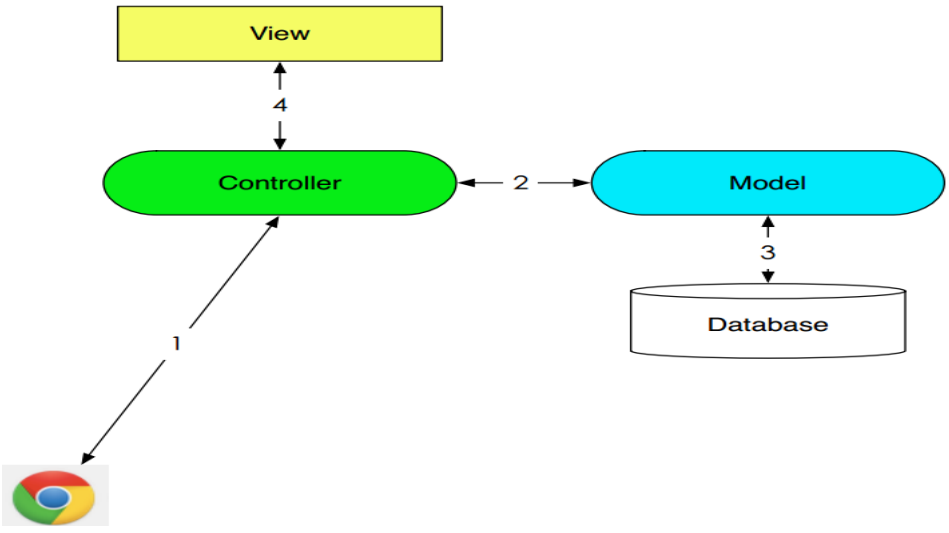

图 2.3 MVC 核心工作流

图 2.3 展示了 MVC 模式的核心工作流：

（1）首先，用户向应用程序程序发送请求，等待服务端响应

（2）控制器接收用户请求，然后由控制器决定是从模型中读取数据还是直接将用户请求交给对应的视图进行响应。

（3）控制器向模型层发送数据请求。

（4）模型层从数据库中读取请求的数据，并将获取到数据返回给控制器。

（5）控制器将模型返回的数据发送给响应的处理视图。

（6）视图层通过从模型中获取数据填充视图内容，并将最终的 HTML 结果返回给控制器。

（7）控制器将结果视图返回给用户。

通过 MVC 模式，实现了应用系统业务逻辑和表现逻辑的分离，数据逻辑和数据库系统的分离，因此降低了模块之间的耦合性，也便于前后端开发人员的相互协作。因此越来越多的应用系统采用 MVC 模式实现，于此同时也产生了大量优秀的遵循 MVC 模式的框架，如 Ruby on Rails，Java Spring MVC，ASP.net MVC 等等。我们所采用的 Nancy 框架也是.net 平台上的一个轻量级的 MVC 框架。

2.5.2 Nancy 框架中的 MVC 实现

Nancy 框架是.net 平台下基于 MVC 模式的轻量级 Web 框架，一个 Nancy 项目结构如图 2.4 所示。

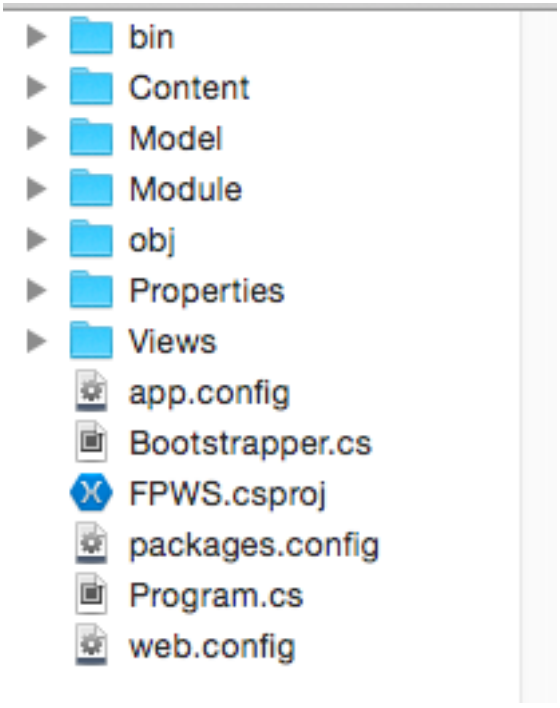

图 2.4 Nancy 项目结构

在 Nancy 框架中，Model 表示模型模块，Views 表示视图模块，Module 表示控制器模块。

1. 控制器遵循 REST 风格，其基本的工作模式是：当用户在浏览器中请求 http://hostaddress/ 时，Nancy 框架中的控制器类会首先通过 Uri 查找对应的控制器，这种机制被称为路由机制，这种路由机制也是 Nancy 框架实现 Rest 风格的 Web Service 的核心。当用户向服务器根目录 Uri 发送 GET 请求时，控制器会首先通过路由机制查找 GET["/"]控制器，在找到响应的控制器之后，用户请求会被对应的控制器进行处理。而对应的控制器可能会调用对应的视图资源或模型资源来响应用户请求。

2. 视图遵循 HTML 作为其基本风格，在 Nancy 框架中，官方支持的视图引擎是 Radar 视图引擎，Radar 引擎的语法与 HTML 基本类似。

3. 视图层将数据库组织成相应的对象，并支持大多数的 ORM 框架。

在论文中，我们将结合以上多种核心技术来构建我们的指纹认证系统原型。

3 模型与需求

本章探讨的主要问题是模型与需求，将主要讨论指纹认证系统的安全策略问题，通过讨论相应的安全策略确立一种优良的系统解决方案模型，在此基础上提出需求建模。在本章中，首先我们会简要地分析系统安全威胁；其次我们会讨论加强系统安全的传统解决方案；然后我们提出了基于可逆数字水印的新的安全解决方案，并对其进行了逐步改进。最后我们确立了解决方案模型和系统需求模型。

3.1 原有的解决方案

3.1.1 安全威胁

正如我们在论文第一章讨论的那样，Web 系统中存在多种类型的安全攻击。其中一种攻击是会话劫持，会话劫持的目的在于窃取信息内容本身。另一种攻击是重放攻击，重放攻击的目的主要在于破坏认证的有效性。在我们的指纹认证系统中，网络攻击者会尝试劫持包含有用户指纹信息的用户数据包，这是一种典型的会话劫持类攻击，攻击者的目的在于盗取用户的指纹信息，在这种情况下，用户信息将会被窃取。此外，攻击者在截获了用户发送的指纹数据包之后，会假冒用户向服务器再次发送该指纹数据包，以骗取服务器信任，即使用户发送的指纹数据包经过加密也无法防止这种攻击，这种攻击是一种典型的重放攻击。攻击者通过重放攻击以达到欺骗认证服务器的目的，这种情况下服务器会把攻击者认定为原始的用户，通过指纹认证。

可见在我们的指纹认证系统中同时包含了以上两类安全隐患，指纹认证系统应该是足够安全的。因此，我们尝试设计安全子系统以加强系统的安全性，最终使我们的系统可以同时防御会话劫持和重放攻击两种类型的攻击。

3.1.2 通常的解决方案

文件加密和数字签名是加强系统安全性的最通用的解决方案。其中文件加密主要用于防止信息内容泄漏，而数字签名主要用来确保信息来源的真实可靠性。

文件加密：文件加密的基本流程是通过特定的算法对原始文件信息进行编码，使经过加密的文件变的不可读，我们可以将其称之为密文。在没有密钥的情况下原始文件信息是无法获取的。通过这种方式我们可以有效的防止数据被非法窃取。对应的，文件加密的逆过程也被称为文件解密。

数字签名: 数字签名技术本质上是通过数学方法来验证消息、软件和数字文

件来源的真实性和可靠性。数字签名的基本流程如图 3.1 所示。

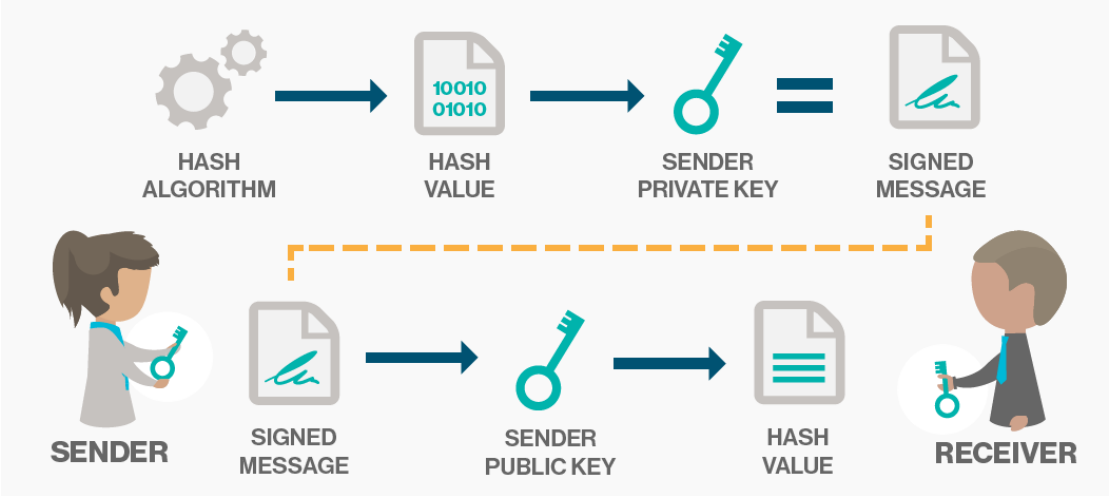

图 3.1 数字签名基本流程

从图 3.1 中我们可以了解到数字签名技术的工作原理。首先，数据发送方使用他的私钥对他需要发送的信息进行加密，这样他就获得了签名消息；其次，数据发送方讲签名消息发送给接收方。数据接收方接收签名消息，并利用公钥对签名消息进行解密，解密后，接收方将获得解密后的 Hash 值，如果接收方解密后的 Hash 值与发送方的 Hash 值相匹配，则证明消息在签名之后没有被篡改，否则则证明数据在签名之后遭到攻击者攻击，信息内容已经被改变。

3.1.3 传统解决方案的问题

上述传统解决方案对于加强系统安全性和抵御网络攻击能够起到良好的效果。但是应用在我们的指纹识别系统中仍然有一些缺陷和问题。

首先是系统复杂性的问题，数字签名需要另外一整套复杂的架构，此外，我们可能需要将安全系统拆分成两个子系统，一个子系统用来进行文件加密，另一个系统进行数字签名。这就导致了系统架构的进一步复杂化。其次，上述两种解决方案是一种通用的解决方案，并没有考虑到指纹识别系统的特殊性，也没有结合指纹图像的特征。鉴于此，我们提出了一个新的安全解决方案。

3.2 基于数字水印的解决方案

为了克服传统解决方案的上述缺陷，在本部分我们将提出一个新的解决方案。首先，我们会分析在没有加入安全措施的情况下系统可能产生的问题，之后我们会在这些问题的基础上，对解决方案进行逐步改进和优化。

3.2.1 安全漏洞

首先，在不采用文件加密和数字签名的情况下，我们会发现原始的指纹认证系统会产生如图 3.2 所示的安全漏洞。

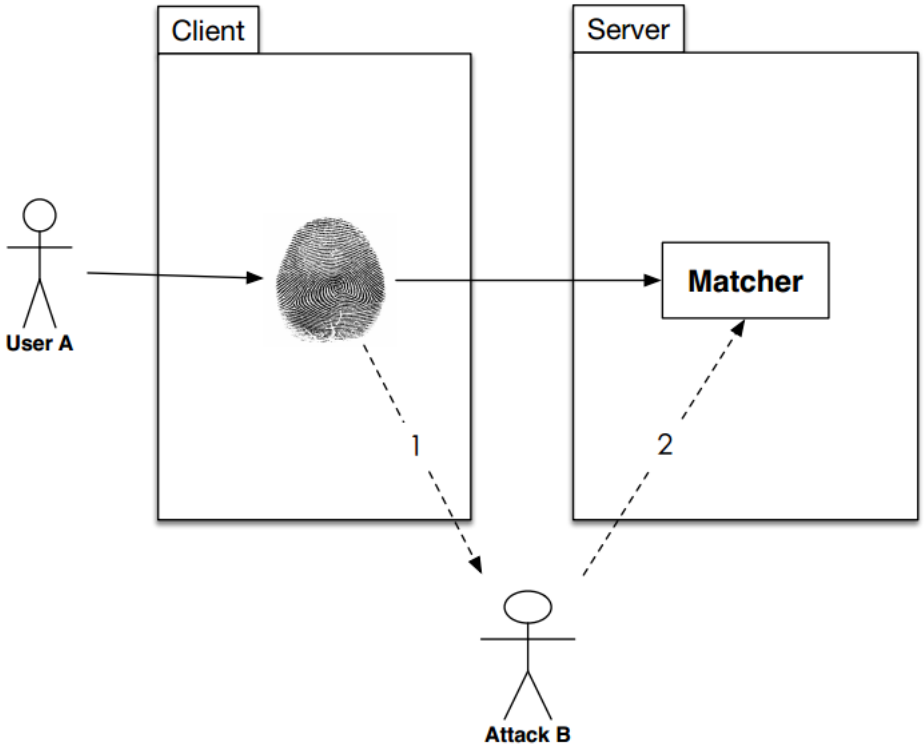

图 3.2 安全漏洞

在图 3.2 所示的业务场景下，用户 A 希望采用本系统进行身份认证，该名用户采用智能手机采集他的指纹图像，之后用户 A 直接向服务器发送该图片以进行匹配。然而此时网络攻击者 B 正在监听此次通信过程，网络攻击者 B 是一个恶意监听者并且盗取了用户 A 的指纹图像信息。几分钟之后，攻击者 B 假扮 A 向服务器发送刚刚窃取到的指纹图像。这样系统会把攻击者 B 当作是真正的用户 A，B 就成功获取了系统认证。在这种情况下，由于 B 获得了与 A 相同的系统认证，对于用户 A 而言显然是不安全的。可见，原始的指纹认证系统无法防止会话劫持和重放攻击，我们需要在系统中加入一些安全策略以增强系统的安全性。

