学号 专业 信息安全 姓名

实验日期 **2024.11.26** 教师签字 成绩

实验报告

**【实验名称】** 可信计算第三次实验

**【实验目的】**

模拟基于Privacy CA的远程证明过程

实验分成3个通信端：示证者、证明者和CA

**初始化：**

CA启动，随机生成公私钥𝑆𝐾\_𝐶𝐴和𝑃𝐾\_𝐶𝐴，示证者和证明者连接过去，并获得这个公钥𝑃𝐾\_𝐶𝐴。（这条流程，其实就是很多密码系统里面的初始化） 示证者注册：CA随机生成一对公私钥𝑆𝐾\_𝑃和𝑃𝐾\_𝑃，并用自己的私钥𝑆𝐾\_𝐶𝐴对公钥进行签名获得𝜎=𝑆𝑖𝑔𝑛\_(𝑆𝐾\_𝐶𝐴 ) (𝑃𝐾\_𝑃 )，最后将{𝑆𝐾\_𝑃, 𝑃𝐾\_𝑃, 𝜎}发送给示证者。（这条流程，在TPM远程证明里面是离线操作的，TPM出厂时候，就烧录进去了）

**平台身份证书颁发：**

示证者随机生成一对AIK公私钥𝑆𝐾\_𝐴𝐼𝐾和𝑃𝐾\_𝐴𝐼𝐾，并用𝑆𝐾\_𝑃对这个𝑃𝐾\_𝐴𝐼𝐾进行签名获得𝜎\_𝐴𝐼𝐾，并将{𝑃𝐾\_𝐴𝐼𝐾,𝜎\_𝐴𝐼𝐾,𝑃𝐾\_𝑃, 𝜎}发送给CA；CA收到后，用自己的公钥𝑃𝐾\_𝐶𝐴验证{𝑃𝐾\_𝑃, 𝜎}元组中签名是否正确，正确的话说明这个𝑃𝐾\_𝑃对应的𝑆𝐾\_𝑃对应的私钥是被合法持有者拥有的，接着再用𝑃𝐾\_𝑃去验证{𝑃𝐾\_𝐴𝐼𝐾,𝜎\_𝐴𝐼𝐾}中签名𝜎\_𝐴𝐼𝐾的合法性，合法的话，证明𝑃𝐾\_𝐴𝐼𝐾是一个拥有正确硬件密钥𝑆𝐾\_𝑃的设备所产生的密钥，最后CA返回𝜎\_(𝐴𝐼𝐾,𝐶𝐴)=𝑆𝑖𝑔𝑛\_(𝑆𝐾\_𝐶𝐴 ) (𝑃𝐾\_𝐴𝐼𝐾 )，完成了AIK公钥的签发。

**远程证明：**

示证者对一组报告消息𝑅𝐸𝑃𝑂𝑅𝑇（里面按序放入实验二的多个PCR数据），使用𝑆𝐾\_𝐴𝐼𝐾进行签名，获得𝜎\_𝑅𝐸𝑃𝑂𝑅𝑇，并将{𝑅𝐸𝑃𝑂𝑅𝑇, 𝜎\_𝑅𝐸𝑃𝑂𝑅𝑇, 𝑃𝐾\_𝐴𝐼𝐾, 𝜎\_(𝐴𝐼𝐾,𝐶𝐴)}发送给验证者；验证者收到消息后，首先利用𝑃𝐾\_𝐶𝐴验证{𝑃𝐾\_𝐴𝐼𝐾, 𝜎\_(𝐴𝐼𝐾,𝐶𝐴)}的合法性，通过的话则再用𝑃𝐾\_𝐴𝐼𝐾验证REPORT, 𝜎\_𝑅𝐸𝑃𝑂𝑅𝑇的合法性，验证通过则认可了示证者的可信性（需要比对REPORT和自己的是否一致）。

**【实验原理】**

**三个通信端的作用**

1. **CA（认证中心）**  
   CA在整个体系中扮演可信第三方的角色。
   * **初始化阶段**：CA生成自己的公私钥对（SK\_CA，PK\_CA），并将公钥PK\_CA分发给示证者和验证者，让他们知道CA的公钥，这样他们就能验证由CA签发的签名数据。
   * **平台身份认证**：在注册过程中，CA为平台（示证者）生成的一对平台密钥（SK\_P，PK\_P）进行签名，形成平台证书。这在TPM的实际环境中通常是出厂烧录的过程。该签名证明了该平台的公钥PK\_P是合法可信的，具有由可信硬件（TPM）持有的私钥SK\_P作为对应。
   * **AIK公钥签发**：当示证者生成AIK密钥对后，CA使用SK\_CA对AIK公钥PK\_AIK进行签名，形成AIK证书（σ\_(AIK, CA)）。这一证书证明了AIK是由一个已被认证的可信平台密钥产生的，从而确保该AIK可用于后续的远程证明。
2. **示证者（Attester）**  
   示证者即需要证明自身可信状态的设备或平台。其职责包括：
   * 从CA获取平台证书，即（SK\_P, PK\_P, σ），该σ是CA对PK\_P的签名，证明此设备的平台密钥对是可信的。
   * 自主产生AIK密钥对（SK\_AIK，PK\_AIK），并将PK\_AIK及其由SK\_P签名后的签名σ\_AIK连同平台证书一起发送给CA请求AIK证书。
   * 在完成AIK证书签发后，示证者利用AIK私钥（SK\_AIK）对自身的可信报告（REPORT）进行签名，并连同AIK证书一同发送给验证者，以证明自身当前软件/硬件运行状态是可信的。
3. **验证者（Verifier）**  
   验证者是远程接收示证者提供的证明数据，并判断示证者是否可信的一方。
   * 接收示证者提供的REPORT、σ\_REPORT、PK\_AIK、AIK证书（σ\_(AIK, CA)）。
   * 首先利用PK\_CA验证AIK证书中的签名是否正确，从而确认PK\_AIK来自可信CA认证的AIK密钥对。
   * 再用PK\_AIK验证REPORT及σ\_REPORT的合法性。若验证通过，则说明示证者是在一个可被信任的状态下产生该REPORT。验证者可进一步比对REPORT内容是否符合预期从而决定是否信任示证者。

