Jussi Salminen

APPLYING PKI TO DEVICES IN A NETWORK

Information Technology

Software Engineering

2013

Satakunnan ammattikorkeakoulu

Satakunta University of Applied Sciences

APPLYING PKI TO DEVICES IN A NETWORK

Salminen, Jussi
Satakunta University of Applied Sciences
Degree Programme in Information Technology
July 2013
Supervisor: Trast, Ismo
Number of pages: 29
Appendices: 2

Keywords: secure communication, cryptography, public key infrastructure

The purpose of this thesis was to develop an application for secure communication in a public network. The goal was to have a certification authority (CA) –based implementation so that each device using the application is identified.

This was achieved by exploring the most recent cryptographic algorithms, public key infrastructure, hardware security modules (HSM) and third party libraries. The application was developed in Java, which natively supports useful cryptographic functionality.

The application has been tested using a HSM/PKCS#11 device simulator (openCryptoki), showing positive results. Due to development conditions, the application could not be tested using real hardware, but the simulator complies with standards, thus it is expected to work with any HSM/PKCS#11 device.

APPLYING PKI TO DEVICES IN A NETWORK

Salminen, Jussi
Satakunnan ammattikorkeakoulu
Tietotekniikan koulutusohjelma
Heinäkuu 2013
Ohjaaja: Trast, Ismo
Sivumäärä: 29
Liitteitä: 2

Asiasanat: turvallinen viestintä, salaustekniikat, julkisten avainten hallintajärjestelmä

Tämän opinnäytetyön tarkoituksena oli toteuttaa sovellus, joka mahdollistaa turvallisen viestinnän avoimessa verkossa. Tavoitteena oli varmentaja (certificate authority) -pohjainen implementaatio, jotta jokainen sovellusta käyttävä laite on identifioitu.

Tavoite saavutettiin tutkimalla nykyisiä salausmenetelmiä, julkisen avaimen hallintajärjestelmää (PKI), HSM:ja (hardware security module) ja kolmannen osapuolen kirjastoja. Sovellus toteutettiin Java -ohjelmointikielellä, joka tukee natiivisti hyodyllisiä salausmenetelmiä.

Sovellus on testattu HSM/PKCS#11 simulaattorilla (openCryptoki) positiivisin tuloksin. Sovelluksen kehitysolosuhteista johtuen sovellusta ei voitu testata oikealla laitteella, mutta simulaattori noudattaa standardeja, joten sen pitäisi myös toimia millä tahansa HSM/PKCS#11 laitteella.

CONTENTS

ABBREVIATIONS

AES	Advanced Encryption Standard
API	Application Programming Interface
CA	Certification Authority
HSM	Hardware Security Module
MAC	Message Authentication Code
NIST	National Institute of Standards and Technology
NSA	National Security Agency
OAEP	Optimal Asymmetric Encryption Padding
PIN	Personal Identification Number
PKCS	Public Key Cryptography Standards
PKI	Public Key Infrastructure
RA	Registration Authority
RSA	Cryptographic public-key algorithm invented by Ron Rivest, Adi Shamir and Leonard Adleman

1 INTRODUCTION

Security and privacy is a major concern when it comes to today's communication in an open network. When two or more parties communicate in public domain in confidence, the communication must be secured. To achieve secure communication, cryptographic mechanisms are applied for the messages between the parties. Cryptographic mechanisms and a properly implemented secure system enable confidentiality, integrity and authenticity for the communication. In other words, they provide ways and means to avoid eavesdropping and data tampering.

The goal of this thesis was to explore cryptography to an extent that an application demonstrating secure communication could be developed. This was achieved by being involved in the development of software focused on security and studying existing literature about the topic. The implementation of the application can be used as a reference or base for those who intend to develop such application for secure communication.

2 CRYPTOGRAPHY

2.1 Context

Cryptography is a practice to secure communication from adversaries. This is achieved by means of applying an encryption mechanism on a message (also known as plaintext or cleartext). When a message is encrypted, it can be securely transferred to its intended receiver in a public network. The receiver decrypts the encrypted message (also known as ciphertext) to retrieve the original message. Cryptography has enabled some real-life applications such as internet banking and on-demand streaming for their intended customers, which is a great feat for today's businesses.

Although cryptography provides security, the system to which it is applied must be well designed. Adversaries may have various ways to attack a system, such as trying to compute the secrets by themselves, attempting to track a flaw in the implementation or simply by bribing the people using the system. Therefore the implementation has to be robust, even at the cost of performance. It is also safer to have a system requiring multiple people to access rather than just one person.

2.2 Cryptographic algorithms

The most common and widely used cryptographic mechanisms are asymmetric and symmetric algorithms. When encryption or decryption is performed on a given message using one of these algorithms, a key is always involved in the operation. Thus, the key is used to protect the message. In symmetric-key algorithms, a single key is shared between the sender and the receiver. This denotes that the key is used for both encryption and decryption. In asymmetric-key algorithms (often mentioned as public key algorithms), the sender and receiver hold key pairs. One of the keys is public and one is private (or secret). These keys are mathematically related such that the public key is used to encrypt a message and the private key is used to decrypt a message (Figure 1). As the words imply, public keys are shared between the sender and receiver and private keys are kept secret by the key owner. (Wikipedia: Cryptography 2013)

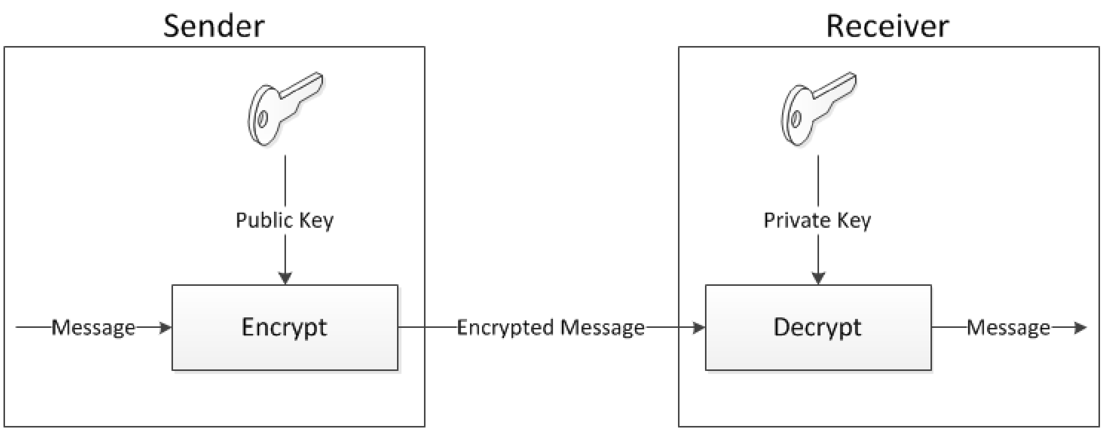

Figure 1. Public-key algorithm in practice.

2.3 Digital signatures

Digital signatures are used to verify that a message was sent by a certain party. In addition to the encryption, the digital signature is calculated on the plaintext and sent together with the message. The receiving party decrypts the ciphertext, calculates and verifies the digital signature from the plaintext. In asymmetric-key algorithms, the digital signature is calculated with the private key and verified with the public key. For symmetric-key algorithms, another key is shared between the sender and the receiver. This key is used for the digital signature calculation which is also known as message authentication code (MAC). Despite it is possible to sign a message using symmetric keys, in practice this method is not common because the MAC does not prove that the message has been sent by the expected sender. Digital signatures are extremely useful and good way to provide for authenticity. (Wikipedia: Message authentication code 2013)

2.4 Applications

Both asymmetric and symmetric-key algorithms have their own applications. For instance, symmetric-key algorithms are useful when you want to protect your own data. In this case, there would be no need to create a public/private key pair for such activity, since you alone would be the only person carrying the secret key. Albeit symmetric-key algorithms are computationally fast and simple, they are unusual in secure communication since the initial key exchange becomes a problem. The receiver and sender have to find a way to share the secret key before the communication. Sharing a secret key via a public channel should never be considered, as there is always the chance of being eavesdropped. Also, it would be inconvenient to establish communication between many parties; each party would have to agree on a secret key with the receiving party. If there were 30 parties, each party would have to perform 29 key exchanges, which is undoubtedly a bad practice. Asymmetric-key algorithms solve this problem, since the public key can be shared with every party. The key exchange is therefore not an issue either, as the private key is kept secret. This way, a symmetric key can be shared by encrypting it with the private key, should there be such requirement for an application. However, asymmetric-key algorithms are computationally slow and decrypting a large amount of data within a limited time frame could be an issue, so the method of exchanging a symmetric key via asymmetric-key mechanism is a good way to provide performance. (Wikipedia: Asymmetric key cryptography 2013)

3 RSA

3.1 Definition

One of the most widely used public-key algorithms is the RSA algorithm. It was invented by Ronald L. Rivest, Adi Shamir and Leonard Adleman in 1977. Its security is based on factoring of large prime numbers. The encryption is based on publicly known values **n** (as modulus) and **e** (as public exponent) such that the message is computed to the power of **e** modulo **n**. This is a quick operation, but computing the plaintext from a ciphertext with the given public values **n** and **e** is extremely difficult and time-consuming. The decryption operation involves a secret value **d** (as private exponent) and is used exactly the same way as in the encryption operation; the ciphertext is computed to the power of **d** modulo **n**. From here it is clear that the (**e**, **n**) values form the public key and the (**d**, **n**) values form the private key.