3.2.2 第一次改进

为了增强系统安全性，我们在系统中引入了安全模块。

引入安全模块的两个关键点在于：

（1）为了防止指纹图片信息泄密，客户端不应该将最原始的指纹图像信息发送给服务器。

（2）为了防止重放攻击，系统可以采用一种类似于一次性密码本的加密技术。这意味着系统需要生成一个单次有效的密钥。为了实现上述系统，我们可以基于时间戳来产生密钥。当用户需要进行指纹认证时，服务器端首先检查时间戳密钥，如果时间戳密钥正确，再进行指纹认证操作，否则系统不能确定请求来源的可靠性，就不再进行指纹认证操作。

为了实现以上两个关键点，我们结合了二维码技术来构建指纹认证系统。首先，服务器端根据时间戳来生成对应的二维码图片然后将该图片发送给客户端。在这种情况下，由于二维码图片包含有时间戳信息，因此可以作为一次性密钥来防止重放攻击。此外，我们可以将二维码图片和原始的指纹图片相结合，这样客户端发送图片时可以采用新的图片代替原始的指纹图片。最终，加入了二维码技术的安全模块满足了上述两点安全需求。

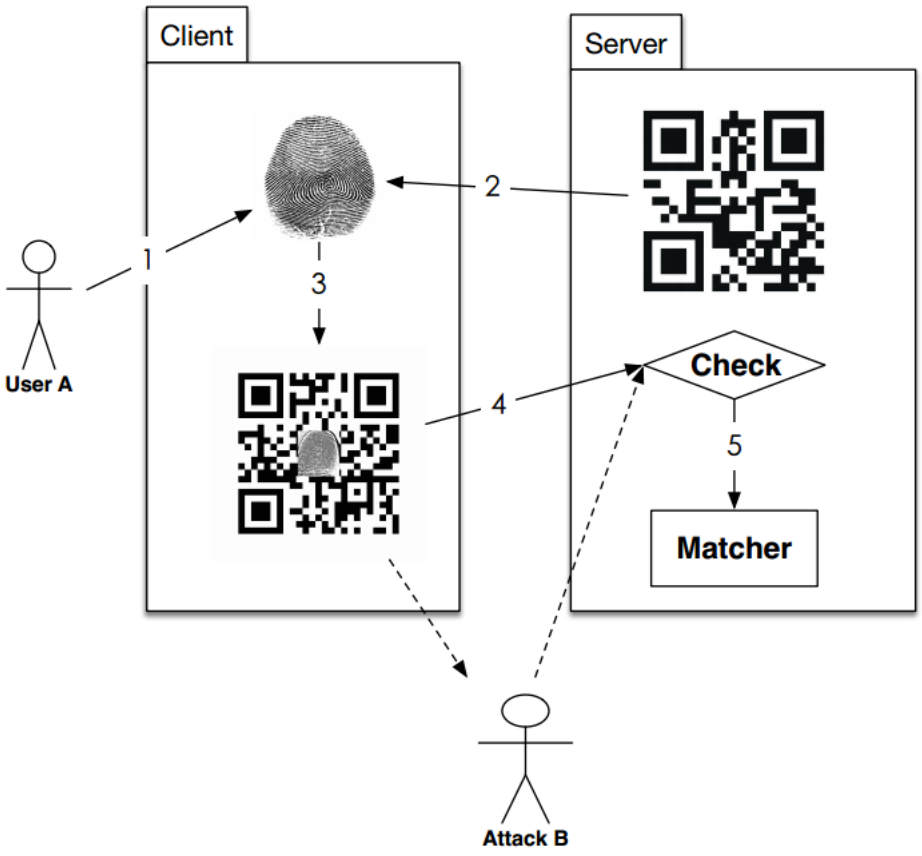

图 3.3 第一次改进策略

改进后的系统解决方案如图 3.3 所示。在这种情况下，用户 A 首先拍摄指纹图片，其次他向服务器发送请求，服务器接收到用户认证请求后首先根据当时的时间戳生成相应的二维码图片并将该图片返回给客户端。客户端接收到二维码图片之后，将拍摄到的原始的指纹图片放置于二维码图片中心位置产生了一个新的

图片。接着，客户端将新的图片发送给服务器，服务器端接收到新的图片以后，首先对二维码进行解码获得时间戳信息，判断时间戳信息是否正确。如果与服务器端生成时保存的时间戳一致，系统就将指纹图片提取出来发送给匹配模块，否则服务器会直接返回认证失败信息给客户端。

现在我们再来分析新系统的安全性。此时，攻击者 B 仍然能够通过会话劫持窃取包含二维码和指纹的新图片。但是，由于图片是二维码和指纹图片的混合，攻击者不能再直接获取原始的指纹图片了。更重要的是，攻击者不再能够进行重放攻击。如果攻击者 B 再次向服务器发送他刚刚劫持到的图片，图片中的时间戳已经不再有效，原因在于时间戳认证在 B 重发消息之前已经被认证过了，而时间戳仅一次有效。因此，服务器不会将攻击者 B 发送的图片分发给匹配模块作进一步处理。服务器会直接返回失败消息给攻击者。

由此可见，指纹认证系统的安全性已经得到了显著的改进与增强。然而该系统仍然是不安全的。攻击者仍有许多办法来获取原始的指纹图片，攻击者只需要将截获到的新图片进行中心部分裁剪就可以轻松地还原出原始的指纹图片，再他们获取到了原始的指纹图片以后，攻击者可以重新向服务器发送认证请求以获取新的二维码图片。这样，他们只需要将二维码图片与原始图片进行结合就可以再次将图片发送给服务器进行认证。这又变成了一种重放攻击，在这种场景下，由于时间戳消息是全新可用的，攻击者可以通过服务器端的时间戳检查。这样攻击者就成功通过了系统认证。为了防止这种情况的发生，该解决方案仍然需要进一步改进。

3.2.3 第二次改进

这次改进的核心思想和第一次改进基本相同。上述解决方案的主要问题在于，客户端只是简单直接地将指纹图片与二维码图片进行了拼接，由于指纹图片本身并没有被隐藏起来，这使得新产生的图片很容易被破解。因此，如果我们能够采用一种算法使指纹图片隐藏起来，攻击者就很难对截获到的图片进行还原。这样系统安全性就会大大提高。

此时，我们将可逆数字水印技术加入我们的指纹认证系统中。采用这一技术，我们可以轻松地将原始指纹图片和二维码图片进行组合，于此同时可以将指纹图片隐藏起来。这样，就可以把指纹图片作为数字水印隐藏起来，而不是将其直接发送给服务器。

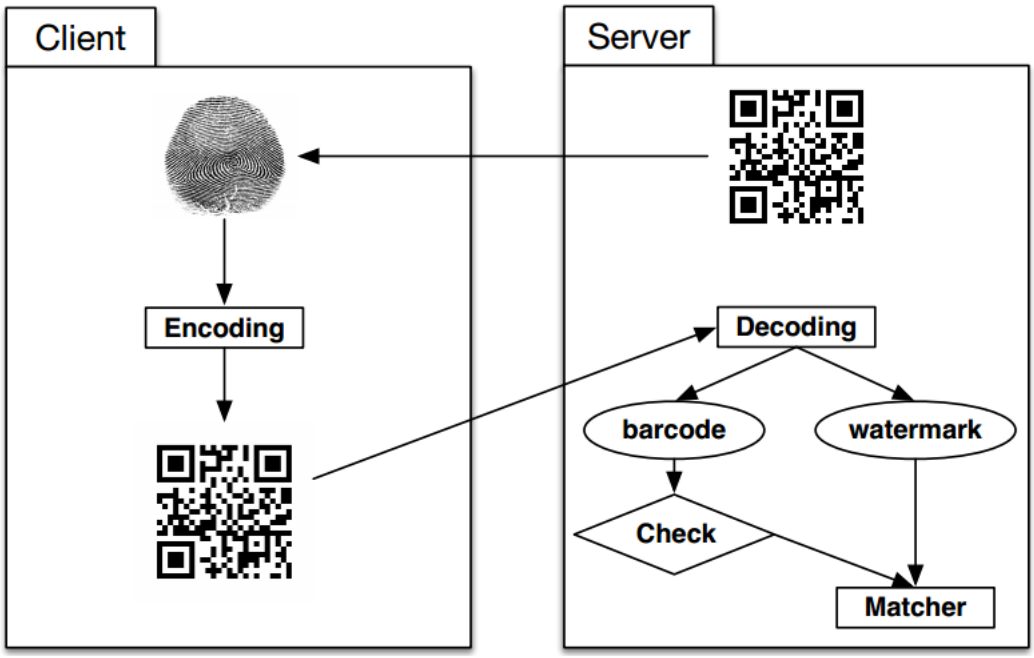

图 3.4 第二次改进策略

图 3.4 展示了本次改进后新解决方案的基本流程。该解决方案与第一次改进后的方案基本类似。我们只是采用了数字水印代替了之前的直接组合。客户端把指纹图片作为数字水印并将其编码进二维码图片当中。服务器端接收包含数字水印的二维码图片，然后调用解码模块对二维码图片进行解码。解码之后，服务器端会将原始二维码图片和原始的指纹图片进行分离，接着服务器就可以检查二维码中的时间戳信息，如果时间戳正确，则对指纹图片进行匹配操作，否则返回认证失败。

经过这样的加密，系统安全性显著提高。攻击者再也不能很轻易地获取原始的指纹图片了。攻击者只能截获到一张二维码图片，而这张图片本身对身份认证是没有任何意义的。如果攻击者尝试还原原始的指纹图片信息，他们必须知道相应的数字水印加密算法，否则他们就无法破解出真实的指纹信息。然而，一旦系统加密算法被泄漏，攻击者仍然可以破解出原始的指纹信息，这样一来，系统仍然是不够安全的。因此，我们还需要对指纹系统进行进一步的改进。

3.2.4 第三次改进

第三次改进则非常简单。我们在最终的系统中加入密钥机制。在第二次系统改进的基础上，我们采用用户密码作为密钥对上次改进中的图片进行再加密，客户端将再加密之后的图片发送给服务器。服务器端接收到图片以后首先根据用户名在数据库中查找相应的用户密钥，然后根据用户密钥对图片进行解密，在解密

之后，服务器端执行与上一部分相同的操作即可。第三次改进如下图 3.5 所示。

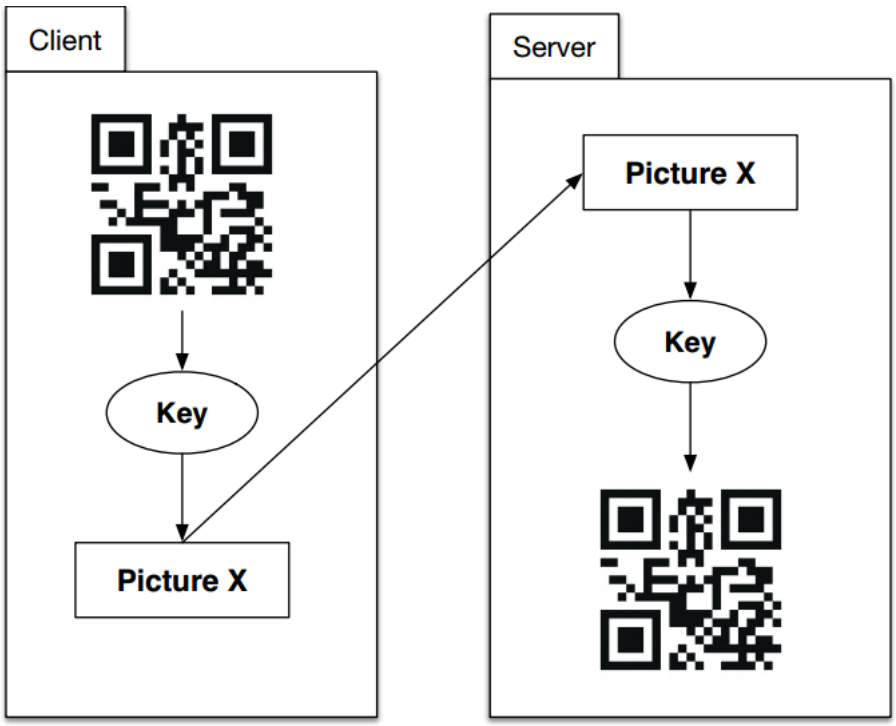

图 3.5 第三次改进策略

这样，攻击者必须通过掌握了系统加密算法和用户密钥之后才能还原出原始的指纹图像，大大增加了破解的难度和成本。因此，我们认为经过三次改进后的系统已经足够安全。

3.3 需求建模

下面，我们基于上述解决方案分析对系统进行需求建模。本部分采用面向对象的需求分析方法对上述解决方案进行需求分析，首先给出了系统整体用例图，然后对用例图进行了具体描述。

