

[认证服务系统开发技术文档]

(密码学原理与实践-实践部分)

[项目名称：天际-认证服务系统以及天际线银行]

[姓名：吴昊]

[学号：120L020818]

[同组成员：袁野(电商)]

目录

目录	I
1. 背景与意义	1
1.1 项目开发意义	1
1.2 国内外现状及技术综述	2
2. 需求分析	4
2.1 总体需求	4
2.2 功能需求	4
2.2.1 用户功能需求	4
2.2.2 管理员功能需求	5
2.2.3 CA 功能需求	5
2.3 性能需求	6
2.3.1 安全性能需求	6
2.3.2 总体性能需求	7
3. 概要设计	8
3.1 模块划分	8
3.2 数据库设计	8
3.3 安全传输	10
3.4 安全注册登陆	11
3.5 公私钥生成和私钥保护	12
3.6 证书申请与审批	14
3.7 证书撤销	15
3.8 证书验证	15
3.9 电子信封	17
3.10 双重签名	18
3.11 系统日志	19
4. 详细设计	20
4.0 各界面的美术设计	20
4.0.1 登陆界面	20
4.0.2 注册页面	20
4.0.3 主页面/欢迎页面	20
4.0.4 各个功能页面	22
①用户功能页面:	22
②管理员功能页面:	24
4.0.5 项目目录结构	25
4.1 安全传输	26
4.2 安全注册与登陆	28
4.3 公私钥对生成和私钥保护	31
4.4 签名与验证算法设计	34
4.5 证书申请与审批	36

4.6 证书撤销	40
4.7 在线验证证书有效性(OCSP)	41
4.8 系统日志	42
5. 实现	44
5.1 技术栈选择	44
5.2 图形/邮件验证码	44
5.2.1 邮件验证码	44
5.3 混合加密+签名	46
5.3.1 前端混合加密+签名:	46
5.3.2 后端对称密钥加密响应:	47
5.3.3 前端解密:	48
5.3.4 后端解密:	48
5.4 加盐存储	49
5.5 公私密钥对的生成与私钥的保护	49
5.5.1 公私密钥对生成	49
5.5.2 私钥保护	50
5.6 CA 签名与验签	53
5.6.1 签名	53
5.6.2 OCSP 验证签名	54
5.7 证书传递	55
6. 测试	59
6.1 注册/登陆/登出	59
6.2 公私钥对生成和私钥保护	60
6.3 证书申请与审批	62
6.4 CA 证书分发、签名验证	63
6.5 证书撤销与 OCSP 验证	66
7. 结束语	68
7.1 Git 提交记录与学习、开发流程	68
7.2 遇到的困难、解决与收获	69
7.3 鸣谢	69
参考文献	70
附录	71
附录 A	71
附录 B	71
附录 C	72

1. 背景与意义

1.1 项目开发意义

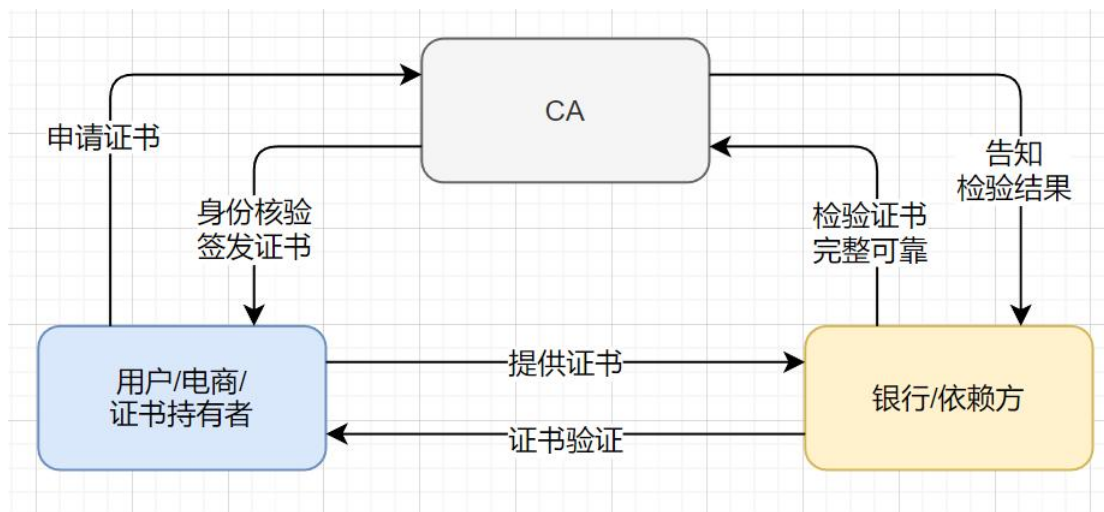


图 1.1 CA 原理图

数字证书认证机构（英语：Certificate Authority，缩写为 CA），也称为电子商务认证中心、电子商务认证授权机构，是负责发放和管理数字证书的权威机构，并作为电子商务交易中受信任的第三方，承担公钥体系中公钥的合法性检验的责任。

CA 为每个使用公开密钥的用户发放一个数字证书，数字证书的作用是证明证书中列出的用户合法拥有证书中列出的公开密钥。CA 的数字签名使得攻击者不能伪造和篡改证书。它负责产生、分配并管理所有参与网上交易的个体所需的数字证书，因此是安全电子交易的核心环节。在 SET(英语:Secure Electronic Transaction)交易中，CA 不仅对持卡人、商户发放证书，还可以对获款的银行、网关发放证书。

CA 是证书的签发机构，它是公钥基础设施的核心。细化来说，CA 是负责签发证书、认证证书、管理已颁发证书的机关。它要制定政策和具体步骤来验证、识别用户身份，并对用户证书进行签名，以确保证书持有者的身份和公钥的拥有权。

为保证用户之间在网上传递信息的安全性、真实性、可靠性、完整性和不可抵赖性，不仅需要用户的身份真实性进行验证，也需要有一个具有权威性、公正性、唯一性的机构，负责向电子商务的各个主体颁发并管理符合国际安全电子交易协议标准的电子商务安全证，并负责管理所有参与网上交易的个体所需的数字证书，因此 CA 是安全电子交易的核心环节。

1.2 国内外现状及技术综述

1976 年 Whitfield Diffie、Martin Hellman、Ron Rivest、Adi Shamir 和 Leonard Adleman 等人相继公布了安全密钥交换与非对称密钥算法后，整个通信方式为之改变。随着高速电子数字通信的发展，用户对安全通信的需求越来越强。密码协议在这种诉求下逐渐发展，造就新的密码原型。全球互联网发明与扩散后，认证与安全通信的需求也更加严苛。仅仅是商务理由便足以解释一切。当时在网景工作的 Taher ElGamal 等人发展出传输安全层协议，包含了密钥创建、服务器认证等。公开密钥基础建设的架构因此浮现。

美国的厂商和企业家察觉了其后的广大市场，开始设立新公司并启动法律认知与保护。美国律师协会项目发行了一份对公开密钥基础建设操作的可预见法律观点的详尽分析，随后，多个美国州政府与其他国家的司法单位开始制定相关法规。消费者团体等则提出对隐私、访问、可靠性的质疑，也被列入司法的考虑中。被制定的法规实有不同，将公开密钥基础建设的机制转换成商务操作有实际上的问题，远比许多先驱者所想的缓慢。

21 世纪的前几年，人们才慢慢发觉，密码工程没那么容易被设计与实践，某些存在的标准某方面甚至是不合宜的。公开密钥基础建设的厂商发现了一个市场，但并非九十年代中期所预想的那个市场，这个市场发展得缓慢而且以不同的方式前进。公开密钥基础建设并未解决所期待的问题，某些厂商甚至退出市场。

公开密钥基础建设最成功的地方是在政府部门，目前最大的公开密钥基础建设是美国防卫信息系统局（Defense Information Systems Agency, DISA）的共同访问卡（Common access Cards）方案。2000 年 6 月 30 日，美国总统克林顿正式签署美国《全球及全国商业电子签名法》，给与电子签名、数字证书以法律上的保护，这一决定使电子认证问题迅速成为各国政府关注的热点。加拿大在 1993 年就已经开始了政府 PKI 体系雏形的研究工作，到 2000 年已经在 PKI 体系方面获得重要进展。加拿大与美国代表了发达国家 PKI 发展的主流。

日本认证服务业 2006 年的产值推估为 225 亿日元。电子化政府服务有大量应用公开密钥，特别是 2003 年发行的国家身份证智能卡。在金融服务上，多数银行之在线系统并没导入公开密钥。日本使用公开密钥的用途十分多样，例如公众公证服务中心就提供合约、组织文件等资料的存证服务。

而在 2005 年，香港有三家 CA，其中两家为政府单位，一家为民间公司。在“数字 21 新纪元”项目的 200 类在线政府服务中，有交通局、税务局、选举事务处、差饷物业估价署、入境事务处的服务使用公开密钥。

台湾其证书管理框架（简称 CA）采用层次结构式的管理机制，其信赖核心是国家发展委员会委托中华电信管理的政府证书管理中心（简称 GRCA），GRCA 将签发的 CA 证书给政府公开密钥基础建设（简称 GPKI）的下层 CA，而内政部证书管理中心则属于 GPKI 中的第一层下属证书机构，并遵循 GPKI 证书政

策所订定之保证等级第三级的规定，对于在台湾设籍登记满 18 岁以上之国民进行签发与管理其自然人公钥证书。

欧洲在 PKI 基础建设方面也成绩显著。已颁布了 93/1999EC 法规，强调技术中立、隐私权保护、国内与国外相互认证以及无歧视等原则。为了解决各国 PKI 之间的协同工作问题，他们采取了一系列策略：如积极主动相关研究所、大学和企业研究 PKI 相关技术；自主 PKI 互操作性相关技术研究，并建立 CA 网络及其顶级 CA。并于 2000 年 10 月成立了欧洲桥 CA 指导委员会，于 2001 年 3 月 23 日成立欧洲桥 CA。

我国的 PKI 技术从 1998 年开始起步，由于政府和各有关部门近年来对 PKI 产业的发展给予了高度重视，2001 年 PKI 技术被列为”十五“863 计划信息安全主题重大项目，并于同年 10 月成立了国家 863 计划信息安全基础设施研究中心。国家计委也在制定新的计划来支持 PKI 产业的发展，在国家电子政务工程中明确提出了要构建 PKI 体系。目前，我国已全面推动 PKI 技术研究与应用。

2. 需求分析

2.1 总体需求

CA 是负责签发证书、认证证书、管理已颁发证书的机关。它要制定政策和具体步骤来验证、识别用户身份，并对用户证书进行签名，以确保证书持有者的身份和公钥的拥有权。

用户若欲获取证书，应先向 CA 提出申请，CA 判明申请者的身份后，为之分配一个公钥，并将该公钥与其身份信息绑定，为该实体签名，签名后的整体即为证书，发还给申请者。

网上的公众用户通过验证 CA 的签名从而信任 CA，任何人都可以得到 CA 本身的证书（含公钥），用以验证 CA 所签发的证书。如果一个用户想鉴别另一个证书的真伪，他就用 CA 的公钥对那个证书上的签字进行验证，一旦验证通过，该证书就被认为是有效的。

2.2 功能需求

2.2.1 用户功能需求

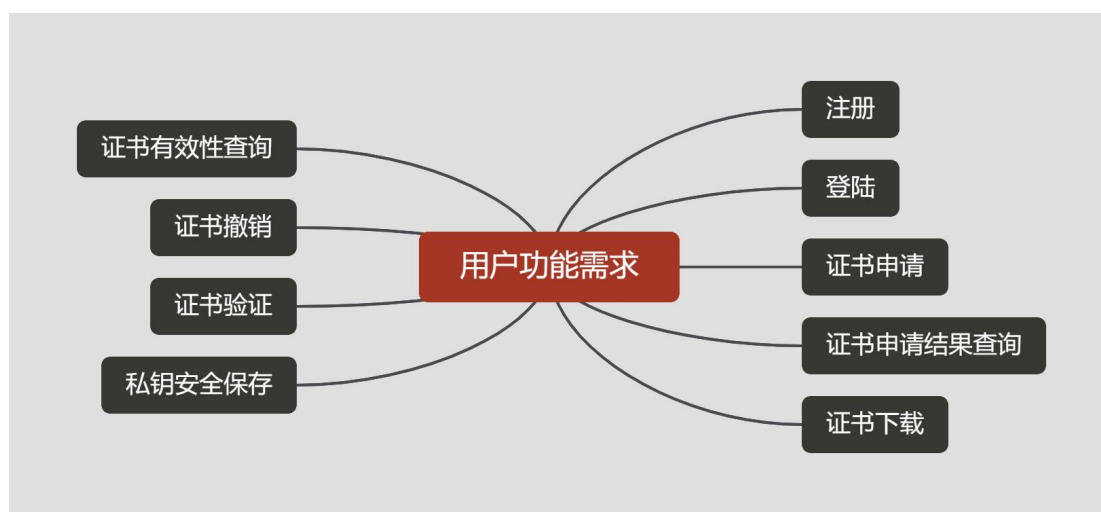


图 2.1 用户功能需求

用户的功能需求包括：注册，登录，证书申请，证书申请结果查询，证书下载，证书的有效性查询，证书撤销，证书的签名验证，私钥安全保存。

2.2.2 管理员功能需求

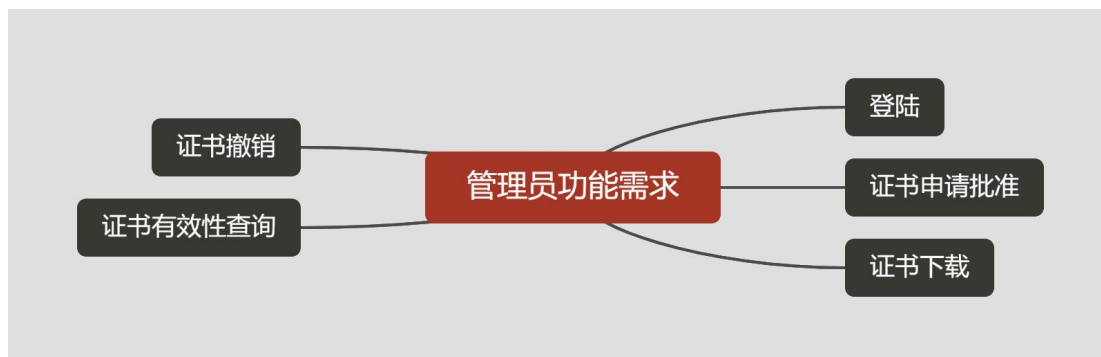


图 2.2 管理员功能需求

管理员功能需求包括：登录，证书申请批准，证书下载，证书撤销以及证书有效性查询。

2.2.3 CA 功能需求

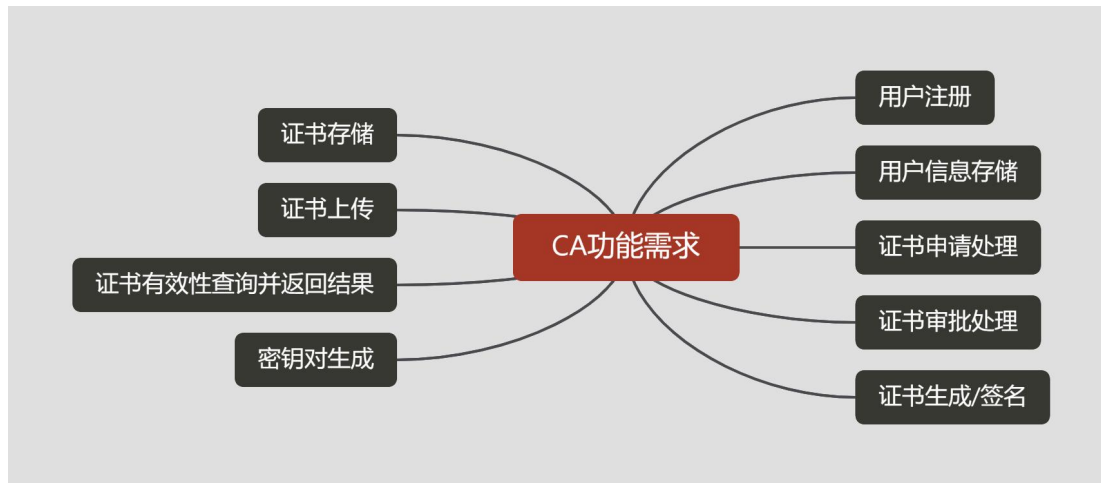


图 2.3 CA 功能需求

CA 的功能需求包括用户注册，用户信息存储，证书申请处理，证书审批处理，证书生成与签名，证书存储，证书上传，证书有效性查询结果返回。

1. 用户注册、登录。
2. 用户密钥对安全生成与保存。
3. 管理员接收用户数字证书的申请，并进行审批颁发（或拒绝颁发）证书。
4. 生成、存储证书，产生和发布证书的有效期。
5. 接收用户数字证书的查询、撤销。

6. 数字证书与密钥归档。

整个项目功能方案的大体逻辑流程如下：

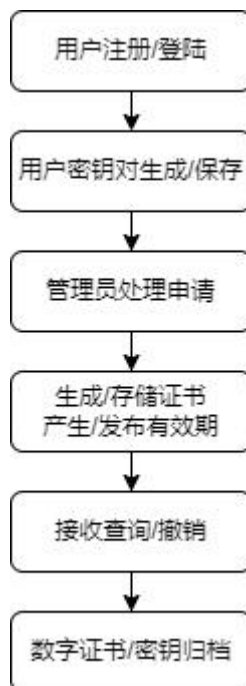


图 2.4 程序流程图功能需求

2.3 性能需求

2.3.1 安全性能需求

1. 安全注册与登录：

完成用户身份认证注册，数据库中安全存储用户密码；

用户登录防止暴力破解

注册与登录信息需要安全传输，保证机密性、完整性与真实性

2. 用户密钥对生成并保证私钥安全：

为用户安全生成密钥对

保证私钥安全不泄露

3. 证书签名与认证

4. 传输中的机密性、完整性、真实性

所有传输过程，均需保证机密性、完整性、真实性，包括：

- ✓ 用户注册和登录的完整性与真实性
- ✓ 用户证书申请信息的完整性与真实性
- ✓ 管理员获得的证书申请信息的完整性与真实性
- ✓ 管理员审批结果信息的完整性与真实性
- ✓ 用户、管理员证书撤销信息的完整性与真实性

- ✓ 用户、管理员证书有效性查询的完整性和真实性

2.3.2 总体性能需求

1. 加密解密速度不能过慢→采用混合加密
2. 网页加载速度控制在 1 秒以内
3. 支持至少 100 人同时访问
4. 平台支持性较好，支持各种主流浏览器
5. 系统可以 24 小时持续运行

3. 概要设计

3.1 模块划分



图 3.1 模块划分

将功能模块化划分为: 数据库设计, 安全注册登陆, 安全传输, 公私密钥对生成与私钥保护, 证书申请与审批, 证书撤销, 证书验证, 包括前端验证 CA 签名以及在线证书状态协议 OCSP, 系统日志记录。

3.2 数据库设计

3.2.1 user 表

表 1 user 表

字段	类型	注释
UserName	Varchar (32)	用户名
Password	Varchar (64)	用户密码
Email	Varchar (32)	电子邮箱
Sex	Varchar (1)	性别
Birthday	Date (N/A)	生日
RegDay	Date (N/A)	申请日期
Phone	Varchar (32)	手机号
Salt	Varchar (32)	盐值

`user` 表用于保存用户信息, 包括用户名, 加盐后的用户密码, 用户邮箱, 用户性别, 用户生日, 账号申请日期, 用户手机号, 以及用于给密码加盐的盐值。注意, 此处性别定义为字符类型, 存储 ‘1’ 时代表为男性, ‘0’ 时代表为女性。

注意, 我们在 `user` 表中会保存一些预先创建好的特殊用户作为管理员, 以对系统进行维护, 并进行证书的审查。

另外, 中国大陆的手机号一般是 11 位, 并且符合一定的正则表达式。但是为了拥有更好的可扩展性, 以适应更大范围的需求, 在数据库中, 我们对手机号的位数没有过多限制, 具体的手机号的合法性判断交给了后端程序处理。

3.2.2 apply 表

表 2 apply 表

字段	类型	注释
RegistrationNumber	Varchar (32)	工商注册号
SerialNumber	Varchar (64)	证书序号
Organization	Varchar (32)	组织名称
StartTime	Date (N/A)	申请时间
EndTime	Date (N/A)	证书失效时间
JuridicalPerson	Varchar (32)	法人姓名
ChargePerson	Varchar (32)	经办人姓名
ChargePhone	Varchar (32)	经办人电话
UserName	Varchar (32)	申请者用户名
PublicKey	Varchar (600)	公钥

`apply` 表保存已申请但是待审批的证书申请信息, 内容包括: 工商注册号, 证书序号, 组织名称, 申请时间, 证书失效时间, 法人姓名, 经办人姓名, 经办人电话, 申请者用户名, 对应公钥。

3.2.3 cert 表

表 3 cert 表

字段	类型	注释
RegistrationNumber	Varchar (32)	工商注册号
SerialNumber	Varchar (64)	证书序号
Organization	Varchar (32)	组织名称
StartTime	Date (N/A)	申请时间
EndTime	Date (N/A)	证书失效时间
JuridicalPerson	Varchar (32)	法人姓名
ChargePerson	Varchar (32)	经办人姓名
ChargePhone	Varchar (32)	经办人电话
UserName	Varchar (32)	申请者用户名
PublicKey	Varchar (600)	公钥
CertPathName	Varchar (32)	证书存储路径

cert 表用于保存已审批、且未撤销的有效证书信息，内容包括：工商注册号，证书序号，组织名称，申请时间，证书失效时间，法人姓名，经办人姓名，经办人电话，申请者用户名，对应公钥，证书存储路径。

在实际实验的过程中，为了方便，证书存储路径就放在了项目目录下，所以这一字段实际上置空，在未来可以做到对证书统一存储，提高了系统的可扩展性。

3.2.4 crl 表

表 4 crl 表

字段	类型	注释
SerialNumber	Varchar (64)	证书序号
Organization	Varchar (32)	组织名称
RevokeTime	Date (N/A)	证书撤销时间

crl 表用于保存已被撤销的证书信息，内容包括：证书序号，组织名称和证书撤销时间。

注意到，在数据库中，为了防止数据库泄露后造成的密码泄露，我们对 user 表中的用户密码做加盐处理，避免明文泄露。而为了便于演示实验，其余字段我们并未进行加盐，实际上对其余字段的加盐只需要进行相同的步骤。

3.3 安全传输

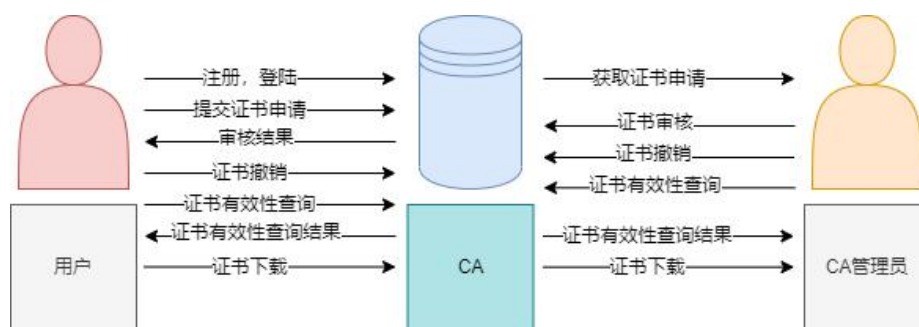


图 3.2 传输过程总览

所有的传输过程如图所示，用户、CA、CA 管理员之间所传输的信息均需要经过“混合加密+MAC”来保护，实现“机密性+完整性+真实性”。

用户和 CA 之间的传输信息，包括：注册，登录，提交证书申请，获得证书的审核结果，证书撤销，证书有效性查询，证书有效性查询结果，证书下载等。

CA 管理员和 CA 之间的传输过程，包括：获取证书申请，提交审批结果(通过或者不通过)，证书撤销，证书有效性查询，证书有效性查询结果，证书下载等。

3.4 安全注册登陆

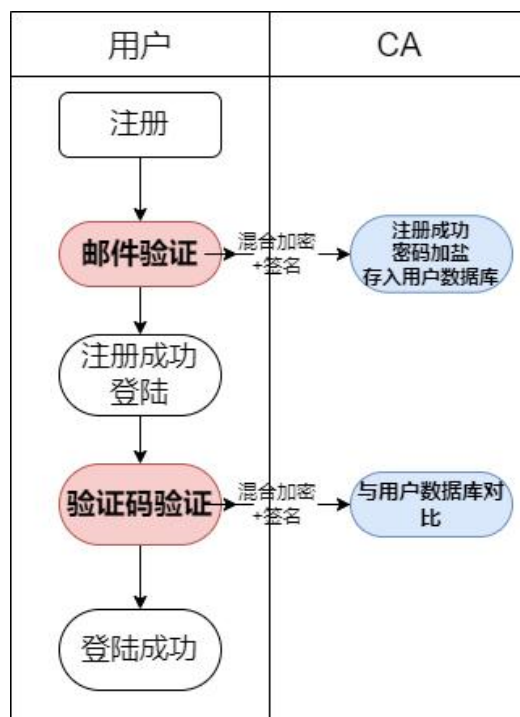


图 3.3 安全注册登陆流程

安全注册过程如下：首先，用户进行注册需通过邮箱的验证码验证，将注册信息经过混合加密和签名之后，发送给 CA 后端，然后，CA 后端进行解密和验证签名，将密码加盐存入用户数据库。