Practically, it is impossible to derive a private key from the given public key. Messages are encrypted with the public key and only the corresponding private key is able to decrypt the encrypted message. Therefore a party may share their public key with anyone and keeps the private key secret. Then anyone can send encrypted messages to this party, which only the receiver can decrypt. (Wikipedia: RSA algorithm 2013)

3.2 Key Generation

An RSA key pair is generated in the following way:

1. Two large prime numbers, commonly called **p** and **q** are randomly chosen. They should be never equal.
2. The modulus is calculated so that **n = pq**.
3. The public exponent **e** is chosen so that **1 < e < n** for which **(p-1)(q-1)** is coprime.
4. The private exponent **d** is chosen so that **de = 1 (mod (p-1)(q-1))**.
5. The RSA key now consists of the public key (**e**, **n**) and the private key (**d**, **n**).

The prime numbers **p** and **q** are sensitive values that must be kept secret because they are used to calculate the private and public exponents. In real-life applications, those values should form at least a modulus of 2048-bit size so that the security is at a reasonable level. (Wikipedia: RSA – Key generation 2013)

3.3 Digital signatures with RSA

One great advantage of RSA is that it can be used to sign messages. The computation is the same as with the encryption, but the private key is used instead. The result of this operation is the signature of the plaintext, which can be verified with the public key by performing the decryption operation on the signature value. (Wikipedia: RSA - Signing messages 2013)

3.4 Padding schemes

RSA by itself is a great algorithm, but it has a disadvantage; the outcome of the encryption is deterministic. It means that there is no randomness involved, so an attacker could attempt to encrypt some plaintext to check if the ciphertext equals another ciphertext of interest. Padding schemes were introduced to avoid this problem. They are used to add some random data into the message so that the encryption results in a reasonable amount of different ciphertexts, even if the input plaintext was the same for each operation. Currently, it is recommended to use the Optimal Asymmetric Encryption Padding (OAEP), since it is proved to be secure against these attacks. (Wikipedia: RSA – Padding schemes 2013)

4 AES

AES is a popular symmetric-key algorithm that is standardized by the U.S. National Institute of Standards and Technology (NIST). It is a block cipher, which means that the encryption and decryption are performed on fixed-size blocks. Common block sizes are 128, 192 and 256 bits. Certain defined transformation and substitution steps to perform the encryption or decryption are repeated multiple times, which are also called **rounds**. The number of rounds depends on the key size; 128-bit keys repeat 10, 192-bit keys repeat 12 and 256-bit keys repeat 14 rounds. AES operations are performed on a 4x4 matrix of bytes, also called the **state**. (Wikipedia: Advanced Encryption Standard 2013)

To set up the encryption or decryption, the secret key is expanded into round keys so that each round will have its own key (also known as **KeyExpansion**). For each state, the input plaintext or ciphertext is XORed with the round key (also known as **AddRoundKey**).

For each round, the following operations are done;

1. Each byte is replaced according to a certain entry in a lookup table (**SubBytes**)
2. Bytes in the second row of the state are shifted one to the left, in the third row two to the left and in the fourth row three to the left (**ShiftRows**)
3. Each column of bytes in the state is multiplied with a certain fixed polynomial (**MixColumns**)

The last round does not include the **MixColumns** step. (Wikipedia: Advanced Encryption Standard – High level description of the algorithm 2013)

5 CRYPTOGRAPHIC HASH FUNCTIONS

A cryptographic hash function is a one-way function that takes data of arbitrary length as input and produces a fixed-size result (usually of 128 to 512 bits length). Typically they are used together with RSA signatures or as check values. This provides for data integrity, since if an adversary attempts to modify the data which the hash result was calculated on, the following hash calculation will result in a different value. Commonly used cryptographic hash functions are SHA-256 (256-bit result) and SHA-1 (160-bit result). (Wikipedia: Cryptographic hash function 2013)

As the results are fixed-size, there are a finite number of results that a cryptographic hash function can provide. Consequently, with an infinite amount of possible input values to the function, certain inputs will produce the same result. These are called collisions, which are required not to be found. Depending on the security level of an application, a function with a larger output size should be considered (at the cost of performance). (Wikipedia: Collision 2013)

6 PUBLIC KEY INFRASTRUCTURE

6.1 Introduction

In terms of key exchanges between communicating parties, public-key cryptography simplifies the key management, but one party still has to find the receivers' public key. If the public keys are shared in an open network, there would be no way to tell that a key belongs to a certain party. The solution for this issue is to use a PKI (Public Key Infrastructure). The solution is to have an authority called certificate authority (CA). Each party takes their public key to the CA and identifies themselves. The CA then signs the party's public key using a digital signature scheme. As result, the party will acquire a certificate. This certificate states that the public key belongs to a certain party. With the CA's public key, it can be then verified that a public key belongs to a certain party. (Wikipedia: Public key infrastructure 2013)

PKI still has some drawbacks; the CA has to be trusted by every party. In certain situations, that is easy. But if the CAs' private key is compromised by an adversary, he or she could issue false certificates and cause major harm to a business. Therefore the implementation of the PKI has to be robust and the CA key has to be managed securely.

6.2 Certificates

Certificates comprise the party’s public key and the signature calculated by the CA. Additionally, some general contents they include are a validity period, name of the certificate issuer and name of the key owner that is also known as subject field. A validity period is used since a same public key should not be used forever. There is always the risk that a key is compromised. A new certificate is to be issued when the current certificate expires. Certificate issuer name is a text field describing the certificate issuer. This field does not guarantee that the issuer is actually the CA – it is the digital signature that is used to verify the identity. There also is a field called Signature Algorithm, which defines the type of algorithm used to calculate the certificate signature. The subject field includes the name of the certificate owner. Although most certificate formats include an identifier field such as a serial number, this field should also be unique within a PKI. The following text is a simple example of a certificate. (Wikipedia: Public key certificate 2013)

```
Certificate:
    Data:
        Version: 1 (0x0)
        Serial Number:
            cf:5b:b2:3a:01:8c:e9:d9
        Signature Algorithm: sha1WithRSAEncryption
        Issuer: C=FI, ST=Satakunta, L=Rauma, O=Test Company Ltd, OU=R&D,
                CN=Jussi
        Validity
            Not Before: Jul 10 20:37:42 2013 GMT
            Not After : Jul 10 20:37:42 2014 GMT
        Subject: C=FI, ST=Satakunta, L=Rauma, O=Test Company Ltd, OU=R&D,
                 CN=Jussi
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
            RSA Public Key: (2048 bit)
                Modulus (2048 bit):
                    00:b5:3b:3c:3b:3c:27:29:7d:42:11:f0:ba:de:07:
                    ... [truncated remaining bytes]
                Exponent: 65537 (0x10001)
    Signature Algorithm: sha1WithRSAEncryption
        a1:71:f4:d5:87:d2:c7:ef:d3:b4:1b:81:08:b1:08:c1:90:4f:
        ... [truncated remaining bytes]
```

6.3 Registration Authority

There are various ways of enrolling certificates to parties. One way is to generate the private key/certificate pair at the CA and securely transmit it to the intended receiver. For instance, temporary RSA keys may be generated and used to encrypt the data before sending. Another, more common way is to have a registration authority (RA) which is an authority that receives certificate requests from parties with the intent to be part of the PKI. A certificate request contains the public key of the requesting party, a digital signature and a subject field which is intended to be in the enrolled certificate. Depending on the certificate enrollment criteria, the request may be approved if the RA decides so. If the RA accepts the request, it tells the CA to issue the certificate. (Wikipedia: Public key infrastructure 2013)

7 HARDWARE SECURITY MODULES

A hardware security module (HSM) is a physical device that is used to manage and protect cryptographic keys and accelerate cryptographic processes such as encryption, decryption, signing and calculating hashes. They are also used as random number generators for key generation purposes. Generally, an HSM comes in the form of a plug-in card (PCI/PCI-Express) or as an external USB or TCP/IP device. HSMs provide both logical and physical protection of sensitive data from adversaries. In other words, they protect cryptographic keys. Most notable applications using HSMs are CA-based PKI and card payment systems.