3.3.1 总体用例图

系统总体用例图如图 3.6 所示。

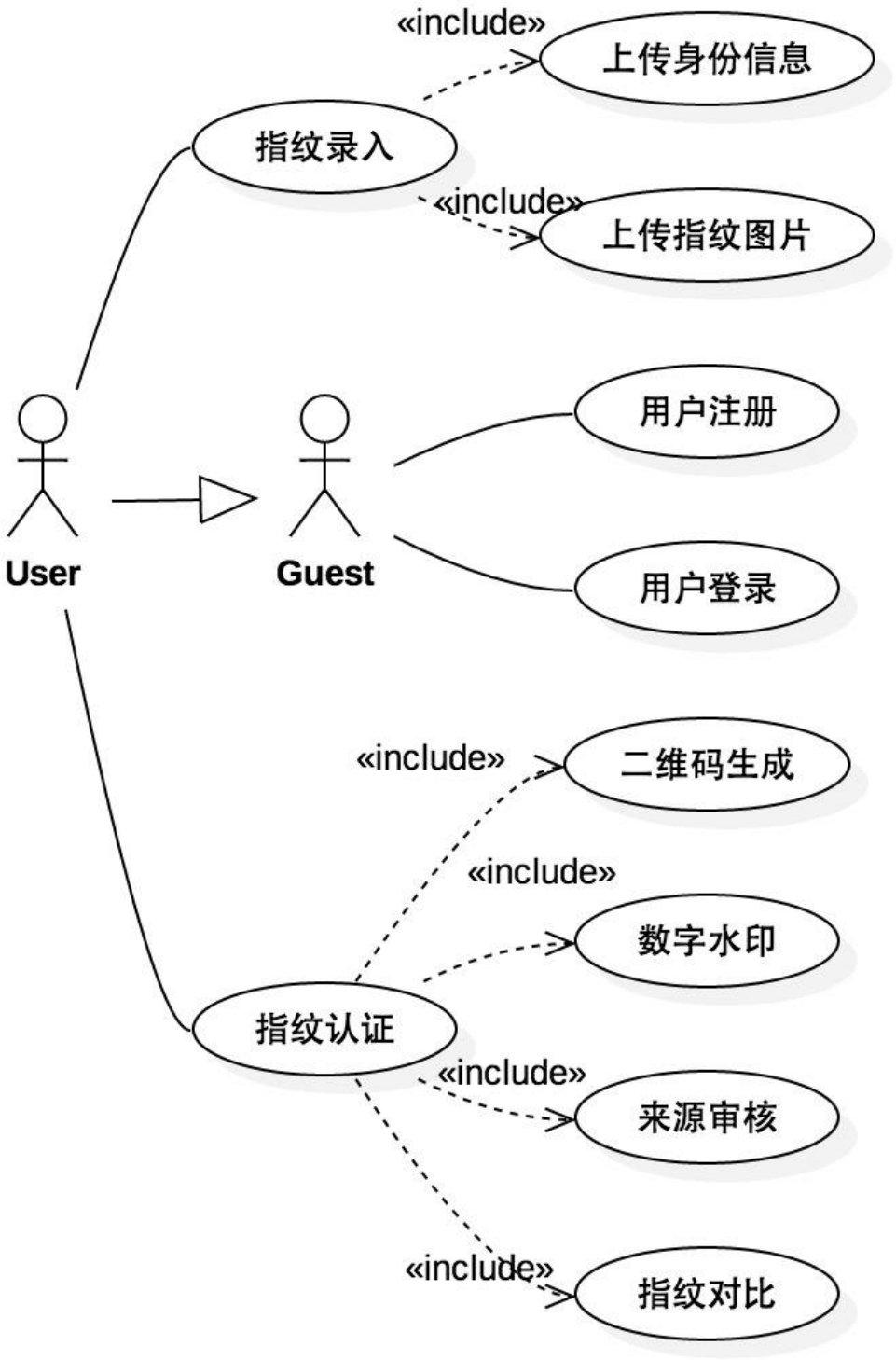

图 3.6 系统总体用例图

3.3.2 用例描述

从总体用例图可以看出，系统参与者主要用两类：游客 Guest 和用户 User,

而由于本系统和拥护身份认证等敏感操作息息相关，因此本系统游客除了注册登录以外并没有任何可用操作。对于游客而言，他们只有用户注册和用户登录两个用例。游客进行登录操作以后，游客就转化为用户 User. User 是本系统的核心参与者，拥有指纹录入和指纹认证两个用例。这两个用例又分别包含一些子用例。现进行详细描述。

指纹录入用例，是用户通过指纹机将自己的所有指纹录入进指纹认证系统后台数据库中，指纹录入是指纹认证的基础。该用例包含 2 个子用例：上传身份信息用例和上传指纹图片用例。在上传指纹图片之前，必须对指纹本身的用户信息先录入后台数据库，即上传身份信息，在上传身份信息之后，才可以开始正式的指纹图片录入。

指纹认证用例，是用户通过手机拍照验证自身指纹是否匹配的过程。该用例包含 4 个子用例：二维码生成用例、数字水印生成用例、来源审核用例、指纹对比用例。其中二维码生成、数字水印、来源审核都属于系统的安全模块，用于增强系统的安全性，防止会话劫持和重放攻击。二维码生成用例会根据请求的时间戳生成相应的二维码图像；数字水印用例会将二维码图像和原始指纹图片进行可逆水印计算生成数字水印；来源审核用例主要用来检查数字水印中包含的时间戳信息以判断请求来源的合法性；指纹对比用例主要用来比较用户指纹图片的匹配度，是指纹匹配的核心用例。

4 系统架构设计

本章探讨的主要问题是系统的架构设计。在确定了系统的解决方案和需求建模之后，必须根据相关的用例模型并结合特定的技术进行系统设计，系统设计与特点的技术选型息息相关。本章共分为 3 部分，第一部分依据系统用例分析结合特定的 API 函数分析系统业务逻辑，第二部分基于 REST Web Service 给出了系统架构设计整体方案和详细设计方案，第三部分在面向对象设计的基础上进行数据库设计。

4.1 系统业务逻辑

本部分讨论系统业务逻辑，首先对系统组件进行详细划分，然后对整个系统的基本业务逻辑进行阐述。

4.1.1 系统组件

系统组件包含以下四部分：

（1）客户端：包含 Android 智能手机、指纹数据采集子模块、指纹特征提取子模块、指纹加密子模块。

（2）服务器端：包含 Web 服务器、指纹解密子模块、用户认证子模块、指纹匹配子模块。

（3）第三方组件：包含指纹匹配 SDK，SourceAFIS SDK、NBIS SDK、Zxing.net、Nancy Framework 等组件。

（4）数据库：包含指纹图像数据库、用户数据库等。

4.1.2 工作流分析

系统基本业务流程可大致分为以下几个步骤：

（1）指纹图像采集：用户登录 android 客户端，android 客户端调用手机相机和图像采集模块获取用户指纹图像。

（2）指纹图像处理与特征提取：客户端调用图像处理模块对指纹图片进行预处理，在此基础上对用户指纹进行特征提取。

（3）图像加密：首先客户端接收来自服务器端生成的二维码图片，然后客户端调用加密模块对原始指纹图片和二维码图片进行数字水印加密。

（4）图片上传：客户端调用 Web API 将加密后的指纹图片上传至服务器。

（5）图像解密：服务器端调用解密模块对加密的指纹图片进行解密，解密

后服务器同时获得二维码图片和原始指纹图片。

（6）用户认证：服务器端调用用户认证子模块来检查二维码图片中的时间戳信息是否一致以确保用户来源的真实性。

（7）指纹匹配：服务器端调用指纹匹配 SDK 对用户上传的指纹图片和数据库中保存的指纹信息进行匹配和对比，得到相应的指纹匹配结果。

（8）结果显示：服务器将匹配结果返回给客户端，客户端根据匹配分数判断用户是否通过指纹匹配认证。指纹认证过程完成。

系统业务流程图如图 4.1 所示。

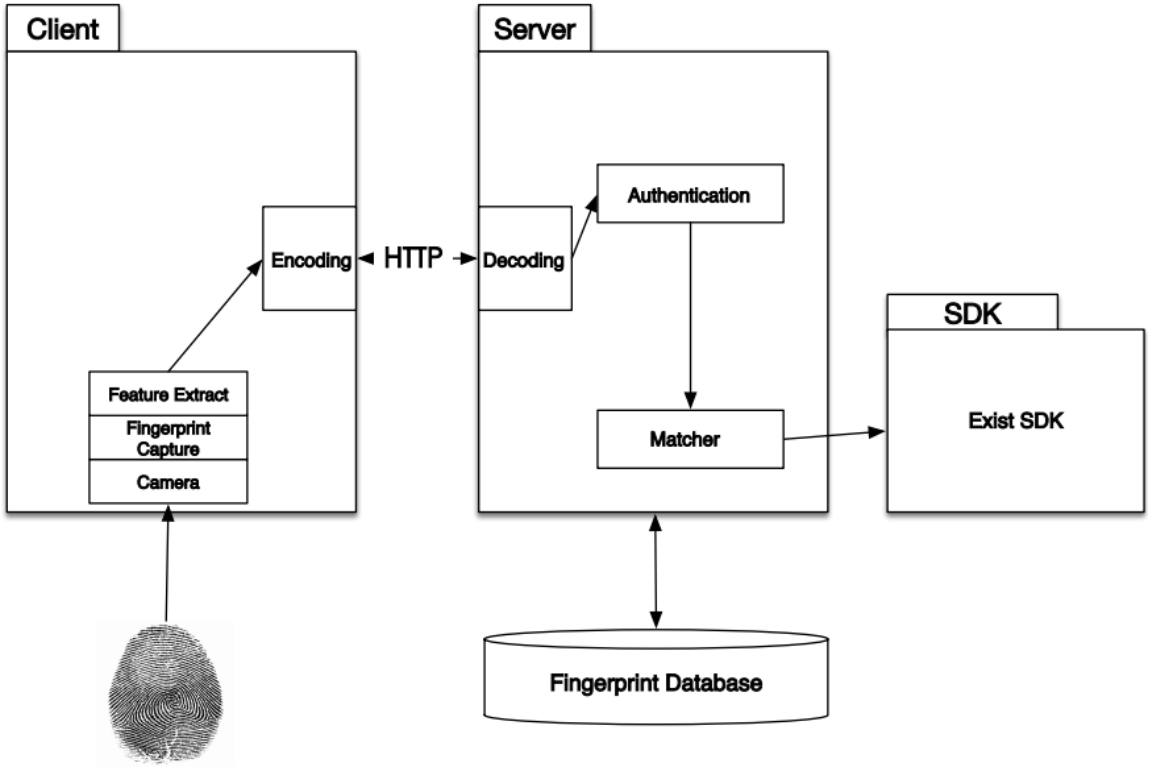

图 4.1 系统业务流程

4.2 基于 REST 的系统架构设计

本部分是系统设计部分，首先进行系统整体架构设计，然后介绍了系统类设计方案，最后分模块进行了模块详细设计。

4.2.1 整体架构设计

在介绍系统架构设计之前，首先介绍系统的技术选型。在不考虑第三方类库的平台支持情况下，由于指纹认证系统规模并不大，因此可以采用一些快速开发框架如 Ruby on Rails 或 Node.js 进行开发，并将服务器部署在 Linux 的 Nginx 服

务器上，但是由于系统中采用了 SourceAFIS 指纹识别开源库，并且该开源库仅提供了完善的.net 版本。因此本系统在技术选型过程中，底层指纹匹配服务只能基于.net 构建。最终采用了 windows 10 作为操作系统平台，IIS8 作为 Web 服务器，MySql 5.6 作为数据库服务器，.net Nancy Framework 作为服务端开发框架。使用 Nancy Framework 提供 Web API，顶层实现了 Android 端、iOS 端等客户端。