安全登录过程如下：首先，用户进行验证码验证，验证成功后，登录信息经

过混合加密和签名发给 CA 后端, 然后, CA 后端解密和验证签名, 将信息与用户数据库中的信息进行比对, 一致则登录成功。

3.5 公钥生成和私钥保护

3.5.1 公钥生成

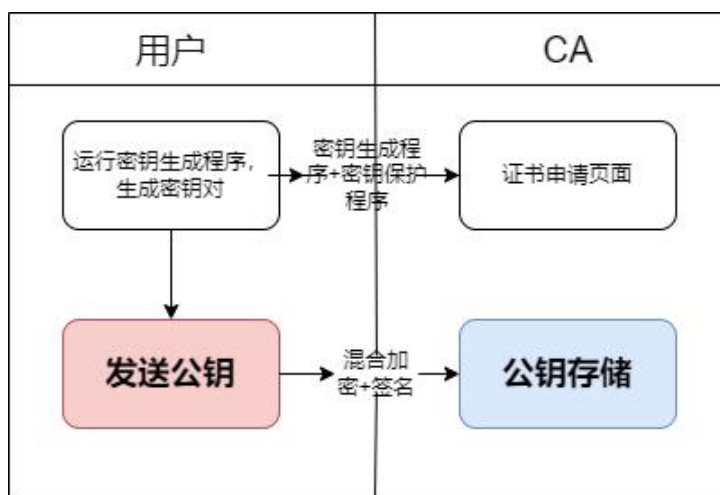


图 3.4 公钥生成

用户在证书申请页面, 获取密钥生成程序以及私钥保护程序 2 个程序, 在本地运行密钥生成程序, 生成密钥对并发送公钥, 经过混合加密和签名之后, 发给 CA; CA 经过解密和验证签名之后, 把公钥进行存储。

另外, 在系统中又增加了公钥自动分发的功能, 现在我们可以直接点选按钮进行公私钥的自动分配生成, 而不需要在本地进行手动运行程序操作了。

3.5.2 私钥保护

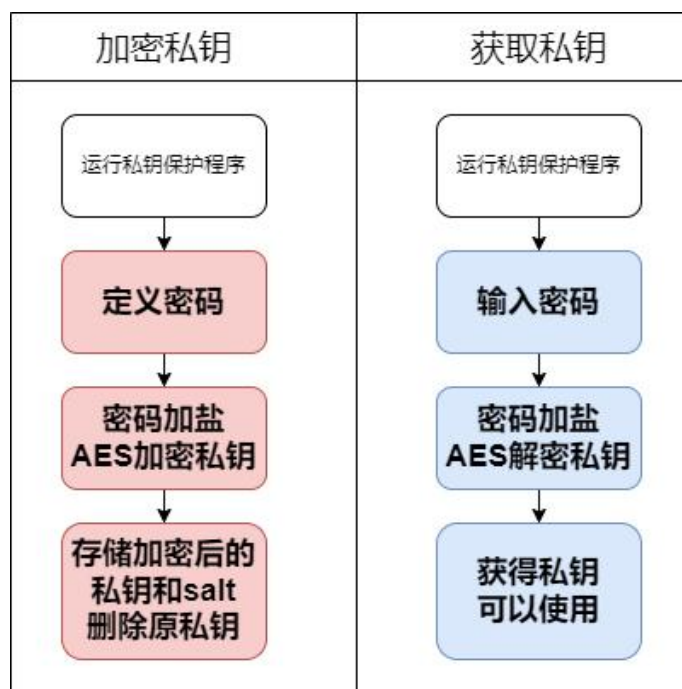


图 3.5 私钥保护

用户在本地运行私钥保护程序, 加密私钥。首先要定义用户密码, 程序随机生成一个 **salt** 对密码进行加盐, 然后使用 **AES** 算法对私钥进行加密, 对加密后的私钥以及随机生成的盐值进行存储, 最后删除原有的私钥文件。

用户若想获取私钥, 首先要输入原先定义的密码程序, 读取 **salt**, 对密码进行加盐, 读取被加密的私钥, 用 **AES** 算法进行解密, 获得私钥, 然后进行使用。

3.6 证书申请与审批

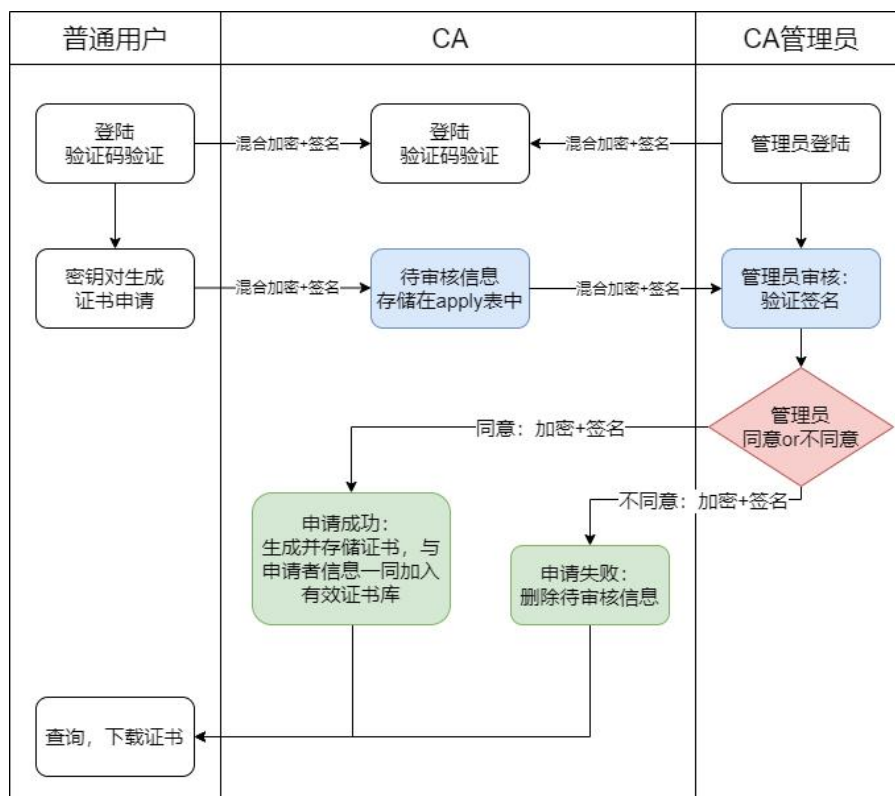


图 3.6 证书申请与审批

证书申请流程如下，首先，普通用户向 CA 提交证书申请，证书申请信息经过混合加密和签名之后，发给 CA 后端，CA 经过解密和验证签名之后，将待审核信息存储在数据库 apply 表中。

CA 管理员登录后，获取待审核的证书信息：CA 后端将信息经过 CA 私钥签名之后发给 CA 管理员；管理员首先进行签名验证，然后才可以进行审核；管理员将审核结果经过加密和签名之后，发给 CA 后端。

CA 后端经过解密和验证签名，获取管理员的审批结果。若审批通过，则生成并存储证书，与申请者的信息一同加入有效证书数据库 cert 表中。若审批结果不通过，则从 apply 表中删除原有的待审核信息。

最后，普通用户可以查询证书申请结果，并下载证书。

3.7 证书撤销

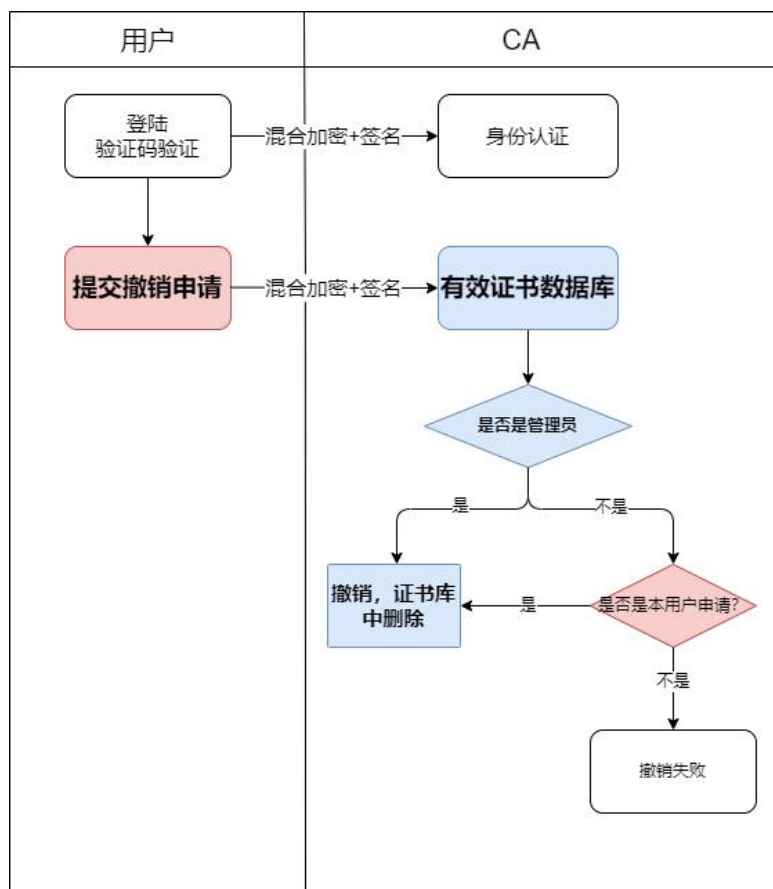


图 3.7 证书申请与审批

用户登陆后，提交撤销申请，撤销申请信息经过混合加密和签名之后，发给 CA 后端。CA 后端在有效证书数据库 cert 表中查询，获得证书信息，以及对应的当初的申请者用户名。

CA 首先进行判断，撤销申请的提交者是否为管理员，若为管理员，则拥有任意撤销各种证书的权限，直接将在有效证书库 cert 表中删除，并将证书信息放入已被撤销的证书数据库 crl 表中。若不是管理员，则需判断要撤销的证书是否为本用户所申请，若不是则撤销失败。

对于普通用户需要进行身份权限隔离，即用户只能撤销自己所申请的证书。而系统管理员则可以撤销任意用户的证书。

3.8 证书验证

3.8.1 前端检验证书的 CA 签名

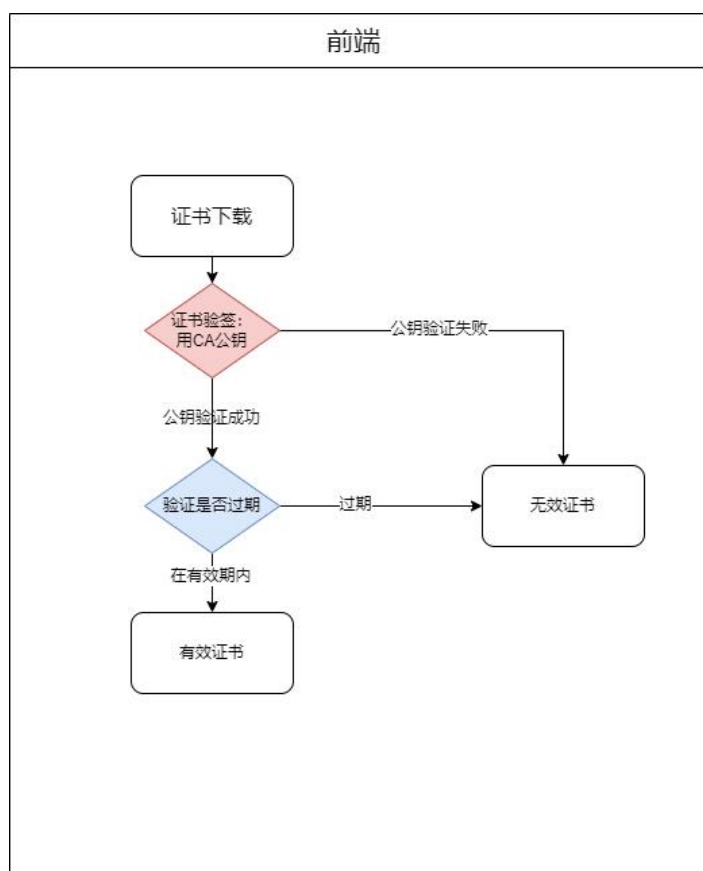


图 3.8 前端验证 CA 签名

前端验证证书的 CA 签名总共分两步，第一步：用 CA 公钥对证书信息进行验证；第二步：验证证书是否过期；

两步均通过之后才为有效证书，否则为无效证书。

3.8.2 在线检验证书有效性 OCSP（是否被撤销）

当该数字证书被放入 crl 数据库后，数字证书则被认为失效。注意，失效并不意味着无法被使用。如果乙窃取到甲的私钥，然后用甲的私钥签名了一份文件发送给丙，并且附上甲的证书。若丙忽视了对 crl 数据库的查看，丙就依然会用甲的证书成功的验证这份非法的签名，并会认为甲确实对这份文件签名。这种攻击即重放攻击。

随之而来一个话题，就是如何让各依赖方能够容易、及时、正确地得到他们需要了解的证书的撤销信息。这涉及到证书撤销状态的发布方式和发布时间这两个方面。

常用的证书撤销状态发布方式有在线证书状态协议（Online Certificate

Status Protocol, 简称为 OCSP), 该协议为依赖方提供对证书状态的实时在线查询。依赖方不用检查证书撤销列表而是向证书状态查询服务器在线询问某证书的状态。但是 OCSP 服务器必须对查询的结果进行数字签名。

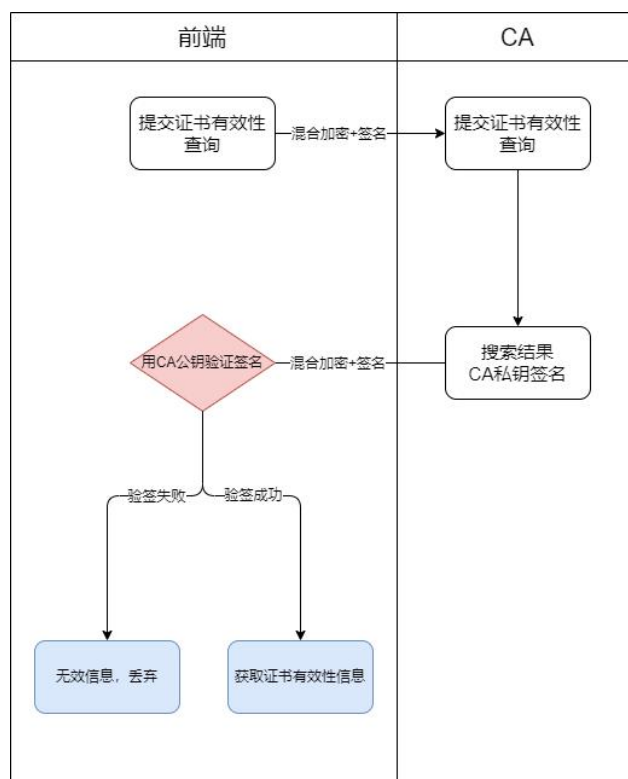


图 3.9 在线验证证书有效性的流程

用户提交证书有效性查询, 查询信息经过混合加密和签名之后发给 CA 后端。CA 后端经过解密和验证签名之后, 得到有效性查询申请, 在有效证书数据库中查询并得到结果, 用 CA 私钥进行签名发送给用户。

前端用户需要先用 CA 公钥验证 CA 后端的签名。若验证签名失败, 则获得的信息为无效信息, 予以丢弃; 若 CA 公钥验证签名成功, 则可以获取证书的有效性信息。

3.9 电子信封

SET 协议使用电子信封来传递更新的密钥, 电子信封涉及到两个密钥: 一个是接收方的公开密钥, 另一个是发送方生成临时密钥 (对称密钥)。其实电子信封机制就是使用混合加密来更新/传递证书, 流程图如下:

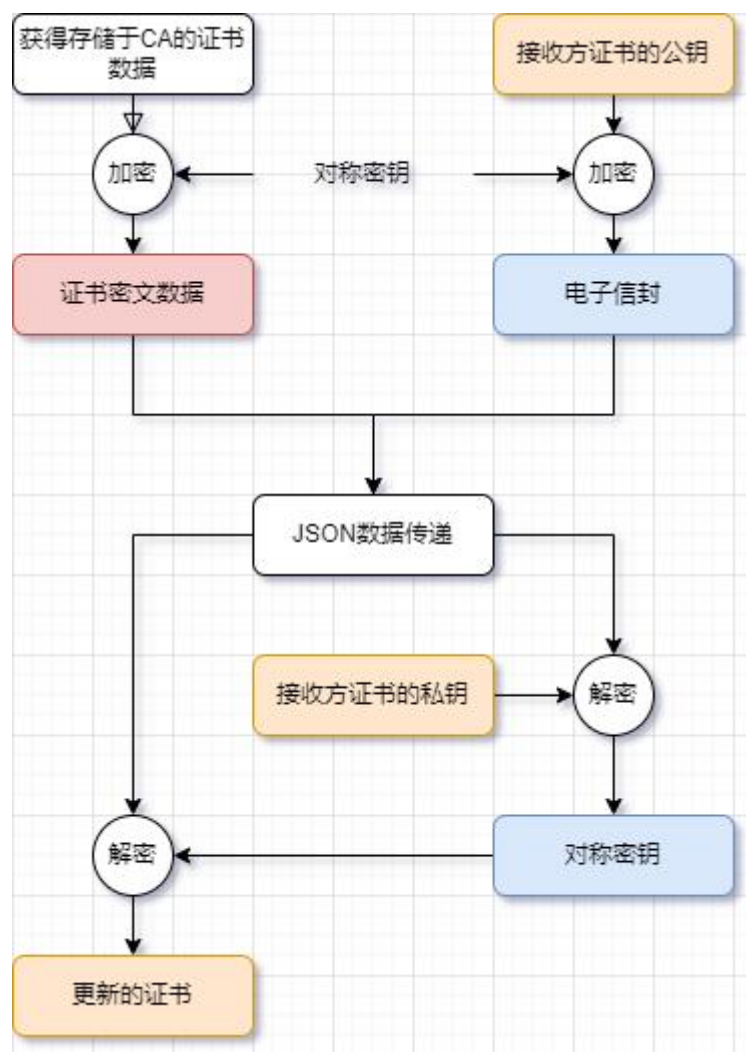


图 3.10 电子信封传递的流程

3.10 双重签名

SET 协议核心内容是订购信息 OI 和支付信息 PI。而 DS (Dual Signature) 技术将 OI 和 PI 这两部分的摘要信息绑定，确保电子交易的有效性和公正性。分离 PI 与 OI，确保商家不知道顾客的支付卡信息，银行不知道顾客的订购细节。实现了信息绑定与信息隐藏。

这一部分原本是银行部分的工作，但由于本次实验中银行部分缺失，故由本人代为实现这一功能。

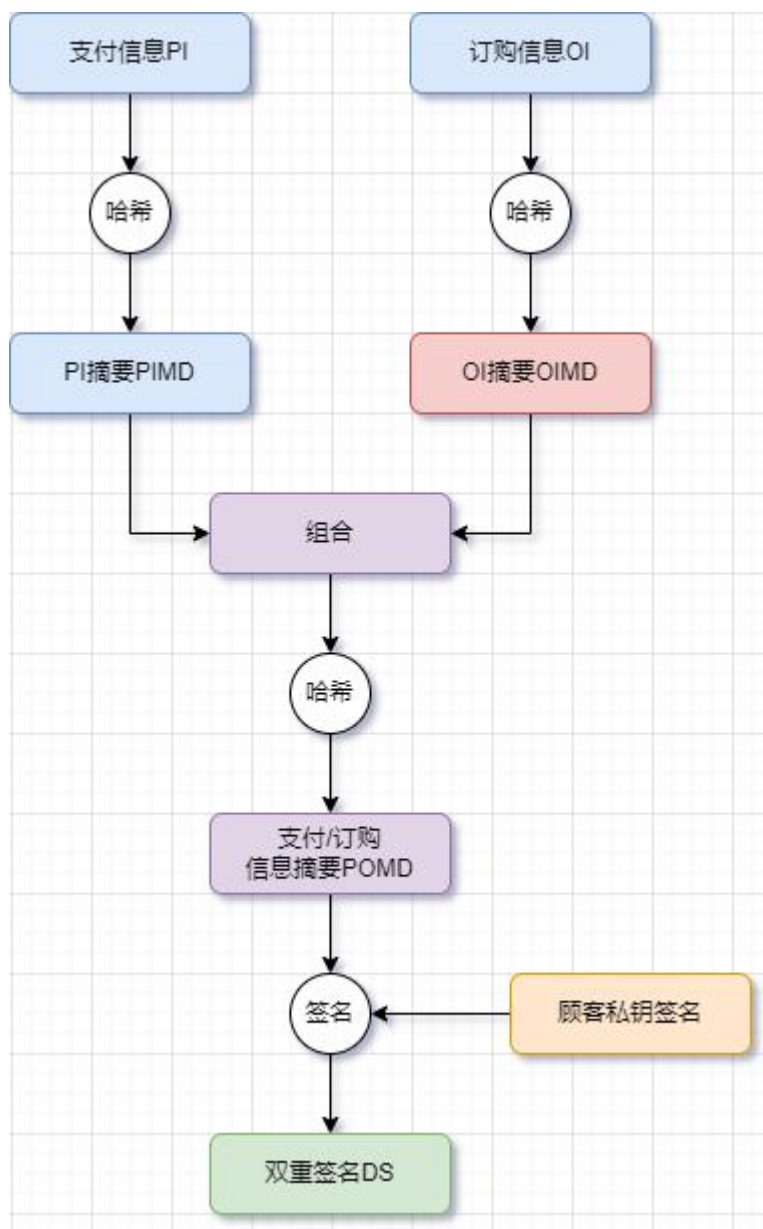


图 3.11 双重签名生成的流程

在使用过程中，顾客通过使用银行的公钥加密自己的对称密钥，并且使用对称密钥加密双重签名 DS，PI 和 OIMD，然后通过商家把加密后的信息转发给银行，银行通过计算 POMD（正向计算，即通过获得的 PI 和 OIMD 生成 POMD）与 POMD'（反向计算，即通过顾客公钥验签），倘若 POMD 与 POMD' 相等，则银行认为 DS 正确，并批准实施该交易。

3.11 系统日志

系统日志方面，本系统中大致分为两部分，一个是 user.log 日志，其中主要记录了用户注册、登陆、下线等情况；另一个是 admin.log，记录了用户申请与撤销证书和管理员颁发证书的情况。

将用户的申请信息，管理员的审批情况，用户或者管理员的撤销证书申请，连同时间，一同写在系统日志中。记录在本地文件里。

4. 详细设计

4.0 各界面的美术设计

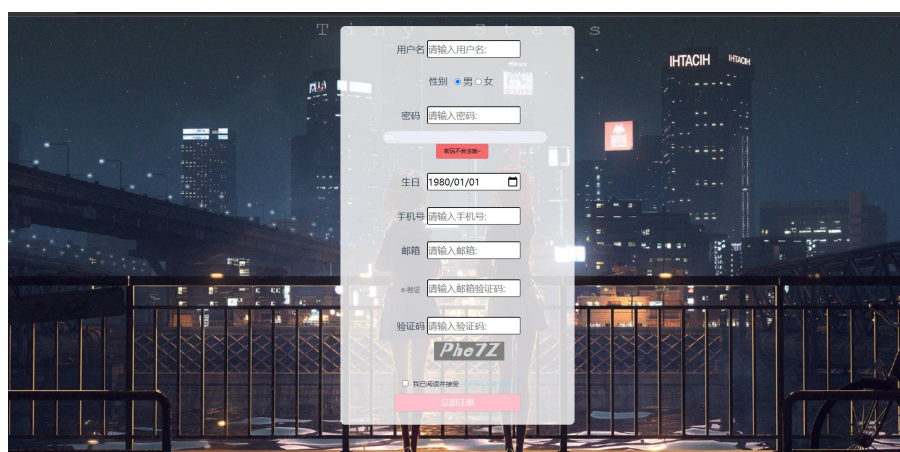
在这里我们详细介绍一下关于本项目的前端页面，以便对该项目拥有一个宏观的把握。在该项目的前端设计，我们在页面的主体功能部分采用了单组件应用的设计思想，在一个大的页面中继承了许多小的组件路由，以实现没有页面的硬跳转但能够显示多个页面的内容。