From a security perspective, the most valuable attributes of an HSM are the secure cryptographic operations, key protection and random number generation. All of this should occur “inside the device”, which means that no plaintext private or secret key is allowed to be transferred or stored outside the device. Keys stored in the HSM are accessed with “handles”, which can be stored on an operating system in encrypted format or on a security token, such as smart card. An HSM should be also tamper-proof; it should reset itself (erase all sensitive data) if it detects physical tampering, irregular electrical activity or irregular temperature. (Wikipedia: Hardware security module 2013)

8 PKCS#11

PKCS #11 is one of the group of standards called Public Key Cryptography Standards (PKCS) produced by RSA Laboratories. It specifies an application programming interface (API), also known as “Cryptoki”, to devices such as HSMs and smart cards which hold cryptographic information and perform cryptographic operations. Cryptoki defines the most common object types used in cryptography, such as asymmetric (RSA) and symmetric (AES) keys and certificates. Generation, modification and deletion of these objects are fully supported by devices that follow the standard. Cryptoki also defines digital signature calculation and commonly used hash functions. The main benefit of Cryptoki is that it enables for application portability. Applications developed with the API do not have to change to interface different types of devices. (Wikipedia: PKCS#11 2013)

For application development, RSA Laboratories provides C header files that define the API. The cryptographic device that supports PKCS #11 API comes with a library which holds the implementation of the functionality defined in the API. Although the header files are defined in C programming language, wrappers have been developed for many other languages. Java includes a native PKCS #11 implementation, so it can be used directly with a library provided with a cryptographic device. (RSA Laboratories - PKCS #11 2013)

9 APPLIED PKI APPLICATION

9.1 Context

The Applied PKI Application is an application that implements a CA-based PKI. Its main components are the CA itself and PKI objects, which can be any devices that can store identity data on a PKCS#11 device issued by the CA. In the context of this application, the PKCS#11 device is the HSM. The main idea is to enable secure communication between the devices and allow the CA to manage them. In addition, the CA system is used to keep track of the devices and to revoke their identities (e.g. in case private key is compromised).

A server application is included in the CA component, which handles the communication between the PKI objects. In practice, the communication between the PKI objects is about sending and receiving files. Files are encrypted and signed with the PKI object's private key as they are sent to the CA server. Upon request, the CA server informs the receiving PKI object about files to be received. The following sequence diagram (Figure 2) describes the communication process between two PKI objects.

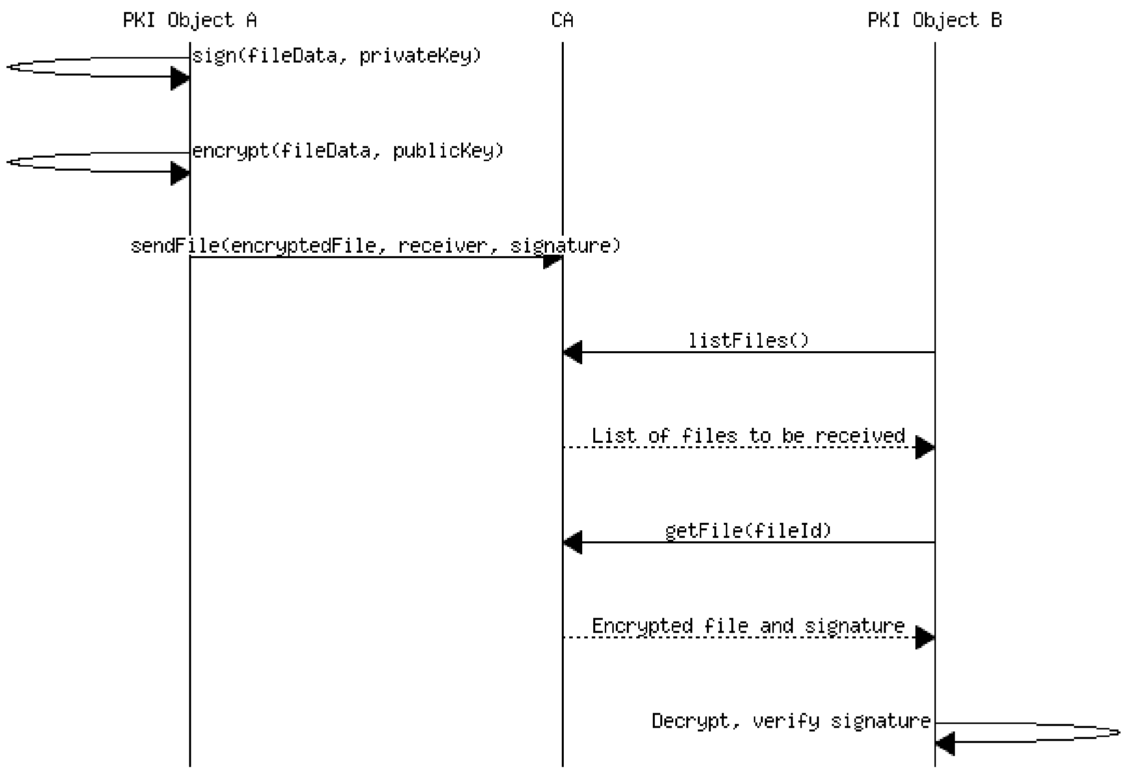
PKI Object A
CA
PKI Object B
sign(fileData, privateKey)
encrypt(fileData, publicKey)
sendFile(encryptedFile, receiver, signature)
listFiles()
List of files to be received
getFile(fileId)
Encrypted file and signature
Decrypt, verify signature

Figure 2. The communication process between two PKI Objects.

A network device may request an identity from the CA by sending a certificate request to the CA. If the CA approves the request, the network device becomes a PKI object. The CA creates a certificate based on the request that the network device originally sent, signs it with the CA private key and sends it to the PKI object. A randomly generated checksum value (by the CA) is used in order to reference the certificate request. CA sends this checksum value to the network device when it has requested the certificate. The certificate requester uses this reference to check if the request has been approved. (Figure 3)

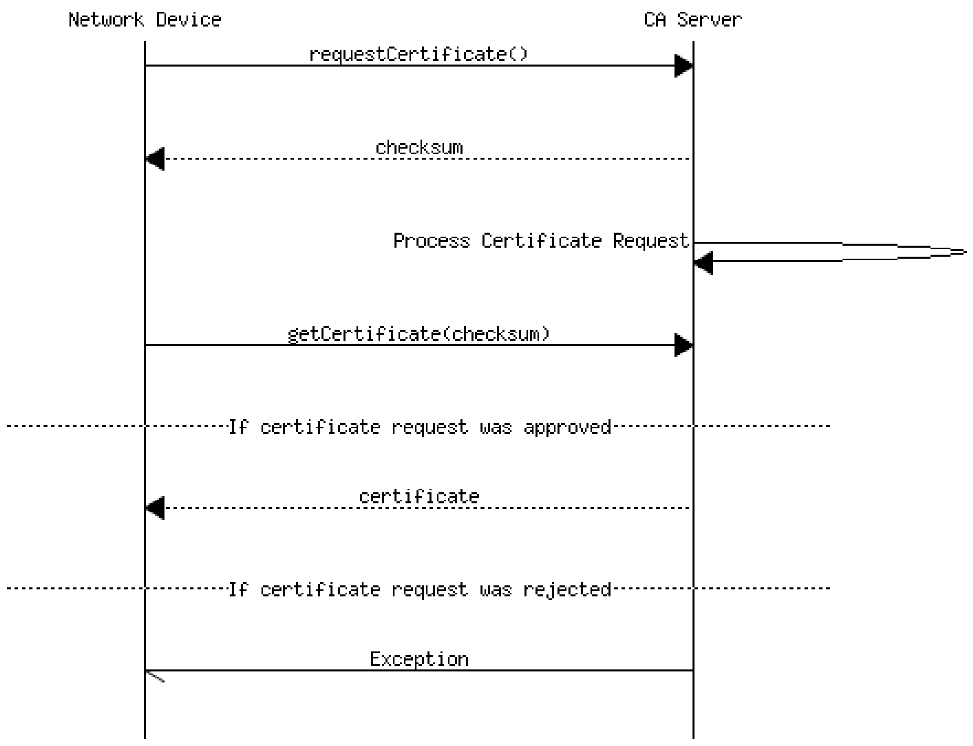
Network Device
CA Server
requestCertificate()
checksum
Process Certificate Request
getCertificate(checksum)
If certificate request was approved
certificate
If certificate request was rejected
Exception

Figure 3. A network device requesting an identity from the CA.