**三对密钥对的作用**

整个流程中涉及了三个关键的密钥对，它们分别是：

1. **CA的密钥对（SK\_CA，PK\_CA）**
   * **SK\_CA（私钥）**：CA使用此密钥对平台公钥和AIK公钥进行签名，从而签发平台证书与AIK证书。
   * **PK\_CA（公钥）**：公开给示证者和验证者，用于验证由CA签发的证书的真实性和完整性。验证者需要通过PK\_CA来确保AIK证书是可信的，从而信任PK\_AIK的合法性。
2. **平台密钥对（SK\_P，PK\_P）**
   * **SK\_P（私钥）**：这是示证者在出厂时已经预植入可信硬件（TPM）的私钥，用于为AIK公钥签名，从而向CA证明此AIK是该可信平台生成的。
   * **PK\_P（公钥）**：对应的公钥由CA签名形成平台证书，证明其是由可信的硬件设备持有的。CA在颁发AIK证书时要验证该PK\_P是否可信，从而确保AIK来自合法平台。
3. **AIK密钥对（SK\_AIK，PK\_AIK）**
   * **SK\_AIK（私钥）**：示证者自主生成，用于对自身的REPORT进行签名，以向验证者证明其REPORT的真实性和来自可信的硬件平台。
   * **PK\_AIK（公钥）**：通过向CA申请证书并获得签名（σ\_(AIK, CA)）证明其来源可信。验证者根据AIK证书确认PK\_AIK可信后，再用PK\_AIK对REPORT签名进行验证。

**签名与验证流程概述**

1. **平台证书签发（离线或出厂阶段）**
   * **签名过程**：  
     CA使用SK\_CA对平台公钥PK\_P签名
   * **验证过程**（在后续使用时验证者或CA再次验证）：  
     使用PK\_CA验证该签名
   * **作用**：  
     获得{SK\_P, PK\_P, σ}后，示证者持有的平台证书证明其平台密钥由可信的CA背书。此时信任链起点为：  
     CA（根信任） ——签署→ 平台密钥对（PK\_P, SK\_P）
2. **AIK密钥对生成与绑定**
   * **AIK生成**：  
     示证者在本地生成（SK\_AIK, PK\_AIK）。
   * **签名过程**：  
     示证者用SK\_P对PK\_AIK签名
   * **验证过程**（由CA进行）：  
     在申请AIK证书时，CA先验证示证者提交的{PK\_AIK, σ\_{AIK}, PK\_P, σ}：
     + 使用PK\_CA验证σ对PK\_P签名的正确性，以确保PK\_P可信：
     + 再使用已经确认可信的PK\_P验证PK\_AIK来自可信平台（合法持有SK\_P）
   * **作用**：  
     通过该验证过程，CA确认PK\_AIK由一个已认证的平台密钥（PK\_P对应的SK\_P）签署，从而证明AIK确实源自可信的设备。
3. **AIK证书签发**
   * **签名过程**：  
     CA使用SK\_CA对PK\_AIK签名，形成AIK证书
   * **验证过程**（在后续远程证明时由验证者执行）：  
     使用PK\_CA对AIK证书进行验证
   * **作用**：  
     通过这一步，AIK公钥得到CA的背书。此时信任链延伸为：  
     CA（根信任） ——签署→ 平台密钥对（PK\_P, SK\_P） ——签署→ AIK密钥对（PK\_AIK, SK\_AIK）
4. **远程证明**
   * **签名过程**：  
     示证者在远程证明阶段，用SK\_AIK对REPORT进行签名：
   * **验证过程**（由验证者执行）：  
     验证者在收到{REPORT, σ\_{REPORT}, PK\_{AIK}, σ\_{(AIK, CA)}}后进行如下验证：

1.使用PK\_CA验证AIK证书：若通过，则确认PK\_AIK可信。

2.使用PK\_AIK验证REPORT签名： 若通过，则确认该REPORT确实来自已认证的可信AIK私钥持有者（即示证者自身）。

**【实验内容】**

开发环境：Qt5.15.2

开发工具：QtCreator 14.0.1

语言：C++ 11

密码学库：cryptopp

详细代码文件与可执行文件见压缩包，本节介绍如何搭建socket通信并展示远程证明模拟过程中关键步骤的代码。

**三端通信的流程：**

采用QWebSocket实现：

在Qt项目的工程文件中加入“QT +=websockets”

引用：

#include<QtWebSockets/QWebSocket>

#include<QtWebSockets/QWebSocketServer>

本次实验中，一共创建了三个独立的项目，分别模拟CA,示证者，验证者



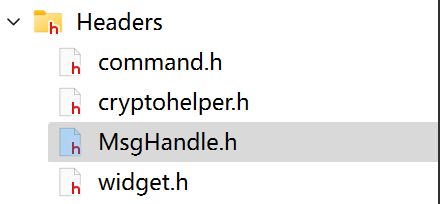
CAServer：CA

Prover：示证者

Verify:验证者

QWebSocket通信需要一个服务端。因此，本次实验以CA作为服务端。另外两端之间的通信需要以CA作为中继。

**CA端：**





创建一个专门的类Session用于处理网络消息，每一个Session对象都代表一个套接字（私有成员变量pSocket实际处理网络活动），用于唯一标识某个客户端与服务端之间的连接。

在Widget类的私有成员中，创建两个Session类对象，用于表示示证者与验证者。

创建一个QWebSocketServer对象用于开启服务端。

创建一个QWebSocket对象用于引用当前网络活动。



**服务端开启监听：**

在Widget的构造函数中，实例化QWebSocketServer私有成员变量，并设置监听端口为443，监听任何IPV4请求。

通过Qt的信号-槽机制，将接收到新网络请求的信号绑定给onnewconnection槽函数。



开启监听

**处理新的连接：**

从等待队列中获取一个网络活动，赋值给pSocket私有成员变量，并以pSocket作为构造函数参数创建Session对象。将Session的私有成员pSocket收到二进制消息的信号与Session::onMessageReceived函数绑定。onMessageReceived函数负责处理接收到的二进制消息。

确认示证者连接与验证者连接：

当启动示证者或验证者程序时，它们会向服务端CA发送一个含参请求（详见后续部分对另外两端的说明）。之前创建的Session类对象在捕获到这个请求后交给onMessageReceived函数处理。Session类对象会根据请求中包含的参数发出相应信号，告诉Widget类这个信号是由哪一端发出的，进而绑定示证者与验证者。