系统整体架构图如图 4.2 所示。

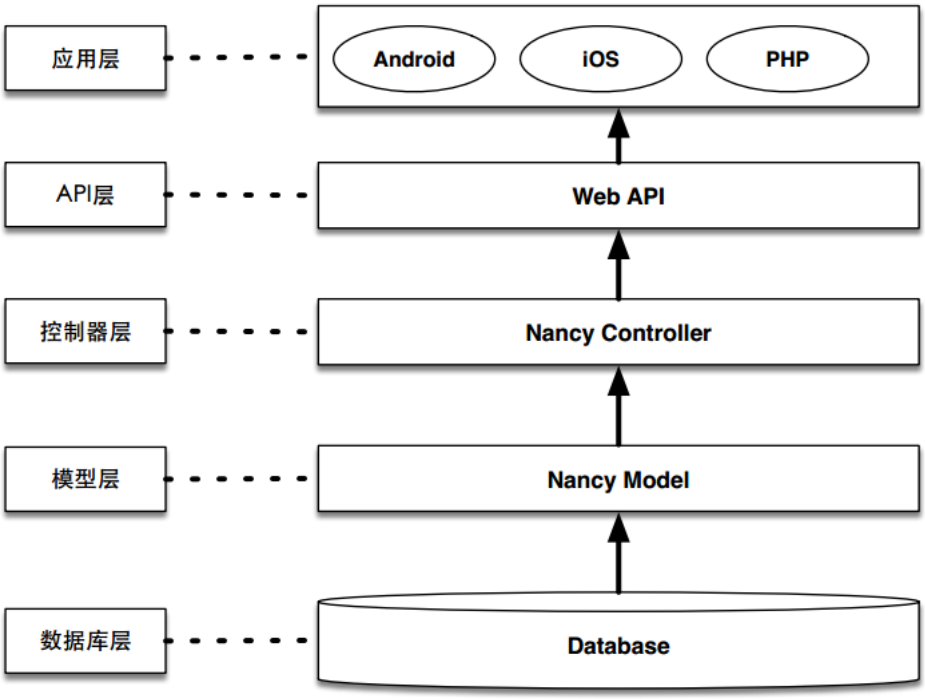

图 4.2 系统整体架构图

从整体架构图中可以看出，系统共分为 5 层。底层为数据库层，采用 MySQL 提供 DBMS 服务，其次为模型层，在 Nancy Framework 中建立相应的模型类，中间为控制器层，使用 Nancy Framework 中的控制器组件编写具体的系统业务逻辑，上层为 API 层，通过路由机制提供对应的 Web API 服务，向应用层提供所需的 JSON 数据，最顶层为应用层，包括但不限于 Android 端、iOS 端、Web 端等具体的应用服务，这些应用服务通过解析 Web API 中提供的 JSON 数据填充其视图。

4.2.2 系统类设计

在整体架构设计的基础上，在 Nancy 框架的基础上进行类设计。类设计的基本思路是：

（1）数据层与控制层分离，控制层与表现层分离，反映边界类和实体类的区别，反映实体类之间之间的继承与组合关系。

（2）通过设计接口实现面向接口的设计而非面向实现的设计。

（3）设计控制器类操作数据实体类

（4）实体类与实体类之间通过关联类建立联系。

系统类图整体设计如图 4.3 所示。

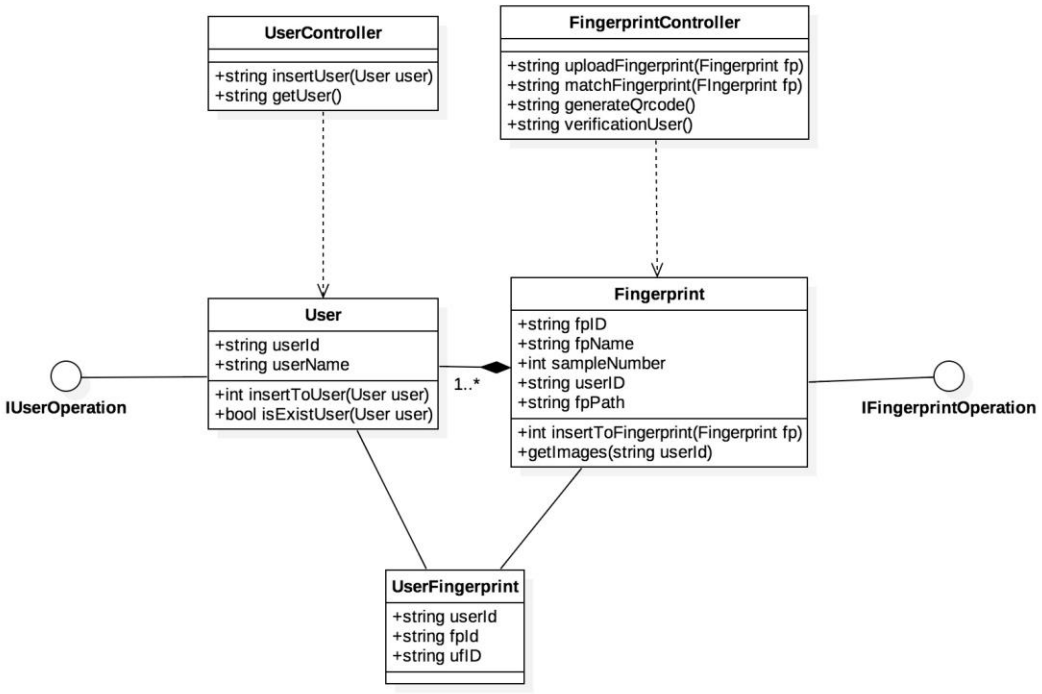

图 4.3 系统整体类设计

先对系统类设计进行详细说明。

首先定义实体类接口，实体类接口定义了实体类应具备哪些操作，在我们的指纹认证系统中，我们定义了 IUserOperation 接口和 IFingerprintOperation 接口。IUserOperation 接口定义了 User 实体类应该具有的操作，具体包含了 insertToUser 方法和 isExistUser 方法，分别用于添加用户和判断用户是否存在；IFingerprintOperation 接口定义了 insertToFingerprint 方法和 getImages 方法，分别用于添加指纹图片和获取所有指纹图片。

其次定义实体类，实体类反映了该系统实际操作的对象，在本系统中共定义了 User 类和 Fingerprint 类两个实体类。User 类定义了用户所具有的属性，包括 UserId 和 UserName 属性，并实现了 IUserOperation 接口，Fingerprint 类定义了指纹具有的属性，包括 fpId，fpName，sampleNumber，userId 和 fpPath 属性，并

实现了 IFingerprintOperation 接口。

接下来定义实体关联类，关联类反映了用户实体和指纹实体之间一对多的关系。共定义了一个关联类 UserFingerprint 类，共包含 3 个属性，ufID 属性，userId 属性和 fpId 属性。通过此类建立起 User 类和 Fingerprint 类之间的关系。

最后定义控制器类，我们主要定义了 UserController 和 FingerprintController 两个控制器类用于操作相应的数据实体类。两个控制器类分别以来对应的数据实体类。其中 UserController 类定义了 insertUser 和 getUser 两个方法，分别用于添加用户和获取所有用户；FingerprintController 类定义了 uploadFingerprint 方法、matchFingerprint 方法、generateQrcode 方法和 verificationUser 方法，分别用于用户上传指纹、用户匹配指纹、生成二维码和验证用户真实性。

在系统中，对应于数据实体类和控制器类，实际上还有视图类，但是由于视图类基本上都是负责显示逻辑的 HTML 代码，与系统业务逻辑关联并不大，因此并不刻意反映在系统整体类图中。

4.2.3 分模块详细设计

在整体类图的基础上，本部分对系统几个核心模块进行详细设计。在本系统中，我们认为用户指纹录入和用户指纹匹配两个模块最为重要和关键。

（1）用户指纹录入模块详细设计

用户指纹录入模块包含两个子模块：用户信息上传子模块和用户指纹上传子模块，其中用户信息上传子模块是对用户自身的信息进行录入，是用户上传指纹图片的基础和前提；在用户信息上传以后，用户才可以进行指纹信息的录入，用户指纹上传子模块是用户分指纹分样本上传指纹信息，用户指纹上传是进一步进行指纹匹配的前提。先分别从时序图角度对两个子模块进行描述。

用户信息上传子模块时序图如图 4.4 所示。

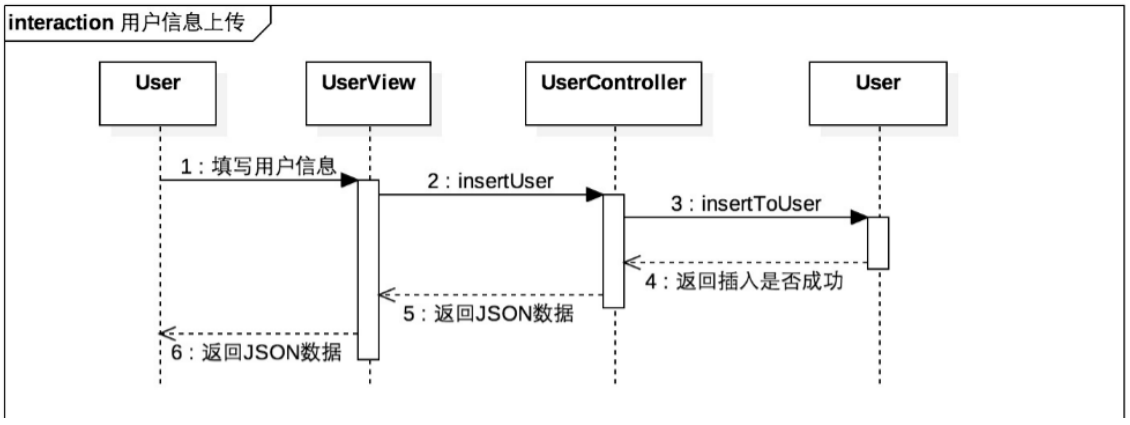

图 4.4 用户信息上传时序图

从该时序图可以看出，用户信息上传共涉及 4 个实体，首先 User 在 UserView 视图中填写用户信息，UserView 收到用户信息之后请求 UserController 类的 insertUser 方法，该方法进一步请求 User 类的 insertToUser 方法将用户信息实际写入数据库中，insertToUser 方法执行以后，返回用户插入是否成功给 UserController 类，控制器根据是否成功组织 JSON 数据，并把 JSON 数据返回给 View，视图最终将 JSON 数据返回给用户（客户端）。

用户指纹上传子模块时序图如图 4.5 所示。

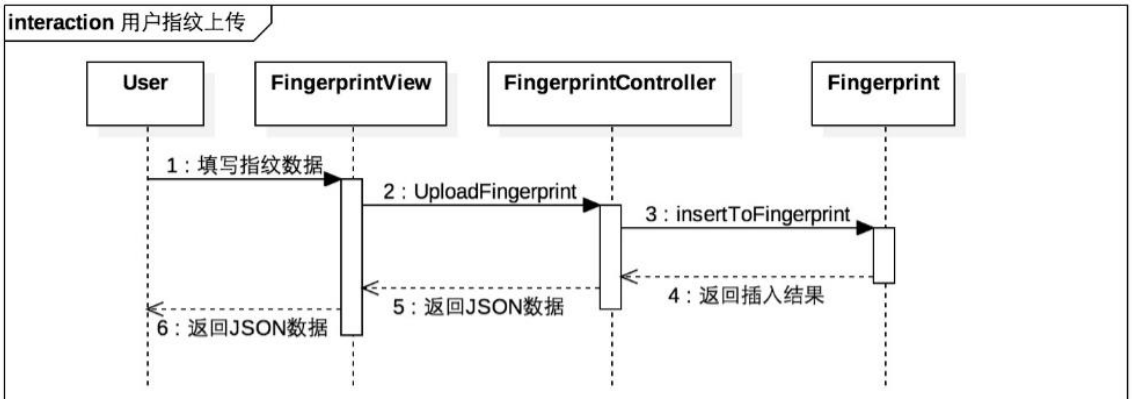

图 4.5 用户指纹上传时序图

从该时序图可以看出，用户指纹上传共涉及 4 个实体，首先 User 在 FingerprintView 视图中填写指纹数据，视图收到用户指纹数据之后请求 FingerprintController 类中的 UploadFingerprint 方法，该方法进一步请求 Fingerprint 类中的 insertToFingerprint 方法将用户指纹数据写入数据库中，insertToFingerprint 方法执行以后，返回指纹插入是否成功给 FingerprintController 类，控制器根据是否成功组织 JSON 数据，并把 JSON 数据返回给 View，视图最终将 JSON 数据返回给用户（客户端）。

（2）用户指纹匹配模块详细设计

用户指纹匹配模块是系统最核心和最关键的功能模块，整个匹配过程又可以细分为 3 个阶段：

第一阶段，获取二维码阶段。首先 User 向 FingerprintView 视图中发送获取二维码请求，视图收到用户请求后调用 FingerprintController 类中的 generateQRCode 方法生成二维码，该方法执行完成以后向视图层返回二维码图像信息，视图层显示二维码图像给用户。

第二阶段，用户验证阶段。客户端将二维码和用户指纹图片进行数字水印加密之后向 FingerprintView 视图提交匹配请求，视图层收到匹配请求以后调用 FingerprintController 中的 verificationUser 方法验证用户的真实性，并返回用户验证结果给 View 类。

第三阶段，指纹匹配阶段。在验证用户真实性以后，视图请求 FingerprintController 类的 matchFingerprint 方法，该方法进一步请求 Fingerprint 实体类的 getImages 方法获取该用户所有指纹图像，然后将用户指纹与数据库中所有该用户指纹进行对比和匹配，得到指纹对比结果，并将 JSON 结果返回给 View，最后视图把指纹匹配 JSON 结果返回给用户（客户端），完成了整个指纹匹配过程。