9.2 Architecture

The Applied PKI Application is designed to be as portable as possible, so that it may be used in different environments. The following diagram (Figure 4) represents the application deployed for a server machine and two PKI objects;

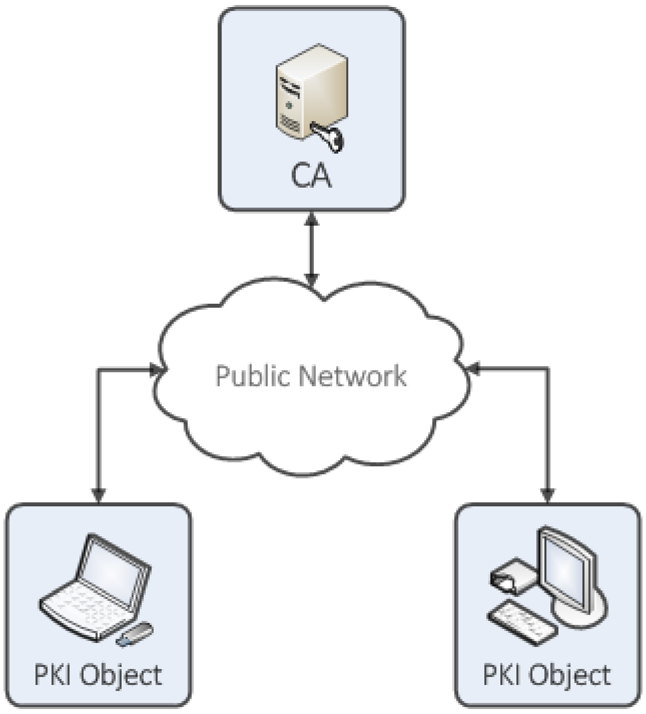

Figure 4. Applied PKI Application deployed on a server machine and two network devices.

In this diagram (Figure 4), the CA component contains the server which is used as a gateway between the PKI objects' communication. It is designed so that one server handles all the communication – meaning, that all the files are stored in the CA server machine, in encrypted format. PKI objects look up these files and may either download or discard them. Therefore, the larger network this application is deployed on, the more powerful the server has to be. There should be also a reasonable amount of storage capacity, depending on the kind of data to be transmitted. Concurrent communication between multiple PKI objects may consume a lot of capacity and require a lot of system resources. Hence it is better to invest in the server machine hardware, should the application be deployed for a large network.

The following diagram (Figure 5) represents the CA component. The database contains data for the certificate requests, PKI objects and the files transmitted between the PKI objects. The server runs the CA server application to which the PKI objects connect to. Also, the actual certificate request is stored in the file system of the server, so the database only contains a reference to the corresponding request file. Same principle applies for the files that are sent between the PKI objects. The reasoning behind this design is to have less load on the database. If all the files were stored on the database as blobs, the network traffic between the database would increase heavily, resulting in longer response time for each query.

The operator workstation is used to manage the CA. It contains an application which may view and respond to certificate requests, revoke PKI objects, look up / search PKI objects, stop the server or tear down the system. These operations require a connection to both database and server components.

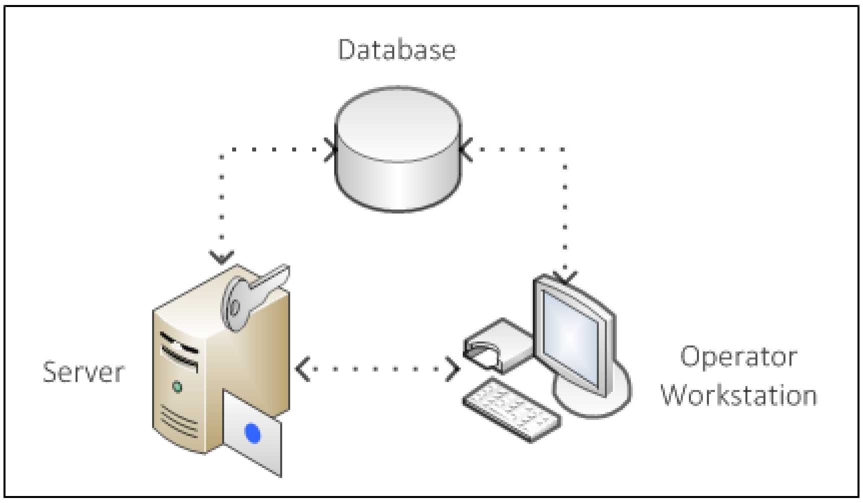

Figure 5. The CA component including an operator workstation, server and database.

9.3 Security

To achieve security to a reasonable extent, some of the most recent cryptographic mechanisms and system design have to be taken into account when developing the application. This chapter describes the used cryptographic mechanisms, application design and key management for both CA and PKI object.

9.3.1 General

For sake of confidentiality and data integrity, the RSA algorithm is used with a key size of 2048 bits to perform the cryptographic operations on the data transmitted within the application. RSA is implemented in many different programming languages, libraries and applications; hence it was easy to choose this algorithm. OAEP is used as padding scheme, since it is considered secure and supported by PKCS #11 devices as well.

To improve the performance on the cryptographic operations, the AES algorithm is used to encrypt the files that are sent between PKI objects, and the AES key is encrypted with the receiving PKI object's RSA public key (Figure 6). The AES key is randomly generated by the PKCS#11 device of the sender. The encrypted AES key and file are then sent to the CA server, in which they are stored. The PKI object receiving this file decrypts the AES key with his/her private key, uses the AES key to decrypt the file and verifies the signature with the sender's public key. If the signature verification fails, the object automatically refuses to receive this file.

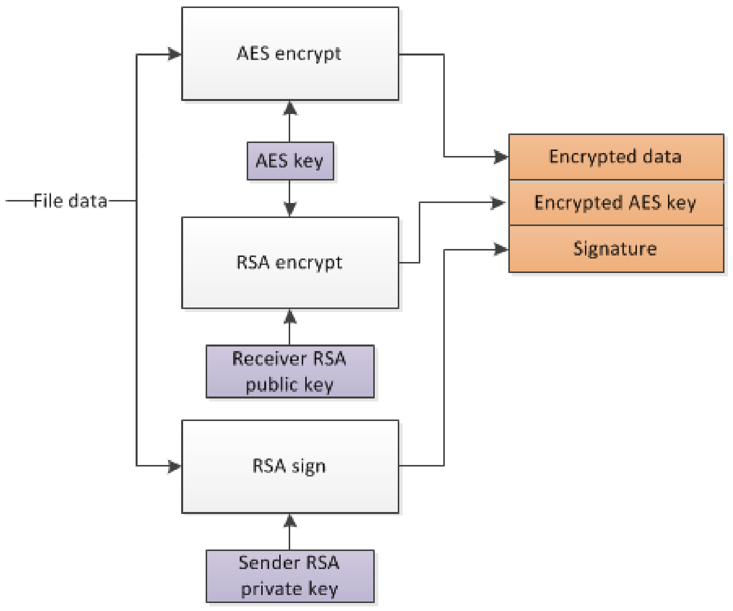

Figure 6. Preparing a file to be sent for another PKI object.

