网络空间安全学院（密码学院）

2024-2025 学年度第 1 学期

J00718 密码系统设计 课程设计论文

论文题目：基于 SM2 和 SHA256 算法的数字签名与验证系统设计与实现

学生学号及姓名：

20213006525 梁浩哲（贡献度 50%）

20213006839 甄五四（贡献度 50%）

学生所在班级：信息安全实验班

课程设计论文提交时间：2024 年 12 月 28 日

授课教师：岳秋玲

教师评阅：

成绩：_____

教师签名：_____

摘　　要

本项目设计并实现了一套基于国密 SM2 算法的数字签名与验证系统，集成了签名系统、验证系统和 CA（证书颁发机构）系统的核心功能。系统以 Python 语言为开发基础，并结合 Tkinter 图形用户界面，提供了一种直观高效的数字签名解决方案。主要功能包括密钥生成与管理、信息加密与解密、数字签名生成与验证，以及数字证书的颁发与撤销。

在技术实现方面，系统采用国密标准的 SM2 椭圆曲线算法执行数字签名与验证，辅以 SHA256 哈希算法，确保数据的完整性和安全性。CA 系统支持用户公钥的签名及数字证书生成，验证系统能够通过证书有效性检查保障通信的合法性与可信性。系统还集成了文件导入导出功能，并通过友好的界面设计提升了用户操作的便捷性。

本系统的开发为数字签名与证书管理的实际应用提供了技术支持，具有高安全性、易用性和较强的通用性，可广泛应用于电子签名、身份认证等数据安全场景。

关键词：数字签名，SM2 算法，国密标准，数字证书，数据安全

Abstract:

This project designs and implements a digital signature and verification system based on the Chinese National Standard SM2 algorithm, integrating core functionalities of a signature system, a verification system, and a CA (Certificate Authority) system. Developed using Python and the Tkinter graphical user interface framework, the system provides an intuitive and efficient solution for digital signatures. Its main features include key generation and management, data encryption and decryption, digital signature generation and verification, and the issuance and revocation of digital certificates.

Technically, the system leverages the SM2 elliptic curve algorithm from the Chinese National Standard for digital signing and verification, combined with the SHA256 hashing algorithm to ensure data integrity and security. The CA system enables signing of user public keys and the generation of digital certificates, while the verification system ensures the validity and authenticity of communication through certificate validation. Additionally, the system supports file import/export functionalities and offers a user-friendly interface to enhance operational efficiency.

This system provides robust technical support for the practical application of digital signatures and certificate management. It features high security, usability, and versatility, making it well-suited for scenarios such as electronic signatures and identity authentication.

Keywords: Digital signature, SM2 algorithm, Chinese National Standard, digital certificate, data security

基于 SM2 和 SHA256 算法的数字签名与验证系统设计与实现

目　　录

1.前言

1.1 背景

随着信息技术和网络技术的飞速发展，电子商务、电子政务、移动支付等领域的应用不断普及，人们的生产生活逐渐与网络深度融合。与此同时，网络安全问题也日益凸显，尤其是信息在网络传输过程中的安全性、真实性、完整性和不可否认性成为亟待解决的重要问题。在数据安全的众多领域中，数字签名技术和密钥交换协议是保障信息安全传输的关键。通过数字签名技术，通信双方可以实现身份认证、数据完整性验证及不可否认性，从而有效抵御网络中的潜在安全威胁。

然而，现有的密钥交换协议和数字签名技术仍面临诸多挑战：

1.中间人攻击：在密钥交换过程中，攻击者可能冒充通信双方中的任意一方，窃取密钥并监听或篡改通信内容。如何有效防范中间人攻击，是密钥交换协议设计中的核心问题。

2.密钥管理复杂性：在大规模或动态网络环境中，密钥的生成、分发、更新和撤销需要一套高效、可扩展的管理机制，否则将面临管理效率低下的问题。

3.算法安全性威胁：随着量子计算的发展，目前广泛使用的加密算法（如 RSA、Diffie-Hellman）面临被破解的风险，迫切需要使用更高强度、更安全的算法来保障通信安全。

4.性能开销：密钥交换和数字签名过程可能带来较大的计算开销，尤其是在资源受限的设备（如物联网设备）上，这一问题尤为突出。因此，如何在保证安全性的同时，降低计算和通信开销成为密钥交换协议设计的重要考虑因素。

针对上述问题，公钥基础设施（PKI）作为一种有效的解决方案被广泛应用。PKI 通过引入权威的第三方——证书颁发机构（CA），为网络实体提供数字证书，确保公钥与身份的绑定关系可信。数字证书包含持有者的身份信息及公钥，并由 CA 使用数字签名进行认证，从而为用户验证公钥的合法性提供了技术支持。

在 PKI 体系中，数字证书通过以下方式解决了密钥管理与网络安全问

题：

1.身份认证：通信双方在交换数字证书后，可通过验证对方证书的签名来确认身份，防止中间人攻击和身份伪造。

2.信息完整性：通过数字签名技术，证书内容一旦被篡改，其签名验证将无法通过，从而确保信息在传输过程中的完整性。

3.信任链机制：用户仅需信任 CA，即可通过 CA 颁发的证书建立与其他实体的信任关系，简化了密钥分发和管理的复杂性。

4.性能优化：数字证书的使用可以减少重复的身份验证操作，从而降低密钥交换过程中的性能开销。

本项目设计中使用了 SHA256 哈希算法对数据进行哈希计算，确保数据的完整性和真实性。此外，系统采用了 SM2 国密算法实现数字签名和加密操作。SM2 是中国国家密码管理局发布的一种基于椭圆曲线的加密算法，具有高效性与安全性特点。在本系统中，SHA256 用于生成待签名数据的哈希值，SM2 算法基于该哈希值生成数字签名，从而确保数据的完整性和不可否认性。通过结合 SHA256 和 SM2，本系统能够有效验证通信双方交换的公钥是否可靠，并保障密钥交换过程的真实性和机密性。