4.0.1 登陆界面

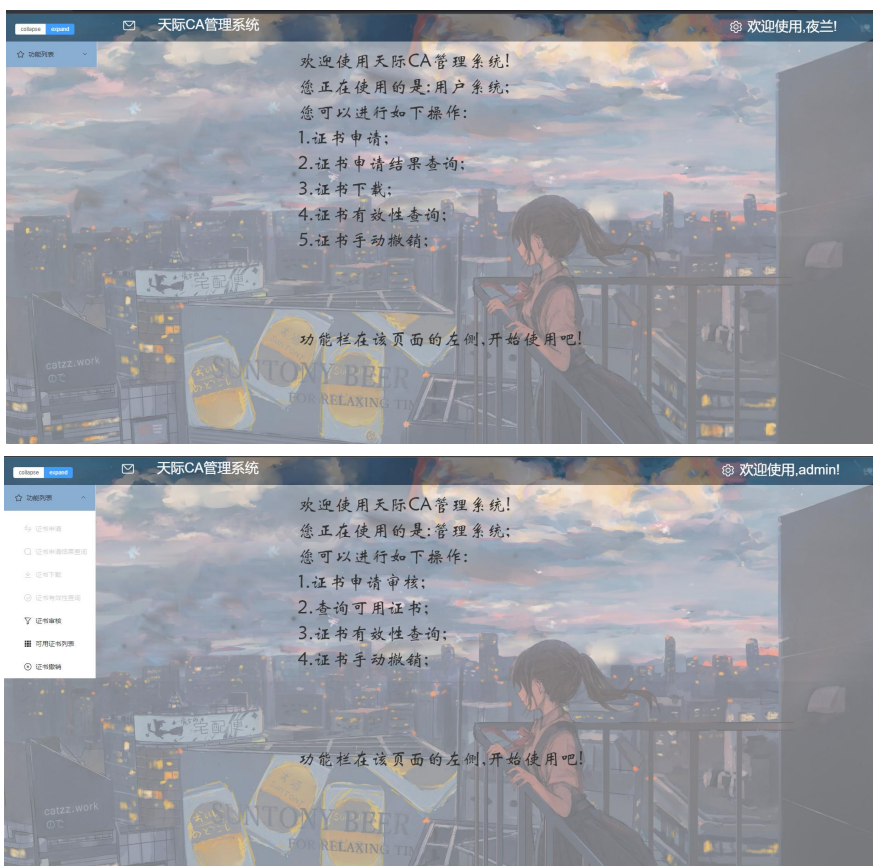


登陆页面大致包括欢迎横幅与登陆框两个部分，其中登录框使用了element-plus 组件生成，并且自己实现了验证码功能。

4.0.2 注册页面



4.0.3 主页面/欢迎页面



在主页面我们会根据是管理员用户还是普通用户显示不同的功能列表与欢迎语。因为我们使用了 `pinia` 技术，所以在多页面之间保存状态是很容易的一件事情。

主页面主要分为两大部分：页头和页身。

页头包含以下部分：

① “扩展/收缩按钮”，这个按钮可以使左侧的功能列表进行扩展和收缩，效果如下：



② “天际 CA 管理系统” 标签，这个标签可以点击，以使在其他页面的用户返回到主页面。

③ “欢迎” 标签，该标签左侧的设置按钮是可以点击的，会出现两个子选项，即个人信息与登出，图例如下：



注意，我们这里欢迎使用之后的字样是会随着 username 的变化而变化的，同样也是使用了 pinia 技术，在 login 页面就记录用户信息。

4.0.4 各个功能页面

①用户功能页面：

证书申请页面：



证书申请结果查询页面：



证书下载页面：

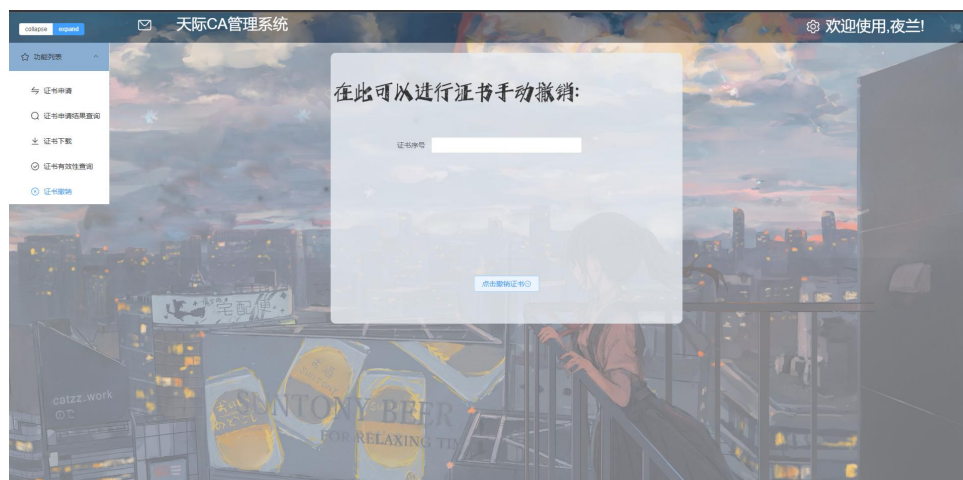
在这个页面可以选择传统的浏览器下载或者更安全的电子信封来获得证书。



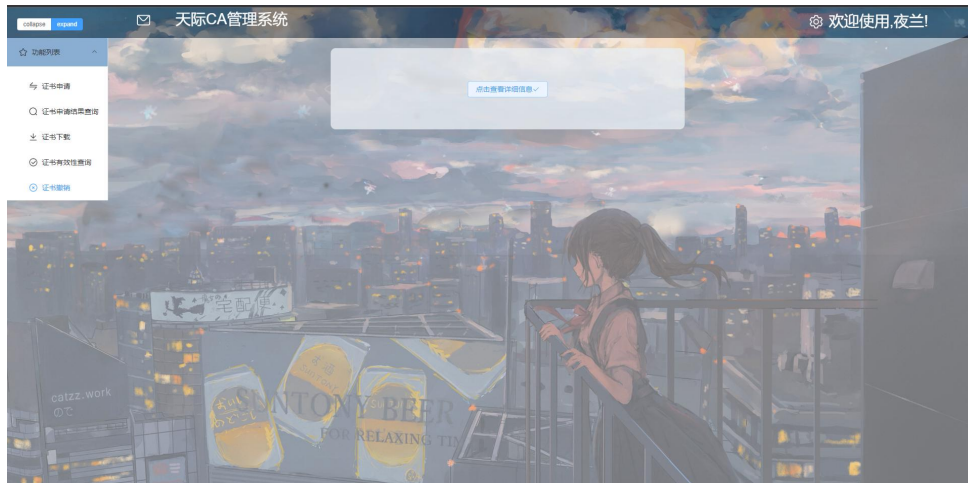
证书有效性查询页面：



证书撤销页面：



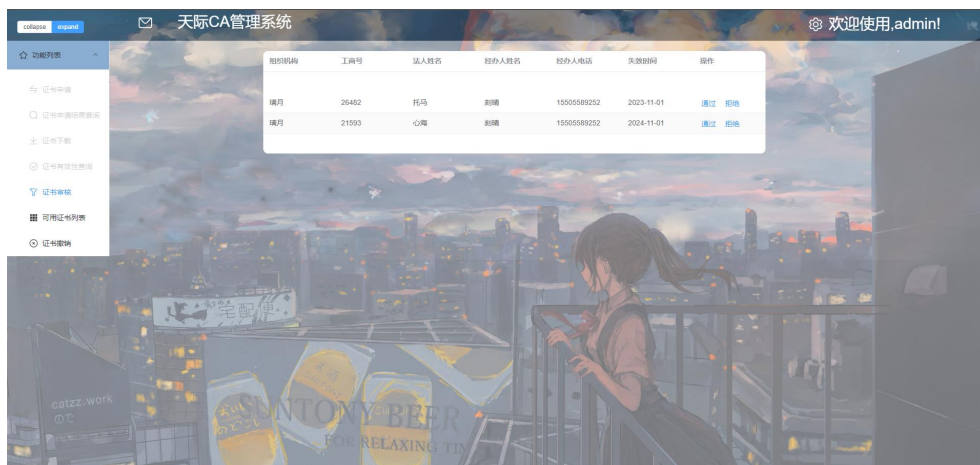
个人账号页面：



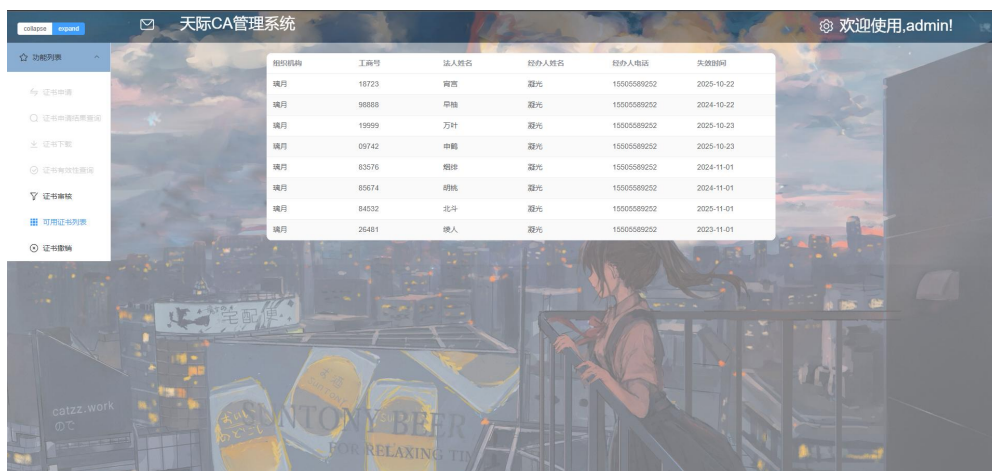
点击这个按钮，会向后端服务器发送请求，返回用户注册时登陆的信息。具体细节会在之后演示。

②管理员功能页面：

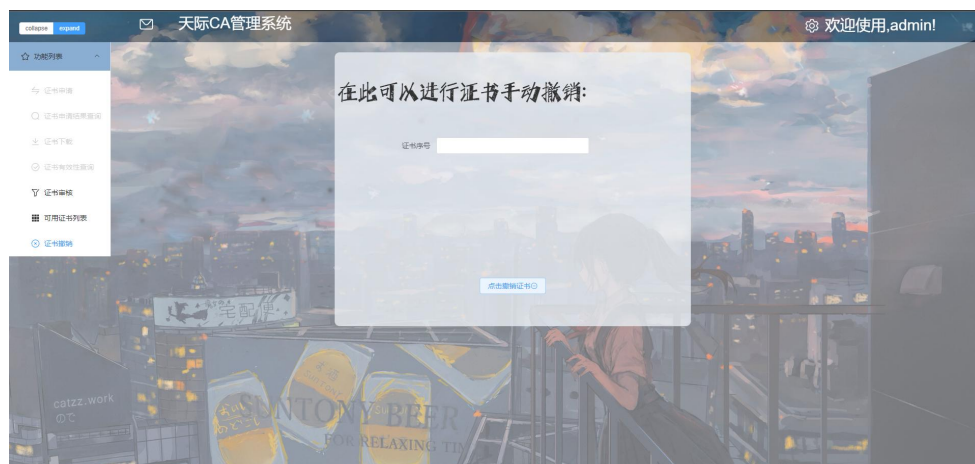
证书审核页面：



可用证书列表页面：



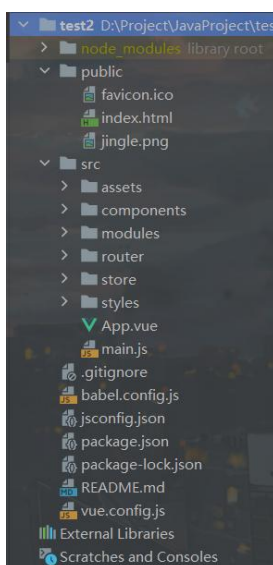
证书撤销页面：



注意到管理员的权限是最高的，可以撤销任何用户的证书。

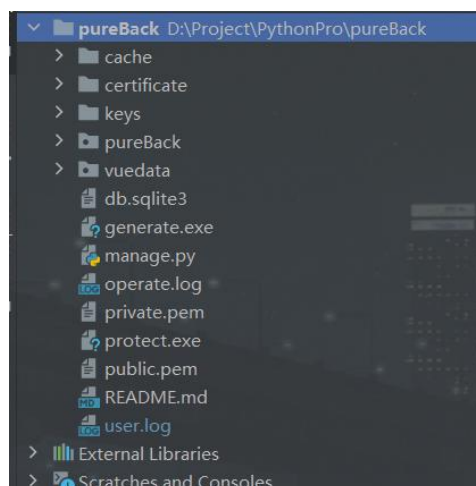
4.0.5 项目目录结构

在这一部分我们详细阐述一下本次项目的后台目录结构。
首先是 Vue（前端）部分：



注意到该目录基本符合通用的 Vue 项目结构目录，基本的各个组件均放置在了 components 文件夹中。modules 文件夹中包含 api.js 文件，该文件批量处理了后端的 url。router 文件夹中主要包含 index.js 文件，该文件规定了 Vue 组件的路由。store 文件夹中也包含一个 index.js 文件，但是该 index.js 文件主要是为了使用 pinia 而使用，即通过 pinia 可以实现前端多页面数据的共享。styles 和 assets 中放置了一些静态文件，如图片和 css 文件等。

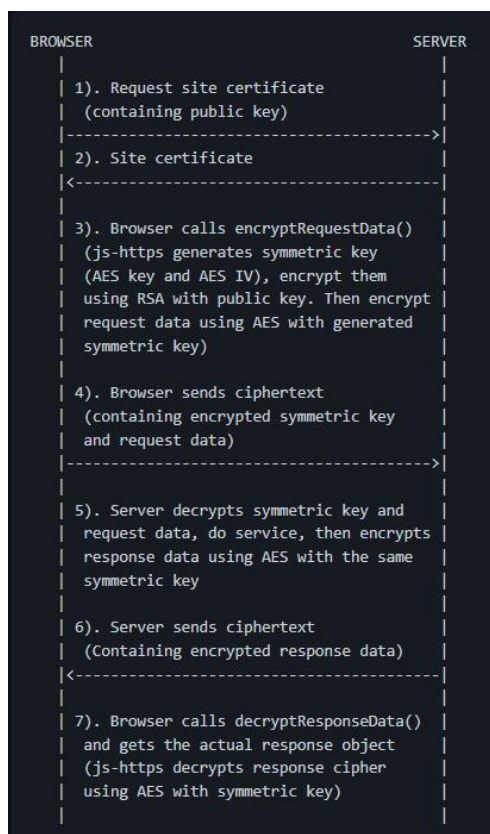
然后是 Django（后端）部分：



对于 Django 项目目录, 首先 pureBack 和 vuedata 都是 Django 的组件, pureBack 中主要包含各类后端处理 URL 与函数, 而 vuedata 主要是利用 Django 的 orm 数据库模型来操作本机 mysql 数据库。另外, cache 文件夹主要是为了处理缓存数据, 例如在后台共享某些数据。而 certificate 文件夹中主要存储生成的证书。在 keys 文件夹中存储了 CA 的公钥与私钥。另外, 一些文件如 generate.exe 与 protect.exe 为了方便起见放在了项目根目录下, 与之处于同一目录下的还有两个日志文件, 即 user.log 和 operate.log 方便管理员归档。

4.1 安全传输

我们先了解一下如何进行安全传输, 首先给出类似于 HTTPS 的工作流:



我们并未实现第一步与第二步，注意到浏览器需要去证实证书以确保其完整性可信性。

我们从第三步开始阐述该 workflow，大致流程如下：

第三步：前端使用 `encryptRequestData` 函数使用 CA 的 RSA 公钥对生成的 AES 密钥与初始向量进行加密，然后使用我们生成的 AES 密钥加密我们需要传输的数据。

第四步：前端发送密文（包含加密的对称密钥与请求数据）。

第五步：后端使用自己的 RSA 私钥解密对称密钥并解密请求数据，进行服务。然后使用相同的对称密钥对 `response` 数据做 AES 加密。

第六步：后端发送密文（包含加密的 `response` 数据）

第七步：前端使用 `decryptResponseData` 函数，并且得到真正的 `response` 对象。

4.1.1 混合加密+MAC 签名

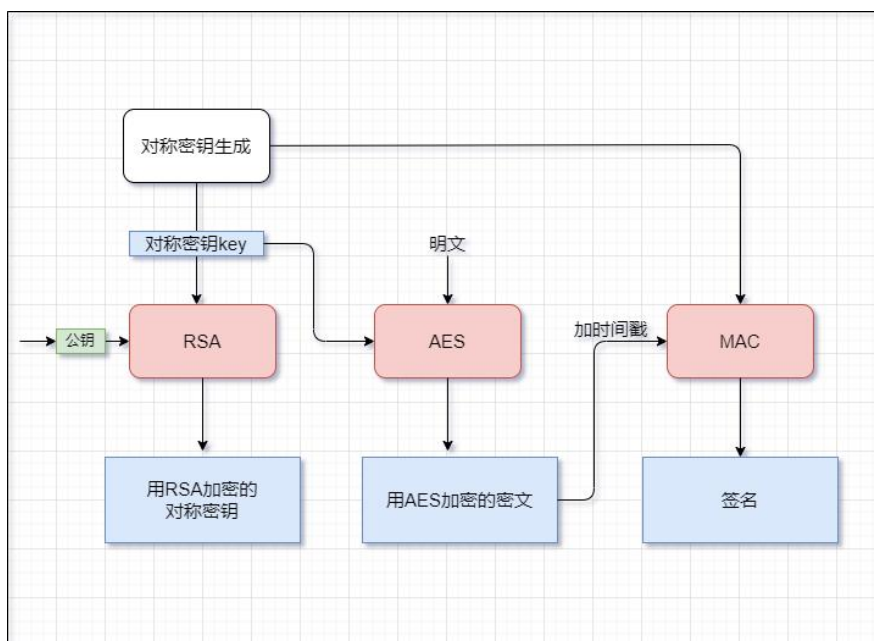


图 4.1 混合加密+MAC 签名算法

混合加密：首先进行对称密钥生成，使用 CA 的公钥采用 RSA 算法对对称密钥进行加密，构成密文的第一部分。然后使用对称密钥对数据使用 AES 算法进行加密，构成密文的第二部分。

签名：使用 Timed one-time passwords 设计思路，即采用密文加上时间戳，使用对称密钥，做一个 MAC。以此实现混合加密+签名。其中我们的签名具体等于 $MAC_k(c+t)$ 。在本次实验中，我们使用 SHA 算法进行实际的 MAC。

4.2 安全注册与登陆

4.2.1 注册与登录页面设计

欢迎界面&登陆界面：



图 4.2 欢迎界面&登陆界面

注意，在登陆界面中，我们可以根据用户所要登陆的账号，并根据数据库内已经存储的信息，来判断该账号是否是管理员账号，从而可以返回不同的操作页面。

注册界面：

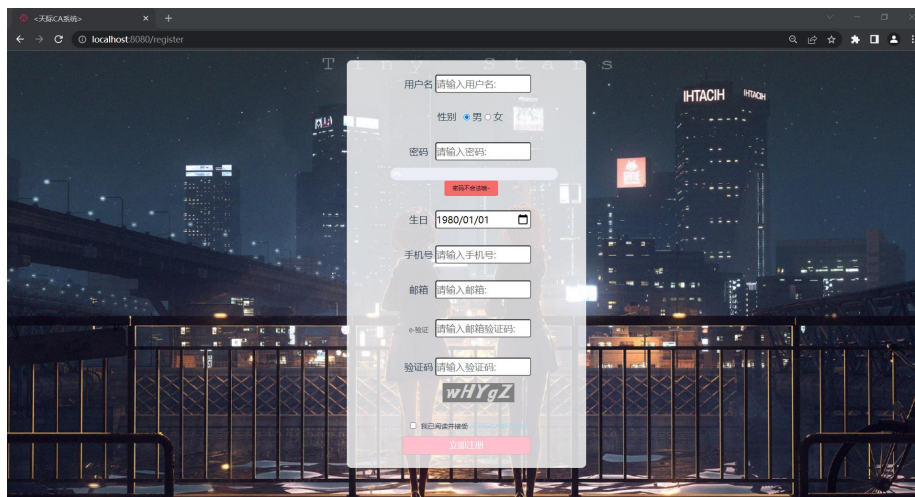


图 4.3 注册界面

注意，在注册界面中，除了通过图形验证码进行验证，我们加入了 e-验证功能（即通过 email 进行人机验证），该功能可以通过前端页面接收用户输入的邮箱，向该邮箱发送邮箱验证码，同时在前端保留该验证码，以便与用户将要输入的验证码进行比对。

在两个页面的设计中，我们通过图形验证码和 email 验证码的方式进行了人机验证，有效地降低了出现脚本自动注册造成产生大量‘僵尸账号’的场景的概率。

邮箱验证码功能演示：

点击“e 验证”标签，即可发送邮箱验证码：



您的注册验证码为:LMBwv,感谢您的使用!