处理新连接，并绑定示证者，验证者

**示证者端（Prover）：**

注：本次实验为简化逻辑，只在CA端中设置了专门的处理网络通信类。在另外两端中，直接将网络通信方法放在了Widget类中。

创建一个私有成员变量QWebSocket \*pSocket，用于引用当前网络活动。

**与服务端建立连接，并发送确认示证者信号：**

在Widget类的构造函数中，实例化pSocket成员变量，并为其绑定事件。指定服务端地址后，打开连接。（此处使用的是127.0.0.1）

当接收到连接建立成功的信号后，调用sendMsg函数向服务端发送一个参数为CommandEnum\_Prover的请求。服务端（CA）在接收到该请求后，会将基于该连接创建的Session类对象赋值给Prover成员变量，完成示证者端的绑定。

****

与服务端建立连接，并发送确认示证者信号

验证者端逻辑大致同上，此处不再介绍。

**三端网络通信的公用方法与格式：**

注：前篇提到，只有CA有专门处理网络通信的类，因此CA的通信方法属于Session类的成员方法。另外两端则属于Widget类的成员方法。不过实现逻辑是一致的。

**通信的数据格式：**

每一个工程下都会有一个command.h头文件，里面定义了三端通信的数据格式。

定义了包头与包体两个结构体以及一个指令枚举command

在处理数据时，首先校验包头是否以CA开头。再判断长度，如果为-1，则表示没有发送实际数据，只是请求。若长度大于0，根据包体中的command判断要执行的操作。command字段的内容是枚举成员。



数据结构

**发送消息方法：sendMsg**

以CA为例：

有两个重载，第二个表示请求。

void sendMsg(void\*buf,int bufLen,int type,int error = 0,int mode = 2);

void sendMsg(int command, int error, int mode);

下面介绍重载一：

接受五个参数，其中起作用的为前三个参数，分别代表数据内容，数据长度，指令类型。

通过强制类型转换，将数据最终转为QByteArray格式，与创建的包头包体凭借后，调用pSocket发送数据。

注：调用该方法时，传入的数据一般为json格式的QByteArray

注：发送数据时，需要对数据进行封装。本次实验中采用的方法是，现创建一个QJsonobj对象，对其键值对进行赋值后，转为一个QJsonDocument对象。调用其toJson方法（参数为QJsonDocument::Compact）转为QByteArray后进行发送



重载一

重载二只需将包头的数据长度设为-1



重载二

**接收消息：onMessageReceived：**

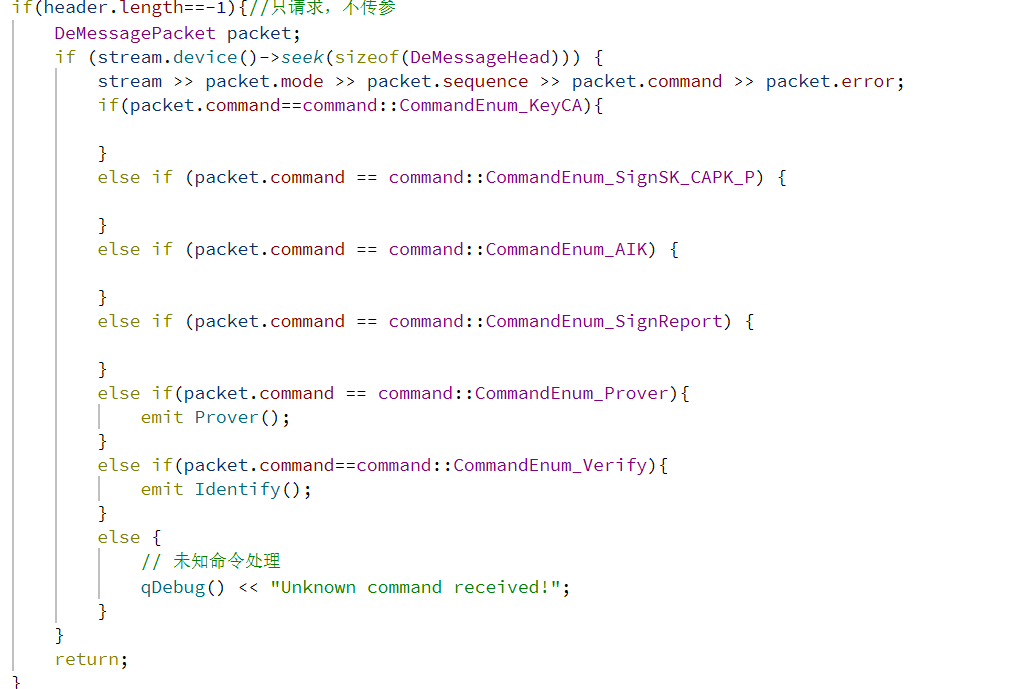
void onMessageReceived(QByteArray message);

该函数即为上文中提到的，与接收到二进制消息信号绑定的槽函数。

首先，会对接收到的数据进行格式校验



根据消息长度判断是否为请求：



处理请求

若不是请求，则检查消息长度，并处理数据，调用handlePacket函数进行处理



处理数据

**处理数据：handlePacket：**

根据解析出的指令，执行相应的函数，每个端的函数不同，具体功能放到后续的远程证明部分介绍

注：接收到的数据类型为QByteArray，应先将QByteArray转为QJsonDocumnet，再转为QJsonobj，根据key提取value。



根据指令类型执行相应操作（CA）

**远程证明过程中，三端的具体操作：**

**公用密码方法：**

Cryptopp库采用的是string格式，而Qt中通常采用QString。为方便公钥，私钥，string，QString之间的转换与编码，编写了如下的辅助方法。存放在cryptohelper文件中。



.h头文件，函数声明

受于篇幅限制，这里不一一介绍每个辅助函数的具体实现。主要作用是将RSA密钥转为文件，string；从文件，string中读取RSA密钥；将string转为base64编码的QString。具体函数实现见cryptohelper.cpp文件。