1.2 需求分析

安全目标：

在本课程设计中，核心目标是通过数字签名与验证技术、密钥管理及证书体系，保障网络通信过程中的数据安全性。具体安全目标包括：

数据真实性：确保通信双方的身份真实有效，防止身份伪造。

数据机密性：保护传输数据不被未经授权的第三方窃取。

数据完整性：确保数据在传输过程中未被篡改，若发生修改能够及时发现。

数据不可否认性：提供数字签名以防止通信双方否认已发送或接收到的信息。

抗攻击能力：防止中间人攻击、重放攻击及其他常见的网络攻击，确

保通信的安全可靠性。

功能需求：

根据设计目标，本系统需实现以下功能模块：

1.签名系统：

密钥管理：生成用户唯一的公私钥对，并支持密钥的导入与导出。

数据签名：基于 SM2 算法对数据进行数字签名，确保数据完整性和不可否认性。

数据加密：使用接收者的公钥对敏感数据进行加密，保障通信的机密性。

文件处理：支持导入和导出文本文件，用于批量处理数据签名和加密。

2.验证系统：

签名验证：使用对方公钥验证数字签名的真实性，确保数据未被篡改。

数据解密：通过私钥解密接收到的密文，恢复原始数据。

证书验证：通过加载并验证数字证书，确保通信双方的公钥可信，防止中间人攻击。

3.CA（证书颁发机构）系统：

证书颁发：为用户生成数字证书，并使用 CA 私钥进行签名。

证书验证：检查数字证书的真实性和有效性，防止证书伪造。

证书管理：支持证书的申请、撤销、更新与存储，简化大规模网络中的密钥分发。

性能指标：

1.安全性指标：

采用国家密码管理局发布的 SM2 算法和 SHA256 哈希算法，确保签名与加密过程的安全性；支持基于数字证书的身份认证，防止中间人攻击和伪造。

2.易用性指标：

提供图形化用户界面（GUI），简化用户操作流程；支持文件批量处理功能，提高用户处理效率。

3.扩展性指标：

系统需具备良好的模块化设计，便于未来扩展（如支持 SM3 哈希算法或 SM4 对称加密算法）。支持与其他系统集成，如电子签名平台和电子政务系统。

4.性能优化指标：

优化加密和签名计算的效率，确保在普通计算机和资源受限设备（如物联网设备）上高效运行。

实现证书分发与验证的高效管理，满足大规模用户的操作需求。

2.系统设计

2.1 系统架构图

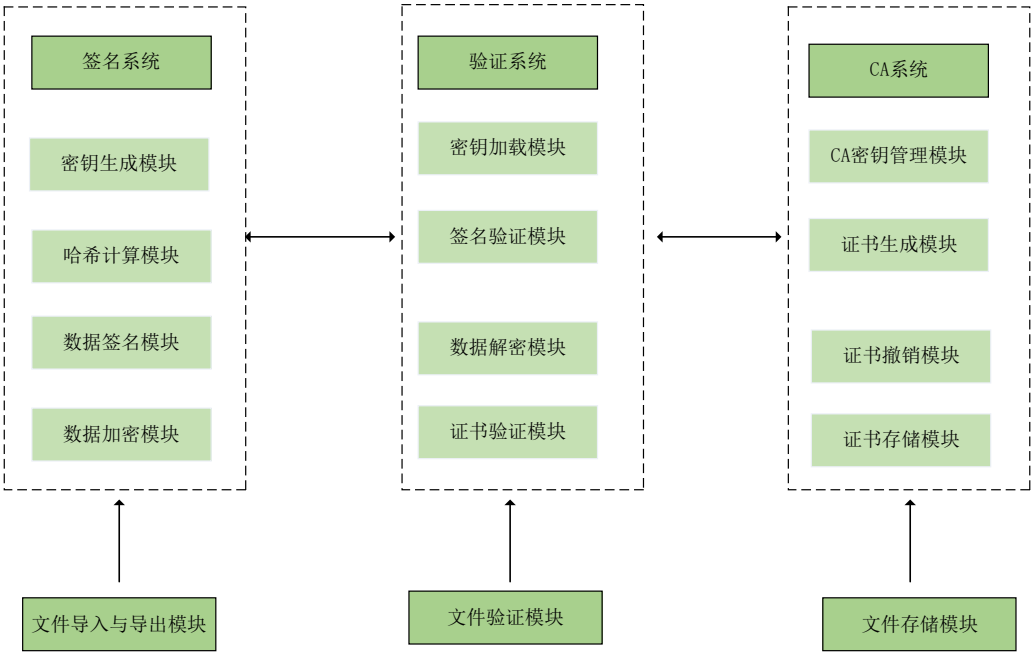

图 1 系统的功能结构框图

2.2 系统组件及交互方式

一、签名系统

1.密钥生成模块：

功能：基于 SM2 算法生成用户唯一的公私钥对。

交互：

与哈希计算模块和数据签名模块交互，提供密钥进行签名和加密。支持用户通过文件导入与导出模块保存或加载密钥。

2.哈希计算模块：

功能：对输入数据进行 SHA256 哈希计算，确保签名基于完整性验证。

交互：

接收来自用户输入或文件的明文数据。将计算的哈希值传递给数据签名模块。

3.数据签名模块：

功能：利用用户的私钥和哈希值生成数字签名。

交互：

与哈希计算模块交互，获取哈希值。与密钥生成模块交互，调用私钥完成签名操作。将生成的签名结果存储至文件或传递至验证系统。

4.数据加密模块：

功能：使用接收者的公钥对敏感数据进行加密。

交互：

接收用户输入或文件中导入的明文。与密钥生成模块交互，加载接收者公钥。输出加密结果至文件或传递至验证系统。

二、验证系统

1.密钥加载模块：

功能：加载签名系统生成的用户私钥或对方的公钥。

交互：

从文件中读取密钥，与签名验证模块和数据解密模块交互。验证所加载密钥的有效性，确保通信双方的安全性。

2.签名验证模块：

功能：验证数字签名是否合法，确保数据未被篡改。

交互：

从文件中读取签名和加密结果，与数据解密模块协作。调用证书验证

模块检查证书合法性，确保签名者身份可信。

3.数据解密模块：

功能：使用接收者私钥解密加密的数据。

交互：

接收加密数据，调用密钥加载模块的私钥完成解密。与哈希计算模块协作，对解密结果重新计算哈希值，用于签名验证。

4.证书验证模块：

功能：加载并验证数字证书的合法性，防止伪造。

交互：

与 CA 系统交互，获取并验证证书内容的真实性。

三、CA 系统

1.CA 密钥管理模块：

功能：生成 CA 系统的公私钥对，负责证书的签名与加密。

交互：

提供签名功能给证书生成模块。将 CA 公钥传递给证书验证模块，确保信任链的建立。

2.证书生成模块：

功能：为用户的公钥生成数字证书，并使用 CA 私钥进行签名。

交互：

从签名系统获取用户公钥信息。将生成的证书存储至证书存储模块，供验证系统使用。

3.证书撤销模块：

功能：支持证书的更新和撤销操作，管理证书生命周期。

交互：

接收用户撤销请求，更新证书存储模块中的证书状态。

四、证书存储模块：

功能：存储生成的数字证书，支持证书的加载与分发。

交互：

提供证书给验证系统的证书验证模块，确保签名验证的合法性。管理

证书的存储路径，支持批量证书操作。

五、文件模块

1.文件导入与导出模块：

功能：支持签名系统导入明文或密钥文件，导出签名和加密结果。

交互：

将导入的明文传递给哈希计算模块。存储签名和加密结果供验证系统使用。

2.文件验证模块：

功能：在验证系统中加载签名、加密结果和证书文件。

交互：

提供文件数据给签名验证模块、数据解密模块和证书验证模块。

3.文件存储模块：

功能：在 CA 系统中存储证书和密钥文件。

交互：

管理 CA 密钥和证书文件，支持证书更新与撤销操作。

2.3 系统交互流程

用户在签名系统中生成密钥，完成数据签名和加密，并导出签名与加密结果。

验证系统加载密钥、签名和加密文件，通过解密和签名验证，确保数据完整性。

CA 系统生成并管理用户证书，为签名和验证过程提供身份验证支持。

2.4 技术选型

1.Python

Python 拥有丰富的第三方库，适合实现密码学算法、文件处理和图形用户界面。语法简洁易懂，开发效率高，便于实现复杂的算法和功能模块。Python 对科学计算和安全算法的支持较为成熟，适合作为本项目的开发语言。

2.SM2 算法

SM2 是中国国家密码管理局发布的椭圆曲线公钥加密算法，符合国密标准，安全性较高。SM2 具有较高的性能，特别适合应用于签名和加密操作。使用 gmssl 库能够便捷地实现 SM2 的密钥生成、签名、验证等功能。

3.SHA256 哈希算法

SHA256 是一种广泛使用的哈希算法，具备良好的安全性，适合确保数据完整性。pycryptodome 提供了高效可靠的哈希算法实现，便于集成到签名生成和验证中。