图 4.4 邮箱验证码

当用户完成所有信息的正确填写，并且勾选“我已阅读并接受《天际 CA 使用须知》”之后，方可点击注册按钮。点击之后，若注册成功会跳出弹窗，提示点击跳转。

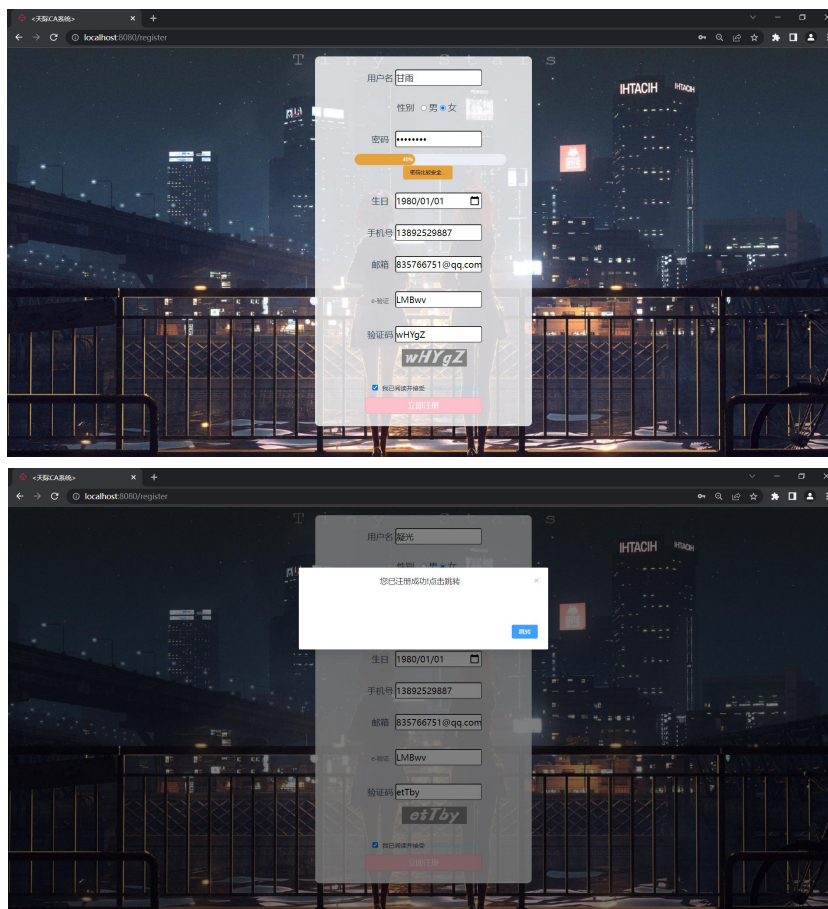


图 4.5 注册成功, 可以跳转

如果用户名冲突，则会注册失败：

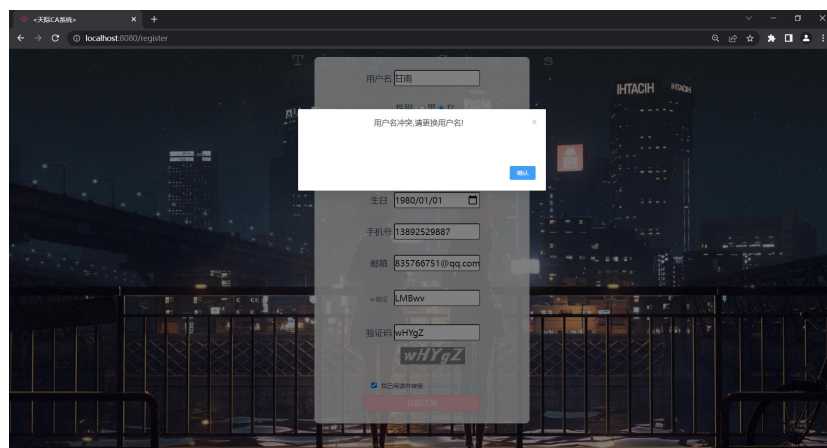


图 4.6 用户名冲突, 注册失败

另外，除了注册时需要使用的邮箱验证码外，我们也使用图形验证码进行人机检验，示意图如下：

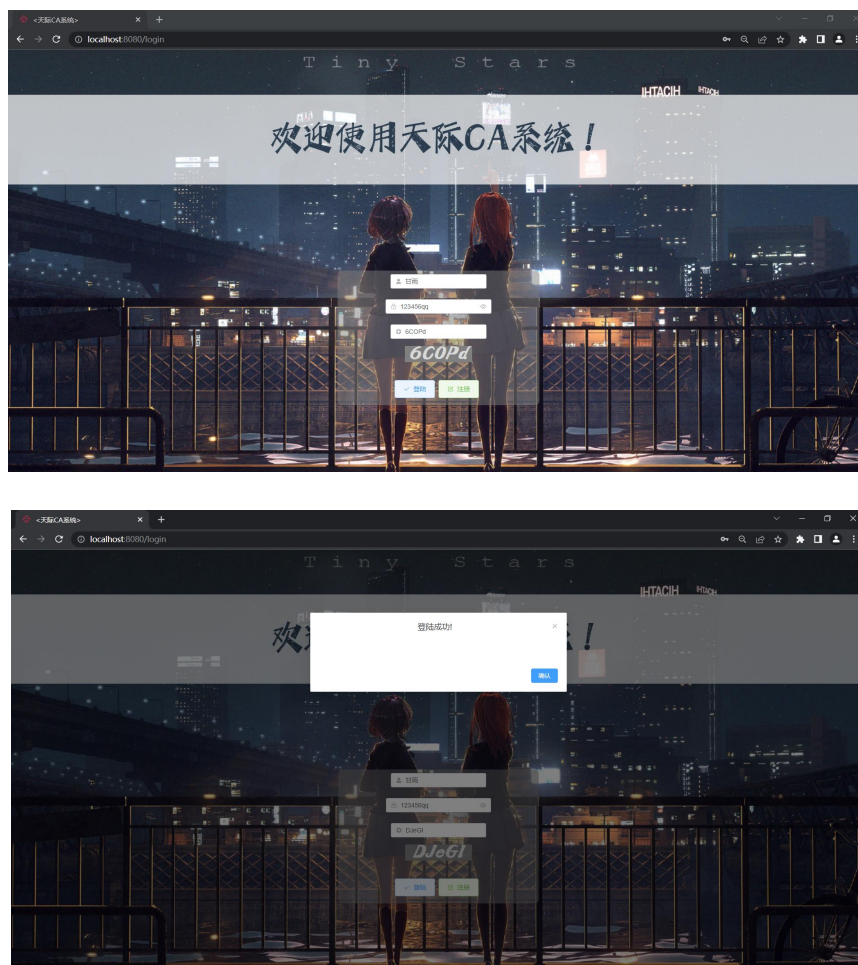


图 4.7 登陆成功!

4.2.2 用户密码存储安全

在后端我们对数据库内的密码进行加盐处理。这里我们调用了 `bcrypt` 库进行加盐操作。

id	用户名	加盐后的密码	盐值
6	admin	\$2b\$12\$0J.HPYCQWd.CuX5e6myrDu7C7TzheH...	\$2b\$12\$0J.HPYCQWd.CuX5e6myrDu
7	刻晴	\$2b\$12\$wB1WJwhN/mkUTk25.YE0ce/w6dPjWR...	\$2b\$12\$wB1WJwhN/mkUTk25.YE0ce
8	甘雨	\$2b\$12\$1HJTTW6J4KQir9Y/A1bc7UsoQTxPqY...	\$2b\$12\$1HJTTW6J4KQir9Y/A1bc7U
9	凝光	\$2b\$12\$TJH87i7L9c/DGA8cnAQ2UuLApC6L0D...	\$2b\$12\$TJH87i7L9c/DGA8cnAQ2Uu
10	夜兰	\$2b\$12\$pKXQThvm2XiLxLcdz75BK0uPkHvVE5...	\$2b\$12\$pKXQThvm2XiLxLcdz75BK0
14	绫华	\$2b\$12\$Rucp/m/5ShykTRR6L/jRXuRtcBK7by...	\$2b\$12\$Rucp/m/5ShykTRR6L/jRXu
15	钟离	\$2b\$12\$jvaVpK5FWLTxdbZqbFsgtuIpxNst0h...	\$2b\$12\$jvaVpK5FWLTxdbZqbFsgtu
16	万叶	\$2b\$12\$xVgpcmc60s.4J5mWZ8pDz.M2PRkC35...	\$2b\$12\$xVgpcmc60s.4J5mWZ8pDz.
17	海森	\$2b\$12\$RfUmyUwP5Hp2I.NVVs6pK0kbCL/xBM...	\$2b\$12\$RfUmyUwP5Hp2I.NVVs6pK0
18	瑶瑶	\$2b\$12\$sXLRx1Db70BkZVzyh.oyTebhjq6z.5...	\$2b\$12\$sXLRx1Db70BkZVzyh.oyTe

图 4.8 用户密码加盐存储

4.3 公私钥对生成和私钥保护

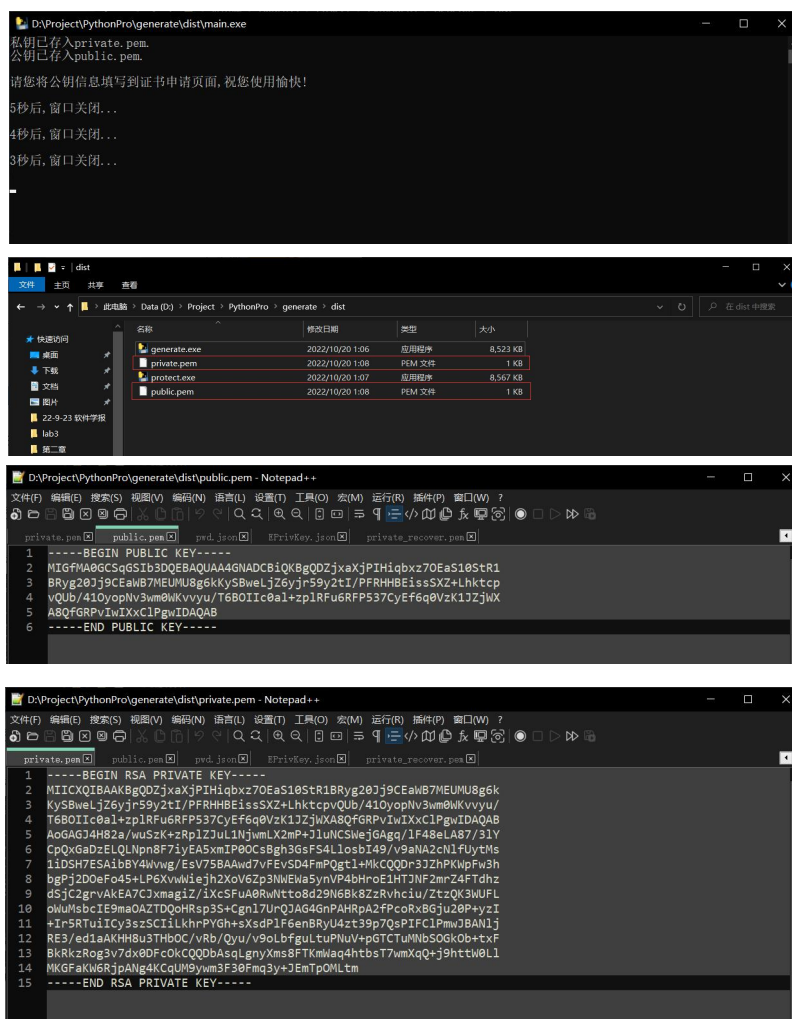
4.3.1 相关程序获取

在证书申请页面，下载密钥生成程序和私钥保护程序



4.3.2 公私钥对生成

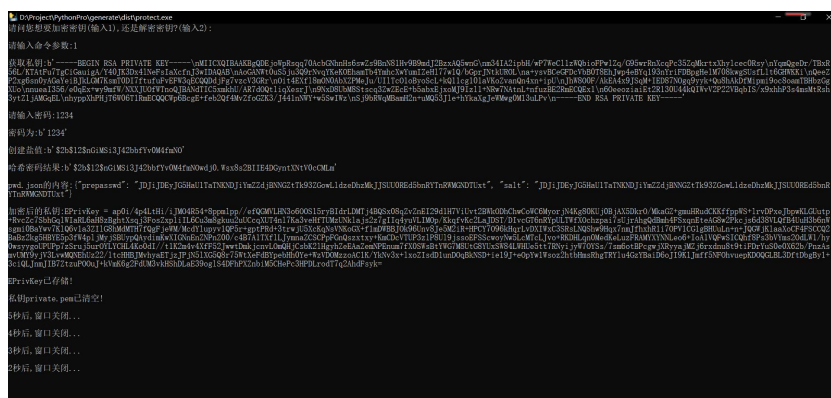
运行密钥生成程序得到公钥和私钥文件。



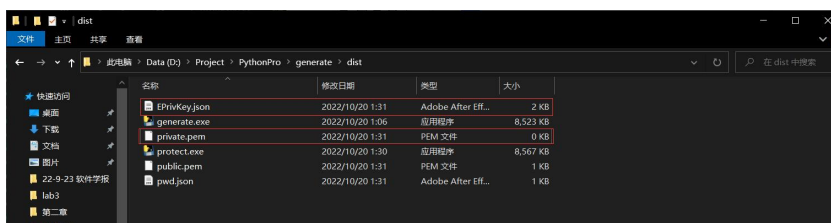
4.3.3 私钥保护

用户在本地运行私钥保护程序，加密自己的私钥。

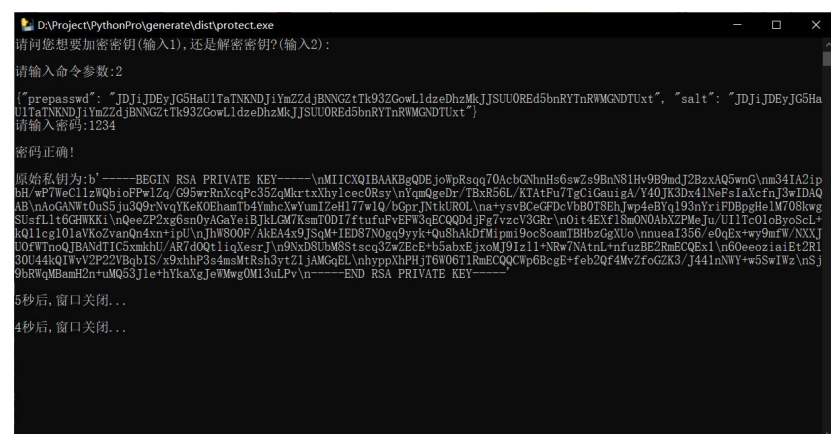
首先定义用户密码，程序随机生成一个 salt 对密码进行加盐。然后用 AES 算法对私钥进行加密，对加密后的私钥以及随机生成的盐值进行存储，最后清空原有的私钥文件。



文件中只剩下 `pwd.json` 以及加密后的私钥 `EPrivKey.json`，而原来的 `private.pem` 文件的大小已经变成了 0KB，表示内容已经被清空了。



用户若想获取私钥，首先在解密程序中输入原先定义的密码，读取 salt，对密码进行加盐，读取被加密的私钥，用 AES 算法进行解密，获得私钥，即可以使用。我们将恢复的私钥放在了 `private_recover.pem` 文件中，以示区别。



```

-----BEGIN RSA PRIVATE KEY-----
MIICXQIBAAKBgQDEjowpRsqg78AcBGNhHs6swZs9BnN81Hv9B9mdJ2BzxAQ5wnG
m34IA2ipbH/wP7WeCl1zWQb1oFPwLzq/G95wrRnXcqPc35zqMkrtXxhy1cec8Rsy
YqmQgeDr/TBxR56L/KTAtFu7TgCiGauigA/Y40JK3Dx41NeFsIaXcfnJ3wIDAQA
AoGANWt0uS5ju3Q9NvqYKeKOEhamTb4YmhcXWYumI2eH177w1Q/bGprJNtkUROL
a+ysvBCeGFDcVb0T8EhJwp4eBYq193nYriFDBpHe1M708kvgSUsfl1t6GHwKki
QeeZP2xg6sn0yAgaYeibJkLGM7KsmT0DI7ftufuFVFEW3qEQCQDdjFg7vzcV3GR
Oit4EXf18mON0AbXZPMeJu/UIITc01oByoScl+kQ11cg101aVKoZvanOn4xn+ipU
Jhw800F/AkEA4x9J5qM+IED87NOgg9yYk+Qu8hKdFh1pmi9oc8oamTBHbzGgXUo
nueaI356/e0qEx+wy9mFw/NXXJUOfwTnoQJBANDTIC5xmKHU/AR7d0Qt11qXesrJ
9Nx8UBM85stscq3ZwZEcE+b5abxExjxMj9Iz11+NRw7NAtNL+nfuzBE2RmECQEx1
6OeeoziaiEt2R130U44kQIwV2P22VBqIS/x9xhhP3s4msMtrsh3ytZ1jAMGqEL
hyppXhPHjT6W06T1RmECQqCwP6BcgE+Feb2Qf4MvZfoGZK3/J441nNWY+w5SwIwz
Sj9bRWqMBamH2n+uMQ53J1e+hYkaXgJewMwg0M13uLpv
-----END RSA PRIVATE KEY-----

```

4.4 签名与验证算法设计

4.4.1 签名算法



CA 获取当前时间与工商注册号后进行拼接，做哈希生成证书序号。选取证书核心信息，将证书序号、证书失效时间和公钥进行拼接作为代签信息，对代签信息做哈希，然后 CA 采用 RSA 算法加密得到签名。

证书格式如下：包括证书序号，证书颁发机构，所采用的签名算法，证书申请时间，证书失效时间，组织机构，公钥，以及证书签名。

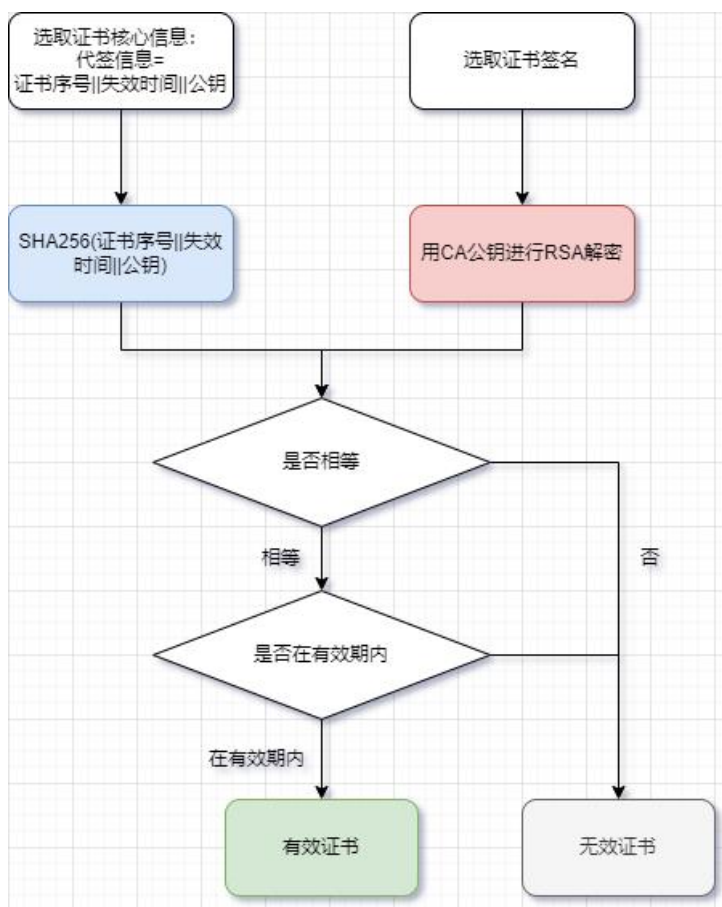
我们可以将该证书存储为.cert 文件或者.json 文件，存储成.cert 文件的格式如下：

```

D:\Project\PythonPro\generate\9109d605c0f1f1db49a6c7f792422b9d05bfd69b93fecb54b20673678088d7.cert - Notepad++
文件(F) 编辑(E) 搜索(S) 视图(V) 编码(N) 语言(L) 设置(T) 工具(O) 宏(M) 运行(R) 插件(P) 窗口(W) ?
9109d605c0f1f1db49a6c7f792422b9d05bfd69b93fecb54b20673678088d7.cert public.pem private.pem
1 Serial Number: 9109d605c0f1f1db49a6c7f792422b9d05bfd69b93fecb54b20673678088d7
2 天际CA系统:www.Skyrim.com
3 Sign Algorithm:sha256+RSA
4 Valid Time From:2022/10/22
5 Valid Time to:2023/10/22
6 User:刻晴
7 User PublicKey:-----BEGIN PUBLIC KEY-----
8 MIGfMA0GCsqGSIb3DQEBAQUAA4GNADCBiQKBgQDuH919x919mUgHf1pBkGGf9urM
9 u11YwKbxwe31wdS3aVg3GP99Civb3p7du4etBCPSvYUojmPKwsEn+N/9Ldtv72SS
10 2S0FvuF8+F7zsxoZ7vQV2xUFHCP1S0poBSWpug6a9/1hModZBw1igj5tE5H/JkHe
11 uIQxYAQyBsJ72j1IJQIDAQAB
12 -----END PUBLIC KEY-----
13 Sign:Krb7L2dAeSnzSqK31zR4B1q0Vjk0+w+2Kaz0vnkX809SCm6t1H2Ykow99g8A0E9viD8F59GoTuOv2K4ggx750tLNx6LQmQ+Sv4MH
14
Normal text file length: 677 lines: 14 Ln: 14 Col: 1 Pos: 678 Windows (CR LF) UTF-8 INS
    
```

为了便于程序处理，我们选择将这种一一映射的关系实际存储为 json 格式。

4.4.2 签名验证算法



签名验证算法主要分两步：第一步是验证签名，第二步是验证有效期。

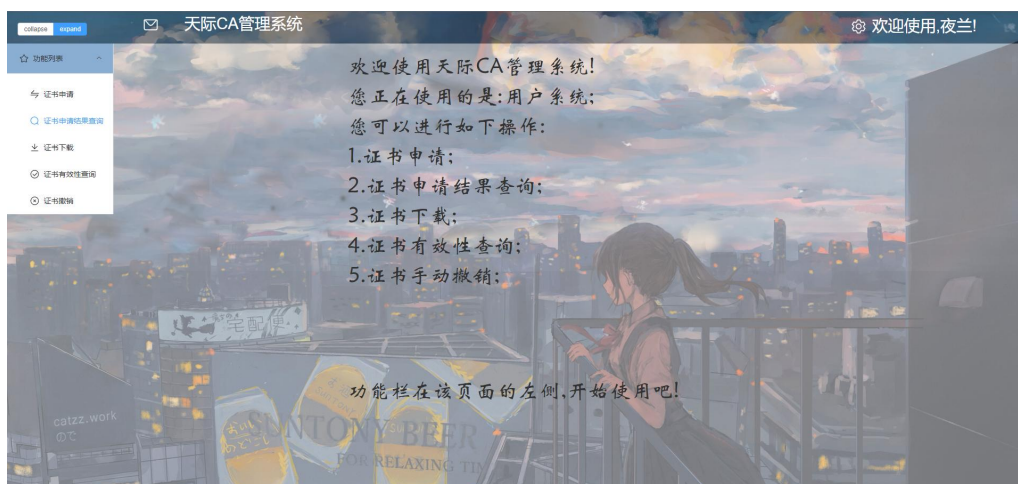
第一步，进行签名验证，首先选取证书核心信息，此处要和后端的签名算法一致。将证书序号、失效时间以及公钥进行拼接，作为代签信息，对代签信息做哈希。然后选取证书签名，用 CA 公钥进行 RSA 算法解密。判断得到的两个结果是否相等，相等则签名验证成功，否则签名验证失败。

第二步，获取失效时间，判断当前是否在有效期内，两步均通过才为有效证书，否则均为无效证书。

4.5 证书申请与审批

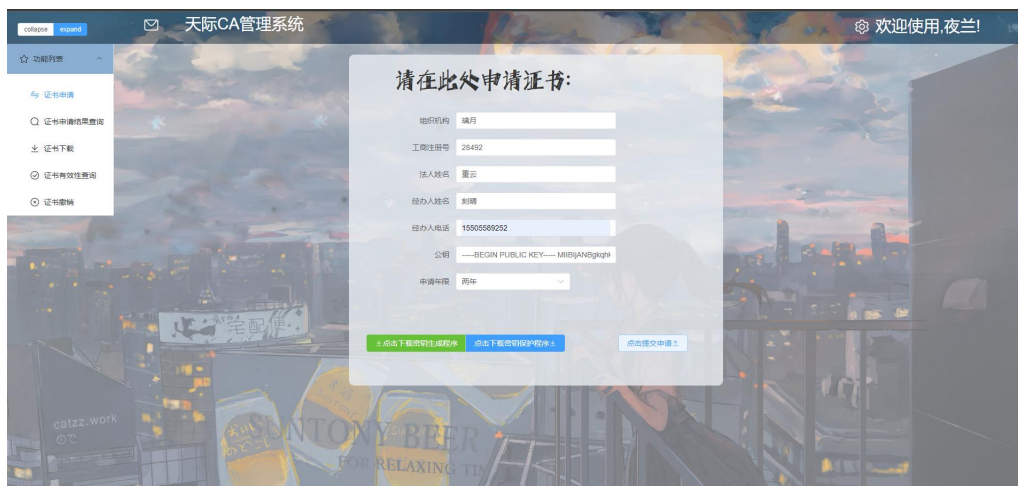
4.5.1 证书申请页面设计

系统欢迎界面，对我们的单页面应用的使用方法进行了简要的介绍。



4.5.2 用户提交申请

用户填写申请表单，提交申请，消息内容经过混合加密+签名发送到后端。



后端解密后, 在 `apply` 表中存储申请信息

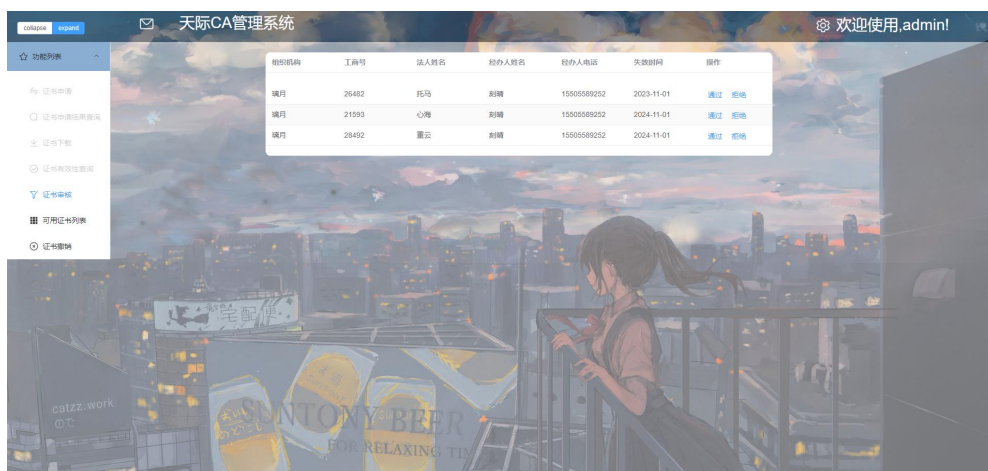
id	RegistrationNumber	SerialNumber	Organization	StartTime	EndTime	JuridicalPerson	ChangeF
1	26 26482	9c33c791b50f9ca428f28aa34376...	璃月	2022-11-01	2023-11-01	托马	刻晴
2	27 21593	aa8572a19282891966d9ca1fddae...	璃月	2022-11-01	2024-11-01	心海	刻晴
3	28 28492	7ebcbf854ef9f7151ff407334a23...	璃月	2022-11-01	2024-11-01	重云	刻晴