用户指纹匹配模块的时序图如图 4.6 所示，该时序图形象反映了以上 3 个阶段。

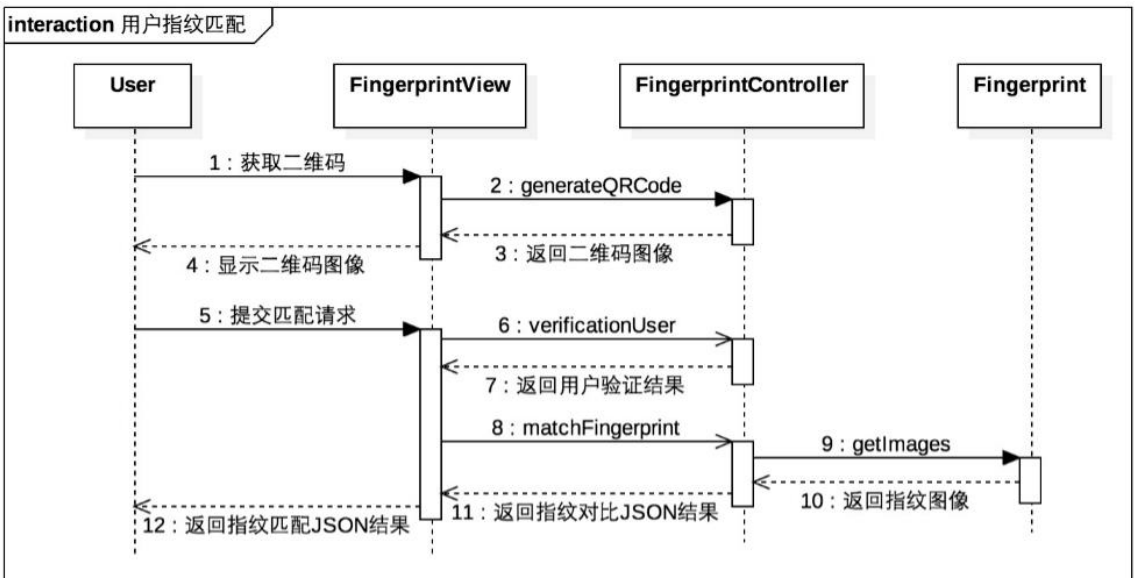

图 4.6 用户指纹匹配时序图

4.3 数据库设计

本部分依据系统整体设计方案进行数据库设计。由于本系统设计的数据库表比较少，因此进行了比较简要的设计。

4.3.1 E-R 图

首先设计数据库系统实体关系图，在指纹识别系统数据库中主要包含了 2 个表：用户信息表 t_user 和指纹信息表 t_fingerprint，由于每个用户可以有多个指纹数据，因此 user 表和 fingerpirnt 表形成一对多的关系。

在 t_user 表中，包含 userid，username 和 password 三个字段，其中以 userId 为主键；在 t_fingerprint 表中，包含 fpId，fpName，sampleNumber，fpPath 和 userId 五个字段，其中以 fpId 为主键，userId 为外键。通过 userId 建立一对多关联。

数据库系统 E-R 图设计如图 4.7 所示。

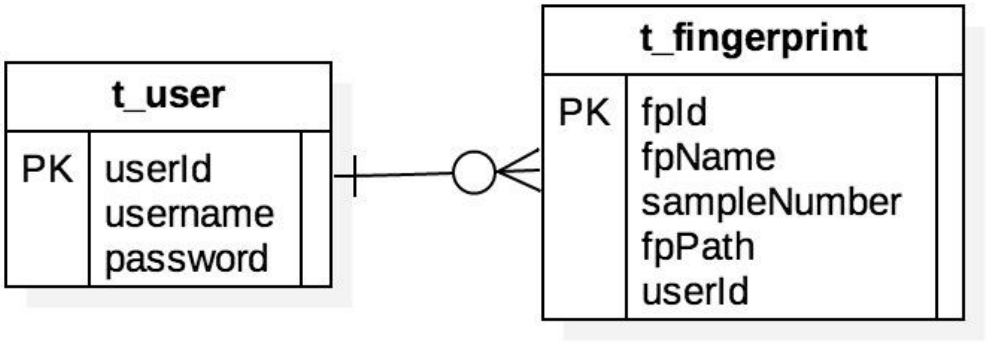

图 4.7 数据库系统 E-R 图

4.3.2 数据库详细设计

（1）t_user 表的设计

t_user 表是用户信息表，存储着用户的个人信息，主要字段的详细描述见如下表格：

字段名称	字段类型	主外键	字段描述
userId	varchar(20)	主键	用户 Id
userName	varchar(20)	否	用户名
userPassword	varchar(20)	否	用户密码

表 4.1 用户信息表

（2）t_fingerprint 表的设计

t_fingerprint 是用户指纹信息表,存储着注册用户的指纹信息，主要字段的详细描述见如下表格：

字段名称	字段类型	主外键	字段描述
fpId	varchar(45)	主键	指纹 Id
fpName	varchar(20)		指纹名称

sampleNumber	int(11)		样本编号
userId	varchar(20)		用户 Id
fpPath	varchar(80)		指纹路径

表 4.2 用户指纹信息表

5 系统实现

本章讨论基于数字水印的指纹认证系统实现，共包括五个部分的内容。第一部分，讨论系统实现的总体结构，借助 Nancy Framework 给出概要的实现框架；第二部分，讨论系统模型层的实现；第三部分，讨论系统视图层的实现；第四部分，讨论系统控制器层的实现；第五部分，讨论系统中关键算法——可逆数字水印算法的实现。

5.1 系统实现整体思路

系统服务器端采用了 Nancy Framework 提供 REST 风格的 Web Service，因此系统实现过程遵循 Nancy Framework 中的 MVC 模式。系统实现步骤如下：

（1）采用 Visual Studio 2013 社区版作为开发工具，首先安装在 Visual Studio 中安装 Nancy Framework 框架工具，建立相应的 Nancy 项目。

（2）添加系统依赖的第三方类库和工具，如 SourceAFIS、Zxing.net、MySQL 驱动程序等。

（3）系统模型层开发，依据系统设计的数据模型开发系统模型层。

（4）系统视图层开发，开发系统与用户交互的 UI 视图。

（5）系统控制器层开发，开发系统的核心业务逻辑，包括指纹上传、指纹匹配、指纹加密、二维码生成等。

系统实现的整体结构图如图 5.1 所示。从图 5.1 可以看出，系统内部主要包括模型层、视图层和控制器层 3 部分，引用了 Zxing、MySQL、Nancy、SourceAFIS 4 个外部库，底层数据库管理系统采用 MySQL.

系统整体设计中的关键问题在于如何区分模型、视图和控制器层，对于模型层应当设计哪些模型类与数据库对应；对于视图层，采用何种模板引擎并如何与后端进行绑定；对于控制器层，应当如何设计路由和访问方法。在本章后续部分，我们将分别从模型层、视图层、控制器层讨论系统如何实现。

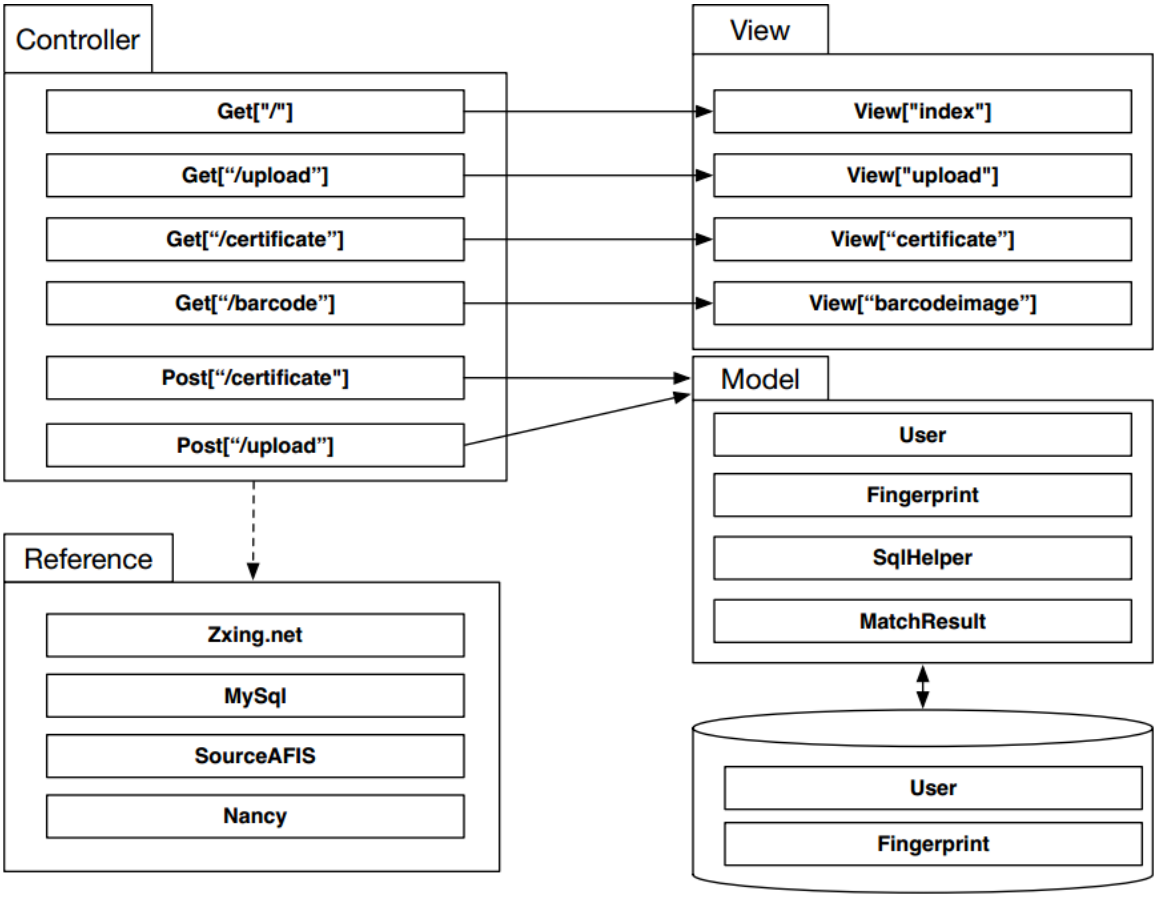

图 5.1 基于 MVC 的系统实现结构图

5.2 模型层实现

在模型层主要设计了 4 个模型类。其中 User 类和 Fingerprint 类是数据库中 User 表和 Fingerprint 表的实体类；MatchResult 封装了匹配结果对象，SqlHelper 类包含了对数据实体类的操作方法封装。现分别实现如下。

5.2.1 User 类实现

User 类实现如图 5.2 所示。

```
namespace FPWS.Model
{
    public class User
    {
        public string userId;
        public string userName;
        public string passWord;
    }
}
```

图 5.2 User 类实现

在 User 类中，定义了 userId、userName、passWord 3 个属性与数据库 t_user 表相对应。

5.2.2 Fingerprint 类实现

Fingerprint 类实现如图 5.3 所示。

```
namespace FPWS.Model
{
    class Fingerprint
    {
        public string fpID;
        public string fpName;
        public int sampleNumber;
        public string userID;
        public string fpPath;
    }
}
```

图 5.3 Fingerprint 类实现

在 Fingerprint 类中，定义了 fpID、fpName、sampleNumber、userId、fpPath 5 个属性与数据库 t_fingerprint 表相对应。

5.2.3 MatchResult 类实现

MatchResult 类实现如图 5.4 所示。

```
namespace FPWS.Model
{
    class MatchResult
    {
        public string fingerprint;
        public float score;
    }
}
```

图 5.4 MatchResult 类实现

在 MatchResult 类中定义了 fingerprint 属性和 score 属性，分别用来表示指纹的匹配分数。在匹配操作中，会实例化该类并进行赋值。

5.2.4 SqlHelper 类实现

SqlHelper 类共实现了 4 个方法：添加用户信息、添加指纹图片信息、判断用户是否存在以及获取指纹图像信息。现分别实现如下。

（1）insertToUser 方法

该方法用于实现添加用户，首先连接数据库，然后将传入的参数写入相应的SQL 语句当中，利用 SQL 语句将用户插入数据库。insertToUser 方法的实现如图5.5 所示。

```
public static int insertToUser(User user)
{
    int rows;
    using (MySqlConnection myconnection = new MySqlConnection(myconn))
    {
        string commandText = "insert into t_user values (@userId,@userName)";
        MySqlCommand command = new MySqlCommand(commandText,myconnection);
        command.Parameters.AddWithValue("@userId", user.userId);
        command.Parameters.AddWithValue("@userName", user.userName);
        myconnection.Open();
        rows = command.ExecuteNonQuery();
    }
    return rows;
}
```

图 5.5 添加用户方法

（2）isExistUser 方法

该方法用于判断用户是否存在。首先连接数据库，然后将用户名作为 SQL 语句中的参数进行查询，根据数据库查询结果来判断用户是否存在，并返回 bool 值。isExistUser 方法的实现如图 5.6 所示。

```
public static bool isExistUser(User user)
{
    int num;
    using(MySqlConnection myconnection = new MySqlConnection(myconn))
    {
        string commandText = "select count(*) from t_user where userId= @userId";
        MySqlCommand command = new MySqlCommand(commandText, myconnection);
        command.Parameters.AddWithValue("@userId", user.userId);
        myconnection.Open();
        num = Convert.ToInt32(command.ExecuteScalar());
    }
    if (num != 0) return true;
    else return false;
}
```