4.Tkinter

Tkinter 是 Python 内置的 GUI 库，无需额外安装，跨平台支持良好。

提供简洁易用的界面设计工具，能够快速实现窗口、按钮、输入框等组件，满足项目功能需求。适合中小型项目的 GUI 设计，学习成本低。

我们使用模块化设计，将签名系统、验证系统和 CA 系统独立开发，便于未来扩展（如添加 SM3 或 SM4 算法支持）。

3.系统实现

在进行数字签名系统设计实现时，我们设计了签名系统、验证系统、CA 系统一共三个系统。

3.1 签名系统实现

在签名系统页面我们设计了密钥管理、操作区域、结果显示三个模块。在密钥管理模块可以生成用户的公私钥，输入用户 ID,按照用户用户 ID 保存公私钥文件在同一个文件夹中，在该模块中我们可以加载用户的私钥用于签名以及上传接收者的公钥文件用于对明文信息进行加密。在操作区域模块我们可以上传明文信息文件，下面的 SM2 加密、SHA 哈希、SM2 签名、保存到文件按钮可以实现用 SM2 加密和 SHA256 计算 hash 的签名。右边结果显示可以显示加密和签名结果，三个系统中的情况按钮均可情况当前系统页面所有变量。

```
# ------------------- 签名系统页面布局 -------------------
sign_content = ttk.Frame(sign_frame)
sign_content.pack(fill='both', expand=True, padx=20, pady=20)
left_panel = ttk.LabelFrame(sign_content, text="密钥管理")
```

```
left_panel.pack(side='left', fill='y', padx=(0,10))
ttk.Label(left_panel, text="用户 ID：").pack(pady=5)
user_id_entry = ttk.Entry(left_panel)
user_id_entry.pack(pady=5)
ttk.Button(left_panel, text="生成密钥", command=lambda:
generate_keys_with_id_signing(user_id_entry.get())).pack(pady=5)
ttk.Button(left_panel, text="保存密钥",
command=save_user_keys).pack(pady=5)
ttk.Button(left_panel, text="加载用户私钥",
command=load_user_keys_signing).pack(pady=5)
ttk.Label(left_panel, text="公钥：").pack(pady=5)
public_key_box = scrolledtext.ScrolledText(left_panel, width=50,
height=4)
public_key_box.pack(pady=5)
ttk.Label(left_panel, text="私钥：").pack(pady=5)
private_key_box = scrolledtext.ScrolledText(left_panel, width=50,
height=4)
private_key_box.pack(pady=5)
ttk.Button(left_panel, text="上传接收者公钥文件",
command=upload_recipient_public_key).pack(pady=5)
middle_panel = ttk.LabelFrame(sign_content, text="操作区域")
middle_panel.pack(side='left', fill='both', expand=True, padx=10)
ttk.Label(middle_panel, text="明文信息：").pack(pady=5)
plain_text_box = scrolledtext.ScrolledText(middle_panel, height=4)
plain_text_box.pack(fill='x', pady=5)
ttk.Button(middle_panel, text="上传文件",
command=load_plain_text_from_file).pack(pady=5)
ttk.Label(middle_panel, text="接收者公钥：").pack(pady=5)
recipient_public_key_box = scrolledtext.ScrolledText(middle_panel,
height=2)
recipient_public_key_box.pack(fill='x', pady=5)
ttk.Label(middle_panel, text="SHA256 哈希结果：").pack(pady=5)
sha_result_box = scrolledtext.ScrolledText(middle_panel, height=2,
state='disabled')
sha_result_box.pack(fill='x', pady=5)
button_frame = ttk.Frame(middle_panel)
button_frame.pack(pady=10)
ttk.Button(button_frame, text="SM2 加密",
command=sm2_encrypt).pack(side='left', padx=5)
ttk.Button(button_frame, text="SHA 哈希",
command=sha_hash).pack(side='left', padx=5)
ttk.Button(button_frame, text="SM2 签名",
command=sm2_sign_h1).pack(side='left', padx=5)
```

```
ttk.Button(button_frame, text="保存到文件",
command=save_to_file).pack(side='left', padx=5)
ttk.Button(middle_panel, text="清空",
command=clear_signing_display).pack(pady=5)
right_panel = ttk.LabelFrame(sign_content, text="结果显示")
right_panel.pack(side='right', fill='y', padx=(10,0))
ttk.Label(right_panel, text="SM2 加密结果（Base64）：").pack(pady=5)
sm2_encrypt_box = scrolledtext.ScrolledText(right_panel, width=50,
height=2)
sm2_encrypt_box.pack(pady=5)
ttk.Label(right_panel, text="数字签名：").pack(pady=5)
sm2_result_box = scrolledtext.ScrolledText(right_panel, width=50,
height=2)
sm2_result_box.pack(pady=5)
```

3.2 验证系统实现

在验证系统页面我们设计了密钥与证书信息、验证操作、验证结果三个模块。密钥与证书信息模块可以加载用户自己的私钥，上传签名文件。对于对方公钥可以直接上传公钥文件，也可以上传数字证书，经过 CA 认证后从数字证书中提取对方公钥。验证操作模块验证签名和解密密文两个按钮用于对数字签名验证以及对密文进行解密。验证结果模块可以显示上传的签名、解密后的明文及其计算的 hash 值，验签结果。

```
# ------------------- 验证系统页面布局 -------------------
verify_content = ttk.Frame(verify_frame)
verify_content.pack(fill='both', expand=True, padx=20, pady=20)
verify_left_panel = ttk.LabelFrame(verify_content, text="密钥与证书信
息")
verify_left_panel.pack(side='left', fill='y', padx=(0,10))
ttk.Button(verify_left_panel, text="加载用户私钥",
command=load_user_keys_verification).pack(pady=10)
ttk.Button(verify_left_panel, text="上传证书",
command=load_certificate_verify).pack(pady=10)
ttk.Button(verify_left_panel, text="上传对方公钥",
command=upload_signer_public_key).pack(pady=10)
ttk.Button(verify_left_panel, text="上传签名文件",
command=load_signature_and_ciphertext_verification).pack(pady=10)
ttk.Label(verify_left_panel, text="私钥：").pack(pady=5)
private_key_box_verify = scrolledtext.ScrolledText(verify_left_panel,
width=50, height=2)
private_key_box_verify.pack(pady=5)
```

```
ttk.Label(verify_left_panel, text="对方公钥：").pack(pady=5)
signer_public_key_box = scrolledtext.ScrolledText(verify_left_panel,
width=50, height=4)
signer_public_key_box.pack(pady=5)
verify_middle_panel = ttk.LabelFrame(verify_content, text="验证操作
")
verify_middle_panel.pack(side='left', fill='both', expand=True,
padx=10)
ttk.Label(verify_middle_panel, text="密文（Base64）：").pack(pady=5)
sm2_decrypt_box = scrolledtext.ScrolledText(verify_middle_panel,
height=3)
sm2_decrypt_box.pack(fill='x', pady=5)
ttk.Button(verify_middle_panel, text="验证签名",
command=sm2_verify_signature).pack(pady=10)
ttk.Button(verify_middle_panel, text="解密密文",
command=sm2_decrypt).pack(pady=10)
ttk.Button(verify_middle_panel, text="清空",
command=clear_verification_display).pack(pady=5)
verify_right_panel = ttk.LabelFrame(verify_content, text="验证结果")
verify_right_panel.pack(side='right', fill='y', padx=(10,0))
ttk.Label(verify_right_panel, text="解密后的明文：").pack(pady=5)
sm2_decrypt_box_result =
scrolledtext.ScrolledText(verify_right_panel, width=50, height=2)
sm2_decrypt_box_result.pack(pady=5)
ttk.Label(verify_right_panel, text="计算哈希：").pack(pady=5)
hash_display_box = scrolledtext.ScrolledText(verify_right_panel,
width=50, height=2)
hash_display_box.pack(pady=5)
ttk.Label(verify_right_panel, text="签名：").pack(pady=5)
verification_signature_box =
scrolledtext.ScrolledText(verify_right_panel, width=50, height=2)
verification_signature_box.pack(pady=5)
ttk.Label(verify_right_panel, text="验证结果：").pack(pady=5)
verification_result_box =
scrolledtext.ScrolledText(verify_right_panel, width=50, height=3)
verification_result_box.pack(pady=5)
```