Digital signatures are calculated so that the SHA-1 hash function is applied on the given data, and its output is signed with the private key. This signing operation is also known as "SHA-1 with RSA". This is done for several reasons; the signature will be shorter because the SHA-1 function returns a 160-bit result – it is more efficient to calculate the signature on a shorter message than possibly on a longer one. The result of a hash function is also safer to use as input for the RSA signing operation – signing just the plaintext is vulnerable to attacks that may result in some message/signature pairs which an adversary could understand. (Wikipedia: Digital signature – how they work 2013)

Private and secret keys are managed by means of PKCS#11 devices. Both CA and PKI objects have PKCS#11 devices in which their private or secret key is stored. This key is accessed by providing a PIN for the application. The private key values cannot be extracted from the PKCS#11 devices – instead, the key is accessed by a handle so that they can be used with the cryptographic operations. In general, PKI objects and CA are allowed to use any PKCS #11 device, as long as they support the algorithms mentioned in this specification.

9.3.2 CA Secure System

The CA Secure System is designed so that the machine running the CA application is placed in a reasonably secure location. Therefore only trusted people with privileges can physically access the machine. This frees some time and cost (in terms of implementation) since data tampering and eavesdropping is more difficult to avoid, if the machine was placed in a more public place. After all, the main point is that the communication in the public channel is secured.

When the CA application is set up, an RSA key pair is generated with the PKCS #11 device. The public half of the key pair is stored into a public location so that the PKI objects and network devices can verify the CA signature, and the private half of the key is kept inside the PKCS #11 device. This private key is accessed only when new certificates are enrolled, as a new certificate is signed.

Additionally, the CA database data is encrypted by means of the AES algorithm. This is directly supported by the third party database application - it just has to be enabled in the database creation. Although the database is in a secure location, it is better to encrypt the values in order to provide for extra security.

9.3.3 PKI objects

PKI objects can be any network devices that support PKCS #11 devices. Within the context of this application, PKI objects are the communicating parties and therefore they require an identity. Without an identity, a network device is not considered as a PKI object and therefore it cannot communicate with the enrolled PKI objects in the Applied PKI Application network.

The PKI objects encrypt/decrypt, sign and verify this data that is being transmitted. The PKI objects are not allowed to send any data that is in the clear or not signed. As an exception, the certificate request is sent in clear so that the CA can respond with a generated certificate.

9.4 Use cases and procedures

This chapter describes the use cases for the application. This list can be definitely extended, should the application be developed further.

CA –specific use cases

- Initialization
    1. Initialize PKCS#11 device
    2. Create an RSA key pair (on the PKCS#11 device)
    3. Create database tables
    4. Start the server application
- Initialized/running
    1. List certificate requests
    2. Respond to certificate requests
    3. Lookup/search for PKI objects
    4. Revoke PKI object’s identities
    5. Tear down the system

PKI object –specific use cases

- Initialization
  1. Initialize PKCS#11 device
  2. Create an RSA key pair (on the PKCS#11 device)
  3. Create a certificate request
  4. Send the certificate request to the CA
  5. Obtain the certificate from the CA, if the request is approved
- Initialized
  1. Send files to another PKI object
  2. List files to be received
  3. Download a file
  4. Discard a file
  5. Reset the device

9.5 Implementation

The Applied PKI Application is developed in Java on Linux platform. It consists of three software components; AppliedPkiCA, AppliedPkiObject and AppliedPkiCommon. The class diagrams of the application can be found in Appendix 1 and the user guide in Appendix 2.

AppliedPkiCA -component includes the relevant CA implementation, which comprises database access, certificate creation, file server and certificate revocation.

AppliedPkiObject includes the relevant PKI object implementation, which includes the sending of the certificate requests and listing, sending, receiving and discarding of the files that are transmitted with other PKI objects.

AppliedPkiCommon includes functionality shared by both AppliedPkiCA and AppliedPkiObject components. This includes:

- The PKCS#11 functionality (key store and generation)

- Interface class for the AppliedPkiCA –server, which the PKI object uses and CA implements
- Implementation for the files sent between PKI objects – CA needs this implementation as well in order to save the data into the file system
- Authentication class to authenticate the PKI object before performing any of its use cases
- General utility classes for file operations, reading password input, cryptography and handling of certificates/RSA keys

9.5.1 User interface

To provide for best portability, the application uses a command line interface for all the operations. The command line interface can be also used with UNIX shell scripts in order to reduce the amount of parameters, for instance.

9.5.2 Connectivity

The connectivity between the PKI objects and the CA is implemented with Java RMI (remote method invocation). A server class is used as an interface; CA implements the functions defined in this interface and the PKI objects call these functions. It was easy to decide to use RMI, since it takes care of concurrency (a thread is created per PKI object) and the only requirement is to provide the IP address of the CA when retrieving the server object from an “RMI” registry. The CA initializes the RMI registry and binds the server object implementation to it, enabling the PKI object to use this functionality.

9.5.3 Dependencies

In this chapter, the application dependencies and their purposes are explained. The application has been developed and tested on a Ubuntu 13.04 for desktops and tested on a Ubuntu Server 12.04 LTS.
The following table describes the dependencies for the application environment;

Name	Purpose
openCryptoki (Linux only)	PKCS#11 device simulator
NetBeans IDE	Java development environment
pkcs11-tool (OpenSC)	Tool to initialize the PKCS#11 device
OpenSSL	Converting certificate formats
Apache Derby	Relational database
BouncyCastle API	Certificate generation/parsing
Apache Commons CLI	Command line parsing classes
VirtualBox	Virtual machines

Table 1. Applied PKI Application environment dependencies.

Due to the development conditions, there was no real PKCS#11 device available, so the development was done with a simulator called openCryptoki. It implements all the functionality to be PKCS#11 standards compliant (Website of openCryptoki 2013), but as it is a software simulator, it should not be used in a real environment. The use of openCryptoki limited the testing to Linux platform, but the application should work in other platforms as well, as long as an appropriate PKCS#11 device is used.

NetBeans IDE was preferred over other IDE’s, since it has a built-in support for Apache Derby and an excellent GUI for managing the databases during the development. (Website of NetBeans 2013)

pkcs11-tool is a command line tool to manage PKCS#11 devices. The functionality that is utilized from this tool is the PKCS#11 device initialization. It is executed by means of a UNIX shell script. (Website of OpenSC 2013)

OpenSSL is an open source implementation for general-purpose cryptographic operations. The functionality that is utilized from this tool is converting of the certificate requests to a format that Java supports. The tool is used from the command line, so the conversion is done by executing a command line command in Java. (Website of OpenSSL 2013)
Apache Derby is a lightweight database implemented in Java. It is based on Java and SQL standards, supports database encryption and is easy to use and deploy. (Website of Apache Derby 2013)

BouncyCastle API extends the Java cryptography, implementing many useful features like certificate revocation, generation and many algorithms that are not supported by the cryptographic libraries of Java. The certificate generation is the most useful part utilized from the BouncyCastle API. (Website of BouncyCastle 2013)

Apache Commons CLI facilitates the command line parsing development, freeing the developer from a lot of low level coding. Command line options may be added and configured according to the purpose of the option; it may have an argument or might not be required, for instance. All the main classes of Applied PKI Application use Commons CLI for command line parsing. (Website of Apache Commons CLI 2013)

VirtualBox was used to create two virtual machines to act as PKI objects during the development process. (Website of VirtualBox 2013)

9.5.4 Recovery measures

If the CA private key or the database password is compromised, the solution is to reset the whole system and deploy the application again. Even if there were files to be sent to other PKI objects. The server is not supposed to be a permanent storage for the files and the sender of the file must have a local backup of this file, in order to send the file again to the intended receiver. Once the system has been re-established, new certificates must be requested by the PKI objects.

10 CONCLUSION

This thesis contains two parts; first, an introduction to cryptography, security and public key infrastructure and secondly, an application to implement a CA-based PKI with a client application. Every entity within the application protects their own identity with a PKCS#11 device. The goal was to enable secure communication, which is now done by means of encrypting and sending files.

After testing the application with different combinations of inputs, it feels very usable utility. The passphrase writing seems definitely tedious, as it is done for almost every operation, but in the end, a cached passphrase would be a security risk.

In the beginning I had a much larger scope for the application – I wanted to stream media, use more properties for the certificates and implement a graphical user interface for both CA and PKI object. When I started to design the classes and look at the implementation from more detailed point of view, I realized I have to leave something out.

So how is the application different from the others? I could indeed find many tools on the internet which deal with similar scope, but everything is CA focused. CA has the PKCS#11 device, database, revocation list etc. But Applied PKI Application includes the clients (or PKI objects) also. The PKI objects are strongly connected to the CA, and are the only entities communicating with the CA and other clients. The key management is implemented for the PKI objects too, which is forced so that the private key has to be stored inside the PKCS#11 device. The PKI object identity security is at a reasonable level.