**CA端：**

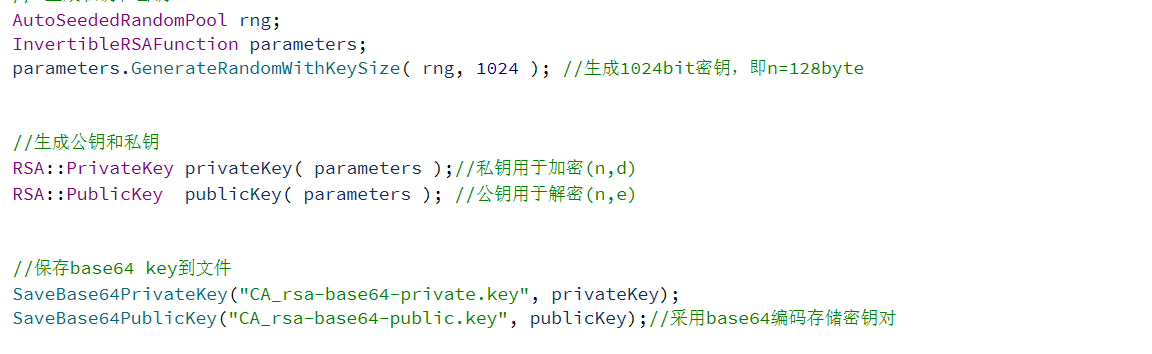
CA需要完成的操作包括：K\_CA密钥对的生成与派发；K\_P密钥对的生成；对PK\_P使用SK\_CA签名，生成平台证书；验证示证者发送的PK\_AIK的合法性；对PK\_AIK使用SK\_CA签名，生成AIK证书。



CA的公有成员函数

**K\_CA密钥对的生成与派发：KeyCA函数：**

该函数生成K\_CA密钥对，保存在本地，并将公钥派发



生成密钥对并保存到文件



向示证者与验证者发送公钥

**K\_P密钥对的生成：KeyP函数**

流程和上一步差不多，不用发送。

****

KeyP函数，生成K\_P密钥对

**对PK\_P使用SK\_CA签名，生成平台证书：VerifySignature函数**

该函数并用自己的私钥对公钥进行签名获得，最后将发送给示证者。

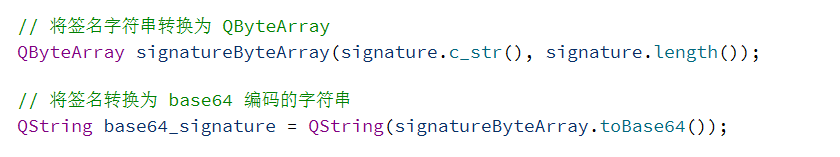
通过RSASS<PSS, SHA1>::Signer signer(CA\_private\_Key);创建签名器，并指定SK\_CA为签名密钥。

指定PK\_P的string形式为待签名的内容



对PK\_P签名

生成的签名为string类型，要想使用QWebSocket进行发送，需要将其转为base64编码的QString类型封装进json。转为QByteArray后进行发送。



格式转换



**封装数据，发送给示证者**

**验证示证者发送的PK\_AIK的合法性；对PK\_AIK使用SK\_CA签名，生成AIK证书：VerifyAIK\_Signature函数**

该函数在CA收到后示证者发送的{𝑃𝐾\_𝐴𝐼𝐾,𝜎\_𝐴𝐼𝐾,𝑃𝐾\_𝑃, 𝜎}后进行调用。首先对Json格式数据校验，查看是否包含正确字段。若包含，则提取出数据，转回string格式以便使用。



字段校验与数据提取

接下来开始验证PK\_P的合法性，使用PK\_CA验签。PK\_P公钥从string中加载；PK\_CA从文件中加载。（调用辅助方法）



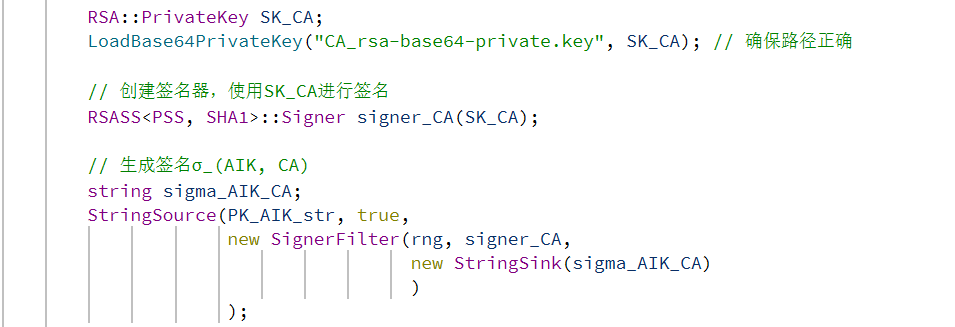
验证PK\_P

若PK\_P验证通过，则使用PK\_P验证PK\_AIK的合法性



验证PK\_AIK

两次验证均通过后，CA颁发AIK证书。使用SK\_CA对PK\_AIK进行签名

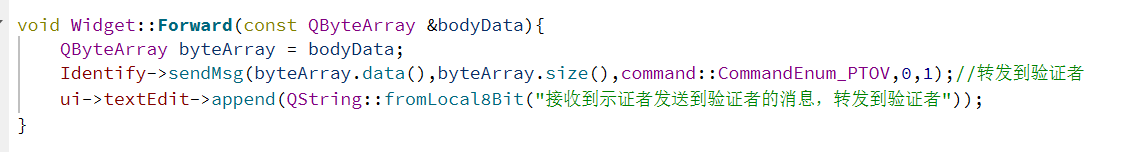


生成AIK平台证书

最后，将证书发送给示证者与验证者

**中继：Forward函数**

远程证明需要示证者给验证者发送消息，QWebSocket的客户端无法直接通信，需要借助服务端作为中继。只需接收示证者的数据，发送给验证者即可。



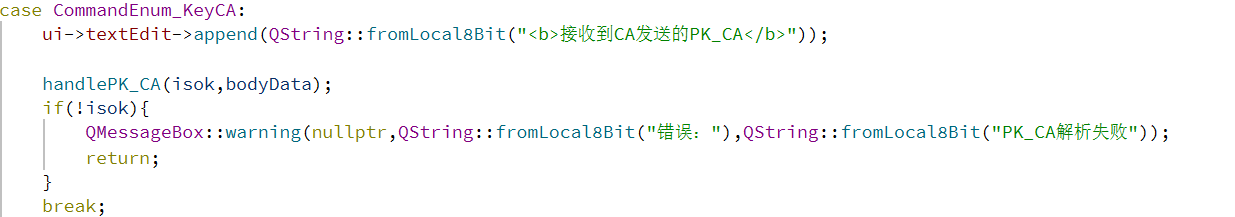
转发

**示证者端（Prover）**

示证者端需要完成的操作有：从CA获取PK\_CA与平台证书；产生AIK密钥对（SK\_AIK，PK\_AIK），并将PK\_AIK及其由SK\_P签名后的签名σ\_AIK连同平台证书一起发送给CA请求AIK证书；用SK\_AIK对报告（REPORT）进行签名，并连同AIK证书一同发送给验证者。