3.3 CA 系统实现

在 CA 系统页面我们设计了 CA 密钥管理、CA 操作、CA 结果显示三个模块。CA 密钥管理模块可以用来生成 CA 的公私钥，可以进行保存，也可以加载已有的公私钥。CA 操作模块可以上传需要提供数字证书的公钥文

件，此处简化仅提供用户名作为身份认证，底部有颁发数字证书按钮实现对公钥签名，以及将该证书信息存储到 CA 库中，验证数字证书按钮用于证书签名校验和有效期检查，撤销数字证书按钮用于删除 CA 存储库中该数字证书的信息。保存证书按钮用于保存，加载证书按钮用于加载已有数字证书。CA 结果显示模块分别显示用户数字证书，验证结果，撤销结果。

```
# ------------------- CA 系统页面布局 -------------------
ca_content = ttk.Frame(ca_frame)
ca_content.pack(fill='both', expand=True, padx=20, pady=20)
ca_left_panel = ttk.LabelFrame(ca_content, text="CA 密钥管理")
ca_left_panel.pack(side='left', fill='y', padx=(0,10))
ttk.Button(ca_left_panel, text="生成 CA 公私钥",
command=generate_ca_keys).pack(pady=10)
ttk.Button(ca_left_panel, text="保存 CA 密钥",
command=save_ca_keys_func).pack(pady=5)
ttk.Button(ca_left_panel, text="加载 CA 公钥",
command=load_ca_public_key).pack(pady=5)
ttk.Button(ca_left_panel, text="加载 CA 私钥",
command=load_ca_private_key).pack(pady=5)
ttk.Label(ca_left_panel, text="CA 公钥：").pack(pady=5)
ca_public_key_box = scrolledtext.ScrolledText(ca_left_panel,
width=50, height=4)
ca_public_key_box.pack(pady=5)
ttk.Label(ca_left_panel, text="CA 私钥：").pack(pady=5)
ca_private_key_box = scrolledtext.ScrolledText(ca_left_panel,
width=50, height=4)
ca_private_key_box.pack(pady=5)
ca_middle_panel = ttk.LabelFrame(ca_content, text="CA 操作")
ca_middle_panel.pack(side='left', fill='both', expand=True, padx=10)
ttk.Label(ca_middle_panel, text="用户名：").pack(pady=5)
ca_username_entry = ttk.Entry(ca_middle_panel)
ca_username_entry.pack(pady=5)
ttk.Button(ca_middle_panel, text="上传用户公钥",
command=upload_user_public_key).pack(pady=5)
ttk.Label(ca_middle_panel, text="已上传用户公钥：").pack(pady=5)
user_public_key_display_box =
scrolledtext.ScrolledText(ca_middle_panel, width=50, height=2)
user_public_key_display_box.pack(pady=5)
ca_button_frame = ttk.Frame(ca_middle_panel)
ca_button_frame.pack(pady=10, fill='x')
ttk.Button(ca_button_frame, text="颁发数字证书",
command=issue_certificate).grid(row=0, column=0, padx=5, pady=5,
sticky='ew')
```

```
ttk.Button(ca_button_frame, text="验证数字证书",
command=verify_certificate_func).grid(row=0, column=1, padx=5,
pady=5, sticky='ew')
ttk.Button(ca_button_frame, text="撤销数字证书",
command=revoke_certificate_func).grid(row=1, column=0, padx=5,
pady=5, sticky='ew')
ttk.Button(ca_button_frame, text="保存证书",
command=save_certificate).grid(row=1, column=1, padx=5, pady=5,
sticky='ew')
ttk.Button(ca_button_frame, text="加载证书",
command=load_certificate).grid(row=2, column=0, padx=5, pady=5,
sticky='ew')
ttk.Button(ca_button_frame, text="清空",
command=clear_ca_display).grid(row=2, column=1, padx=5, pady=5,
sticky='ew')
ca_button_frame.grid_columnconfigure(0, weight=1)
ca_button_frame.grid_columnconfigure(1, weight=1)
ca_right_panel = ttk.LabelFrame(ca_content, text="CA 结果显示")
ca_right_panel.pack(side='right', fill='y', padx=(10,0))
ttk.Label(ca_right_panel, text="用户数字证书：").pack(pady=5)
certificate_box = scrolledtext.ScrolledText(ca_right_panel, width=50,
height=6)
certificate_box.pack(pady=5)
ttk.Label(ca_right_panel, text="验证结果：").pack(pady=5)
verification_result_box_ca =
scrolledtext.ScrolledText(ca_right_panel, width=50, height=3)
verification_result_box_ca.pack(pady=5)
ttk.Label(ca_right_panel, text="撤销结果：").pack(pady=5)
revoke_result_box = scrolledtext.ScrolledText(ca_right_panel,
width=50, height=3)
revoke_result_box.pack(pady=5)
```

3.4 核心函数实现

generate_key 用于生成公私钥，基于 SM2 算法

```
def generate_key():
    """生成 SM2 公私钥对"""
    private_key = random.randint(1, n-1)          # 随机生成私钥
    public_key = point_mul(private_key, Gx, Gy)  # 利用私钥生成公钥
    return private_key, public_key
```