Unfortunately, I never got to test the application with a real PKCS#11 device. As sometimes the real PKCS#11 devices include some additional setup steps, it could have been good to have one example in this project. But there are so many of them that choosing one would have been difficult.

In future, I plan to extend and maintain the application, as new cryptographic mechanisms are introduced and more ideas of functionality come into my mind. It has been definitely a great learning experience in terms of designing, developing, documenting and gathering information. My English skills in terms of writing have also improved very well.

REFERENCES

Wikipedia: Cryptography 2013. Referred 13.3.2013.
http://en.wikipedia.org/wiki/Cryptography

Wikipedia: Message authentication code 2013. Referred 13.3.2013.
https://en.wikipedia.org/wiki/Message_authentication_code

Wikipedia: Asymmetric key cryptography 2013. Referred 13.3.2013.
http://en.wikipedia.org/wiki/Asymmetric_key_cryptography

Wikipedia: RSA algorithm 2013. Referred 15.3.2013.
http://en.wikipedia.org/wiki/RSA_(algorithm)

Wikipedia: RSA – Key generation 2013. Referred 8.12.2013.
http://en.wikipedia.org/wiki/RSA_(algorithm)#Key_generation

Wikipedia: RSA - Signing messages 2013. Referred 20.3.2013.
http://en.wikipedia.org/wiki/RSA_(algorithm)#Signing_messages

Wikipedia: RSA – Padding schemes 2013. Referred 22.3.2013.
http://en.wikipedia.org/wiki/RSA_(algorithm)#Padding_schemes

Wikipedia: Advanced Encryption Standard 2013. Referred 22.3.2013.
http://en.wikipedia.org/wiki/Advanced_Encryption_Standard

Wikipedia: Advanced Encryption Standard – High-level description of the algorithm 2013. Referred 25.3.2013.
http://en.wikipedia.org/wiki/Advanced_Encryption_Standard#High-level_description_of_the_algorithm

Wikipedia: Cryptographic hash function 2013. Referred 31.3.2013.
http://en.wikipedia.org/wiki/Cryptographic_hash_function

Wikipedia: Collision 2013. Referred 4.5.2013.
http://en.wikipedia.org/wiki/Hash_collision

Wikipedia: Public key infrastructure 2013. Referred 2.4.2013
http://en.wikipedia.org/wiki/Public_key_infrastructure

Wikipedia: Public key certificate 2013. Referred 2.4.2013
http://en.wikipedia.org/wiki/Digital_certificate

Wikipedia: Public key infrastructure 2013. Referred 2.4.2013
http://en.wikipedia.org/wiki/Public-key_infrastructure

Wikipedia: Hardware security module 2013. Referred 6.4.2013
http://en.wikipedia.org/wiki/Hardware_security_module

Wikipedia: PKCS#11 2013. Referred 7.4.2013.
http://en.wikipedia.org/wiki/PKCS11

RSA Laboratories - PKCS #11 2013. Referred 14.4.2013.
http://www.rsa.com/rsalabs/node.asp?id=2133

Wikipedia: Digital signature – how they work 2013. Referred 14.4.2013.
http://en.wikipedia.org/wiki/Digital_signature#How_they_work

Website of openCryptoki 2013. Referred 15.6.2013.
http://sourceforge.net/projects/opencryptoki/

Website of NetBeans 2013. Referred 15.6.2013.
https://netbeans.org/features/index.html

Website of OpenSC 2013. Referred 15.6.2013.
https://www.opensc-project.org/opensc/wiki/Tools

Website of OpenSSL 2013. Referred 15.6.2013.
http://www.openssl.org/

Website of Apache Derby 2013. Referred 15.6.2013.
http://db.apache.org/derby/

Website of BouncyCastle 2013. Referred 15.6.2013
http://www.bouncycastle.org/

Website of Apache Commons CLI 2013. Referred 15.6.2013.
http://commons.apache.org/proper/commons-cli/

Website of VirtualBox 2013. Referred 15.6.2013.
https://www.virtualbox.org/

CA Server classes

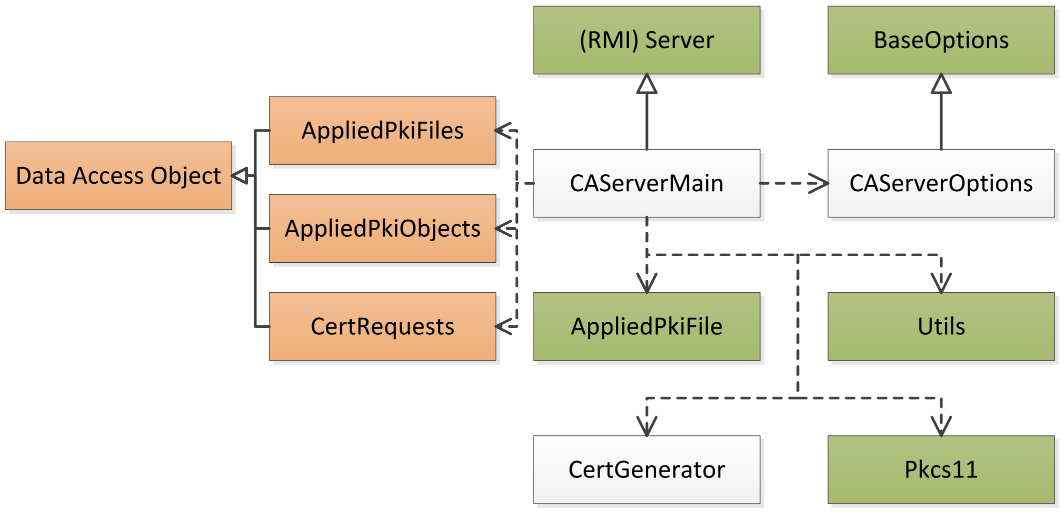

Figure 7. CA Server application class structure

The above diagram represents the CA Server class hierarchy. The CAServerMain functions as the main class.

CAAdmin classes

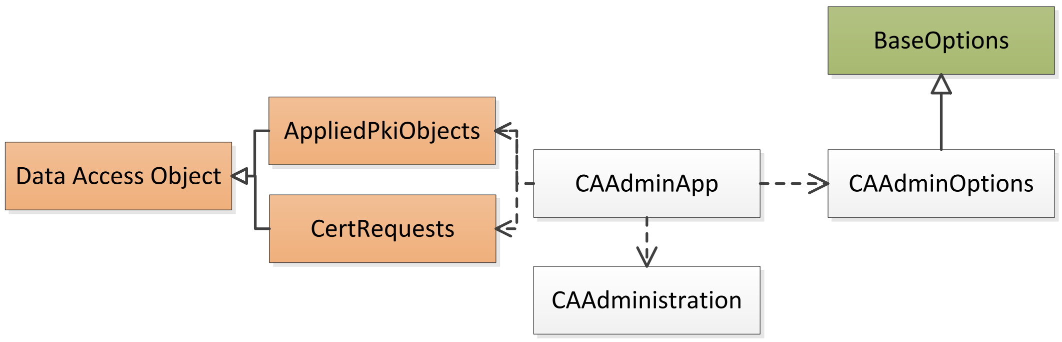

Figure 8. CAAdmin application class structure.

The above diagram represents the CA Administration class hierarchy. The CAAdminApp functions as the main class.

CAInit classes

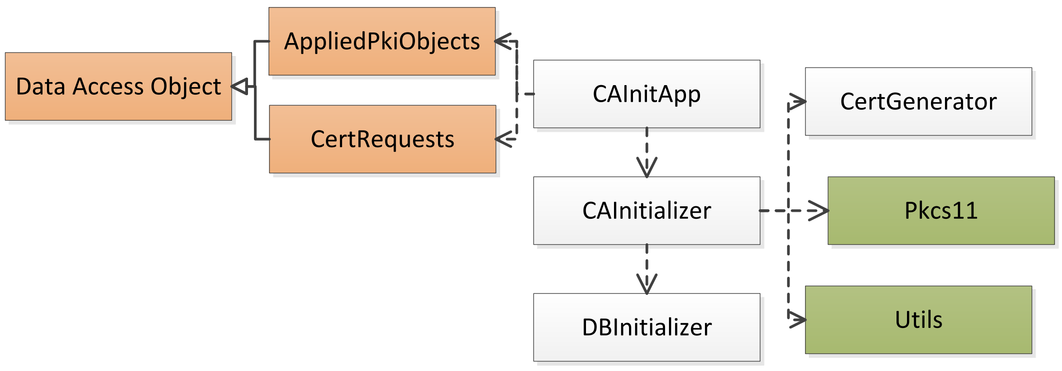

Figure 9. CAInit application class structure.