图 5.6 判断用户是否存在方法

（3）insertToFingerprint 方法

该方法用于插入用户指纹信息。首先连接数据库，然后将用户指纹信息作为SQL 参数执行 SQL 语句，通过执行 SQL 语句将指纹信息插入数据库。

insertToFingerprint 的实现如图 5.7 所示。

```
public static int insertToFingerprint(Fingerprint fp)
{
    int rows;
    using (MySqlConnection myconnection = new MySqlConnection(myconn))
    {
        string commandText = "insert into t_fingerprint values (@fpID,@fpName,@sampleNumber,@userI
        MySqlCommand command = new MySqlCommand(commandText, myconnection);
        command.Parameters.AddWithValue("@fpID", fp.fpID);
        command.Parameters.AddWithValue("@fpName", fp.fpName);
        command.Parameters.AddWithValue("@sampleNumber", fp.sampleNumber);
        command.Parameters.AddWithValue("@userID", fp.userID);
        command.Parameters.AddWithValue("@fpPath", fp.fpPath);
        myconnection.Open();
        rows = command.ExecuteNonQuery();
    }
    return rows;
}
```

图 5.7 添加指纹方法

（4）获取指纹图片方法

该方法根据 userId 查询数据库，获取当前用户的所有指纹数据。然后通过循环读取 SQL 结果，将所有的指纹数据信息存储到 List 容器当中。

getImages 方法的实现如图 5.8 所示。

```
public static List<Model.Fingerprint> getImages(string userId)
{
    List<Model.Fingerprint> images = new List<Model.Fingerprint>();
    using(MySqlConnection myconnection = new MySqlConnection(myconn))
    {
        string commandText = "select * from t_fingerprint where userId=@userId";
        MySqlCommand command = new MySqlCommand(commandText, myconnection);
        command.Parameters.AddWithValue("@userId", userId);
        myconnection.Open();
        using(MySqlDataReader reader = command.ExecuteReader())
        {
            while(reader.Read())
            {
                Model.Fingerprint fp = new Model.Fingerprint();
                fp.fpID = reader.GetString(0);
                fp.fpName = reader.GetString(1);
                fp.sampleNumber = reader.GetInt32(2);
                fp.userID = reader.GetString(3);
                fp.fpPath = reader.GetString(4);
                images.Add(fp);
            }
        }
    }
    return images;
}
```

图 5.8 获取指纹图片方法

5.3 视图层实现

视图层主要用来负责与用户进行交互，由于指纹认证系统服务器端主要用于提供 Web Service，因此视图层非常简单。大体上有 3 个主要视图页面。

5.3.1 Index 页面的实现

Index 页面主要负责指纹认证系统功能导航，通过 Index 页面用户可以获取指纹认证系统所有功能，包括指纹上传、获取二维码、指纹匹配等页面。

Index 页面的实现如图 5.9 所示。

```
<body>
    <h1>Fingerprint Certification System</h1>
    <img src="~/Content/logo.jpg" alt="FP logo" /><br />
    <p>
        <ul>
            <li><b><a href="upload">Upload Fingerprint</a></b></li>
            <li><b><a href="certificate">Fingerprint Certification</a></b></li>
            <li><b><a href="barcode">Generate QR-Code</a></b></li>
            <li><b><a href="securityUpload">Upload Fingerprint with barcode</a></b></li>
            <li><b><a href="result">Get the result</a></b></li>
        </ul>
    </p>
</body>
```

图 5.9 Index 页面实现

Index 页面效果如图 5.10 所示。

- Upload Fingerprint
- Fingerprint Certification
- Generate QR-Code
- Upload Fingerprint with barcode
- Get the result

图 5.10 Index 页面展示

5.3.2 指纹上传页面的实现

指纹上传页面主要用于上传用户名及其指纹图片信息，包括用户指纹图片、指纹名称、指纹样本编号等基本信息，该页面提交以后通过 POST 方法找到相应的控制器处理。

指纹上传页面的实现如图 5.11 所示。页面展示效果如图 5.12 所示。

```
<form method="POST" enctype="multipart/form-data" action="/upload">
    <ul>Student ID:<input type="text" name="id"></ul>
    <ul>Student Name:<input type="text" name="name"></ul>
    <ul>
        Choose File 1:
        <input type="file" name="file1">
        FingerName:
        <input type="text" name="fpname1">
        SampleNumber:
        <input type="text" name="samplenumber1">
    </ul>
    <ul>
        Choose File 2:
        <input type="file" name="file2">
        FingerName:
        <input type="text" name="fpname2">
        SampleNumber:
        <input type="text" name="samplenumber2">
    </ul>
```

图 5.11 指纹上传页面实现

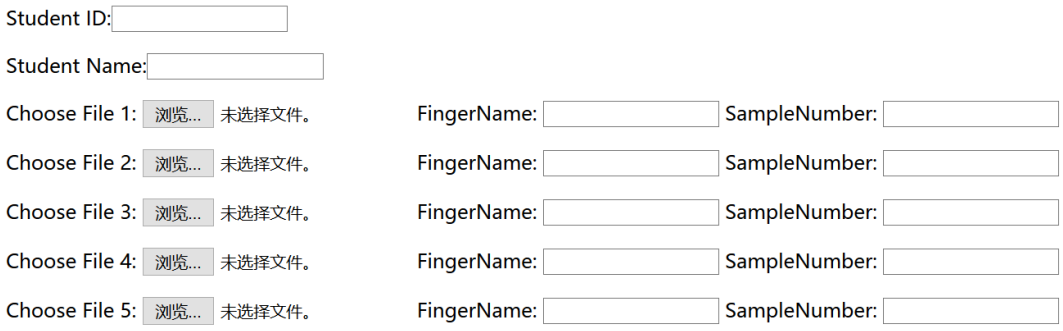

图 5.12 指纹上传页面展示

5.3.3 指纹认证页面的实现

在指纹认证页面中，用户提交用户 ID 和指纹图片，然后系统进行指纹匹配，返回匹配结果给用户。指纹认证页面的实现如图 5.13 所示。

```
<p>
    Fingerprint Certification
</p>
<p>
    <form method="post" action="/verification" enctype="multipart/form-data">
        <ul>
            <li>UserId:<input type="text" name="userId" /></li>
            <br />
            <li>FP Image:<input type="file" name="finger" /></li>
            <br />
            <li><input type="submit" value="submit"/></li>
        </ul>
    </form>
</p>
```

图 5.13 指纹认证页面的实现

指纹认证页面效果如图 5.14 所示。

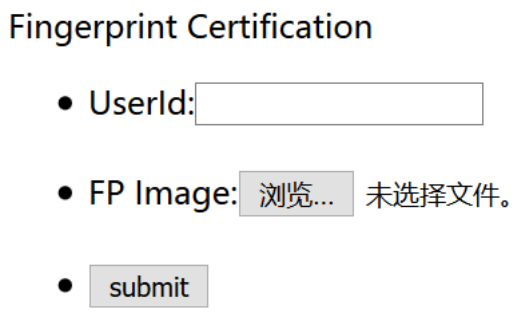

图 5.14 指纹认证页面效果

5.4 控制器层实现

控制器层是 REST 风格 Web Service 的核心。包含了系统核心业务逻辑，主要实现了指纹图像上传、指纹匹配、二维码生成、用户认证等一系列功能。

5.4.1 指纹上传的实现

指纹上传分为两个过程。首先控制器从视图层获取用户指纹的图片、指纹样本编号、指纹名称等信息，并在控制器生成指纹路径等信息，最终将这些指纹信息存储在指纹类中。这一过程的实现代码如图 5.15 所示。

```
Post["/upload"] = parameters =>
{
    User user = new User();
    Model.Fingerprint fp1 = new Model.Fingerprint();
    Model.Fingerprint fp2 = new Model.Fingerprint();
    Model.Fingerprint fp3 = new Model.Fingerprint();
    Model.Fingerprint fp4 = new Model.Fingerprint();
    Model.Fingerprint fp5 = new Model.Fingerprint();

    user.userId = (string)this.Request.Form.id;
    user.userName = (string)this.Request.Form.name;

    fp1.fpName = (string)this.Request.Form.fpname1;
    fp2.fpName = (string)this.Request.Form.fpname2;
    fp3.fpName = (string)this.Request.Form.fpname3;
    fp4.fpName = (string)this.Request.Form.fpname4;
    fp5.fpName = (string)this.Request.Form.fpname5;
```

图 5.15 从视图层获取指纹数据

控制器获取指纹数据之后，将这些数据组合成对应指纹类，然后通过 Model 层将指纹信息写入数据库，将指纹图片保存在服务器制定文件位置。并将上传结果以 JSON 形式返回给客户端。这一过程的实现代码如图 5.16 所示。

```
    if (!SqlHelper.isExistUser(user)) SqlHelper.insertToUser(user);
    int i1 = SqlHelper.insertToFingerprint(fp1);
    int i2 = SqlHelper.insertToFingerprint(fp2);
    int i3 = SqlHelper.insertToFingerprint(fp3);
    int i4 = SqlHelper.insertToFingerprint(fp4);
    int i5 = SqlHelper.insertToFingerprint(fp5);
    if (i1!=0 && i2!=0 && i3!=0 && i4!=0 && i5!=0)
        return Response.AsJson(new { Result = "Insert Sucess" });
    else
        return Response.AsJson(new { Result = "Insert Failed" });
};
```

图 5.16 将指纹信息上传至服务器

5.4.2 指纹匹配的实现

指纹匹配是服务器端的核心模块之一。指纹匹配的过程比较复杂，如果我们需要比较指纹图片 P1 和指纹图片 P2 的匹配度，首先需要分别提取图像 P1 和 P2 的特征点，然后匹配引擎会调用匹配方法比较 P1 和 P2 的特征点，如果二者特征点有相似之处，则可能产生匹配分数，否则匹配分数为 0.

那么如何理解指纹图片的特征点呢？图 5.17 说明了指纹图像中的两类特征点。在图中，圆形标示部分为指纹中的一类特征点——终止点；而方形标示部分为指纹中的另一类特征点——分叉点。在指纹匹配算法中，大多是根据这两类特征点的位置和相似度来进行指纹匹配的。

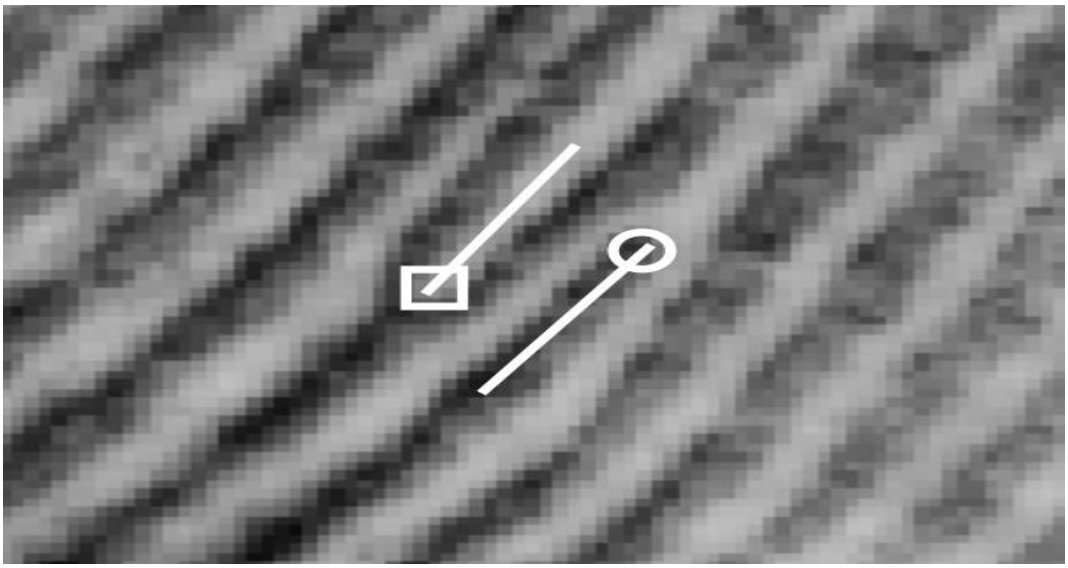

图 5.17 指纹特征点

在本系统中，通过调用 SourceAFIS 开源库进行指纹匹配，使用 SourceAFIS 进行指纹匹配的基本模式如图 5.18 所示。

```
static AfisEngine Afis = new AfisEngine();
Fingerprint fp1 = new Fingerprint();
Person person1 = new Person();
person1.Fingerprints.Add(fp1);

Fingerprint fp2 = new Fingerprint();
Person person2 = new Person();
person2.Fingerprints.Add(fp2);

Afis.Extract(person1);
Afis.Extract(person2);
score = Afis.Verify(person1, person2);
```

图 5.18 SourceAFIS 基本流程

使用 SourceAFIS 的基本步骤为：

（1）实体化 AFIS 匹配引擎；

（2）通过指纹图片实例化两个指纹对象；

（3）将两个指纹对象存入 Person 容器中；

（4）采用匹配引擎中的 Extract 方法对指纹进行特征提取；

（5）采用匹配引擎中的 Verify 方法对指纹进行匹配并获取匹配分数；

（6）客户端根据匹配分数来判断指纹认证是否通过，一般而言，如果两个指纹互相匹配，那么匹配分数应该大于 0，否则匹配分数一般为 0.