sm2_sign 和 sm2_verify 分别为基于 SM2 进行签名和验证，因为我们会把密文传输给对方，密文中可以得到 k,所有验签函数中不需要额外提供 k,仅需提高 hash 值，公钥，签名值，通过用公钥对签名值解密得到的结果与 hash 值比较相等则验证成功，返回真。

```python
def sm2_sign(message_hash, private_key, k=None):
    """
    使用 SM2 对消息哈希进行签名
    message_hash: 16 进制形式的哈希字符串
    private_key: 私钥(int 类型)
    k: 可选的随机数，若不传则自动生成
    """
    if k is None:
        k = random.randint(1, n-1)  # 随机选取 k
    x1, y1 = point_mul(k, Gx, Gy)                       # 计算 k*G
    r = (int(message_hash, 16) + x1) % n                # 计算 r
    if r == 0 or r + k == n:
        # 若出现 r = 0 或 r+k = n，则重新签名
        return sm2_sign(message_hash, private_key)
    # 计算 s = ( (k - r*private_key) * (1 + private_key)^-1 ) mod n
    s = (invert(1 + private_key, n) * (k - r * private_key)) % n
    if s == 0:
        # 若 s = 0 则重新签名
        return sm2_sign(message_hash, private_key)
    # 返回 r 与 s 的 16 进制形式（填充至 64 个字符）
    return (hex(r)[2:].zfill(64), hex(s)[2:].zfill(64))
def sm2_verify(message_hash, signature, public_key):
    """
    使用 SM2 公钥验证签名
    message_hash: 16 进制形式的哈希字符串
    signature: (r, s) 元组
    public_key: (Px, Py) 元组
    """
    r, s = int(signature[0], 16), int(signature[1], 16)
    # 检查 r、s 范围
    if r < 1 or r > n-1 or s < 1 or s > n-1:
        return False
    t = (r + s) % n
    if t == 0:
        return False
    # 计算 s*G 与 t*Q
    x1, y1 = point_mul(s, Gx, Gy)
    x2, y2 = point_mul(t, public_key[0], public_key[1])
    # 点加
    x, y = point_add(x1, y1, x2, y2)
    # 计算 R
    R = (int(message_hash, 16) + x) % n
    return R == r
```