The above diagram represents the CA Initialization application class hierarchy. The CAInitApp functions as the main class.

PkiObjectInit classes

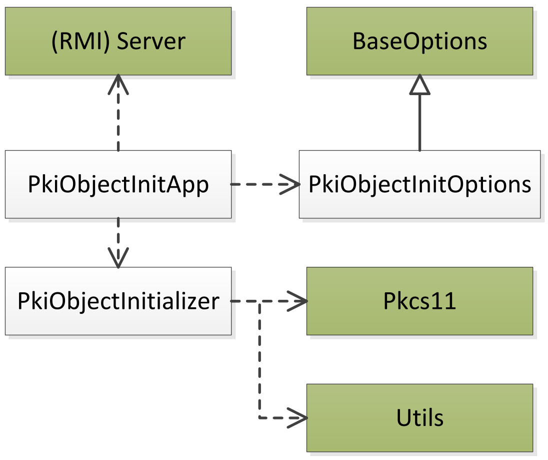

Figure 10. PkiObjectInit application class structure.

The above diagram represents the Pki Object Initialization application class hierarchy. The PkiObjectInitApp functions as the main class.

PkiObjectClient classes

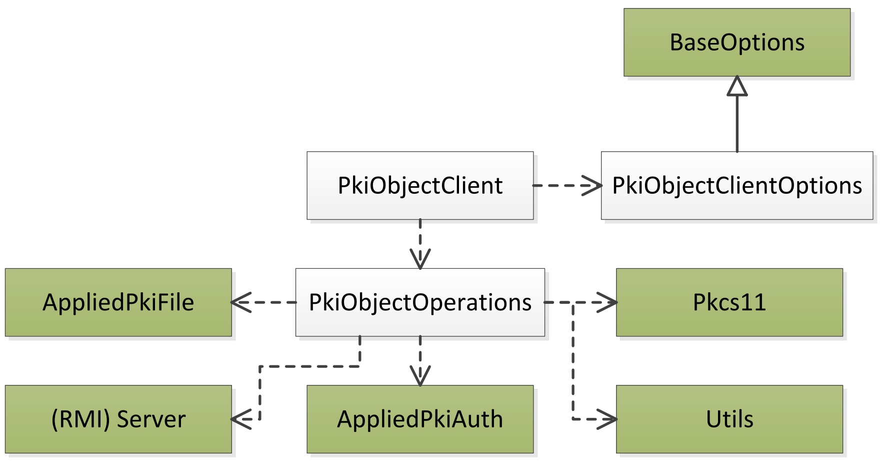

Figure 11. PkiObjectClient application class structure.

The above diagram represents the Pki Object Client application class hierarchy. The PkiObjectClient functions as the main class.

APPENDIX 2

User Guide

APPLIED PKI APPLICATION

Jussi Salminen

July, 2013

CONTENTS

1 INTRODUCTION

This user guide describes the usage of the Applied PKI Application, which is documented in the thesis APPLYING PKI TO DEVICES IN A NETWORK.

2 APPLICATION PACKAGE CONTENTS

This chapter lists the files and folders provided with the Applied PKI Application.

2.1 AppliedPkiCA

- **dist** (directory) – Contains the Java classes for this application, should be only accessed with the launch scripts (see scripts folder).
- **etc** (directory) – Contains example configuration files
    - **cacert.properties** – Example CA certificate configuration file
    - **database.properties** – Example CA database configuration file
    - **pkcs11.cfg** – Example PKCS#11 configuration file
    - **server.policy** – Policy settings file for RMI, must not be edited. Must be kept in this directory
- **scripts** (directory) – Contains the application launch scripts
    - **admin.sh** – Used for administrative operations like revoking certificates or issuing new certificates. The CA must be set up in order to use this script
    - **init.sh** – Initializes the PKCS#11 device for the and the CA server
    - **opencryptoki-reset.sh** – Reset/initialize the PKCS#11 device PIN codes
    - **run-server.sh** – Starts the CA server process

2.2 AppliedPkiObject

- **dist** (directory) – Contains the Java classes for this application, should be only accessed with the launch scripts (see scripts folder).
- **etc** (directory) – Contains example configuration files
    - **pkcs11.cfg** – Example PKCS#11 configuration file
    - **req.properties** – Example certificate request file
- **scripts** (directory) – Contains the application launch scripts
    - **init.sh** – Initializes the PKCS#11 device, prepares the certificate request and sends it to the CA
    - **object.sh** – Used for the PKI object operations like sending/receiving/discarding files. Can be used when the PKI object has a certificate issued by the CA
    - **opencryptoki-reset.sh** – Reset/initialize the PKCS#11 device PIN codes

3 PKCS#11 DEVICE PIN INITIALIZATION

Common to both CA and PKI object is the PKCS#11 device PIN initialization. It is the first step that is done during the application setup. The script `opencryptoki-reset.sh` is used to perform this operation.

The following command starts the PKCS#11 device PIN initialization process:

```
/opencryptoki-reset.sh
```

Expected output:

```
Using slot 0 with a present token (0x0)
Please enter the new SO PIN:
Please enter the new SO PIN (again):
Token successfully initialized
Using slot 0 with a present token (0x0)
Logging in to "test".
Please enter SO PIN:
Please enter the new PIN:
Please enter the new PIN again:
User PIN successfully initialized
```

As seen above, the script executes pkcs11-tool and prompts for the SO (also known as Security Officer) code and PIN code a few times. These codes enable the usage of the secret data (keys) in the PKCS#11 device. This operation erases all data on the PKCS#11 device which makes it ready to use with this application. The PINs must be kept secret by the user, since it will be prompted every time when performing an operation (e.g. issue certificate, list files, get file, send file) within the Applied PKI Application.

4 SETUP

4.1 CA

4.1.1 Server Initialization

First step is the initialization of the server, a script called `init.sh` is provided to facilitate this process. This script starts the Java RMI registry, initializes the database and executes a Java application that creates the CA RSA key pair and certificate.

Command line parameters:

1. **pkcs11-config-file** – (Required) A PKCS#11 configuration file, specifying the device provider PKCS#11 library file, name and slot to use. An example is included for openCryptoki in the **etc** folder
2. **ca-cert-properties** – (Required) Configuration for the CA certificate, the Subject field of the certificate, validity period and signature algorithm are specified in this file. An example configuration is included in the **etc** folder

The parameters are positional, so the **-pkcs11-config-file** is the first parameter and **-ca-cert-properties** is the second.

Example command:

```
./init.sh ../etc/pkcs11.cfg ../etc/cacert.properties
```

Expected output:

```
Applied PKI CA Initializer
==========================
Generating RSA key...
Generating CA certificate...
Enter CA passphrase:
Storing keys...
Creating database...

Applied PKI CA initialized - ready to run server
================================================
```

As seen in the output, the server application is now ready to be executed. The server is run with a script called run-server.sh, which is provided with the CA source code.

4.1.2 Start the Server

When the server is initialized as described in the previous chapter, it can be started with the script `run-server.sh`. This will start the server process which runs until it is terminated.

Command line parameters:

- **-reqfolder** <cert-req-folder> - (Required) Location in the file system in which the certificate requests are stored
- **-repo** <file-repository-folder> - (Required) Location in the file system in which all the files transmitted between the PKI objects are stored
- **-pkcs11** <pkcs11-config-file> - (Required) PKCS#11 configuration file, the same as specified for the `init.sh` script
- **-ipadd** <host-ip-address> - (Required) IP address of the host machine. There may be multiple network interfaces in the host and the server needs to know which one to use
- **-dbconfig** <database-config> - (Required) Database configuration file including DB address and username. An example file is included in the **etc** folder

Example command:

```
./run-server.sh -reqfolder ../reqfolder/ -repo ../repo/
-pkcs11   ../etc/pkcs11.cfg   -ipadd   192.168.56.1   -dbconfig
../etc/database.properties
```

Expected output:

```
Configuration file: ../etc/database.properties
Repository: ../repo/
Certificate request directory: ../reqfolder/
PKCS#11 config file: ../etc/pkcs11.cfg
IP Address to use: 192.168.56.1
Applied PKI CA Server initialized!
```

After this output is printed, the command line window should be reserved for this process. It can be also run in background. Tearing down the system is simply done by stopping the run-server.sh script manually. After this, the `init.sh` and `opencryptoki-reset.sh` scripts can be called again - this will reset the system.