在我们的指纹认证系统中，实现匹配算法基本上采用同样的原理实现，对于每一个需要比较的用户指纹，我们首先从数据库中查询当前用户的所有指纹信息，然后将用户当前指纹与数据库中指纹信息进行逐一对比与匹配，并得到该指纹与数据库中所有指纹的匹配分数，最终以 JSON 形式返回给客户端。这一过程的实现如图 5.19 所示。

```
SourceAFIS.Simple.Fingerprint fp1 = new SourceAFIS.Simple.Fingerprint();
fp1.AsBitmap = new Bitmap(Bitmap.FromFile(uploadPath));
Person person1 = new Person();
person1.Fingerprints.Add(fp1);
Afis.Extract(person1);

List<MatchResult> results = new List<MatchResult>();

foreach (var fp in fingerprints)
{
    SourceAFIS.Simple.Fingerprint fp2 = new SourceAFIS.Simple.Fingerprint();
    fp2.AsBitmap = new Bitmap(Bitmap.FromFile(fp.fpPath));
    Person person2 = new Person();
    person2.Fingerprints.Add(fp2);
    Afis.Extract(person2);


    MatchResult result = new MatchResult();
    result.fingerprint = fp.fpName + fp.sampleNumber.ToString();
    result.score = Afis.Verify(person1, person2);

    results.Add(result);
}

return Response.AsJson<List<MatchResult>>(results);
```

图 5.19 指纹匹配的实现

图 5.20 展示了指纹匹配之后服务器返回的 JSON 结果数据。


```
[
  ▾ {
        fingerprint: "LIndex1",
        score: 54.00148
    },
  ▾ {
        fingerprint: "LIndex2",
        score: 56.52282
    },
  ▾ {
        fingerprint: "LIndex3",
        score: 41.92099
    },
  ▾ {
        fingerprint: "LIndex4",
        score: 52.7956963
    },
  ▾ {
        fingerprint: "LIndex5",
        score: 42.4737053
    }
]
```


图 5.20 指纹匹配结果数据

5.4.3 生成二维码的实现

二维码生成采用 Zxing.net 开源库实现。通过 Zxing，程序员只需要选定二维码的编码标准、宽度和长度以及其他参数信息就可以生成相应的二维码图片，在本系统中使用 DateTime.Now.ToString()方法生成时间戳消息，然后把它作为参数嵌入到二维码当中。二维码生成的实现如图 5.21 所示。

```
Get["/barcode"] = parameters =>
{
    BarcodeWriter writer = new BarcodeWriter();
    writer.Format = BarcodeFormat.QR_CODE;
    writer.Options = new QrCodeEncodingOptions
    {
        DisableECI = true,
        CharacterSet = "UTF-8",
        Width = 1840,
        Height = 1840,
        ErrorCorrection = ErrorCorrectionLevel.H
    };
    requestTime = DateTime.Now.ToString();
    Bitmap qrImage = writer.Write(requestTime);

    string path = Path.Combine("data", "qrdata");
    if (!Directory.Exists(path)) Directory.CreateDirectory(path);
    string imagePath = path + @"\" + "qrImage.bmp";
    qrImage.Save(imagePath);

    return Response.AsImage(imagePath);
};
```

图 5.21 二维码生成的实现

生成的二维码图片如图 5.22 所示。

图 5.22 二维码图片效果

5.4.4 用户认证的实现

用户认证本质上是二维码的解码过程，首先仍然采用 Zxing 库对接收的二维码图片进行解码，解码之后可以得到一个字符串数据，通过比较该字符串数据与之前的时间戳信息是否相等来进行用户认证，确保用户的真实性。用户认证的实现如图 5.23 所示。

```
string rawFPImage = Path.Combine("data", "qrdata") + @"\" + "fingerprint.bmp";
bmpCrop.Save(rawFPImage);

LuminanceSource source = new BitmapLuminanceSource(QRcodeImage);
BinaryBitmap newbitmap = new BinaryBitmap(new HybridBinarizer(source));
Result result = new MultiFormatReader().decodeWithState(newbitmap);

if (result.Text != requestTime)
{
    return Response.AsJson(new { Result = "Authentication failure" });
}
```

图 5.23 用户认证的实现

5.5 系统关键算法

本部分讨论指纹认证系统的关键算法——可逆水印加密算法。正如我们在需求和设计部分描述的那样，我们采用可逆数字水印技术来整合二维码图片和指纹图片。本部分讨论的重点是可逆水印算法的具体实现，在我们的系统中，可逆水印算法是基于最低有效位算法实现的。

5.5.1 图像的矩阵表示

首先假设二维码图像的表示矩阵为 Q：

$$Q = \begin{bmatrix} 255 & 254 & 255 & 1 & 2 & \cdots \\ 255 & 255 & 255 & 1 & 1 & \cdots \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \end{bmatrix}$$

它的二进制表示形式为：

$$Q = \begin{bmatrix} 11111111 & 11111110 & 11111111 & 00000001 & \cdots \\ 11111111 & 11111111 & 11111111 & 00000001 & \cdots \\ \cdots & \cdots & \cdots & \cdots & \cdots \end{bmatrix}$$

接着假设指纹图像的表示矩阵为 M：

$$M = \begin{bmatrix} 167 & 63 & 15 & \cdots \\ 255 & 127 & 128 & \cdots \\ \cdots & \cdots & \cdots & \cdots \end{bmatrix}$$

它的二进制形式为：

$$M = \begin{bmatrix} 10100111 & 00111111 & 00001111 & \cdots \\ 11111111 & 01111111 & 10000000 & \cdots \\ \cdots & \cdots & \cdots & \cdots \end{bmatrix}$$

现在我们把指纹图像进行拆分，使得指纹图像可以作为数字水印嵌入到二维码图像矩阵当中。让我们首先关注指纹图像。

5.5.2 指纹图像矩阵

首先，对于指纹图像矩阵 M，我们可以看到 M(1,1)=167，其二进制形式为 M(1,1) = 10100111. 从二进制形式出发，我们可以将其分为四部分。

第一部分：二进制 1-2 位；

第二部分：二进制 3-4 位；

第三部分：二进制 5-6 位；

第四部分：二进制 7-8 位；

基于以上划分，我们可以将矩阵 M 分成 4 个子矩阵：

1. 由第一部分构成的矩阵 M1：

$$M1 = \begin{bmatrix} 10 & 00 & 00 & \cdots \\ 11 & 01 & 10 & \cdots \\ \cdots & \cdots & \cdots & \cdots \end{bmatrix} = \begin{bmatrix} 2 & 0 & 0 & \cdots \\ 3 & 1 & 2 & \cdots \\ \cdots & \cdots & \cdots & \cdots \end{bmatrix}$$

2. 由第二部分构成的矩阵 M2：

$$M2 = \begin{bmatrix} 10 & 11 & 00 & \cdots \\ 11 & 11 & 00 & \cdots \\ \cdots & \cdots & \cdots & \cdots \end{bmatrix} = \begin{bmatrix} 2 & 3 & 0 & \cdots \\ 3 & 3 & 0 & \cdots \\ \cdots & \cdots & \cdots & \cdots \end{bmatrix}$$

3. 由第三部分构成的矩阵 M3：

$$M3=\begin{bmatrix} 01 & 11 & 11 & \cdots \\ 11 & 01 & 00 & \cdots \\ \cdots & \cdots & \cdots & \cdots \end{bmatrix}=\begin{bmatrix} 1 & 3 & 3 & \cdots \\ 3 & 1 & 0 & \cdots \\ \cdots & \cdots & \cdots & \cdots \end{bmatrix}$$

4. 由第四部分构成的矩阵 M4：

$$M4=\begin{bmatrix} 11 & 11 & 11 & \cdots \\ 11 & 11 & 00 & \cdots \\ \cdots & \cdots & \cdots & \cdots \end{bmatrix}=\begin{bmatrix} 3 & 3 & 3 & \cdots \\ 3 & 3 & 0 & \cdots \\ \cdots & \cdots & \cdots & \cdots \end{bmatrix}$$

5.5.3 数字水印加密

由于二维码图片要比指纹图片要大得多，二维码图像对应的图像矩阵其行数是指纹图像矩阵的 3 倍，列数是指纹图像矩阵的 5 倍多。正因为如此，我们才可以把指纹图像作为数字水印嵌入到二维码图像中，这是至关重要的。在我们的算法中，指纹图像矩阵 M 是一个 392*357 的矩阵，二维码图像矩阵 Q 是一个 900*5000 的矩阵。此外，二维码图像和指纹图像均为灰度图。

我们使用最低有效位算法进行数字水印加密。首先，我们把二维码图像矩阵 Q 划分为以下 5 个部分。

$$Q=\begin{bmatrix} A & B & \\ C & D & \\ & & E \end{bmatrix}$$

对于子矩阵 A、B、C、D，均为 392*357 的矩阵，矩阵 E 是矩阵 Q 的其它部分，这时我们就可以使用 A、B、C、D 四个矩阵进行 M1、M2、M3、M4 矩阵的加密。我们将矩阵 A、B、C、D 中每个二进制数字的最低两位置为 0.举例而言，

对于矩阵 A（二进制表示）：

$$A=\begin{bmatrix} 11111111 & 11111110 & 11111111 & 00000001 & \cdots \\ 11111111 & 11111111 & 11111111 & 00000001 & \cdots \\ \cdots & \cdots & \cdots & \cdots & \cdots \end{bmatrix}$$

将其后两位置为 0 之后的矩阵 A1：

$$A1=\begin{bmatrix} 11111100 & 11111100 & 11111100 & 00000000 & \cdots \\ 11111100 & 11111100 & 11111100 & 00000000 & \cdots \\ \cdots & \cdots & \cdots & \cdots & \cdots \end{bmatrix}$$

这时我们用矩阵 M1 代替矩阵 A 每个数字的最低两位，即：

$$A2=A1+M1$$

对于矩阵 B，C，D 和 M2，M3，M4，我们也执行以下同样操作：

$$B2=B1+M2$$

$$C2=C1+M3$$

$$D2=D1+M4$$

这样，我们就得到了一个新的矩阵，称为 Q2：

$$Q2=\begin{bmatrix} A2 & B2 & \\ C2 & D2 & \\ & & E \end{bmatrix}$$

矩阵 Q2 包含了所有的指纹图片信息，实现了指纹图片作为数字水印进行加密。

5.5.4 数字水印解密

在了解数字水印加密过程后，实现其解密过程就相对容易了。从上述矩阵 Q2 中，我们抽取出 A2，B2，C2，D2 四个子矩阵，我们可以通过 mod([A2,B2,C2,D2],4)运算得到 M1，M2，M3，M4 矩阵。即：

```
M1(i,j) = A2(i,j) mod 4
M2(i,j) = B2(i,j) mod 4
M3(i,j) = C2(i,j) mod 4
M4(i,j) = D2(i,j) mod 4
```

通过 M1，M2，M3，M4，我们可以轻松地还原出加密前的指纹图片矩阵：

$$M(i,j) = M4(i,j) + 4 \times M3(i,j) + 16 \times M2(i,j) + 64 \times M1(i,j)$$

由此，矩阵 M 即为加密前的指纹图片矩阵。

5.5.5 图像加密算法

正如论文第三章讨论的那样，我们仍然需要使用用户密码对上述图片进行加密。使用最低有效位算法，我们可以拆分指纹图片矩阵的最低有效位用以存储密钥信息。我们可以把指纹图片矩阵的最低 2 位/4 位/6 位置为 0，然后利用这些最低位存储用户密钥。如果我们使用该种算法，用户指纹图片的低位信息就会丢失，这就会造成指纹图片的失真，所以我们必须控制好最低有效位的位数并挑选合适的位数以保证指纹图片与原始的指纹图片变化不大。针对这一场景，我们需要选取不同的位数来进行试验和仿真。在下一章的实验部分，我们会对仿真试验进行讨论。

6 实验结果

系统开发完成以后，应该对系统进行系统测试和优化。但是由于本系统是是一个实验性的指纹认证系统，对系统进行仿真实验或许更为有效。本章主要讨论基于 Matlab 的仿真实验，在实验之后对系统成本、可用性和安全性进行分析和验证。

6.1 基于 Matlab 的仿真实验

正如第五章所描述的那样，本部分主要使用 Matlab 工具对图像加密和解密过程进行模拟，通过模拟可以控制指纹图片加密时的最低有效位，分别取得最低有效位为 0 位、2 位、6 位时的指纹图片，最后分析了不同最低有效位的指纹图片匹配结果。