**获取PK\_CA: handlePK\_CA函数**

当检测到操作类型为CommandEnum\_KeyCA时，触发该操作。



调用接收PK\_CA的函数

从QByteArray中提取出数据后，保存到私有成员变量CA\_Public\_Key\_str中。



接收PK-CA

**接收平台证书：VerifySignature函数**

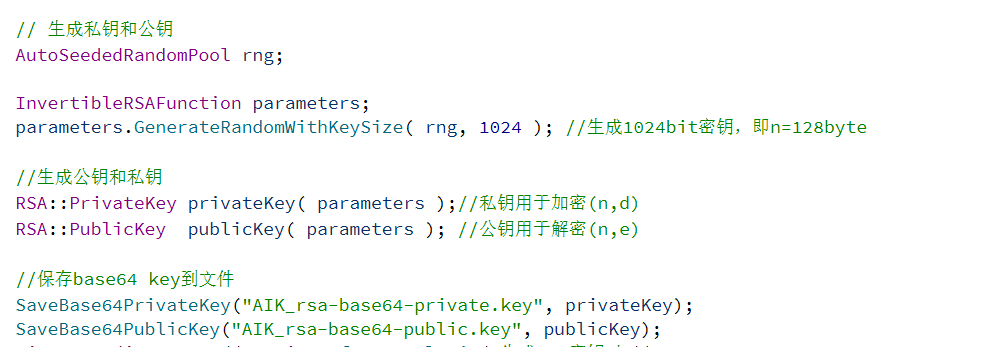
大体流程同上



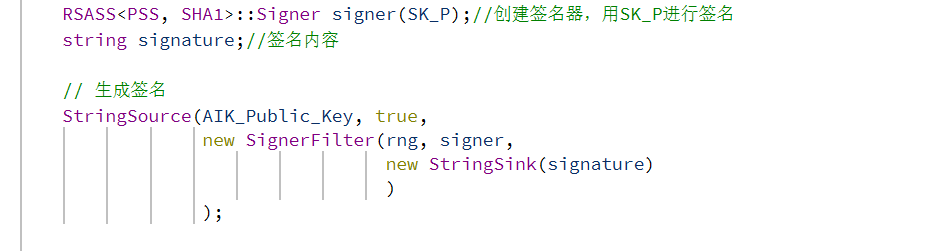
接收平台证书

**创建AIK密钥对，并将PK\_AIK及其由SK\_P签名后的签名σ\_AIK连同平台证书一起发送给CA：CertificateAIK函数**

首先创建AIK密钥对，保存到本地，并转为base64编码的QString便于传输



接着使用SK\_P对PK\_AIK签名，这一步的作用是向CA证明平台合法持有PK\_P与SK\_P



将发送给CA



发送数据

**接收AIK证书：Recieve\_sigma\_AIK\_CA函数**

同上解析QByteArray即可

**用SK\_AIK对报告（REPORT）进行签名，并连同AIK证书一同发送给验证者：Sign\_Report函数**

这里的REPORT采用的是文件的哈希值。首先打开文件并读取内容，计算出文件哈希值。