4. 测试结果

4.1 签名系统测试

1.首先生成两个用户的公私钥，用户 ID 为 zws,lhz。

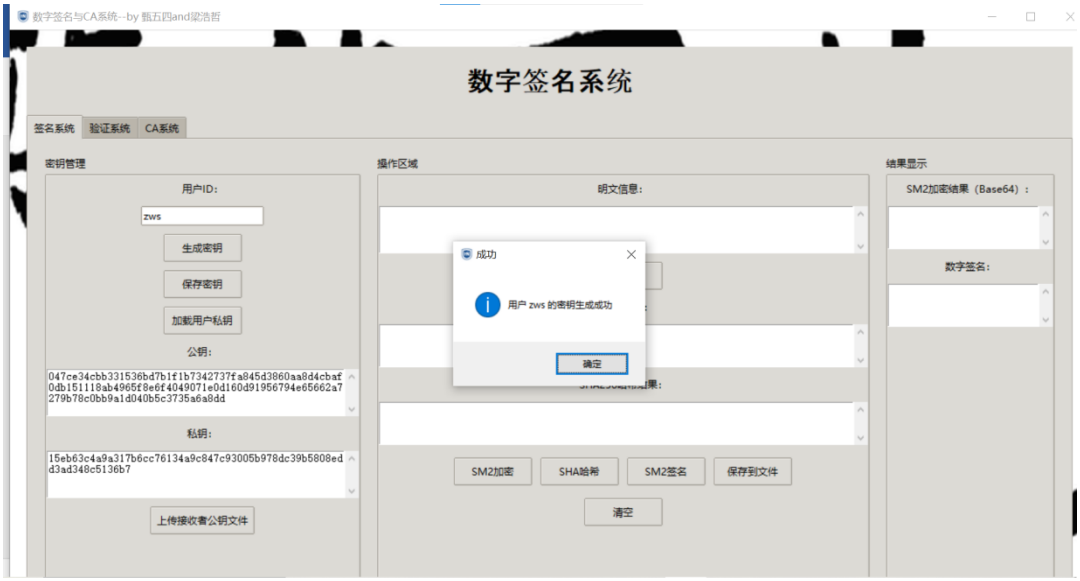

图 2 生成用户公私钥

2.将用户密钥保存

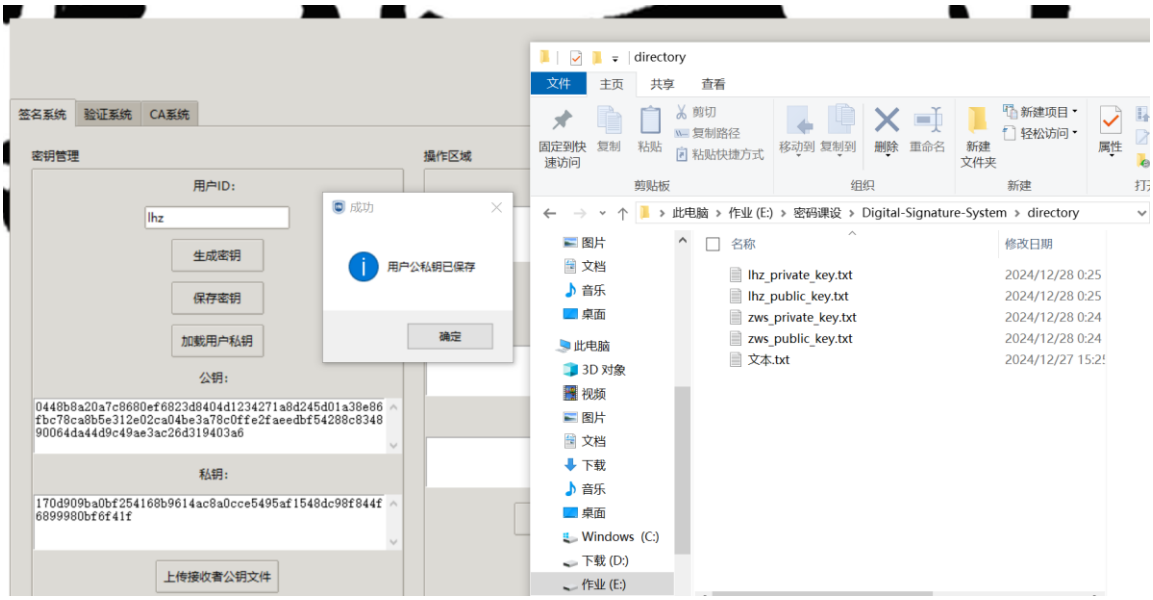

图 3 保存用户公私钥

3.点击加载用户私钥加载 zws 的私钥，点击上传接收者公钥文件上传 lhz 的公钥，然后点击上传文件。

图 4 上传公私钥和文件

4.依次点击 SM2 加密、SHA 哈希、SM2 签名、保存到文件，实现签名保存。

图 5 加密签名保存

4.2 验证系统测试

1.加载 lhz 用户私钥，上传 zws 用户公钥，上传签名文件。

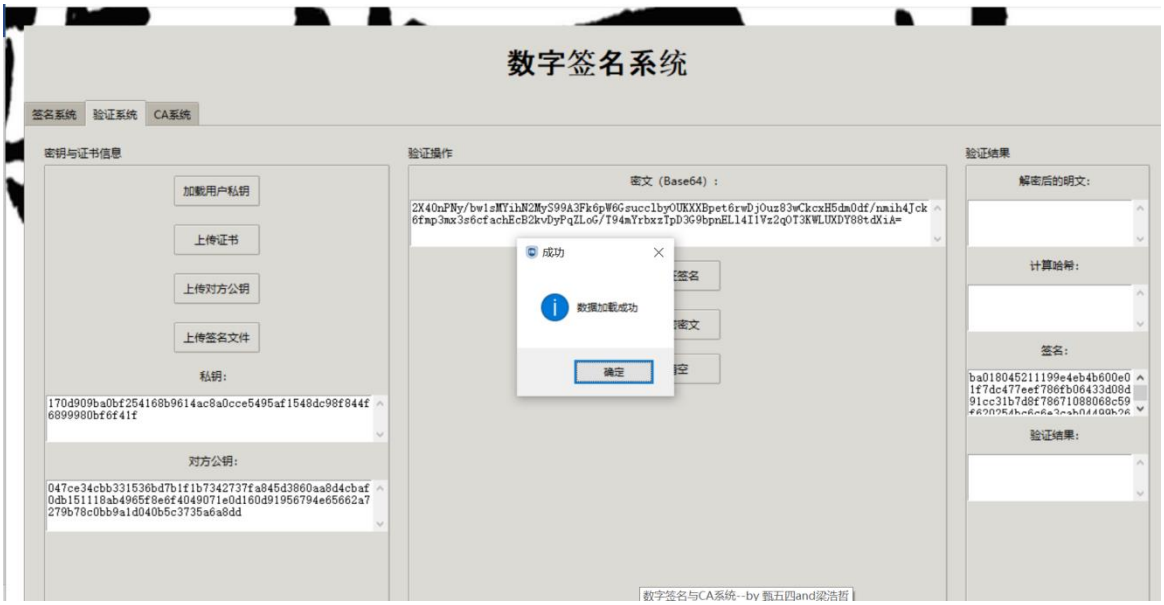

图 6 上传公私钥和签名文件

2.点击验证签名、解密密文按钮

图 7 验证签名和解密密文

4.3 CA 系统测试

1.点击生成 CA 公私钥生成密钥并保存，也可加载已有公私钥。

图 8 生成加载 CA 公私钥

2.输入用户名，上传用户公钥，即可颁发数字证书，点击保存证书可以进行保存。

图 9 颁发数字证书

3.点击加载证书按钮先进行证书加载，后面可以点击验证和撤销。

图 10 加载证书和验证数字证书

4.4 异常情况测试

1.生成密钥时需要输入用户 ID

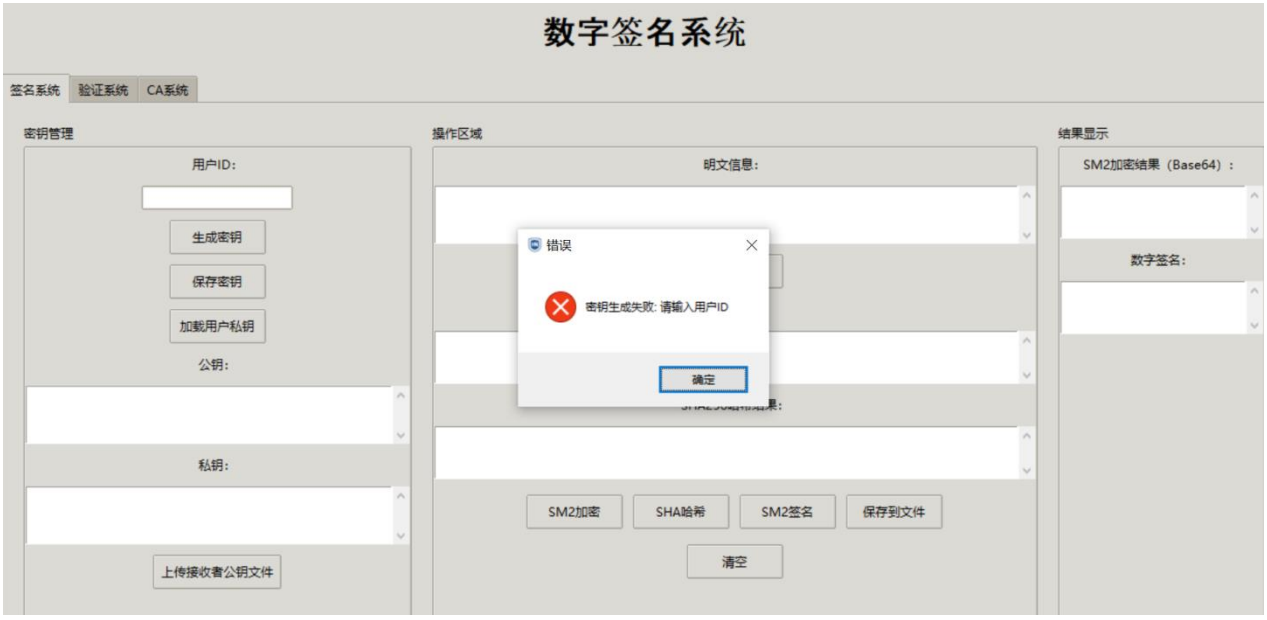

图 11 生成密钥无 ID 输入

2.上传非公钥、私钥文件