4.2 PKI Object

The first step for initializing a PKI object is the same as for the CA – initialization of the PKCS#11 device PIN. This is done with the `opencryptoki-reset.sh` script. After this, the script `init.sh` is used to perform the rest of the initialization process.

The next step is to create an RSA key pair and certificate request and send it to the CA.

Command line parameters:

- **-cahost** <ip-address> - (Required) The CA server IP address to connect to
- **-pkcs11** <pkcs11-config-file> - (Required) The PKCS#11 configuration file
- **-reqsettings** <request-settings> - (Required) The settings of the certificate request. Includes only one field which is the requested subject for the certificate
- **-checksum** <checksum-value> - (Optional) Checksum of a previously sent certificate request. Used to query for certificate request status – if the certificate is signed by the CA, it is automatically downloaded and installed

Example command:

```
./init.sh -cahost 192.168.56.1 -pkcs11 ../etc/pkcs11.cfg
-reqsettings ../etc/req.properties
```

Expected output:

```
Enter PKI object passphrase:
Cert Request Checksum:
e6eb3462f141c0ba5d594caff5a3ef70ab4c6eb46dfcd7100a5f22cf191604f6
```

The passphrase (PKCS#11 device PIN) is prompted and the server returns the checksum that can be also seen in the certificate requests listing on the CA server side. This checksum is used to refer to the certificate request when querying the CA if the certificate has been signed. This checksum value is in the clear and does not need to be encrypted, since it returns a public certificate that is meant to be public for other PKI objects. A digital signature is calculated using the checksum to verify the identity of the certificate requester. This way, the CA can be sure that it is the intended receiver.

Next step is to query the CA with the checksum from the previous command for the certificate. If the certificate is signed, it will be stored into the PKCS#11 device.

Example command:

```
./init.sh -cahost 192.168.56.1 -pkcs11 ../etc/pkcs11.cfg
-reqsettings ../etc/req.properties -checksum
e6eb3462f141c0ba5d594caff5a3ef70ab4c6eb46dfcd7100a5f22cf191604f6
```

So the command remains the same as in the previous one, except that the **-checksum** parameter is added.

Expected output:

```
Enter PKI object passphrase:
Certificate is signed, preparing a signature of the check-
sum...
PKI object initialized!
```

5 OPERATIONS

5.1 CA

When the CA is set up as described in the previous chapter, the CA operations can be performed as described in the initialized/running use cases of the CA. This chapter describes those operations and how they are used in practice. The script `admin.sh` is used to execute the operations. As a common parameter, there is a database configuration file for each operation. This could be also hard-coded into the admin.sh, but it is described here for clarity.

5.1.1 List Certificate Requests

Listing the certificate requests is executed with the following command:

```
./admin.sh -dbconfig ../etc/database.properties –listreq
```

Where `-listreq` tells the application to list the certificate requests.

Expected output:

```
Applied PKI CA Administration Tool
==================================

Cert Requests (tab-delimited):

139666     e6eb3462f141c0ba5d594caff5a3ef70 REQ_139666.req        0
```

The first column in the result indicates the certificate request id, which can be used to approve or discard the request. The second column is a checksum of the request, which the certificate requester gets and refers to this request when checking if the certificate request has been approved. The third column is the certificate request file-name. It is stored in the folder that was specified with the **-reqfolder** parameter in the `run-server.sh` script. The last column tells if the request has been approved. 0 means that it has not been approved, 1 means that it is.

5.1.2 Issue a Certificate

Responding to a certificate request is done with the following command:

```
./admin.sh -dbconfig ../etc/database.properties -respondreq
139666
```

Where the `–respondreq` and its argument 139666 tell that the certificate request with this id is approved. It is up to the CA users to determine if the certificate should be approved – there are no criteria in the application that would affect the certificate enrollment.

Expected output:

```
Applied PKI CA Administration Tool
==================================

Cert request 139666 ready for signing.
```

From the output it is now obvious that this certificate will be now signed.

5.1.3 Discard a Certificate Request

Discarding a certificate request is done with the following command:

```
./admin.sh -dbconfig ../etc/database.properties -discardreq
139666
```

Where the `–discardreq` 139666 tells that the certificate request 139666 will be discarded. This command can be called even if the request was approved, but for any reason it is decided that this certificate request should not be approved after all.

Expected output:

```
Applied PKI CA Administration Tool
==================================

Cert request 139666 deleted.
```

This means that the whole certificate request has been deleted and the PKI object has to create a new request in order to have a pending request.

5.1.4 List PKI Objects

Listing of PKI objects is done with the following command:

```
./admin.sh -dbconfig ../etc/database.properties -listobj
```

Where the `–listobj` parameter indicates to list the PKI objects.

Expected output:

```
Applied PKI CA Administration Tool
==================================

PKI Objects:
TestCn   0
TestCn2  0
```

Where the first column is the CN (Common Name) field of the certificate subject, or PKI object name and second column tells if the PKI object has been revoked. 1 means that it is revoked and 0 the opposite.

5.1.5 Revoke PKI Object

Revoking of a PKI object is done with the following command:

```
./admin.sh -dbconfig ../etc/database.properties -revokeobj
TestCn
```

Where the `–revokeobj testCn` indicates that the PKI object with a name `testCn` should be revoked.

Expected output:

```
Applied PKI CA Administration Tool
==================================

Object TestCn revoked!
```

The PKI object will no longer be part of the Applied PKI Application network once it is revoked. Revoked objects cannot be brought back to the network – the network device has to request a new certificate from the CA in order to be part of the network again.

5.2 PKI Object

When the PKI object is initialized, it may perform the operations as described in the use cases. For these operations, the script `object.sh` is used.

Parameters `-pkcs11` and `-ipadd` are common for all the operations, so they could be also added in the `object.sh` script. The parameters are included in this guide for clarity.

5.2.1 Send File

This chapter describes the procedure to send files to another PKI object. As a requirement, the receiver CN (Common Name), or the PKI object name must be known prior to sending the file. It is specified with the `-receiver` parameter. The way to exchange the CN's between PKI objects is up to the users.

Command line parameters:

- **-pkcs11** <pkcs11-config-file> - (Required) The PKCS#11 configuration file
- **-ipadd** <ip-address> - (Required) The IP address of the CA server
- **-sendfile** <file-path> - (Required) Path of the file to be sent
- **-receiver** <receiver> - (Required) Name of the recipient PKI object

Example command:

```
./object.sh -pkcs11 ../etc/pkcs11.cfg -ipadd 192.168.56.1
-sendfile testFile.txt -receiver PkiObj1
```

If the file sending is successful, the command does not produce any output.

5.2.2 List Files

Files to be received can be listed with the following command:

```
./object.sh -list -pkcs11 ../etc/pkcs11.cfg -ipadd 192.168.56.1
```

The `-list` tells the application to query for file to be received.

Expected output:

```
List files (format: id <tab> filename <tab> sender)
565973    testFile.txt        VirtualMachineTester
```

5.2.3 Download/Get File

When files to be received are downloaded, they are decrypted by the receiving PKI object's private key and the signature is verified with the sender's public key.

Command line parameters:

- **-pkcs11** <pkcs11-config-file> - (Required) The PKCS#11 configuration file
- **-ipadd** <ip-address> - (Required) The IP address of the CA server
- **-get** <file-id> - (Required) ID of the file to be downloaded (seen in the file listing)
- **-getfilepath** <file-path> - (Required) Path in which the file is stored in the local machine

Example command:

```
./object.sh -get 565973 -getfilepath /home/jussi/testFile.txt
-pkcs11 ../etc/pkcs11.cfg -ipadd 192.168.56.1
```

The `–get` parameter takes the file id as input. This id can be seen in the file list as shown in the previous command. The `–getfilepath` is the file path in which the file is stored in the local machine. This command produces no output if it executes successfully.

5.2.4 Discard File

Discarding a file is done with the following command:

```
./object.sh -discard 565973 -pkcs11 ../etc/pkcs11.cfg -ipadd 192.168.56.1
```

Where `–discard` 565973 means that the file with id 565973 is discarded and thus does not show up in the file listing anymore.