计算哈希

对计算出的文件哈希使用SK\_AIK进行签名



SK\_AIK签名

封装数据，将发送给验证者。这里需要注意采用的操作类型需要特别定义，告诉CA中继转发这条消息到验证者端。

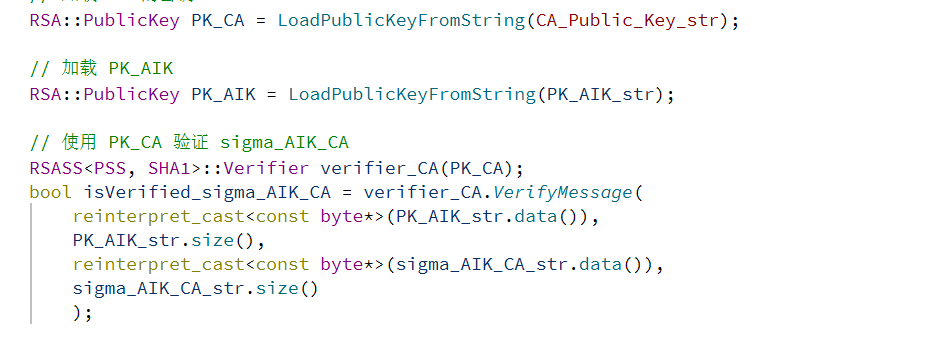


向验证者发送数据

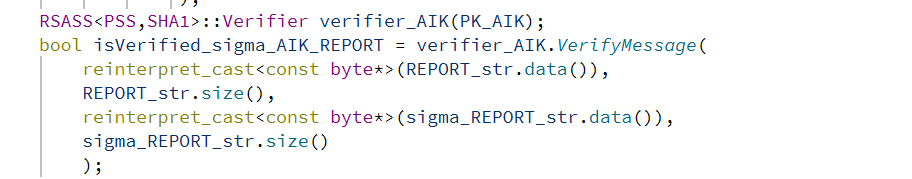
**Verify（验证者）端：**

接收数据的操作和Prover验证者端相同，这里只介绍验证者远程证明过程的操作，即验证PK\_AIK与REPORT。由RemoteProof函数实现

从QByteArray中提取数据的方法同上。

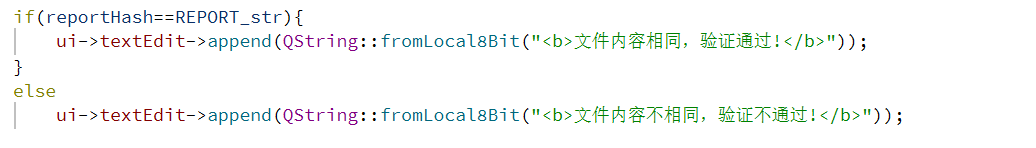


验证PK\_AIK



验证REPORT



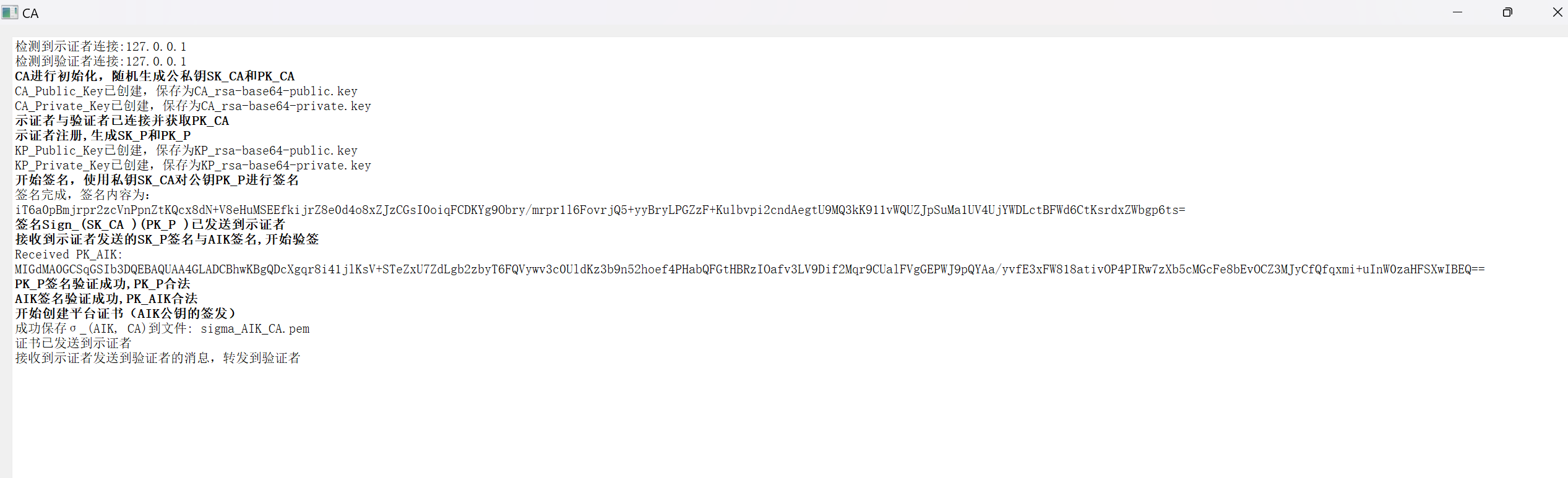


比较本地文件哈希与接收的是否一致

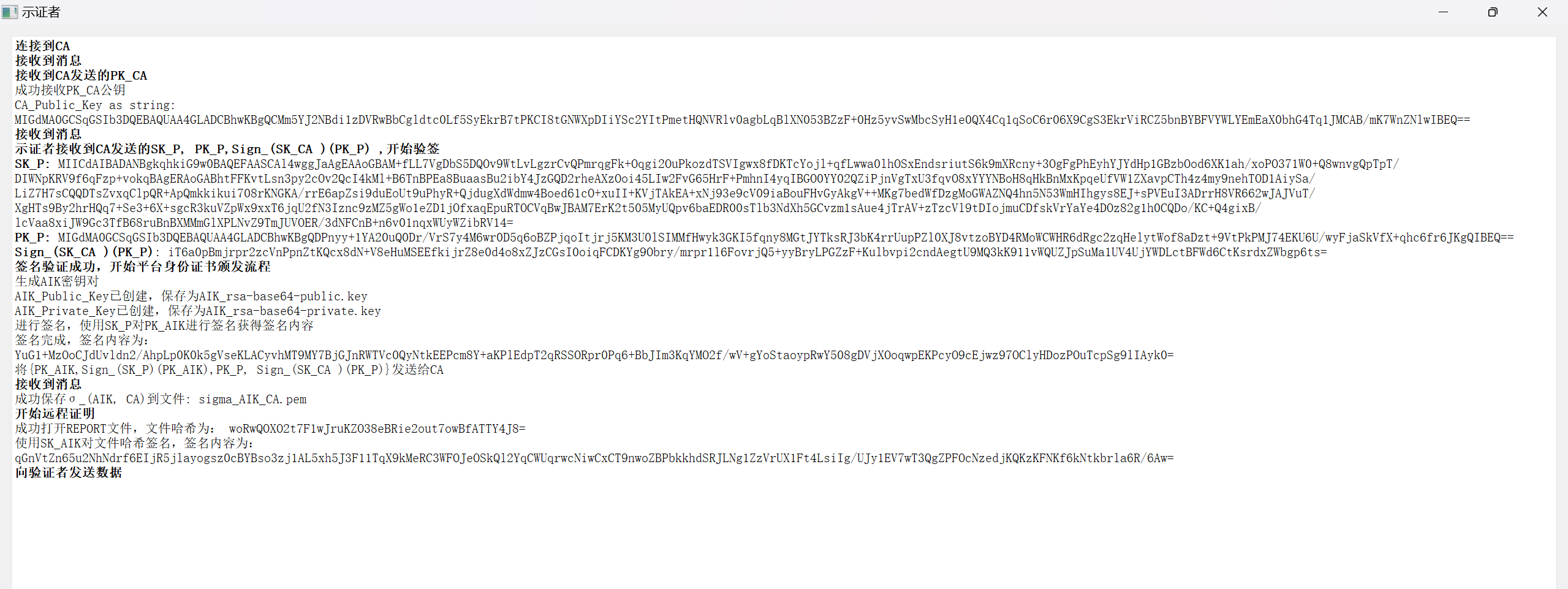
**系统运行截图：**

注：运行时，要先启动CA，打开监听。之后再启动验证者与示证者端（顺序无影响）