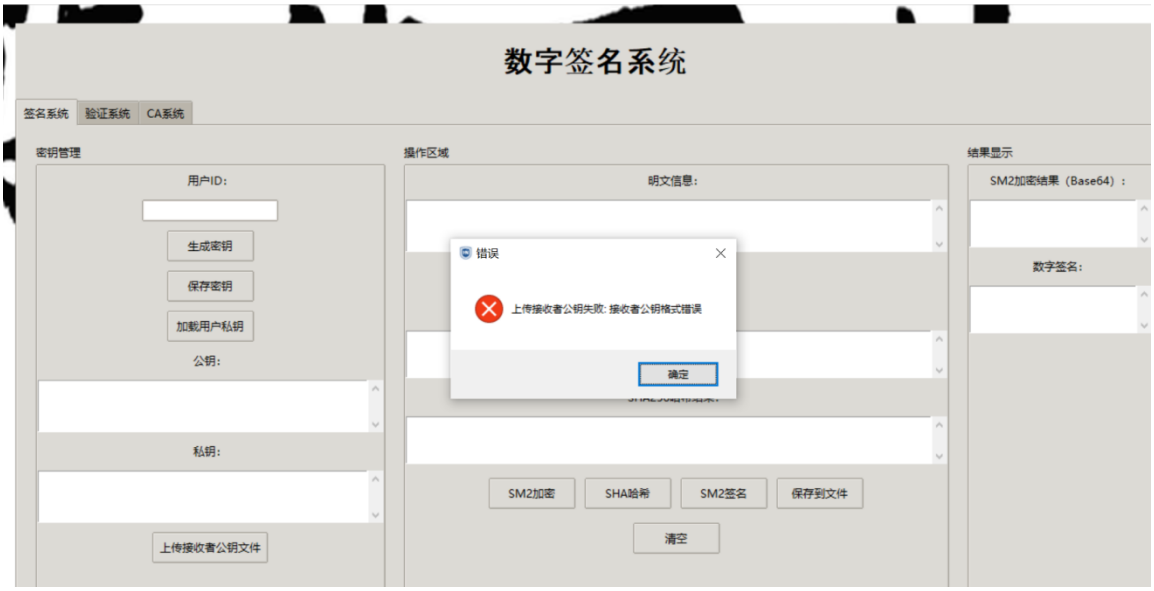

图 12 上传密钥错误

3.已撤销证书验证

图 13 撤销证书验证

5. 总结

本课程设计以国密算法 SM2 为核心，结合数字签名技术、公钥加密技术和证书管理机制，成功实现了一套完整的数字签名与验证系统。通过系统的签名模块、验证模块以及 CA（证书颁发机构）模块，达成了保障网络通信安全的目标，确保了数据的真实性、完整性、机密性和不可否认

性。

在项目开发过程中，针对现代网络安全所面临的主要问题，如中间人攻击、密钥管理复杂性和加密算法安全性等，系统采用了以下技术方案：

通过 SM2 算法生成密钥对，并实现了签名与验证功能，确保通信双方的身份真实可信。使用 SHA256 哈希算法进行数据完整性校验，确保签名的基础安全性。借助 CA 系统颁发和验证数字证书，解决了公钥分发和管理过程中的信任问题。

系统功能模块分工明确，涵盖了密钥管理、数字签名、加密解密、证书颁发与撤销等完整的功能链条。与此同时，系统通过图形化用户界面（GUI）友好地展示了复杂的密码学操作，使用户能够轻松完成签名生成、验证和证书管理等操作。

不足与改进方向：

尽管系统功能和安全性已达到预期目标，但在以下方面仍有改进空间：

性能优化：

当前系统对大规模密钥管理和高频操作的支持能力有限，可通过并行计算和算法优化提高运行效率。

跨平台支持：

目前系统基于 Python 开发，未来可以通过 Web 化改造或打包为桌面应用，增强跨平台性能。

安全增强：

引入 SM3 算法作为哈希函数，进一步提升系统的密码学强度。