当成功发送申请信息之后, 前端会打印汇总到的申请信息通知用户, 便于用户记录检查。



4.5.3 管理员审核

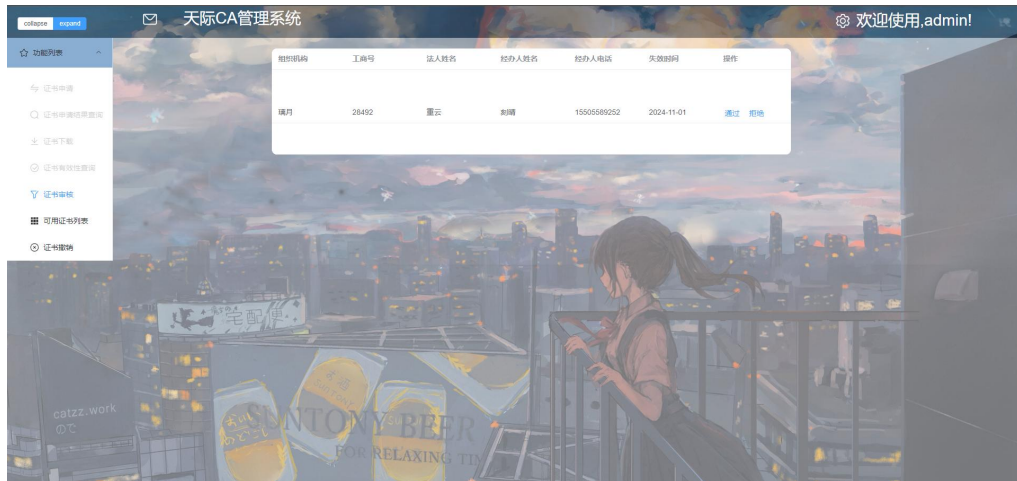
管理员登录后, 获取待审批的证书申请。



在页面挂载的时候我们发送了 `axios` 请求, 并从后端的响应中得到了相应的数据, 符合我们之前所说的安全工作流程。

管理员可以点击通过以发送 `axios` 请求以提交审核结果, 并将原 `apply` 表中的表项删除, 加入到 `cert` 表中。同时后端返回给前端以更新后的 `apply` 列表。当然, 管理员也可以点击拒绝, 以直接将原 `apply` 表中的表项删除, 加入到 `cri` 表中。拒绝给用户颁发证书。

下面我们就接收‘法人姓名’字段为‘心海’的申请, 而拒绝为‘托马’的申请, 操作后页面自动更新后的结果如下:



可见只剩下重云字段。

我们在后端观察 apply 表，cert 表与 crl 表，结果如下：

①待审核信息从 apply 表中删除。

```

WHERE
ORDER BY
id : RegistrationNumber : SerialNumber : Organization : StartTime : EndTime : JuridicalPerson : Charge
1 28 28492 7ebcbf054ef9f7151ff407334a23... 璃月 2022-11-01 2024-11-01 重云 刻晴
    
```

②证书信息加入 cert 表。

```

WHERE
ORDER BY
id : RegistrationNumber : SerialNumber : Organization : StartTime : EndTime : Jurid
1 4 18723 c4188cb4b9207f76ad6de11f372122d4c1e09ac6085bae5e8... 璃月 2022-10-22 2025-10-22 宵宫
2 5 98888 c131dbb5108d2fbfc2f17f5be0160e370f57c38f1567033c... 璃月 2022-10-22 2024-10-22 早柚
3 7 19999 177d71211f111389b7dbffbb2f7be718b8a051a060958bfa8... 璃月 2022-10-23 2025-10-23 万叶
4 23 09742 4167f4324b24cff3ba780e9e355fe6f45e660951f1694f8d6... 璃月 2022-10-23 2025-10-23 申鹤
5 25 83576 04bc76a6eb7439caf3fa3ed9e8dc240fdbb9b7c3b39414322... 璃月 2022-11-01 2024-11-01 燧绯
6 26 85674 b149ace78f901eda794d88f54ebad12d401005a134b546f0f... 璃月 2022-11-01 2024-11-01 胡桃
7 27 84532 7f27f1fc472b62deae9cac5fbacc2cd6af433a1f14865def0... 璃月 2022-11-01 2025-11-01 北斗
8 28 26481 a3f6412ea07452935f14cfee3d218330b85b04c308724203e... 璃月 2022-11-01 2023-11-01 绀人
9 30 21593 aa0572a19202091966d9ca1fddae4a7fbc9ddcbaf5a2659ec... 璃月 2022-11-01 2024-11-01 心海
    
```

③拒绝的证书放入 crl 表。

```

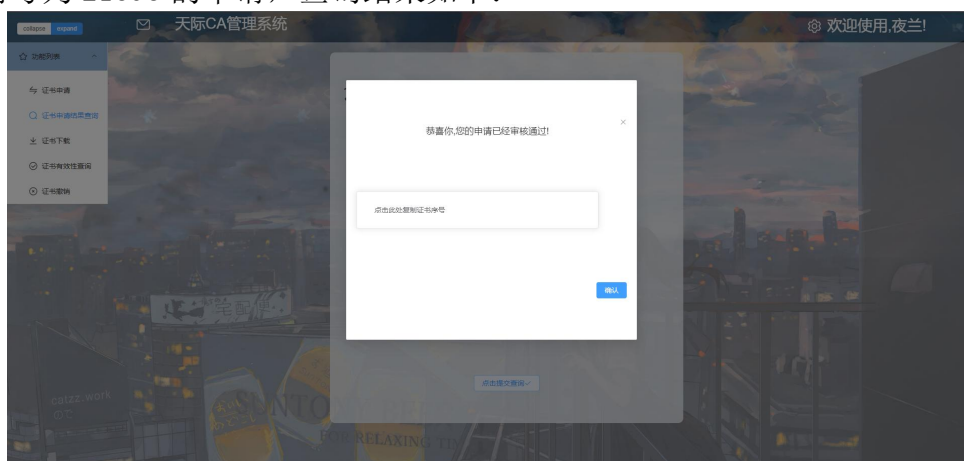
WHERE
ORDER BY
id : SerialNumber : Organization : RevokeTime
1 59a736db4ede43c68a5782954536e9c243454ca4118f51289b76dfdbc6464af3 璃月 2022-10-22
2 ae175ce0387fb817a6b220021834925c75aeb224bf3a5abac15ce83a21f9c23d 璃月 2022-10-22
3 2bd530e7e01247b07809b9f4d8ef33f24f24c8c12b7d4c4b734b13fdcb39371 璃月 2022-10-23
4 8463286f1affd1f1c478f6f04414559fcb3f6f53df8cf862cc1858c1c1327b23 璃月 2022-10-23
5 274867cb6ad4f0c41c5b28498517cea9afa8fc5626587153b5891f55a2af360c 哈工大 2022-11-01
6 f14ea4f0a4b7eaccbb96693bbdb6d68fbf4e35743e2f4c4318b00e2a04ee7f50 璃月 2022-11-01
7 265c6318111068da58f0e5ab34198354579de88009ef6071954b96d7fe5b5cf6 璃月 2022-11-01
    
```

④证书在 BASE_DIR 文件夹下存储，便于获得

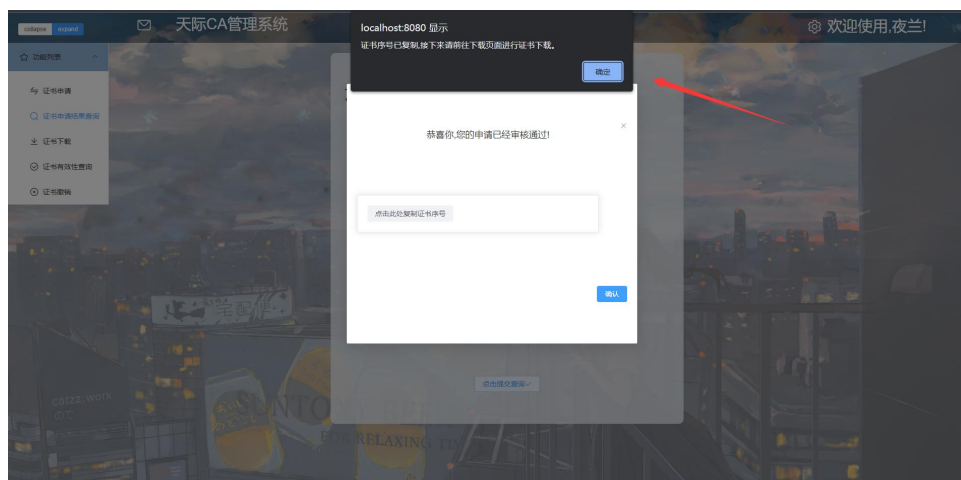
名称	修改日期	类型	大小
.git	2022/11/1 22:33	文件夹	
.idea	2022/11/2 0:08	文件夹	
__pycache__	2022/10/22 16:27	文件夹	
keys	2022/11/1 19:00	文件夹	
pureBack	2022/11/2 0:10	文件夹	
vuedata	2022/11/1 19:50	文件夹	
3abd804ca7a62584f4459fd5bd5f9c8c46be932531fef1769035ad3002bf9d03.json	2022/11/2 0:09	Adobe After Eff...	2 KB
7f27f1fc472b62deae9cec5fbacc2cd6af433a1f14865def0a669f536fc498d3.json	2022/11/1 20:46	Adobe After Eff...	2 KB
177d71211f11389b7dbffbb2f7be718b8a051a060958bfa8cfc59d323c655f7.json	2022/10/23 0:42	Adobe After Eff...	1 KB
4167f4324b24cff3ba780e9e355fe6f45e660951f1694f8d68226583379daa8b.json	2022/10/23 9:25	Adobe After Eff...	1 KB
a3f6412ea07452935f14cfce3d218330b85b04c308724203e9e0eee37ce9445c.json	2022/11/1 20:46	Adobe After Eff...	2 KB
b149ace78f901eda794d88f54ebad12d401005a134b546f0fe7ea4c9ef69a4fa.json	2022/11/1 20:45	Adobe After Eff...	2 KB
db.sqlite3	2022/10/14 18:32	SQLite3 文件	0 KB
generate.exe	2022/11/1 18:45	应用程序	8,523 KB
manage.py	2022/10/14 18:30	Python File	1 KB
operate.log	2022/11/2 0:09	文本文档	2 KB
private.pem	2022/11/1 20:41	PEM 文件	2 KB
protect.exe	2022/10/20 10:40	应用程序	8,567 KB
public.pem	2022/11/1 20:41	PEM 文件	1 KB
user.log	2022/11/1 23:06	文本文档	4 KB

4.5.4 用户申请结果查询

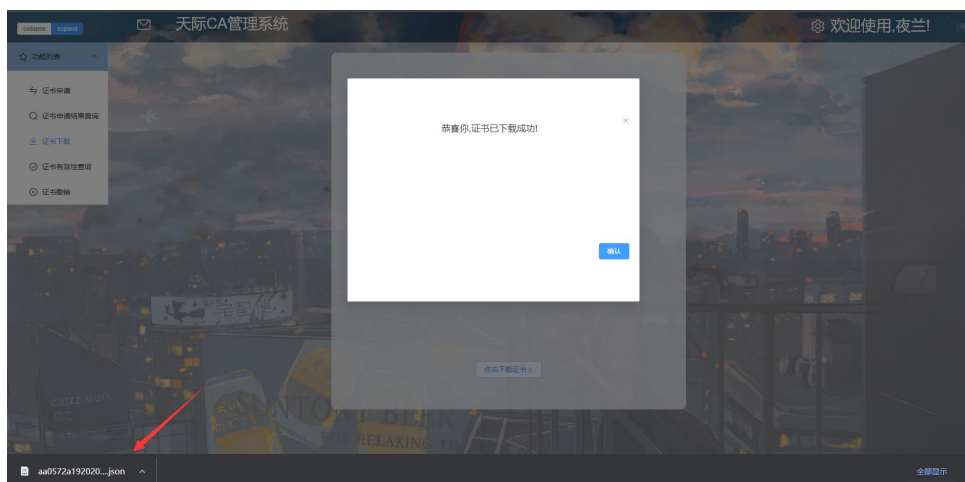
用户可以通过工商注册号查询申请结果, 并获取证书序号。我们这里查询工商注册号为 21593 的申请, 查询结果如下:



查询成功之后可以点击窗口的复制按钮进行一键复制, 效果如下:



获得了证书序号之后, 我们前往证书下载页面进行证书的下载即可: 通过证书序号进行证书下载:



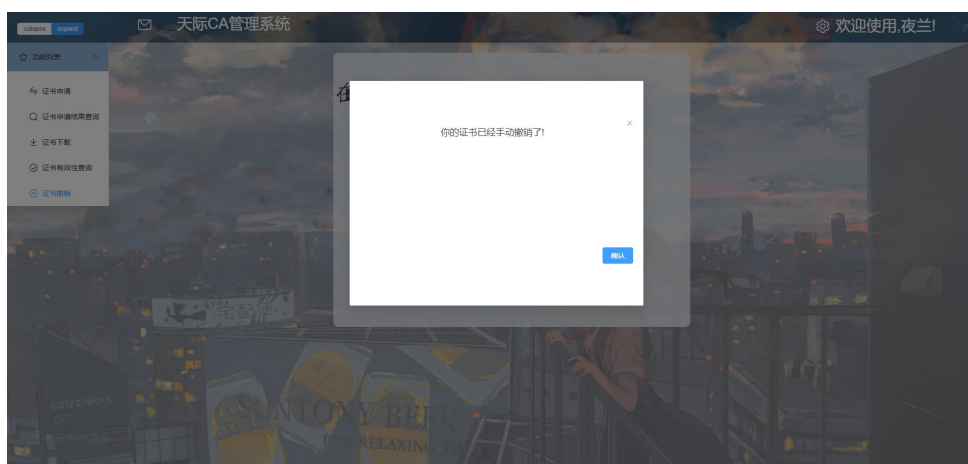
4.6 证书撤销

用户提交证书撤销申请, 经过混合加密+签名后发给后端; 后端进行解密+签名验证, 并验证用户身份。

后端判断: 证书的申请者和撤销者必须为同一人, 或者撤销者为 **admin** 管理员, 才能够进行撤销。

id	RegistrationNumber	UserName	PublicKey	CertPathName
1	4 18723	育富	-----BEGIN PUBL...	1
2	5 98888	早袖	-----BEGIN PUBL...	
3	7 19999	万叶	-----BEGIN PUBL...	
4	23 09742	申鹤	-----BEGIN PUBL...	
5	25 83576	烟绯	-----BEGIN PUBL...	
6	26 85674	胡桃	-----BEGIN PUBL...	
7	27 84632	北斗	-----BEGIN PUBL...	
8	28 26481	绀人	-----BEGIN PUBL...	
9	30 21593	心海	-----BEGIN PUBL...	
10	31 28492	重云	-----BEGIN PUBL...	
11	32 28361	行秋	-----BEGIN PUBL...	

下面我们以 **UserName** 为‘夜兰’的账号撤销法人姓名为‘重云’的证书: 在页面内输入证书序号后, 发现弹窗提示证书已经被手动撤销了。

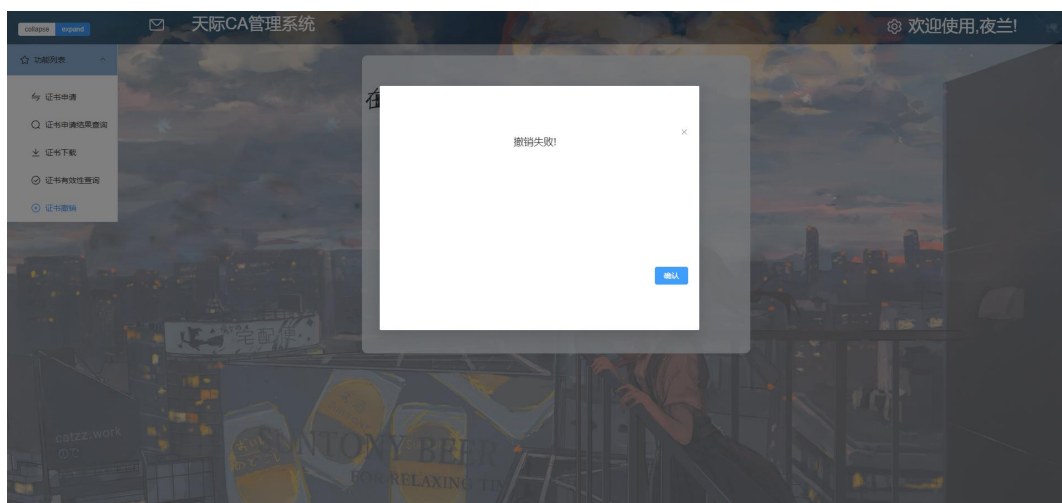


将 **cert** 表中的项删除, 并删除证书文件, 将证书序号和撤销时间存入 **crl** 表。对应项在 **cert** 表中被删除, 由下图可知 **cert** 表中已不存在该证书, 而在 **crl** 表中出现了该证书的历史信息。

id	RegistrationNumber	CentPathName	UserName	PublicKey	CentPathName
4	18723	c4188...	璃月 .. 20... 宵宫
5	98888	c131d...	璃月 .. 20... 早柚
7	19999	177d7...	璃月 .. 20... 万叶
23	09742	4167f...	璃月 .. 20... 申鹤
25	83576	04bc7...	璃月 .. 20... 烟绯
26	85674	b149a...	璃月 .. 20... 胡桃
27	84532	7f27f...	璃月 .. 20... 北斗
28	26481	a3f64...	璃月 .. 20... 绀人
30	21593	aa057...	璃月 .. 20... 心海
32	28361	3abd8...	璃月 .. 20... 行秋

id	SerialNumber	Organization	RevokeTime
1	59a736db4ede43c68a5782954536e9c243454ca4118f51289b76dfdbc6464af3	璃月	2022-10-22
2	ae175ce0387fb817a6b220021834925c75aeb224bf3a5abac15ce83a21f9c23d	璃月	2022-10-22
3	2bd530e7e01247b07809b9f4d8ef33f24f24c8c12b7d4c4b734b13fdbcbe39371	璃月	2022-10-23
4	8463286f1affd1f1c478f6f04414559fcb3f6f53df8cf862cc1858c1c1327b23	璃月	2022-10-23
5	274867cb6ad4f0c41c5b28498517cea9afa8fc5626587153b5891f55a2af360c	哈工大	2022-11-01
6	f14ea4f0a4b7eaccbb96693bbdb6d68fbf4e35743e2f4c4318b00e2a04ee7f50	璃月	2022-11-01
7	265c6318111068da58f0e5ab34198354579de88009ef6071954b96d7fe5b5cf6	璃月	2022-11-01
8	9c33c791b50f9ca428f28aa34376803a0e3f2a8c95b97710f39c6edf59a9eebf	璃月	2022-11-01
9	7ebcbf054e9f7151ff407334a2350cf4238d2897b72b962b53e5c0742ca42df	璃月	2022-11-01

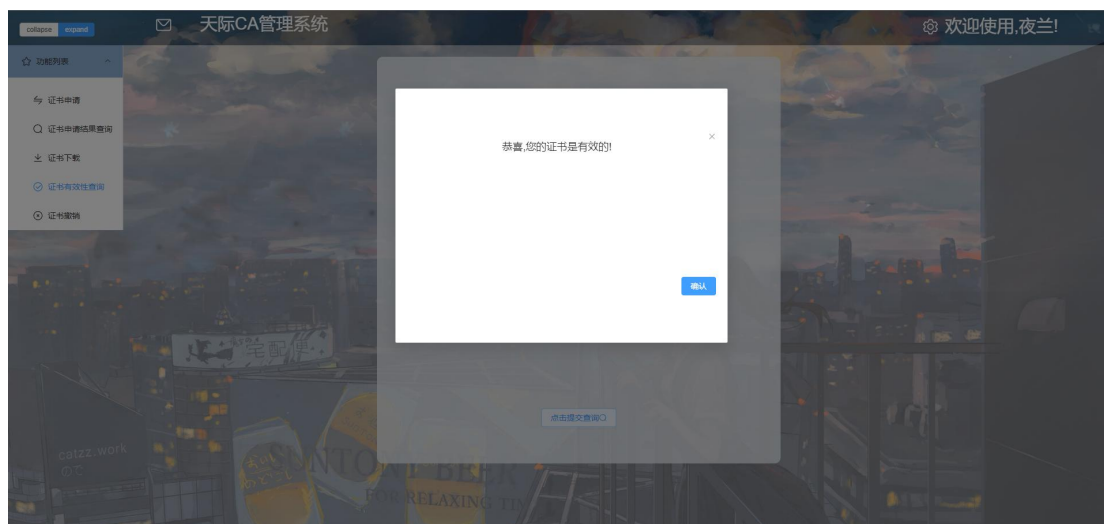
此时我们尝试撤销法人姓名为‘北斗’的证书（由图上可知其并非由用户‘夜兰’申请），输入证书序列号后，弹窗如下：



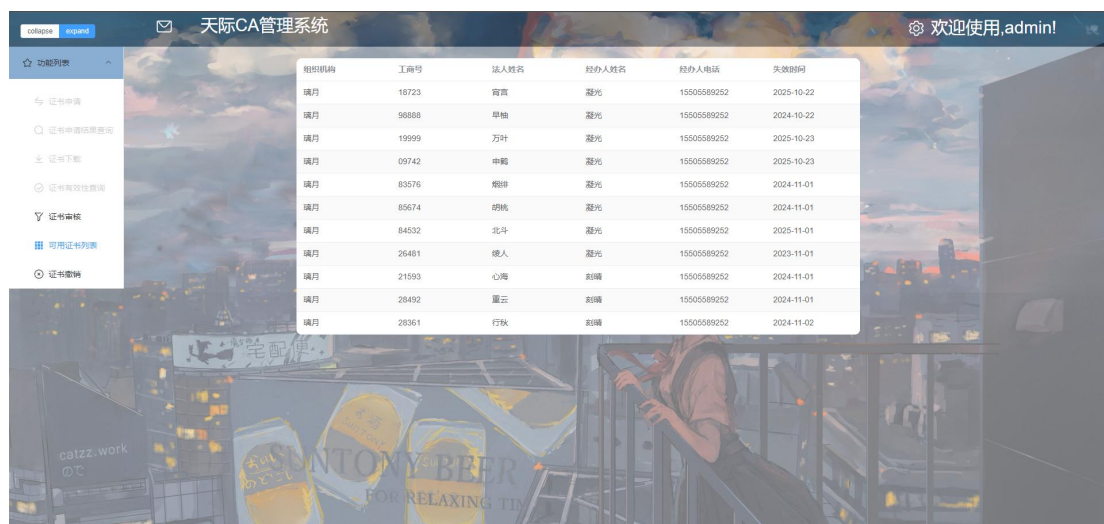
可见由于权限限制，CA 中心拒绝了用户的该次撤销，并且该证书依然存在于 cert 表中。

4.7 在线验证证书有效性(OCSP)

用户输入证书序号进行查询：



而管理员可以直接看到所有的合法证书列表:



4.8 系统日志

在本系统中，我们存放了两个日志，其一为 user 日志，其用于存放用户注册、登入、登出的信息；其二为 operate 日志，其用于存放用户和管理员所做的关于证书相关的操作，如：申请、审批、撤销。具体日志信息如下所示，二者都用“时间-操作-用户名”的方式记录情况。

```

File Edit View Navigate Code Refactor Run Tools Git Window Help pureback - user.log
user.log
60 user-2022-10-23 09:32:21,749-[下线]:刻晴
61 user-2022-10-23 09:32:37,967-[登陆]:续华
62 user-2022-10-31 23:29:08,016-[登陆]:刻晴
63 user-2022-11-01 17:44:55,909-[登陆]:刻晴
64 user-2022-11-01 17:45:10,040-[下线]:刻晴
65 user-2022-11-01 17:45:19,455-[登陆]:admin
66 user-2022-11-01 17:56:18,180-[下线]:admin
67 user-2022-11-01 17:58:18,887-[登陆失败]
68 user-2022-11-01 17:58:46,267-[登陆]:夜兰
69 user-2022-11-01 18:21:34,043-[登陆失败]
70 user-2022-11-01 18:21:41,971-[登陆]:夜兰
71 user-2022-11-01 19:37:15,245-[登陆]:刻晴
72 user-2022-11-01 19:38:52,686-[登陆]:刻晴
73 user-2022-11-01 19:53:23,146-[登陆失败]
74 user-2022-11-01 19:53:35,189-[登陆失败]
75 user-2022-11-01 19:53:54,304-[登陆]:admin
76 user-2022-11-01 19:56:32,197-[登陆]:刻晴
77 user-2022-11-01 20:44:06,056-[登陆]:admin
78 user-2022-11-01 21:27:56,588-[登陆]:admin
79 user-2022-11-01 22:18:14,462-[登陆]:续华
80 user-2022-11-01 22:24:29,934-[登陆失败]
81 user-2022-11-01 22:24:40,667-[登陆]:admin
82 user-2022-11-01 22:25:54,669-[登陆失败]

operate.log
20 operate-2022-11-01 20:45:39,414-[通过]:admin
21 operate-2022-11-01 20:46:07,141-[通过]:admin
22 operate-2022-11-01 20:46:10,487-[通过]:admin
23 operate-2022-11-01 20:48:02,281-[下载]:刻晴
24 operate-2022-11-01 20:49:15,243-[下载]:刻晴
25 operate-2022-11-01 20:52:05,481-[下载]:刻晴
26 operate-2022-11-01 21:11:33,967-[申请]:鹿野苑
27 operate-2022-11-01 22:15:16,282-[通过]:admin
28 operate-2022-11-01 22:19:35,408-[申请]:九条
29 operate-2022-11-01 22:20:12,527-[拒绝]:admin
30 operate-2022-11-01 22:31:43,465-[撤销]:admin 265c6318111
31 operate-2022-11-01 23:07:05,817-[申请]:托马
32 operate-2022-11-01 23:07:33,013-[申请]:心海
33 operate-2022-11-01 23:54:58,007-[申请]:重云
34 operate-2022-11-02 00:02:31,561-[拒绝]:admin
35 operate-2022-11-02 00:02:34,828-[通过]:admin
36 operate-2022-11-02 00:08:33,417-[通过]:admin
37 operate-2022-11-02 00:09:11,249-[申请]:行秋
38 operate-2022-11-02 00:09:20,530-[通过]:admin
39 operate-2022-11-02 00:18:06,472-[下载]:夜兰
40 operate-2022-11-02 00:22:47,440-[下载]:夜兰
41 operate-2022-11-02 00:24:37,244-[下载]:夜兰
42 operate-2022-11-02 00:24:51,369-[下载]:夜兰
43 operate-2022-11-02 00:25:52,145-[下载]:夜兰

```

5. 实现

5.1 技术栈选择

采用 Vue+Element-plus 组件库+Django+MySql 进行前后端开发。

- 前端页面: Vue, Element-plus, CSS
- 前端业务代码: Vue, JavaScript, pinia
- 前后端交互: Vue, Axios
- 后端业务代码: Python, Django
- 数据库操作: Django Orm, MySql

5.2 图形/邮件验证码

5.2.1 邮件验证码

由于前端无法使用 smtp 协议, 所以我们需要借助后端来发送这个请求。
前端: 发送邮箱给后端。

```
sendMyEmail() {
  console.log(this.email);
  axios.post(APIIS.ecode, {
    email: this.email,
  }).then(res => {
    console.log(res.data);
    this.trueEmailCode = res.data.captcha;
  }).catch(reason => {
  }).finally(() => {
  })
},
```

后端: 调用自带的 SMTPlib 包发送验证码, 邮件发送并且可以完成注册。

```
@csrf_exempt
def register_email(request):
    from_addr = '565852435@qq.com'
    password = 'gplblhnga1nybfhe'
```

```
req = json.loads(request.body)
to_addr = req['email']

smtp_server = 'smtp.qq.com'

list1 = list(range(97, 123))
list2 = list(range(65, 91))
list3 = list(range(48, 58))
list4 = list1 + list2 + list3
# 总共 62 个元素
space = [chr(i) for i in list4]
captcha = ''
for i in range(0, 5):
    captcha = captcha + (space[random.randint(0, 61)])
print(captcha)
msg = MIMEText(f'您的注册验证码为:{captcha},感谢您的使用!',
'plain', 'utf-8')
msg['From'] = Header('天际 CA 管理系统')
msg['To'] = Header('用户')
msg['Subject'] = Header('天际 CA 管理系统验证邮件')

server = smtplib.SMTP_SSL(smtp_server)
server.connect(smtp_server, 465)

server.login(from_addr, password)

server.sendmail(from_addr, to_addr, msg.as_string())
# 关闭服务器
server.quit()
return JsonResponse({
    "success": True,
    "captcha": captcha,
})
```

5.2.2 图形验证码

图形验证码可以完全由前端自主生成。具体细节如下：

```
const createCode = function (length) {
    var code = "";
    var codeLength = parseInt(length); //验证码的长度
    var checkCode = document.getElementById("checkCode");
    //所有候选组成验证码的字符，当然也可以用中文的
    var codeChars = new Array(0, 1, 2, 3, 4, 5, 6, 7, 8, 9,
        'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l',
        'm', 'n', 'o', 'p', 'q', 'r', 's', 't', 'u', 'v', 'w', 'x', 'y',
```



```
'z', 'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L',  
'M', 'N', 'O', 'P', 'Q', 'R', 'S', 'T', 'U', 'V', 'W', 'X', 'Y',  
'Z');  
    //循环组成验证码的字符串  
    for (var i = 0; i < codeLength; i++) {  
        //获取随机验证码下标  
        var charNum = Math.floor(Math.random() * 62);  
        //组合成指定字符验证码  
        code += codeChars[charNum];  
    }  
    if (checkCode) {  
        //为验证码区域添加样式名  
        checkCode.className = "code";  
        //将生成验证码赋值到显示区  
        checkCode.innerHTML = code;  
    }  
    return code;  
}
```

5.3 混合加密+签名

5.3.1 前端混合加密+签名:

```
encryptRequestData(data: object, publicKey: string):  
EncryptedRequestParam {  
    const rsaEncryptor = new JSEncrypt();  
    rsaEncryptor.setPublicKey(publicKey);  
  
    // AES-128  
    this.aesKey = cryptoRandomString({ length: 16 });  
    // Must be 16 Bytes  
    this.aesIv = cryptoRandomString({ length: 16 });  
  
    const aesKeyCipher = rsaEncryptor.encrypt(this.aesKey) as  
string;  
    const aesIvCipher = rsaEncryptor.encrypt(this.aesIv) as string;  
  
    return {  
        bodyCipher:  
aesEncrypt(JSON.stringify(data), this.aesKey, this.aesIv),  
        ksCipher: aesKeyCipher,  
        ivCipher: aesIvCipher
```

```
};
}
```

在实现防窃听之后，我们对该加密添加相应的 mac，具体模型如下：

```
onClick() {
  const adminpublickey = this.store.publickey
  const jsHttps = new JsHttps();
  const myRequestData = {
    test: "hello!!"
  }
  var encdata=jsHttps.encryptRequestData(myRequestData,
adminpublickey)
  const mac={
    mac: CryptoJS.SHA1(encdata.bodyCipher).toString()
  }
  const resdata={
    data:encdata,
    resmac:mac
  }
  axios.post(APIS.test, resdata
).then(res => {
  console.log(jsHttps.decryptResponseData(res.data));
}).catch(reason => {
  console.log(reason);
}).finally(() => {
  console.log("FINALLY");
})
}
}
}
```

5.3.2 后端对称密钥加密响应：

```
def encrypt_response_data(self, response_data) -> str:
  if self.aes_key == b"" or self.aes_iv == b"":
    return ""

  response_str = json.dumps(response_data, ensure_ascii=False)
  encrypted_bytes = crypto_helper.aes_encrypt(self.aes_key,
                                             self.aes_iv,
                                             response_str.encode(encoding="UTF-8"))
  return base64.b64encode(encrypted_bytes).decode("UTF-8")
```

5.3.3 前端解密:

```
decryptResponseData(data: string): object|null {
  if (this.aesKey === "" || this.aesIv === "") {
    return null;
  }
  return JSON.parse(aesDecrypt(data, this.aesKey, this.aesIv));
}
```

注意到因为 HTTP 在传输层使用的是 TCP 协议，所以我们在链接建立之后就可以相互确定发送方与接收方的身份。

5.3.4 后端解密:

```
def decrypt_request_data(self, encrypted_request):
    sk: bytes = b""
    with open(os.path.join(settings.BASE_DIR,
"keys/adminprivate.pem"), "br") as f:
        sk = f.read()
        aes_key: bytes = crypto_helper.rsa_decrypt(sk,
base64.b64decode(encrypted_request["ksCipher"]))
        aes_iv: bytes = crypto_helper.rsa_decrypt(sk,
base64.b64decode(encrypted_request["ivCipher"]))
        self.aes_key = aes_key
        self.aes_iv = aes_iv
        return json.loads(crypto_helper.aes_decrypt(
            aes_key, aes_iv,
base64.b64decode(encrypted_request["bodyCipher"])))
```

在加入了 mac 之后，在我们的主程序中同时需要进行如下的判断操作，即当验证 mac 成功时才能继续进行解密，否则我们返回错误信息。

```
def test_controller(request):
    request_params = json.loads(request.body.decode("utf-8"))

    helper = http_crypto_helper.HttpCryptoHelper()
    print(request_params)
    request_params_mac = request_params['resmac']
    request_params=request_params['data']

    vrfy=hashlib.new('sha1',request_params['bodyCipher'].encode('u
tf-8'))
    print(vrfy.hexdigest())
    print(request_params_mac)
    flag=0
    if vrfy.hexdigest()==request_params_mac['mac']:
```

```

    print("相等")
    flag=1
    req = helper.decrypt_request_data(request_params)

    print(req)
    if flag==1:
        return HttpResponse(helper.encrypt_response_data({
            "success": True,
        }))
    else:
        return HttpResponse(helper.encrypt_response_data({
            "success": False,
        }))

```

我们在此处采用了 sha1 充当 mac。

5.4 加盐存储

加盐存储与其验证均使用 bcrypt 包即可实现。

此处代码是我们判断用户名未冲突之后，进行密码加盐与数据存入数据库操作。

```

salt = bcrypt.gensalt()
hashed = bcrypt.hashpw(to_addr3.encode(), salt)
print(f"salt:{salt}")
print(f"hashed:{hashed}")
data = userTable(UserName=to_addr1, Sex=to_addr2, Password=hashed,
    Birthday=to_addr4, Phone=to_addr5,
    Email=to_addr6, Salt=salt)
data.save()

```

5.5 公私密钥对的生成与私钥的保护

5.5.1 公私密钥对生成

我们使用 Cryptodome 包下的 RSA 函数族即可进行 RSA 公私钥对的产生

```

from Crypto import Random
from Crypto.PublicKey import RSA
import time

random_generator = Random.new().read
rsa = RSA.generate(2048, random_generator)
rsa_private_key = rsa.exportKey()

```

```
with open('private.pem', 'w') as f:
    f.write(rsa_private_key.decode())
rsa_public_key = rsa.publickey().exportKey()

with open('public.pem', 'w') as f:
    f.write(rsa_public_key.decode())

print("私钥已存入 private.pem.\n 公钥已存入 public.pem.\n")
print("请您将公钥信息填写到证书申请页面,祝您使用愉快!")
print()

count = 5
while count >= 0:
    print(f"{count}秒后,窗口关闭...")
    print()
    time.sleep(1)
    count = count - 1
```

5.5.2 私钥保护

该段代码中,我们大体上实现了两个功能,即本地 RSA 密钥的加密与解密:

```
import time

from Crypto.Cipher import AES
from base64 import b64encode, b64decode
from Crypto.Random import get_random_bytes
from Crypto.Util.Padding import pad
from Crypto.PublicKey import RSA
import json
import bcrypt
from Crypto.Util.Padding import unpad
import os

print("请问您想要加密密钥(输入 1),还是解密密钥?(输入 2):")
print()
command = input("请输入命令参数:")
print()
while command != "1" and command != "2":
    command = input("命令参数无效!请输入命令参数,或者输入 3 终止程序:")
    print()
    if command == "3":
        exit("see you~")
if command == "1":
```

```
f = open('private.pem', 'r')
data = RSA.import_key(f.read()) # rsa 私钥是要加密的数据
key = get_random_bytes(32) # 生成 256 位的密钥
print(f"获取私钥:{data.export_key()}")
print()
pwd = input("请输入密码:")
print()
pwd = pwd.encode()
print(f"密码为:{pwd}")
print()
salt = bcrypt.gensalt()
hashed = bcrypt.hashpw(pwd, salt)
print(f"创建盐值:{salt}")
print()
print(f"哈希密码结果:{hashed}")
print()

f2 = open('pwd.json', 'w', encoding='utf-8', newline='')
hashtar = b64encode(hashed).decode('utf-8')
salttar = b64encode(salt).decode('utf-8')
result2 = json.dumps({'prepasswd': hashtar, 'salt': salttar})
json.dump(result2, f2)

print(f"pwd.json 的内容:{result2}")
print()

cipher = AES.new(key, AES.MODE_CBC)
ct_bytes = cipher.encrypt(pad(data.export_key(),
AES.block_size))
iv = b64encode(cipher.iv).decode('utf-8')
ct = b64encode(ct_bytes).decode('utf-8')
k = b64encode(key).decode('utf-8')

result = json.dumps({'iv': iv, 'ciphertext': ct, 'key': k})
f = open('EPrivKey.json', 'w', encoding='utf-8', newline='')
json.dump(result, f)

print(f"加密后的私钥:EPrivKey = {ct}")
print()
print(f"EPrivKey 已存储!")
print()

f = open('private.pem', 'w', encoding='utf-8', newline='')
f.truncate(0)
```

```
print(f"私钥 private.pem 已清空!")
print()
else:
    json_input2 = ''
    with open("pwd.json", "r") as dump_f:
        json_input2 = json.load(dump_f)
    print(json_input2)

    p = json.loads(json_input2)
    # passwd = b64decode(p['target'])

    passwd = input("请输入密码:")
    print()

    passwd = passwd.encode()
    salt = b64decode(p['salt'])
    hashed = b64decode(p['prepasswd'])

    if bcrypt.checkpw(passwd, hashed):
        print("密码正确!")
        print()

    else:
        print("密码错误!")
        print()
        exit("see you~")

    json_input = ''
    with open("EPrivKey.json", "r") as dump_f:
        json_input = json.load(dump_f)

    b64 = json.loads(json_input)
    iv = b64decode(b64['iv'])
    ct = b64decode(b64['ciphertext'])
    key = b64decode(b64['key'])
    cipher = AES.new(key, AES.MODE_CBC, iv)
    pt = unpad(cipher.decrypt(ct), AES.block_size)
    print(f"原始私钥为:{pt}")
    print()

    with open('private_recover.pem', 'w') as f:
        f.write(pt.decode())
```

```
count = 5
while count > 0:
    print(f"{count}秒后,窗口关闭...")
    print()
    time.sleep(1)
    count = count - 1
```

接着我们使用 `pyinstaller` 库对该程序进行打包，生成可执行 `exe` 文件。

5.6 CA 签名与验签

5.6.1 签名

```
def generate_certificate(request):
    request_params = json.loads(request.body.decode("utf-8"))
    helper = http_crypto_helper.HttpCryptoHelper()
    req = helper.decrypt_request_data(request_params)

    ID = req['ID']
    origin = applyTable.objects.filter(RegistrationNumber=ID)
    li = list(origin)
    target = li[0]

    User = target.UserName

    userpubk = RSA.import_key(target.PublicKey)

    nowtime = target.StartTime.strftime("%Y-%m-%d")
    expiretime = target.EndTime.strftime("%Y-%m-%d")

    justicenumber = target.RegistrationNumber

    serialNumber = target.SerialNumber

    hardcore = serialNumber + expiretime +
userpubk.exportKey().decode() # 代签信息

    hash_object2 = SHA256.new()
    hash_object2.update(hardcore.encode('utf-8'))

    f = open('keys/adminprivate.pem', 'r')
    adminprivk = RSA.import_key(f.read())
```



```
signer = PKCS1_v1_5.new(adminprivk)
signature = signer.sign(hash_object2)
result = base64.b64encode(signature)
result = result.decode('utf-8')

data = json.dumps({'Serial Number': serialNumber, 'Skyrim CA
System': 'www.Skyrim.com',
                  'Sign Algorithm': 'sha256+RSA', 'Valid Time
From': nowtime,
                  'Valid Time to': expiretime,
                  'User': User, 'User Publickey':
userpubk.export_key().decode(),
                  'Sign': result})
f3 = open(serialNumber + '.json', 'w', encoding='utf-8',
newline='')
json.dump(data, f3)
```

5.6.2 OCSP 验证签名

```
def isvalid_controller(request):
    print("已收到 isvalid 页面的请求")
    request_params = json.loads(request.body.decode("utf-8"))
    helper = http_crypto_helper.HttpCryptoHelper()
    req = helper.decrypt_request_data(request_params)

    ID=req['SerialNumber']
    username=req['username']
    print(ID)
    li = list(certTable.objects.filter(SerialNumber=ID))
    flag = 0
    if len(li) > 0:
        flag = 1
    if flag==1:

        logger = controller_logger.logger2
        logger.info(f'[查询]:{username}')

        return HttpResponse(helper.encrypt_response_data({
            "success": True,
        }))
    else :
        return HttpResponse(helper.encrypt_response_data({
            "success": False,
        }))
```

5.7 证书传递

本次实验中证书传递功能有两种实现方式，这两种方式可以通过‘证书获取’页面的按钮进行灵活的切换，使用起来比较方便，能够适应不同的安全性需求。

方案 1: 直接从浏览器下载证书

在该方案中，前端向后端 APIS.download 发送请求，后端处理该请求，并返回文件流。获得了证书的电商/银行可以主动读取证书文件，并将其部署在本地。

前端请求代码如下：

```
axios.post(APIS.download, resdata, {responseType:
'blob'}).then(res => {
  const a = document.createElement('a')
  a.style.display = 'none'
  a.href = window.URL.createObjectURL(new Blob([res.data]))
  a.setAttribute('download', this.SerialNumber + '.json') //设置
文件名
  document.body.appendChild(a)
  a.click()
  document.body.removeChild(a)
  this.message = "恭喜你,证书已下载成功!";
  this.dialogVisible = true;
}).catch(reason => {
  console.log(reason);
}).finally(() => {
  console.log("FINALLY");
})
```

后端处理代码如下，注意，在处理证书的时候采用了懒处理的方法，即仅当前台请求下载证书的时候，后端才开始生成证书并返回；在此之后的获得证书则可以直接读取。

```
def download_controller(request):
    print("已收到 download 页面的请求")
    request_params = json.loads(request.body.decode("utf-8"))
    req = 0
    helper = http_crypto_helper.HttpCryptoHelper()
    print(request_params)
    flag1 = 0
    if helper.dec_vrfy_data(request_params):
        request_params_data = request_params['data']
        req = helper.decrypt_request_data(request_params_data)
        flag1 = 1

    ID = req['SerialNumber']
    username = req['username']
```

```
li = list(certTable.objects.filter(SerialNumber=ID))
flag = 0
if (len(li) > 0):
    flag = 1
if flag == 1 and flag1 == 1:
    logger = controller_logger.logger2
    logger.info(f'[下载]:{username}')
    try:
        f = open(f'certificate/{ID}.json', 'rb')
        # f = open(ID + '.json', 'rb')
    except Exception as e:
        origin = certTable.objects.filter(SerialNumber=ID)
        li = list(origin)
        target = li[0]

        User = target.UserName

        userpubk = RSA.import_key(target.PublicKey)

        nowtime = target.StartTime.strftime("%Y-%m-%d")
        expiretime = target.EndTime.strftime("%Y-%m-%d")

        justicenumber = target.RegistrationNumber

        serialNumber = target.SerialNumber

        hardcore = serialNumber + expiretime +
userpubk.exportKey().decode() # 代签信息

        hash_object2 = SHA256.new()
        hash_object2.update(hardcore.encode('utf-8'))

        f = open('keys/adminprivate.pem', 'r')
        adminprivk = RSA.import_key(f.read())

        signer = PKCS1_v1_5.new(adminprivk)
        signature = signer.sign(hash_object2)
        result = base64.b64encode(signature)
        result = result.decode('utf-8')

        data = {'serialNumber': serialNumber, 'skyrimCASystem':
'www.Skyrim.com',
                'signAlgorithm': 'sha256+RSA', 'validTimeFrom':
nowtime,
```

```

        'validTimeto': expiretime,
        'user': User, 'userPublickey':
userpubk.export_key().decode(),
        'sign': result}
    print(f"初次生成证书,证书是:{data}")
    f3 = open(f'certificate/{ID}.json', 'w',
encoding='utf-8', newline='')
    json.dump(data, f3)

    f = open(f'certificate/{ID}.json', 'rb')
    # f = open(ID + '.json', 'rb')

    response = FileResponse(f)
    # response['Content-Type'] = 'application/octet-stream'
    # response['Content-Disposition'] =
f'attachment;filename="{ID}.json"'

    # logger.info(f'[下载生成器]:{username}')
    return response

```

方案 2: 通过电子信封更新证书

在该方案中, 前端直接向电商/银行的相关接口发送请求与证书数据, 电商/银行处理该请求, 记录并部署该证书。

```

axios.post(API.email, resdata
).then(res => {
    var target = jsHttps.decryptResponseData(res.data);
    console.log(target);
    var myurl = "http://192.168.0.102:9876/shop/user/getCA";
    var encdata2 = jsHttps.encryptRequestData(target,
adminpublickey)
    console.log(encdata2)
    axios.post(myurl,
        encdata2,
    ).then(res => {
        this.message = "恭喜你,证书已传输成功!";
        this.dialogVisible = true;
        res = jsHttps.decryptResponseData(res.data);
    }).catch(reason => {
        console.log(reason);
    }).finally(() => {
        console.log("FINALLY");
    })
}).catch(reason => {
    console.log(reason);
}).finally(() => {

```

```
console.log("FINALLY");  
})
```

相关的部署细节可以参见电商/银行的该部分代码。

6. 测试

6.1 注册/登陆/登出

成功发送验证邮件并完成注册。



Registration form for '天际CA管理系统' (Tianji CA Management System). The form includes the following fields and options:

- 用户名: 凌华
- 性别: 男 女
- 密码: (Strength indicator: 40%, 密码比较安全...)
- 生日: 2003/01/08
- 手机号: 13899992222
- 邮箱: 835766751@qq.com
- e-验证: Yy7tJ
- 验证码: 7Kreh
- 验证码提示: 7kreh
- 我同意并阅读并接受 [天际CA管理系统](#) 的[用户协议](#)
- 立即注册

天际CA管理系统验证邮件 ☆

发件人: 天际CA管理系统 <> 

(由 565852435@qq.com 代发)

时 间: 2022年10月22日 (星期六) 下午9 : 55

收件人: 用户

您的注册验证码为: Yy7tJ,感谢您的使用!