**CA端：**

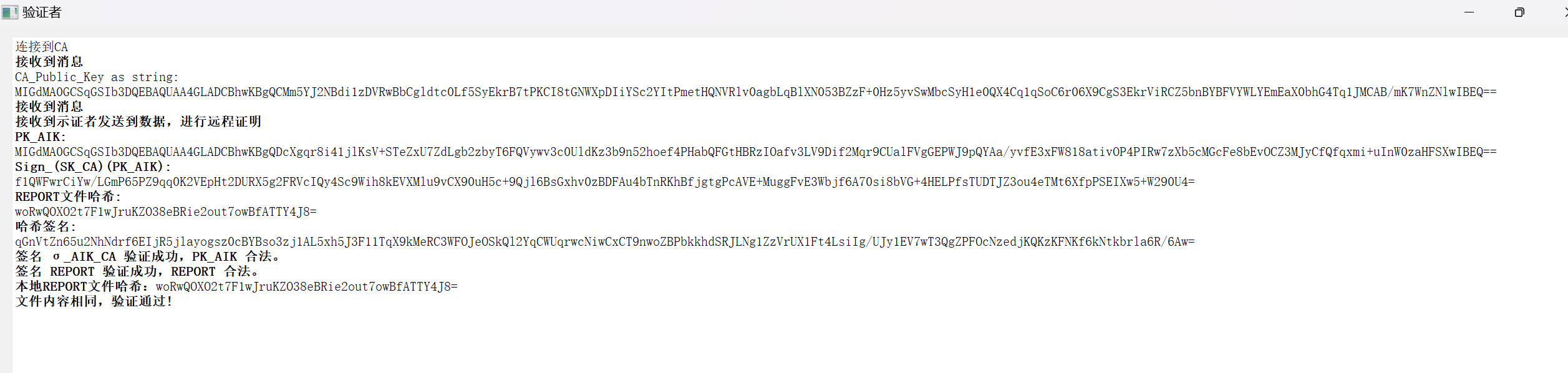


**Prover端：**



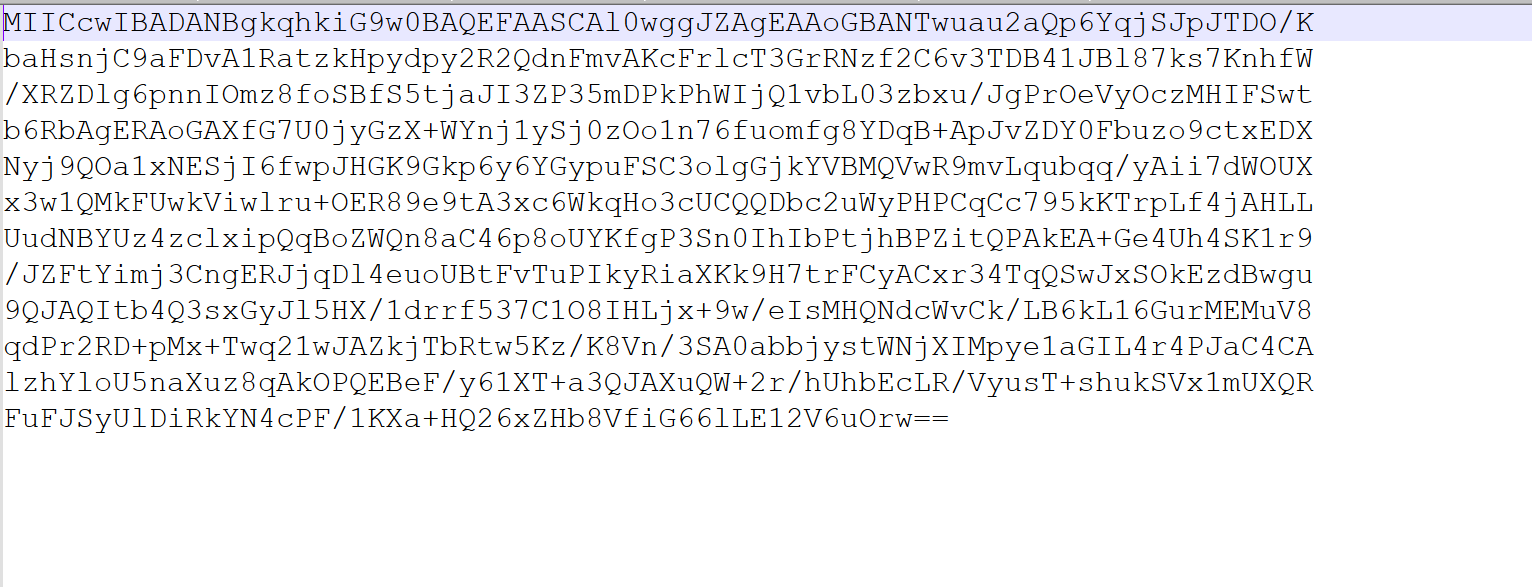
（这里多验证了一步CA的平台证书，即验证了PK\_P的合法性这是不需要的）

**Verify端：**

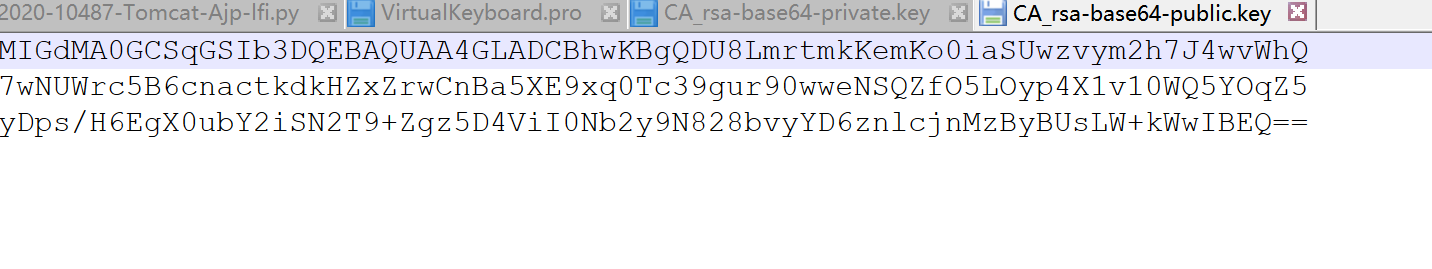


注：密钥在文件中以DER格式存储，系统提供了转为.pem格式的方法。

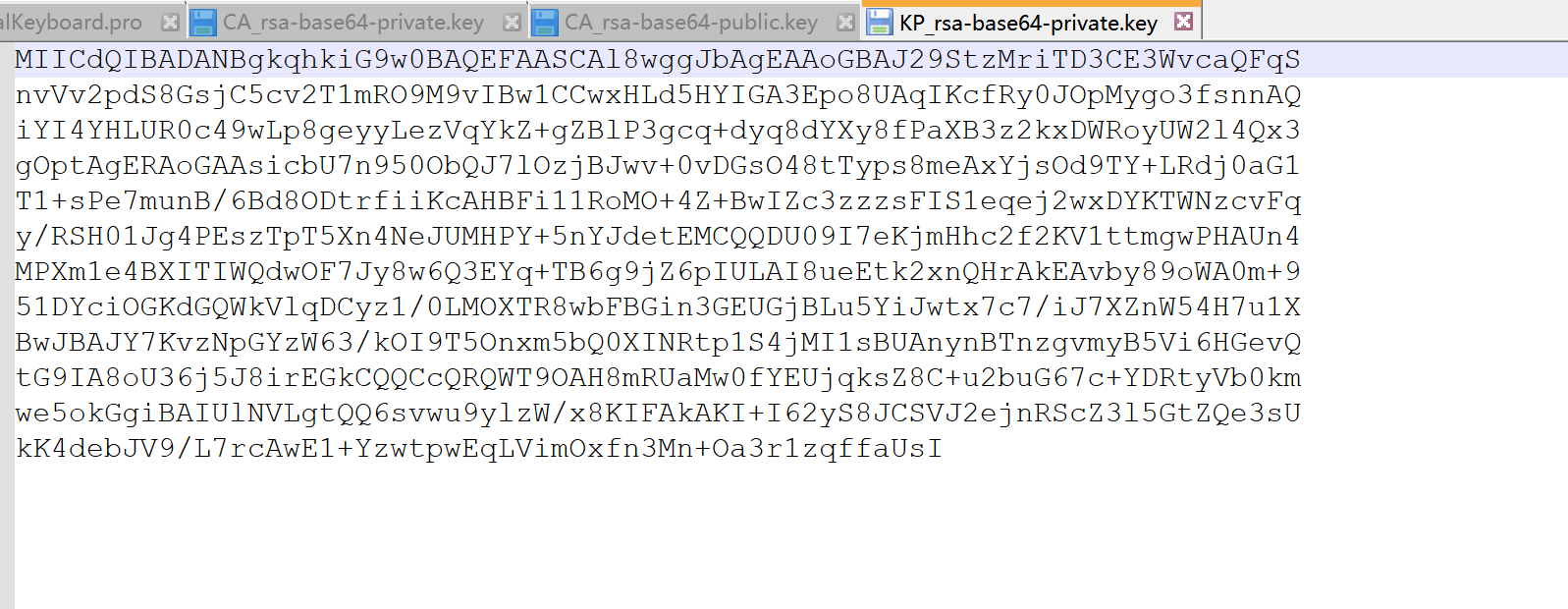
SK\_CA内容



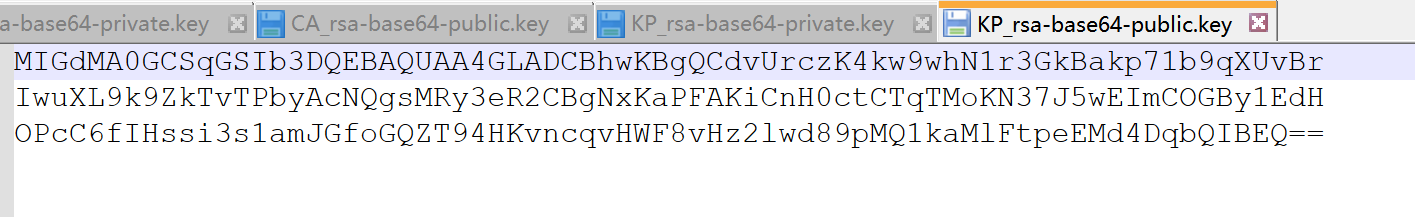
PK\_CA内容:



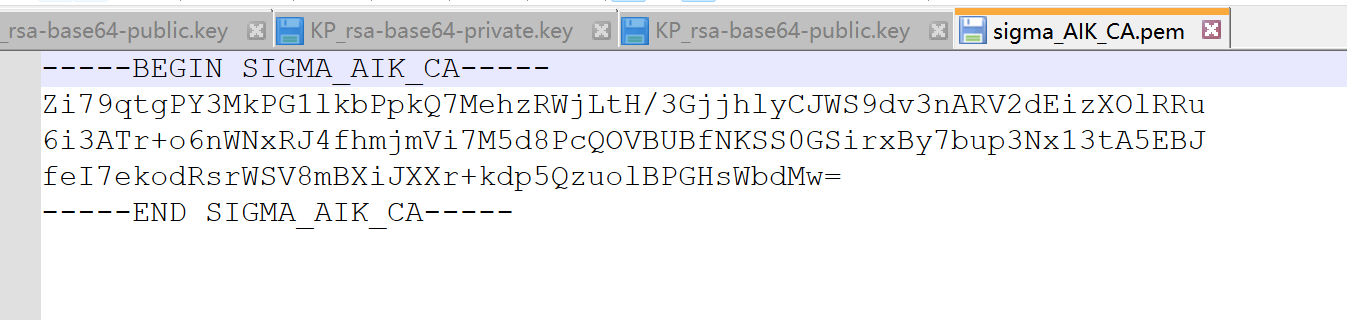
SK\_P内容：



PK\_P内容：



AIK证书内容（.pem）：

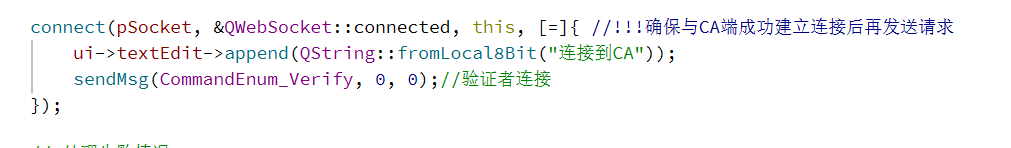


**【实验总结】**

这次实验模拟了基于Privacy CA的远程证明。下面说一下实验中踩的坑。

1.CA接收不到验证者与示证者端发来的确认连接请求：