6.1.1 加密过程

对指纹图像加密过程的 Matlab 仿真程序如图 6.1 所示。该仿真程序的实现与论文 5.5 部分讨论的加密算法基本一致，对原始指纹图片和二维码图片应用该加密算法以后，指纹图片被加入到二维码图片的最低有效位中。

```
for i=1:m
  for j=1:n
    orimod(i,j)=mod(qrImage(i,j,1),4);
    orimod(i+m,j) = mod(qrImage(i+m,j,1),4);
    orimod(i,j+n) = mod(qrImage(i,j+n,1),4);
    orimod(i+m,j+n) = mod(qrImage(i+m,j+n,1),4);

    qrImage1(i,j,1) = qrImage(i,j,1)-orimod(i,j);
    qrImage1(i+m,j,1) = qrImage(i+m,j,1)-orimod(i+m,j);
    qrImage1(i,j+n,1) = qrImage(i,j+n,1)-orimod(i,j+n);
    qrImage1(i+m,j+n,1) = qrImage(i+m,j+n,1)-orimod(i+m,j+n);


    multiwatermarkImage(i,j) = m1(i,j);
    multiwatermarkImage(i+m,j) = m2(i,j);
    multiwatermarkImage(i,j+n) = m3(i,j);
    multiwatermarkImage(i+m,j+n) = m4(i,j);

    qrImage2(i,j,1) = qrImage1(i,j,1) + multiwatermarkImage(i,j);
    qrImage2(i+m,j,1) = qrImage1(i+m,j,1) + multiwatermarkImage(i+m,j);
    qrImage2(i,j+n,1) = qrImage1(i,j+n,1) + multiwatermarkImage(i,j+n);
    qrImage2(i+m,j+n,1) = qrImage1(i+m,j+n,1) + multiwatermarkImage(i+m,j+n);
  end
end
```

图 6.1 加密过程仿真程序

在加密算法中，可以对指纹图片的最低位进行设置。设置算法如图 6.2 所示。

```
if bits == 0
  k1 = fix(watermarkImage(i,j));
  k2 = fix(k1/4);
  k3 = fix(k2/4);
  k4 = fix(k3/4);

  x1 = mod(k1,4);
  x2 = mod(k2,4);
  x3 = mod(k3,4);
  x4 = mod(k4,4);
end

if bits == 1
end

if bits == 2
end
```

图 6.2 指纹图片的最低有效位设置

如果设置指纹图片最低有效位为 0，那么指纹图片没有丢失任何信息；如果设置为 2，指纹图片会产生失真，丢失了最低两位的信息。同样的，如果将最低有效位设置为 6，那么指纹图片就会丢失更多信息，如果将最低有效位设置为 8，那么就意味着将图像的 8 位表示全部设置为 0，此时图片就变成了纯黑图片，原始的图片信息就完全失去了。在实验结果部分，会对不同失真状态下的指纹匹配分数进行统计分析。

6.1.2 解密过程

对指纹图片的解密过程同样与论文 5.5 部分所阐述的解密算法相似。对指纹图片解密的 Matlab 仿真实现如图 6.3 所示。

```
function y = reverse(QrImagewithwatermarkImage,watermarkImage)
  [m,n] = size(watermarkImage);
  newwatermark=zeros(m,n);
  newwatermark2 = zeros(m,n);

  for i=1:m
    for j=1:n
      x1 = mod(QrImagewithwatermarkImage(i,j,1),4);
      x2 = mod(QrImagewithwatermarkImage(i+m,j,1),4);
      x3 = mod(QrImagewithwatermarkImage(i,j+n,1),4);
      x4 = mod(QrImagewithwatermarkImage(i+m,j+n,1),4);

      newwatermark(i,j) = x1 + x2*4 + x3*16 + x4*64;
    end
  end
  newwatermark2 = uint8(newwatermark);
  y = newwatermark2;
```

图 6.3 解密过程仿真程序

6.1.3 实验结果

图 6.4 展示了指纹图片在不同失真条件下的指纹匹配平均结果。从实验结果中可以看出，当设置图片最低有效位为 6 位时，由于图片有较大失真，导致图片在大多数样本条件下匹配分数低于 0 位失真和 2 位失真的情况。当图片最低有效位设置为 2 时，由于其失真程序较小，匹配分数与无失真情况比较接近。

从上述实验结果分析，我们认为选取 2 位的最低有效位存储密钥信息，既可以保证良好的图像加密，由可以保证还原出来的指纹图片失真较小，对正常的指纹匹配不构成大的影响。

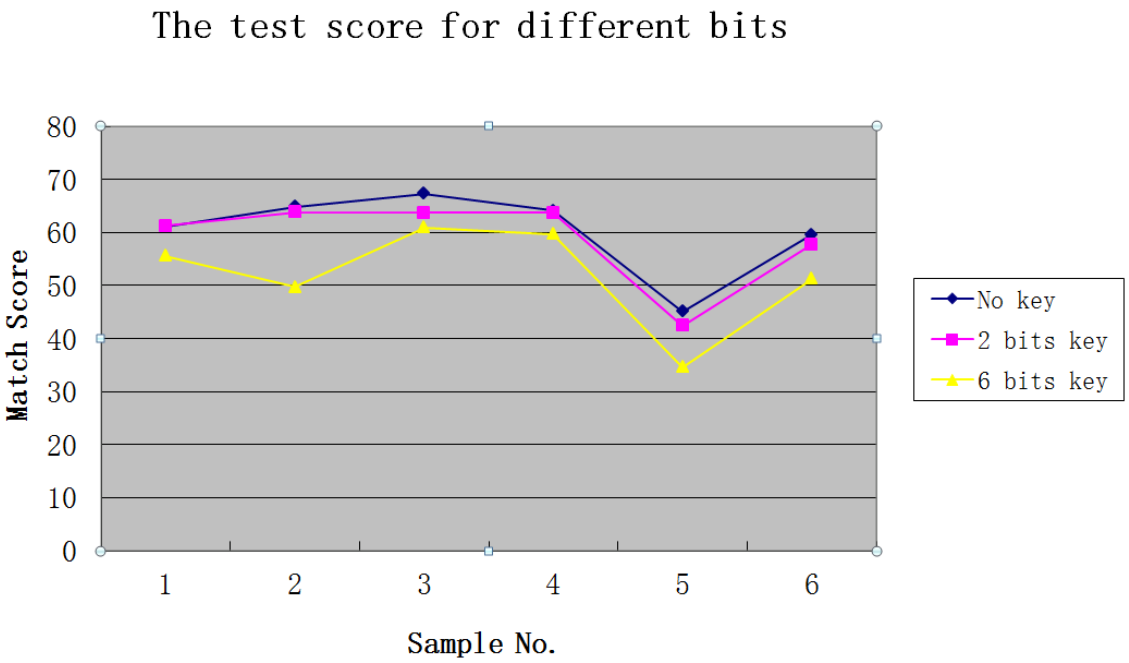

图 6.4 实验结果

6.2 成本与可用性分析

新的指纹认证系统降低了指纹认证系统的成本，于此同时提升了指纹认证系统的易用性。新旧系统的成本与易用性对比如表 6.1 所示。

比较项目	旧系统	新系统
客户端	指纹读取器	智能手机终端
服务端	服务端软件	Web 服务程序
硬件成本	昂贵	便宜
软件成本	高	中
部署成本	高	低
便携性	无	优

表 6.1 新旧系统成本与易用性对比表

从表格中可以看出，新的解决方案以智能手机取代了指纹传感器，并使用 Web Service 提供标准化服务，因此，硬件和软件成本与传统解决方案相比大幅度降低，易用性显著提升。我们认为，基于智能手机的指纹认证系统解决了传统指纹认证系统中存在的成本和易用性问题。

6.3 安全性分析

系统采用了可逆数字水印技术加强系统安全性。正如仿真试验中看到的那样，我们采用不同的二进制位数来控制指纹图像矩阵。首先我们采用最低两位进行指纹图像的加密，在这种情况下，指纹图像的失真较小，新的图像与原始指纹图像相比只丢失了很少的信息；随后，我们采用最低四位和最低六位进行指纹图像加密，在这些情况下，加密后的指纹图像较之于原始指纹图像丢失了更多的信息。图 6.5 展示了上述不同情形下生成的二维码图片信息。

图 6.5 不同条件下的二维码图片

然而，从二维码图像中，我们并不能看出不同条件下的区别，图 6.6 展示了由上述二维码解码获得的指纹图片。

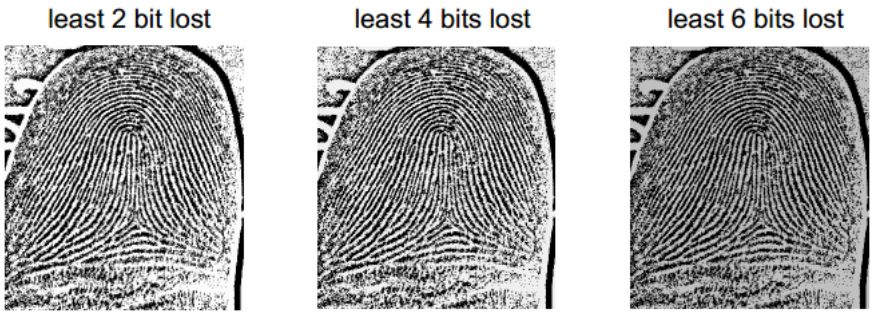

图 6.6 不同条件下的指纹图片

从解码后获得的指纹图片中可以看出，与失真 2 位相比，失真 6 位的指纹图片比原始图片更暗，这意味着损失了更多的信息用于加密。从仿真实验中可以看出，采用指纹图片最低两位进行加密，不仅能够加强系统的安全性，而且不影响正常的指纹匹配结果。因此，我们认为，基于数字水印的指纹认证系统是足够安全的。

7 总结与展望

7.1 总结

论文在移动互联网背景下探讨了基于智能手机的加强安全的指纹认证系统的设计和实现。回顾系统的开发过程，我们在整个过程中主要进行了以下几方面工作：

（1）从宏观上分析了当前传统指纹识别系统的工作方式，并结合移动互联网的特点，提出了一个新的指纹认证系统解决方案。新的指纹认证系统旨在解决传统指纹认证系统中的高成本和低可用性问题。

（2）调查了实现指纹认证系统的相关技术成果，包括指纹识别技术、数字水印技术、二维码技术、SOA 技术、Web Service 技术以及 Nancy Framework.

（3）基于新的指纹认证系统解决方案，分析系统中存在的安全漏洞及其防御措施，结合二维码技术和可逆数字水印技术加入了一个新的安全模块。在此基础上进行了系统需求建模。给出了系统总体用例图并分用例进行了用例描述。

（4）在面向对象的需求分析基础上，进行了面向对象设计，结合 REST Web Service 技术进行了系统架构设计和数据库设计。

（5）围绕系统设计，在.net 平台上采用 Nancy Framework 分别实现服务器模型层、视图层和控制器层，此外还对系统关键技术、匹配算法、指纹加密算法进行了详细分析。

（6）在系统实现的基础上，采用 Matlab 进行了系统仿真实验，并对实现结果从成本和安全性等方面进行了分析。

在上述系统设计和开发过程中，我遇到和解决了很多问题，总结出了以下经验教训。

（1）在.net 平台上开发服务端时应注意解决好配置问题。在系统开发过程中，由于 net 平台配置的复杂性，产生了许多非代码的问题，只有解决好相对应的配置问题，才能提高平台的开发效率和运行质量。对于 Nancy 框架，必须注意该框架的 Web 服务宿主问题，必须解决好 Nancy 框架和 IIS Web 服务器的协作问题，才能更好地提供 Web 服务。

（2）系统开发过程应遵循需求－设计－实现－测试的瀑布模型。需求依据产品，设计依据需求，实现依据设计，测试面向需求，整个过程环环相扣，互相制约和保证，这样的开发闭环有助于开发者保证开发效率，也可以使最终开发出

来的产品符合最初的期望。在许多软件开发过程中，人们经常重代码轻需求和设计，这最终导致开发出来的系统并不能符合最初的用户期望，这也证明了代码和设计文档同等重要，不可偏废。

（3）指纹认证系统开发必须特别注重安全性问题。由于指纹认证系统关乎用户身份认证，因此需要防止用户身份信息的泄露，同时也必须确保指纹认证者身份的有效性。采用可逆数字水印加密可以有效的防止会话劫持和重放攻击。

（4）采用开放灵活的技术架构提供跨平台可扩展的 Web 服务。开放灵活的技术架构是面向对象设计原则的要求之一，面向对象的设计原则要求面向接口编程而不是面向实现编程。采用基于 REST 的 Web Service 可保证系统功能的跨平台和扩展。

（5）没有最好的架构，只有最合适的架构。在指纹认证系统的开发过程中，分别采用了 Ruby on Rails、Node.js、Nancy Framework 等多种技术架构，深感架构之美；但是由于指纹识别类库所限，最终只能采用.net 平台下的 Nancy Framework 来实现整个系统。

7.2 展望

基于数字水印的指纹认证系统是对移动互联网时代下新的指纹识别系统的有效尝试。经过这次系统开发，我认为以下一些技术和方法会更适应移动互联网的未来：

（1）服务化越来越称为移动互联网的基石，SOA 和 Web Service 将成为移动互联网的基础理念。未来的移动互联网会更多的采用服务化接口提供多种跨平台的服务。

（2）轻量级框架会越来越流行，由于未来移动互联网的发展，移动设备对 Web API 的需求远远大于对 Web 页面的需求，鉴于此许多只实现 Web API 功能的轻量级框架必将越来越流行。

（3）敏捷化开发方法会在移动开发中占据一席之地。由于移动互联网瞬息万变，因此敏捷化方法更加适应了移动端快速发展的需求。快速开发、快速上线、快速迭代，这种新的开发方法可能会随着移动互联网的发展迅速流行。

（4）基于智能手机的指纹识别技术会在未来几年取得愈发广泛的重视和发展，会有越来越多的应用场景采用基于智能手机的指纹识别技术。因此，应当加强对移动互联网背景下指纹识别系统开发的研究。相信在不久的将来，这种技术必将大放光彩。

参考文献