注册成功后,可以直接跳转登陆,也可回到开始页面登陆。我们此处不进行直接跳转,而是回到开始页面,进行用户名密码输入并登陆。



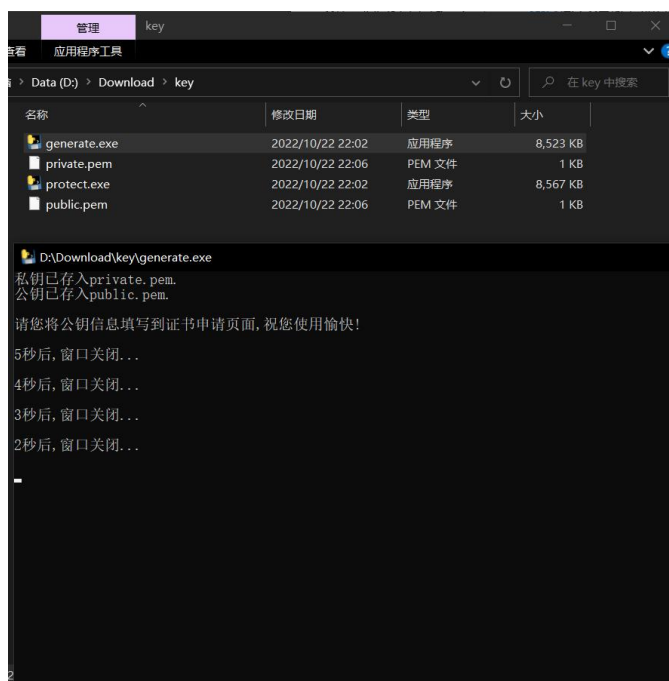
在主页面的右上角，点击退出登录，我们就可以登出当前的账号，并在后台进行日志的记录。



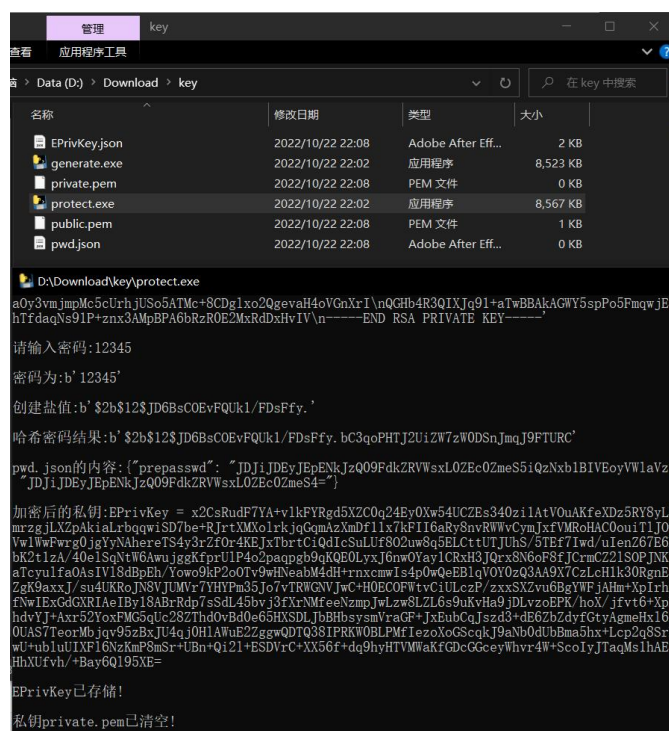
6.2 公私钥对生成和私钥保护

我们先把两个程序下载到本地，然后进行用户密钥相关的操作。

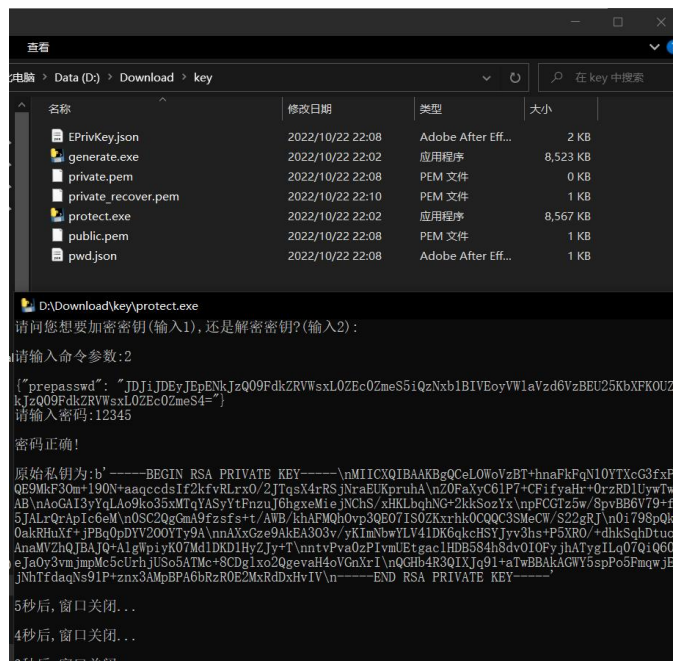
首先进行公私钥对生成，运行 `generate.exe`，生成公钥和私钥，并分别存入 `public.pem` 和 `private.pem` 中。



然后进行私钥加密，运行 protect.exe，输入参数 1，并根据程序的指示输入密码，即可进行私钥的加密，并且将原先的密钥文件清空。



最后演示私钥解密，运行 protect.exe，输入参数 2，并正确输入原先的密码，即可恢复密钥。可以看到生成了 private_recover.pem 文件，里面存储的即为恢复的密钥。

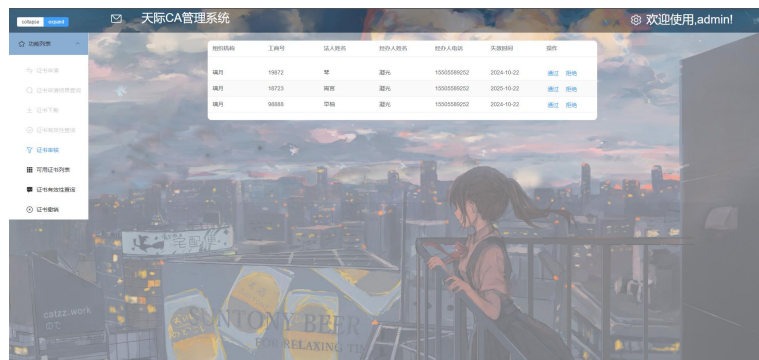


6.3 证书申请与审批

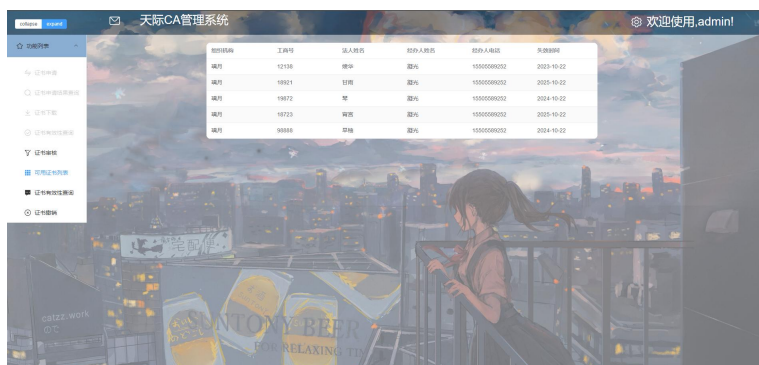
用户在该页面提交申请。在这个页面，我们可以选择两种公钥分发的方式，第一种即手动生成公私钥，并将私钥安全保存在本地；另外一种就是通过电子信封分发的方式，直接进行分发公私钥。我们这里选择通过电子信封分发公私钥。提交申请之后，会弹窗汇总用户输入的信息。



管理员审核：
管理员可以查看所有申请信息的列表，并进行判断。



我们此处让这些申请全都通过。



我们在用户方进行查询。可见审核通过，查询申请成功。

6.4 CA 证书分发、签名验证

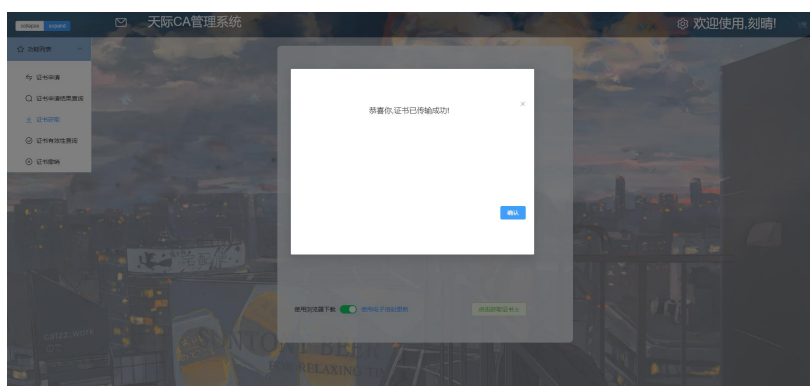
电商与银行想要获取证书，需要先在证书申请结果页面输入工商注册号查询证书是否通过审核。



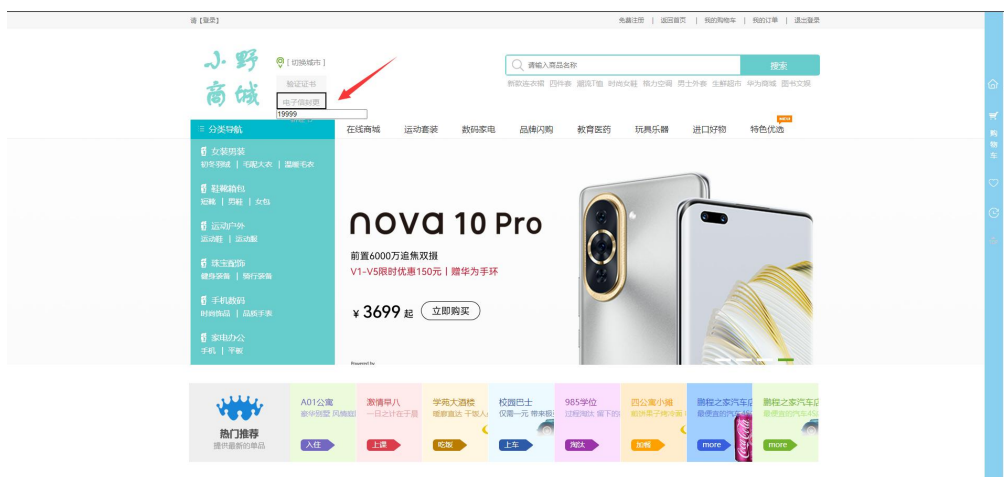
如果通过审核，则会弹出以下窗口，我们点击复制证书序号即可。



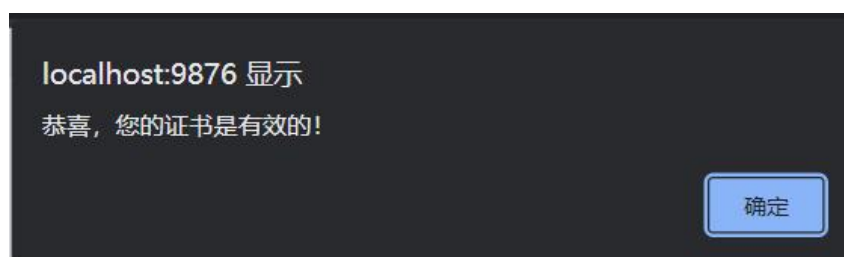
复制得到证书序号之后，我们来到证书获取页面进行证书的获取，该页面提供了两种证书获取的方法，一种是直接从浏览器中下载，另一种则是通过电子信封进行传输。



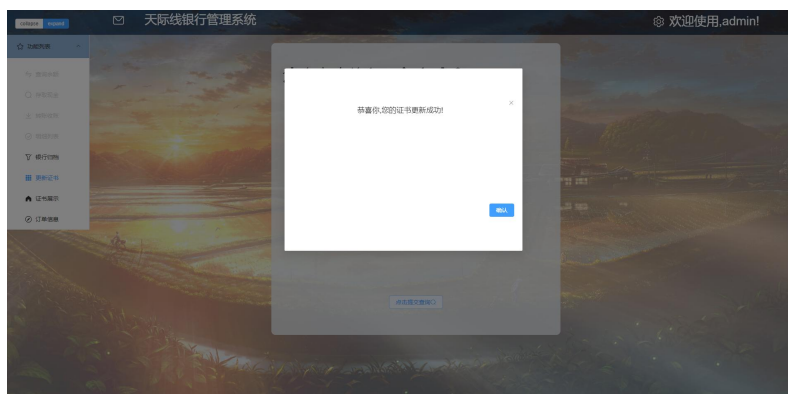
在此处通过电子信封传递，我们可以直接将证书传递到用户端，比如此处我们主动使用电子信封传递，即可将证书传递到商城端。另外，在商城端使用工商注册号进行请求，也能主动获取到我们的证书。



在商城端进行验证，倘若证书有效，则会弹出以下弹窗。



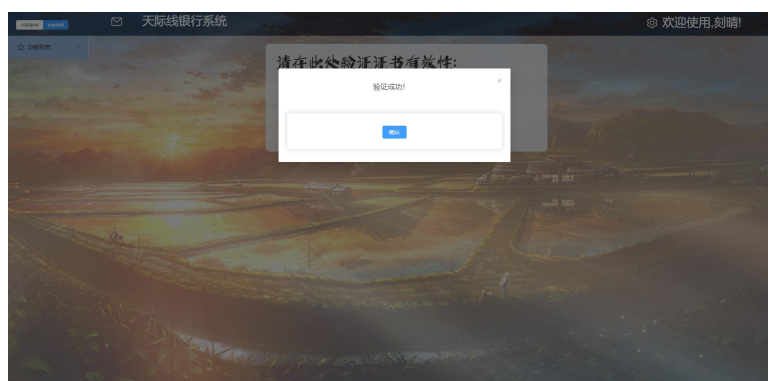
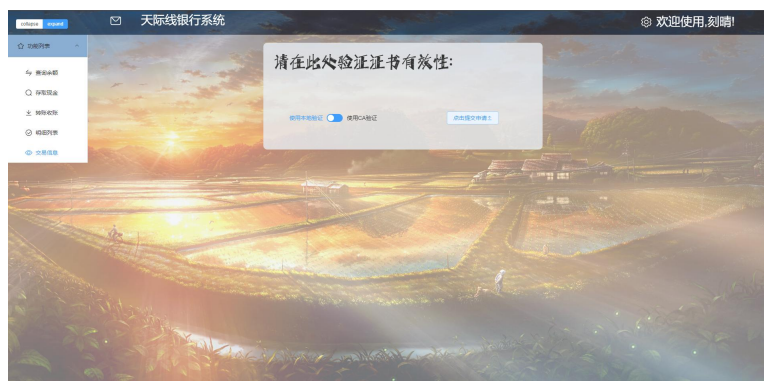
另外在银行端，客户可以自行更新最新的证书。只需要输入所需证书的工商注册号，即可获得最新的证书。



银行端证书更新成功之后，我们可以在用户界面进行验证证书。在右上角的功能栏点击验证证书即可。

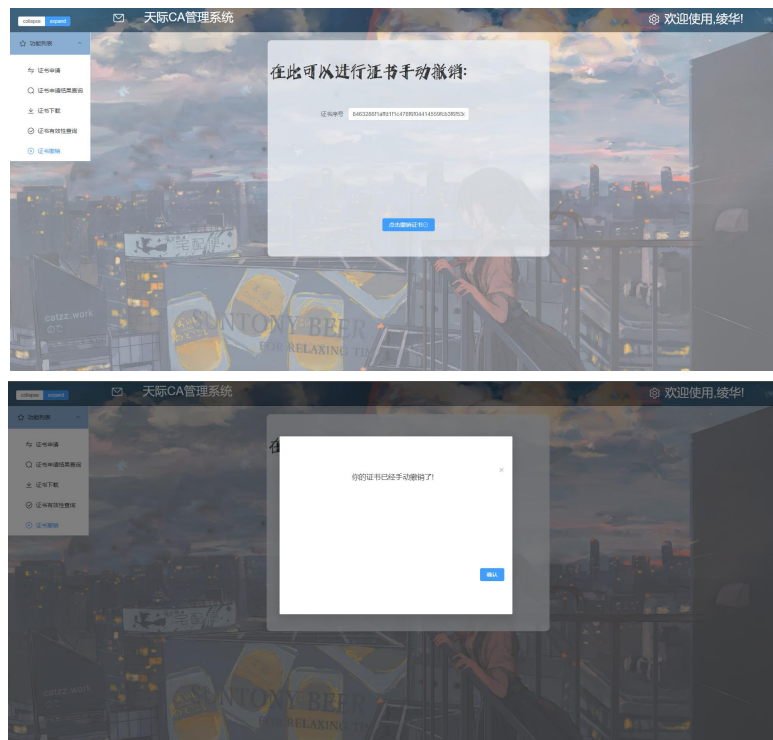


在验证证书页面，我们提供了两种验证的方法即本地验证与利用 CA 预留的接口进行 OCSP 验证，经过检验这两种验证方法均可以通过证书验证。



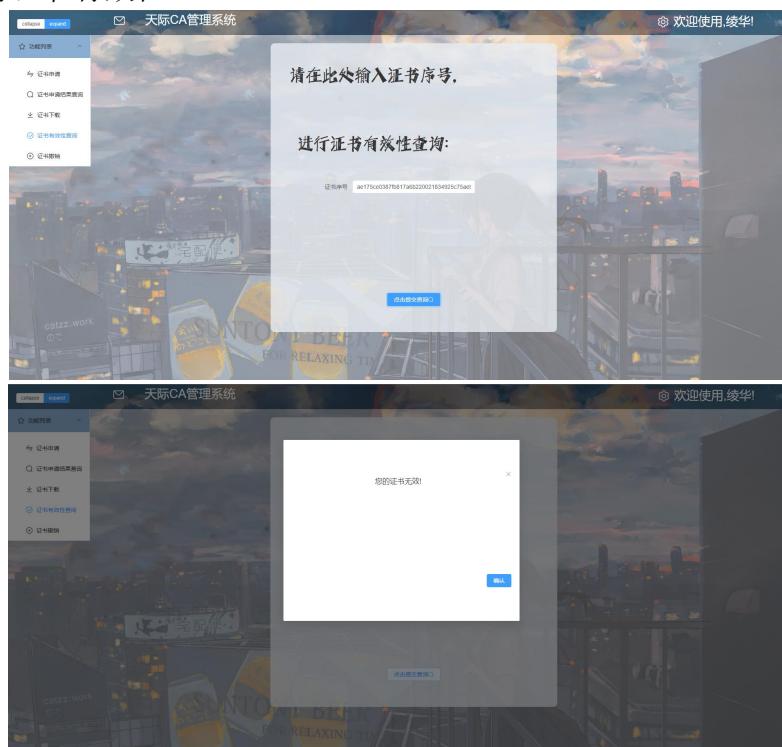
6.5 证书撤销与 OCSP 验证

用户撤销自己申请的证书:



在这一部分，需要注意到用户无法撤销非自己用户申请的证书，即做了用户的权限隔离。比如，在用户名为‘刻晴’的账号申请撤销用户名为‘绫华’的账号所申请的证书，则会得到撤销失败的提示。另外注意到管理员账号拥有最高权限，可以撤销任意用户的证书。

在线查询证书有效性：



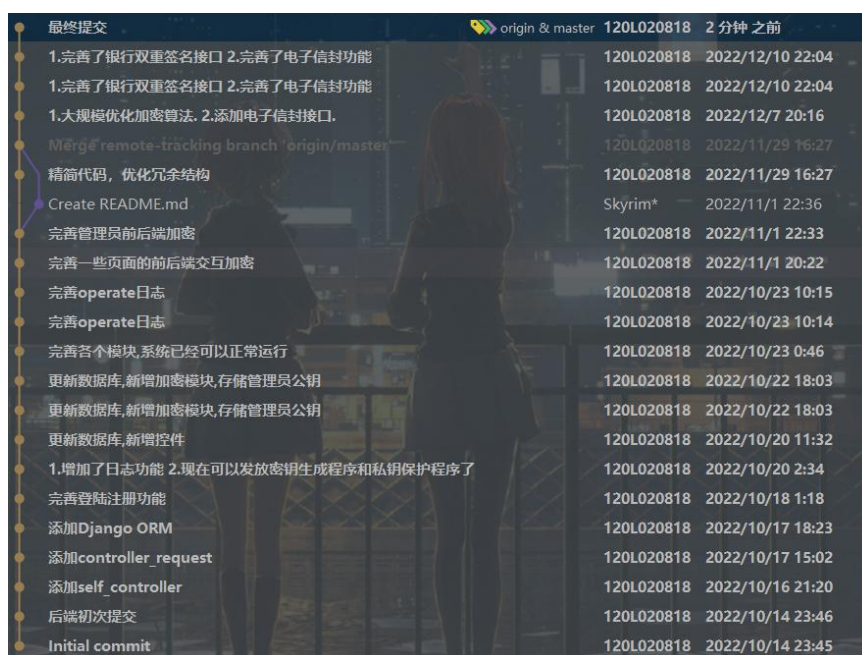
7. 结束语

7.1 Git 提交记录与学习、开发流程

WebStorm 项目 Git 提交记录:



PyCharm 项目 Git 提交记录:



长达两个多月的征程，看着这个项目从无到有、一点点完善。从最初的注册和登录做起，添加日志系统，一步一步的完成注册和登录的安全性，完成申请证书的安全性，下载证书的安全性，验证证书的安全性；并增加了管理员功能，对证书申请进行审核以及撤销。之后在登录和注册中加入了验证码功能，最后增加了混合加密与时间戳签名。

在开发的过程中，学习了 python 中的 Django、bcrypt 和 Cryptodome 等技术，学习了前端的 Vue 和 Element-plus，复习了 HTML，CSS 和 JavaScript，学习了

数据库的 Django ORM 和 MySQL，并学习了单组件页面理论和前后端交互方法 axios。

另外，本项目使用了前后端分离开发，因此分别使用了两个 github 的仓库进行管理，vue 与 django 的版本记录如上图所示。

7.2 遇到的困难、解决与收获

在“前端加密与签名”，以及“证书生成”的过程中，由于前后端使用了不同的技术，前端 JavaScript，后端 Java，经常出现前后端接口对不上的情况。

在调试的过程中，花去了大量的时间去解决，最初通过不同解决方案的排列组合，企图有一次能得到正确的结果，但最后无果，并浪费了大量时间。仅 RSA 签名验签就花费了 2 天的时间。最后通过查看 AES、RSA 等算法的底层源码，才解决了前后端加解密对不上的问题。当输出签名验证成功的一瞬间，我直接高兴的跳了起来。

在这个过程中，深化了自己对 AES、RSA 和 SHA256 等算法的理解，增强了 Python 与 Vue 的使用能力，学习去构建一个安全的系统，锻炼了开发中解决问题的能力，最为重要的是，培养了看源码的习惯，并且自己能够写一些好用的工具类，有能耐更有勇气去造轮子。

7.3 鸣谢

感谢翟健宏老师的悉心指导与问题解答，尤其是对我们的鼓励，让我有信心与勇气去完成这样一个大的项目。

感谢我的一位队友：袁野(商城)，与队友的讨论，pair programming，共同推进项目进行。另外，由于组内缺少成员，我也在本次实验中进行了部分银行接口的编写以便与队友进行对接。

感谢一直以来为我解答问题的崔子健同学，在他的帮助下我掌握了 Vue 这门技术，对前端开发有了更深入的了解。

参考文献

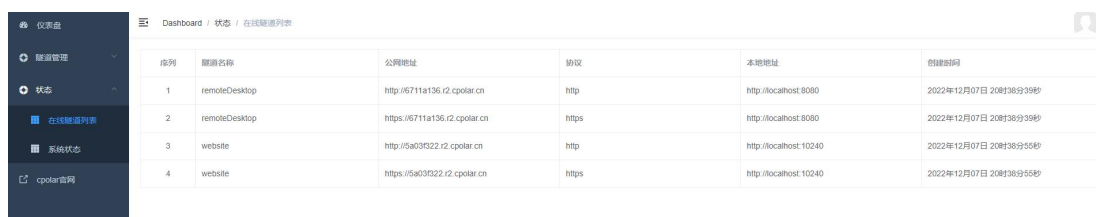
- [1] GB/T 30272-2021, 信息安全技术 公钥基础设施 标准符合性测评[S].
- [2] <https://zh.wikipedia.org/wiki/%E5%85%AC%E9%96%8B%E9%87%91%E9%91%B0%E5%9F%BA%E7%A4%8E%E5%BB%BA%E8%A8%AD> 公开密钥基础建设
- [3] https://en.wikipedia.org/wiki/Public_key_infrastructure Public_key_infrastructure
- [4] <https://zh.wikipedia.org/wiki/X.509> “X.509, 维基百科”
- [5] <https://docs.oracle.com/cd/E19146-01/820-0872/gdagx/index.html> “管理证书撤销列表 (Certificate Revocation List, CRL)”
- [6] [https://baike.baidu.com/item/%E8%AF%81%E4%B9%A6%E6%92%A4%E9%94%80/747891?fr=aladdin\(https://baike.baidu.com/item/证书撤销/747891?fr=aladdin\)](https://baike.baidu.com/item/%E8%AF%81%E4%B9%A6%E6%92%A4%E9%94%80/747891?fr=aladdin(https://baike.baidu.com/item/证书撤销/747891?fr=aladdin)) “证书撤销”
- [7] <https://www.jianshu.com/p/c65fa3af1c01> “PKI/CA 工作原理及架构”
- [8] <https://blog.csdn.net/ayang1986/article/details/80810072> “CA 认证简单介绍与工作流程”
- [9] 江为强, 陈波. PKI / CA 技术的起源、现状和前景综述[J]. 西南科技大学学报, 2003, 18(4):75-78.
- [10] [Vue.js - 渐进式 JavaScript 框架 | Vue.js \(vuejs.org\)](#)
- [11] [一个 Vue 3 UI 框架 | Element Plus \(gitee.io\)](#)
- [12] [python 读写 json 文件 - Bigberg - 博客园 \(cnblogs.com\)](#)
- [13] [Python RSA 签名、AES 密钥加密 - 知乎 \(zhihu.com\)](#)

附录

附录 A

由于疫情原因，最后一段时间的实验并未在学校内进行，故各端之间的通信就成了问题。在学校内大家可以连接局域网协同开发，但在家中没有这种方便的条件。故需要使用一些内网穿透工具进行远程协同开发。