这是因为最初把向CA发送请求的数据直接写在了建立连接后，但这是不正确的，客户端与服务端建立连接是需要时间的，如果这样做会导致在没有成功建立连接时就调用了发送消息的函数，这显然是不成功的。因此，应当把向CA发送请求这一事件写在接收到成功建立连接信号的槽函数中。



2.编码问题

之前的部分也提到了，Cryptopp用的string，Qt用QString。虽然Qt由fromstdstring和tostdstring方法，但如果直接调用的话会莫名其妙多出来很多长度。网上说要把string格式的密钥转为base64编码的QString，试了一下确实没问题了。怀疑是fromstdstring和tostdstring方法把’\’变成了’\\’。（网上教程还有用CString的，C++的string真是五花八门）

3.验签不匹配问题

这个问题困扰了我很久，明明发送数据接收数据没问题，为什么验签就是不通过？排查很久才发现，写代码的适合签名和验签用的哈希算法不一样，一个是SHA1,一个是SHA256。签名和验签过程中使用的哈希方法一定得保持一致。

上面三个算是排查了比较久的问题。其他的小问题那就数不清了。Cryptopp库的使用实在网上找文章照葫芦画瓢的，AI写密码学方面的还是很不靠谱。这次实验加深了我对远程认证的理解，也算复习了RSA算法，证书的概念等知识。对QWebSocket的使用也更熟练了一些，以后可以考虑用多线程优化。