增加抗量子计算的加密算法支持，如基于椭圆曲线的后量子密码学。

本次课程设计通过理论与实践相结合，深入探索了数字签名与验证技术的应用场景及实现过程，完成了一个功能完整、安全性高、易于使用的系统开发任务。该系统不仅巩固了我们对密码学算法和网络安全知识的理解，同时也提升了我们在需求分析、系统设计、功能开发和协作实践等方面的能力。通过此次设计，我们更加深刻地认识到密码学技术在现代网络通信中的重要作用，为日后的进一步学习和研究奠定了良好基础。

6. 参考文献

[1] 谭杰. 基于 PKI/CA 体系的电子签章系统研究与实现[D].南昌大学.2013

[2] 丁惠春,谷建华,张凡.面向电子政务应用的电子签章中间件设计与实现[J].计算机应用研究.2005,(3):135-137.

[3] 张福宾，《PKI 在电子政务中的应用》,信息化建设[J],2003.5。

[4] 黄元飞，陈麟，唐三平，《信息安全与加密解密核心技术[M]》， 浦东电子出版社，2001.08。

[5] 关振胜,《公钥基础设施 PKI 与认证机构 CA[M]》, 北京电子工业出版社 2002.04。

[6] 王礼强，《PKI 理论研究和实现[D]》，东南大学硕士论文，2000.5。

[7] 杨尔明，《数字证书技术的进展》,计算机安全[J],2004,9。

[8] 林乐文. CA 认证系统设计与实现 [D].山东大学.2007

[9] 宗倧.CA 系统中的 RSA 加密算法原理[J]. 广播电视信息.2013,(07)

7. 附件

系统源码以及架构图已经附在课程论文的文件夹。