本次实验中采用的内网穿透工具是 Cpolars。



The screenshot shows the Cpolars dashboard interface. On the left is a dark sidebar with navigation options: '仪表盘' (Dashboard), '隧道管理' (Tunnel Management), '状态' (Status), '在线隧道列表' (Online Tunnel List), '系统状态' (System Status), and 'cpolar官网' (Cpolars Official Website). The main content area is titled 'Dashboard / 状态 / 在线隧道列表' and contains a table with the following data:

序号	隧道名称	公网地址	协议	本地地址	创建时间
1	remoteDesktop	http://6711a136.r2.cpolars.cn	http	http://localhost:8080	2022年12月07日 20时38分39秒
2	remoteDesktop	https://6711a136.r2.cpolars.cn	https	http://localhost:8080	2022年12月07日 20时38分39秒
3	website	http://5a03022.r2.cpolars.cn	http	http://localhost:10240	2022年12月07日 20时38分55秒
4	website	https://5a03022.r2.cpolars.cn	https	http://localhost:10240	2022年12月07日 20时38分55秒

由于我本人使用了前后端分离的开发方式，所以需要建立两个 Web 隧道进行协同使用。在使用内网穿透工具后，可以正常地进行远程访问。

另外还有一种办法是将所有项目都部署在一台机器上，这种办法也是可行的，并且部署好之后非常方便，不再需要解决跨域 CORS 等问题，甚至可以仅仅使用 localhost 端口进行访问。

附录 B

关于浏览器 CORS 策略的禁用问题，在 Django 中可以直接使用 corsheaders 来在 settings 中将其关掉。而在 java 的 spring 中，一种最简单的方法就是采用注解将其取消，操作起来比较快捷。

```
MIDDLEWARE = [  
    'django.middleware.security.SecurityMiddleware',  
    'django.contrib.sessions.middleware.SessionMiddleware',  
    'corsheaders.middleware.CorsMiddleware',  
    'django.middleware.common.CommonMiddleware',  
    # 'django.middleware.csrf.CsrfViewMiddleware',  
    'django.contrib.auth.middleware.AuthenticationMiddleware',  
    'django.contrib.messages.middleware.MessageMiddleware',  
    'django.middleware.clickjacking.XFrameOptionsMiddleware',  
    'django.middleware.http.ConditionalGetMiddleware',  
]  
  
CORS_ALLOW_CREDENTIALS = True  
CORS_ORIGIN_ALLOW_ALL = True  
# 允许所有的请求头  
CORS_ALLOW_HEADERS = ('*')  
ROOT_URLCONF = 'pureBack.urls'
```

附录 B.图 1.Django 中解决 CORS 问题

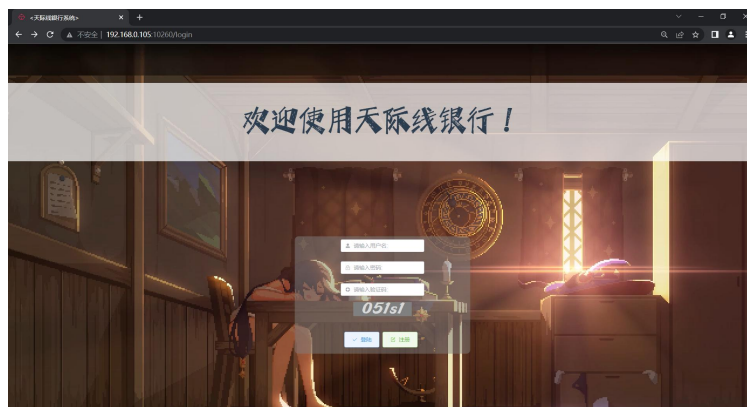
```
no usages  yuanye *  
@Controller  
@RequestMapping("/user")  
@CrossOrigin(origins = {"http://192.168.0.104:8080"})  
public class UserController {  
    4 usages
```

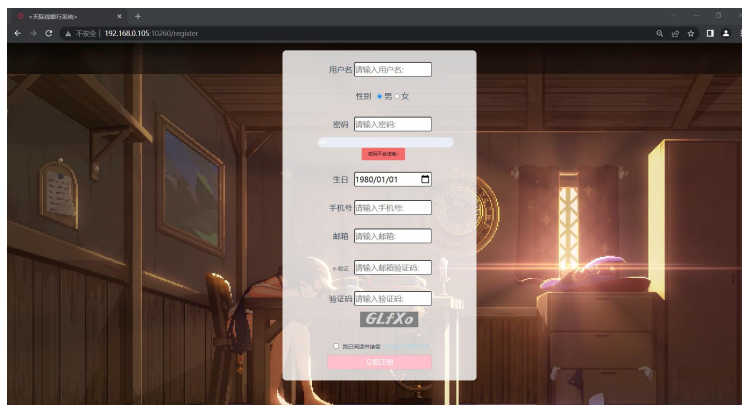
附录 B.图 2.Spring 中解决 CORS 问题

附录 C

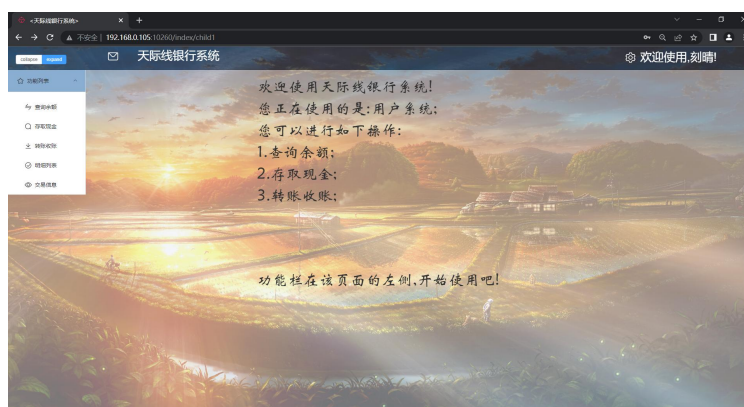
由于不可抗力因素，所以在最后一段时期又自己实现了一个简单的银行，以实现存取款、转账等交易功能，并实现了与 CA、电商之间的对接。其大体框架与本人实现的 CA 中心一致。下面对银行进行简单的介绍。

首页与注册页面的外观与功能，和 CA 中心基本保持一致。

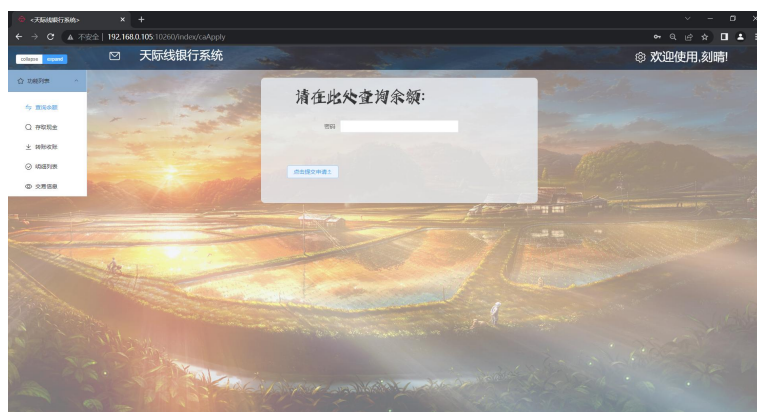


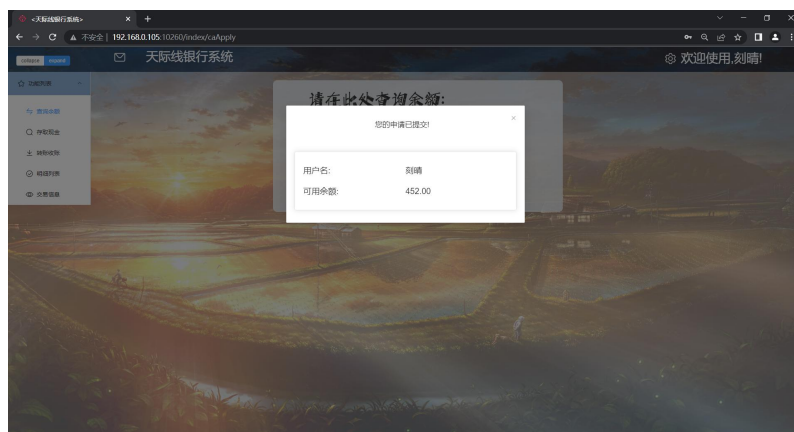


功能区外观与 CA 中心的外观类似，但功能则有了显著的变化。下面对功能展开介绍。

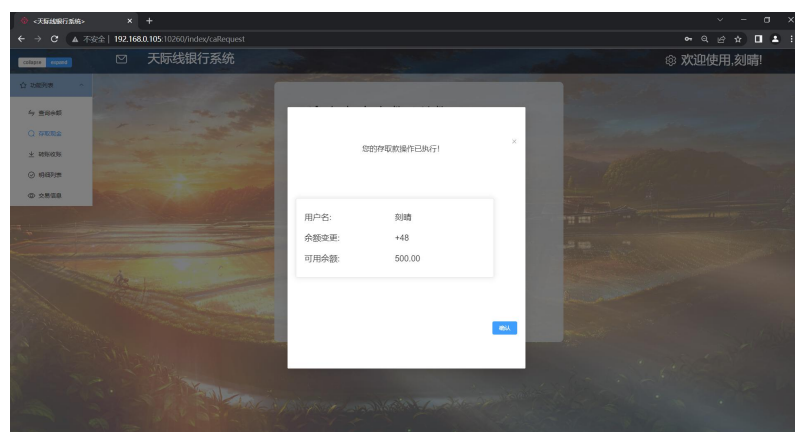


首先是查询余额的界面，在此处输入密码即可对用户账户内的余额进行查询。效果如下图所示：

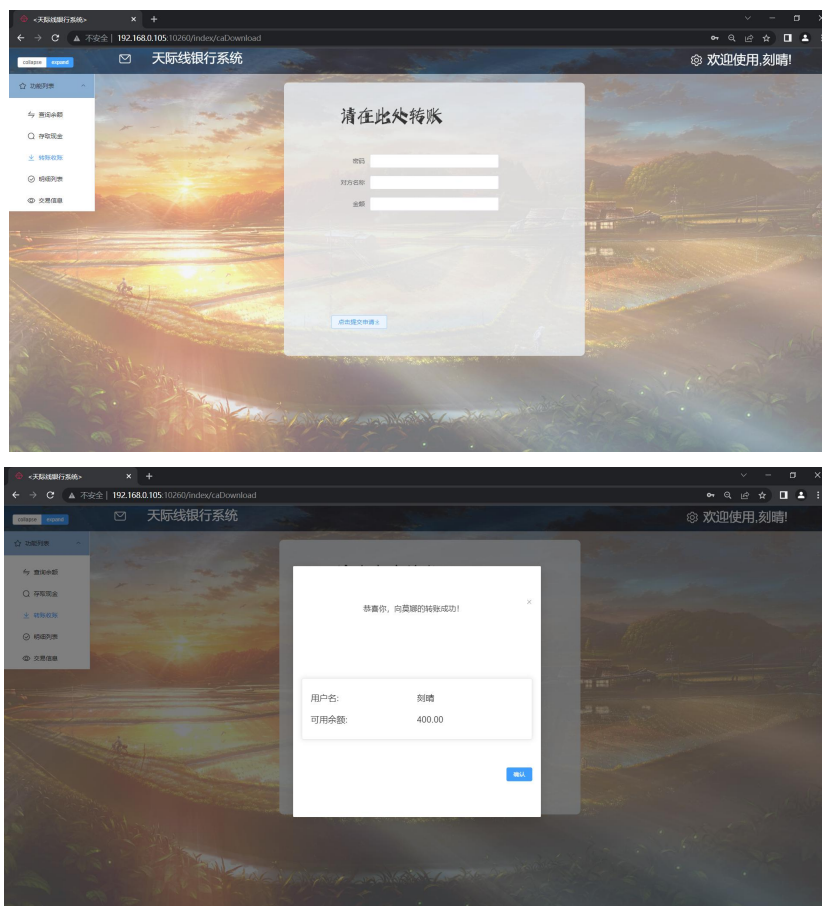




然后是存取现金的界面，在此处输入密码和金额，并选择存/取操作即可对用户账户内的余额进行存储与取出。效果如下图所示：在这个操作中，我们向账户内存了 48 元。



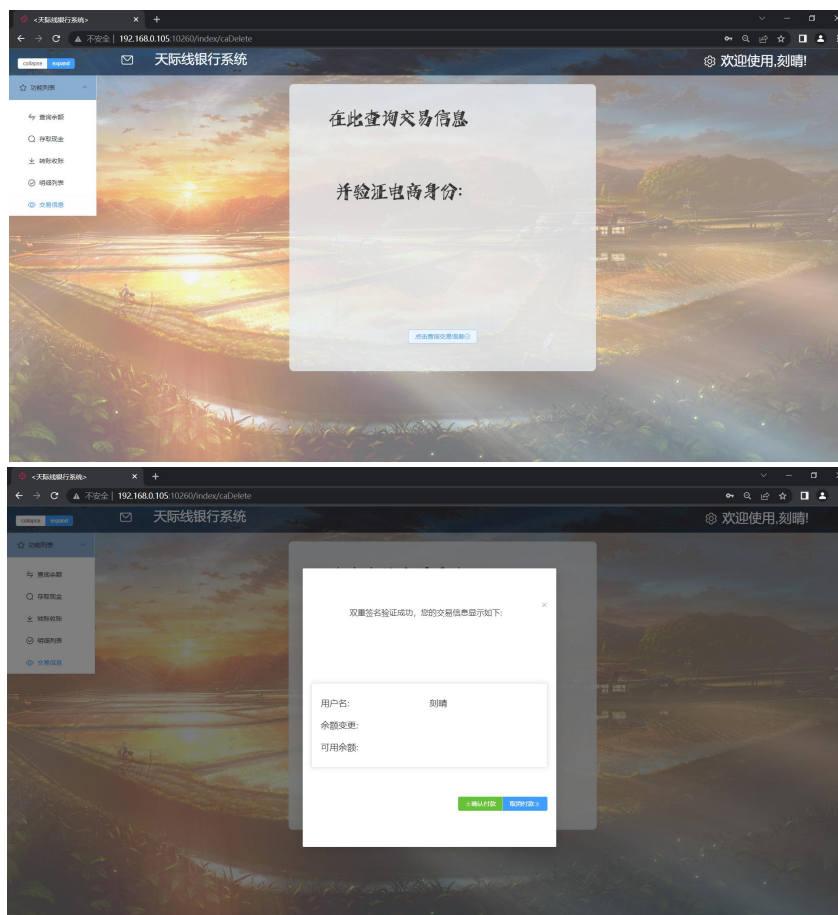
接下来是转账/收账的界面，在此处输入密码和金额，与希望转出的用户即可实现向对方用户进行转账。效果如下图所示：在这个操作中，我们向‘莫娜’账户内转了 100 元。



在明细列表的界面，我们可以查询本账号的所有交易信息。效果如下图所示：

交易日期	用户名	操作类型	转出金额
2022-12-15 23:06:4	刻晴	[存款类]	900.00
2022-12-15 23:07:0	刻晴	[转入]	900.00
2022-12-16 10:53:4	刻晴	[提现]	0.00
2022-12-16 10:53:5	刻晴	[存款类]	900.00
2022-12-16 17:19:0	刻晴	[存款类]	3233.00
2022-12-16 17:19:0	刻晴	[存款类]	900.00
2022-12-16 17:19:2	刻晴	[转出]	700.00
2022-12-16 21:13:3	刻晴	[存款类]	680.00
2022-12-16 21:13:3	刻晴	[存款类]	710.00
2022-12-16 21:13:5	刻晴	[转出]	680.00
2022-12-17 21:53:4	刻晴	[转出]	402.00
2022-12-18 08:58:1	刻晴	[存款类]	500.00
2022-12-18 09:01:3	刻晴	[转出]	400.00

在交易信息页面，我们可以查询本账号是否在电商平台进行了交易，并且可以对是否付款进行判断，同时这个页面也实现了检验双重签名的功能。效果如下图所示：当点击确认付款后，会清空交易信息缓存，防止重放攻击，然后在电商和用户之间进行转账。当点击取消时，同样会清除缓存，但是并不会转账。

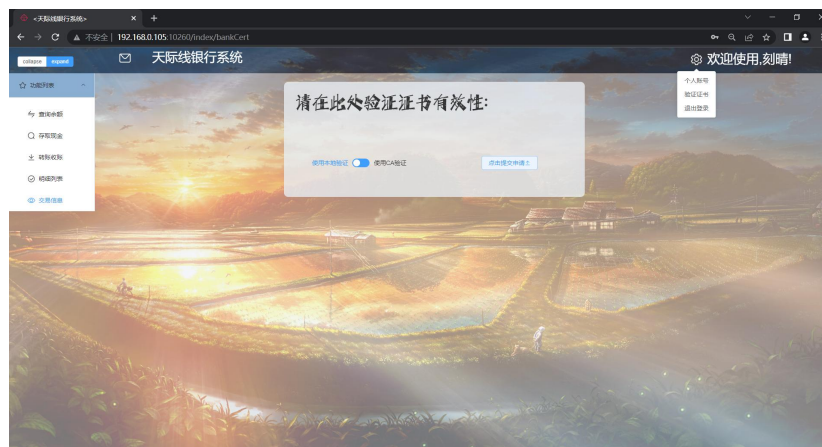


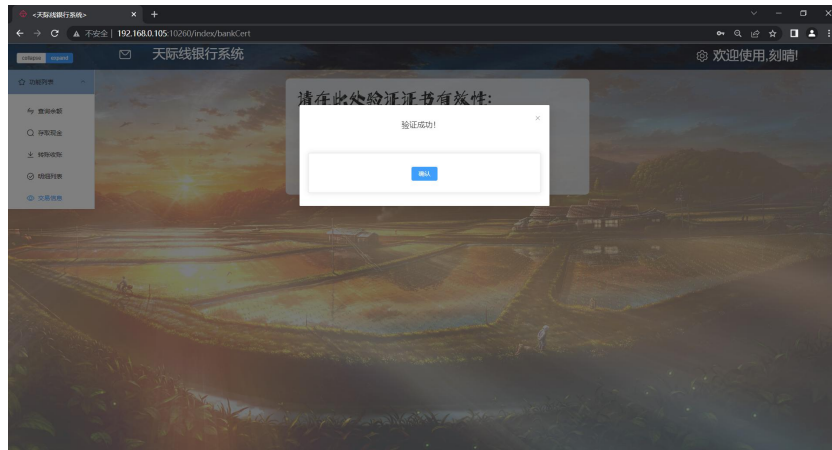
此外，在右上角我们也预留了一些接口。比如个人信息接口，在这个接口中客户可以向后台查验自己的账号信息，此处与 CA 的对应功能基本一致故不做演示。另外我们留下有一个验证证书接口。我们的银行在页面挂载的时候会在前端初始化我们的证书并记录。我们的验证证书就是对该证书进行手动验证。并且验证的具体实现有以下两种：

实现方法 1：直接在银行端进行验证。即使用 RSA 验签的方法，在银行侧直接验签。

实现方法 2：通过 CA 预留的接口，对该证书进行有效性的查询，这种方法需要与 CA 通信。

两种方法验签成功后，均会出现如下页面：

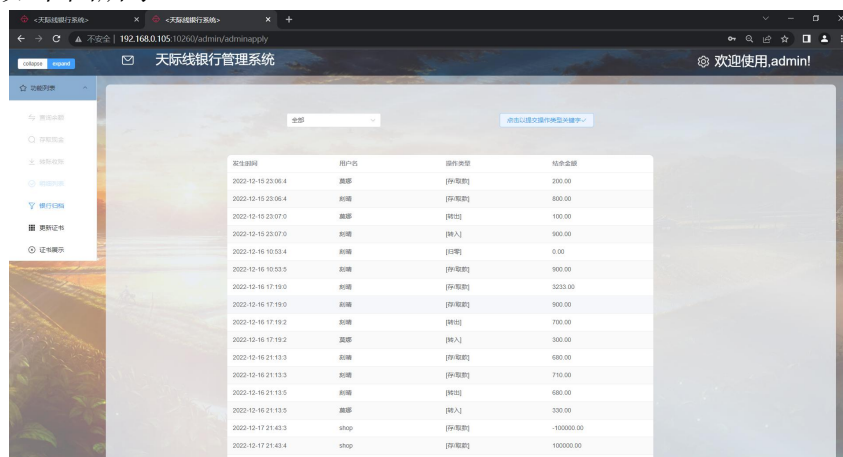


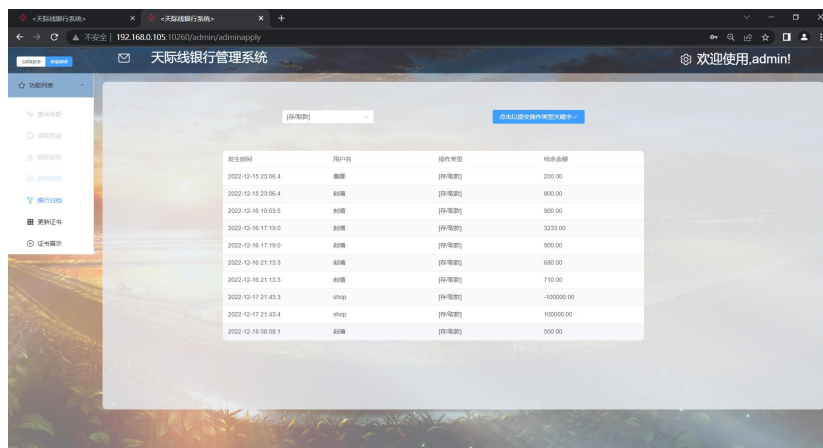


下面演示管理员页面。

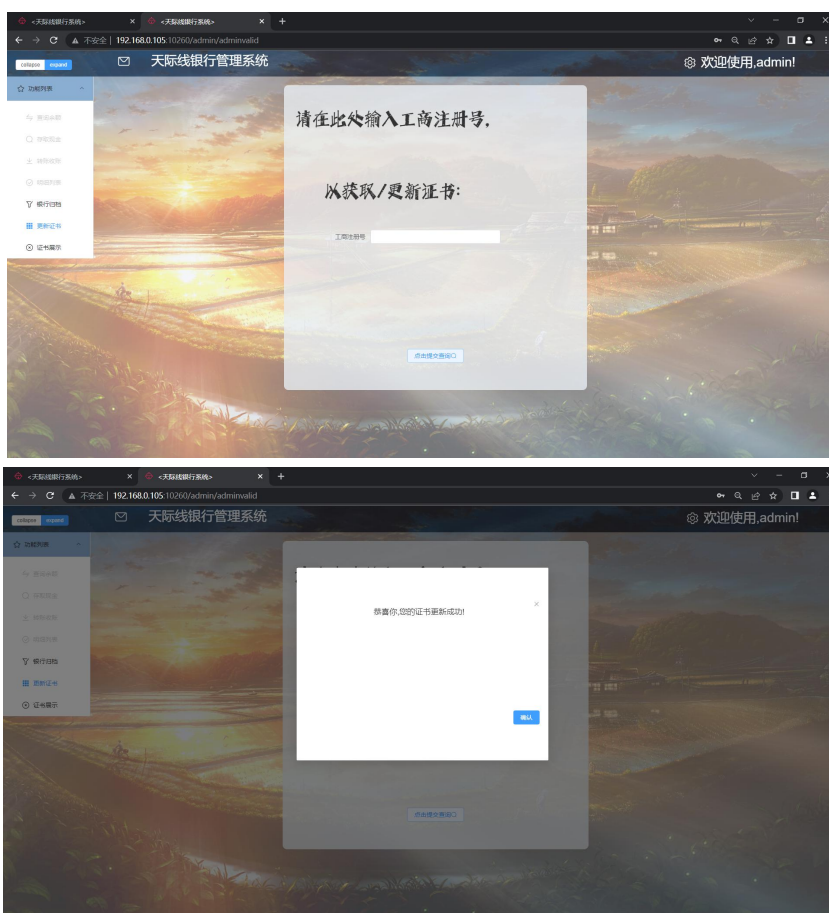


首先是银行归档的界面，在此处可以观察到所有已经进行的交易。并且可以筛选出所有特定操作的交易，比如我们可以专门筛选出所有[存/取款]的操作，如下图所示：





然后是更新证书的页面，在这个页面，我们可以根据我们在 CA 中心所填写的工商注册号来获取/更新证书，如下图所示：



最后一个页面是证书展示页面，在这个页面我们对获取的证书进行了明文展示。如下图所示：



另外，银行与电商，银行与 CA 的交互所做的加密均已经在上文介绍过，即与 CA 本身实施的加密方法相类似。

以上就是本银行系统的简要介绍。