Faculdade de Engenharia da Universidade do Porto

Digital Receipts for Local Transactions in Commercial Spaces

Inês Sofia Ferraz Moreira de Sá

Tese submetida no Âmbito do

Mestrado Integrado em Engenharia Electrotécnica e de Computadores

Major de Telecomunicações

Orientador: Ricardo Santos Morla PhD

Junho de 2008

Abstract

Web technology is propagating at a high speed to every corner of the society's services, and commerce is one of the services that suffered the biggest impact. E-commerce is a combination of traditional business with improvements from the Internet and information technologies. A particular kind of traditional business where e-commerce may be beneficial is store commerce, where there is a great improvement potential in ecologic, logistic and safety behaviors. An important aspect of the shopping experience is the receipt that is given to customers.

The problem that this thesis is trying to solve is the absence of a system that supports local digital receipt transactions. The major contribution of this thesis is the analysis and design of a client-store communication architecture which supports these transactions, and in particular that supports locally and securely sending the client's tax identification number to the store and getting back a digital receipt issued to the client that is digitally signed by the store, validated by the client, and that could be used in the client's tax statement.

To evaluate this architecture, a prototype of a local digital receipt transaction system that abides to the architecture was developed. This thesis describes how this prototype was built using available technologies and tools; digital certificate and secure sockets layer technologies were critical in this development. The prototype implementation validated basic requirements of the architecture that were identified beforehand. An important observation that arises from the experience of developing such a prototype is the lack of compatibility of different technologies on the client's mobile device and the store's fixed device.

This is an important starting point for studying similar systems behavior and their limitations and for future research into e.g. creating a cost model of the deployment of such a system.

This thesis also opens the door to debate several privacy issues concerning e.g. both parties' tax statement data crossing.

Acknowledgments

I wish to thank to my supervisor Ricardo Morla for his help, knowledge, suggestions, motivation and availability.

I would also like to thank to my family and close friends for the support and caring they have given me in the difficult moments. Specially to Célia, for her love, friendship and wise advices, and to Telmo for the support and friendship. To my parents, for being a constant challenge in my life, always making me trying to be a better person.

Last but not least, I would like to express my deeply gratitude to my best friend and my soul mate, Hernâni, whose unconditional love, friendship, support, sacrifice and belief in me allowed me to be where I am today. This thesis could not have been done without him in my life.

Inês Sá

"The core of mans' spirit comes from new experiences."

Chris McCandless

Acronyms

APDU Application Protocol Data Unit

API Application Programming Interface

B2B Business to Business

B2C Business to Commerce

BNEP Bluetooth Network Encapsulation Protocol

BSEG Bluetooth Security Expert Group

CA Certificate Authority

CRL Certificate Revocation Lists

CRM Customer Relationship Management

DDH Decisional Diffie-Hellman

EEPROM Electrically-Erasable Programmable Read-Only Memory

GUI Graphical User Interface

HTTP HyperText Transfer Protocol

HTTPS HTTP over Secure Socket Layer

IDE Integrated Development Environment

IP Internet Protocol

IrDA Infrared Data Association

JSP JavaServer Pages

JSR Java Specification Request

J2ME Java 2 Micro Edition

LAN Local Area Network

LDAP Lightweight Directory Access Protocol

L2CAP Logical Link Control and Adaptation Protocol

WAP Wireless Application Protocol

MD5 Message-Digest algorithm 5

NFC Near Field Communication

OCSP Online Certificate Status Protocol

OSI Open Systems Interconnection

PAN Personal Area Network

PDA Personal Digital Assistant

PDF Portable Document File

PIN Personal Identification Number

PKI Public Key Infrastructure

QoS Quality of Service

RFID Radio Frequency Identification

RMI Remote Method Invocation

RPC Remote Procedure Call

SAT SIM Application Toolkit

SDP Service Discovery Protocol

SIG Special Interest Group

SIM Subscriber Identity Module

SSL Secure Sockets Layer

SQL Structured Query Language

Contents

List of Figures

Chapter 1

Introduction

1.1 Digital Receipts for Local Transactions

E-commerce (electronic commerce) is a combination of traditional business with improvements from the Internet and information technologies. This combination allows companies to exchange sales information, do financial transactions, deliver goods and services and process invoices in an automatic way. E-commerce is leveraged by cryptographic techniques which provide safe access to world-wide buyers.

Major advantages of e-commerce to a company (B2C and B2B) include [1]:

- Reducing the values associated with consumer/ supplier and partner/partner transactions,
- Reducing product delivery time,
- Reducing the bureaucracy associated with a purchase,
- Obtaining information directly from a CRM (Customer Relationship Management)system: e.g. the customer's personal tastes and the market evolution.

A particular kind of traditional business where e-commerce may be beneficial is store commerce. Store commerce typically involves human interaction, waiting in line, exchanging money, and dealing with paper receipts and invoices. An e-commerce approach to store commerce may help address economic, ecologic, and user-satisfaction issues and potentially bring a profit margin. Studying how intelligent environments can help customers make the best purchasing decisions has become an important research area. One research issue in particular is to try and understand how mobile devices can help users to shop intelligently and intuitively.

An important aspect of the shopping experience is the receipt that is given to customers. The customer may find a paper receipt less than ideal because it can be easily lost, damaged, or destroyed. This could be a problem when the customer wishes to use paper receipts to organize his personal finances or to return purchased items. If the receipt is damaged to a point where it's no longer valid, the customer may not be able to properly document and manage his accounting.

1.2 Characterization of a Digital Receipts System

An improved system that keeps digital receipts of local commercial transactions would address the problems identified in the previous section. We assume that this system is divided into different elements which interact with each other, as Figure 1.1 illustrates.

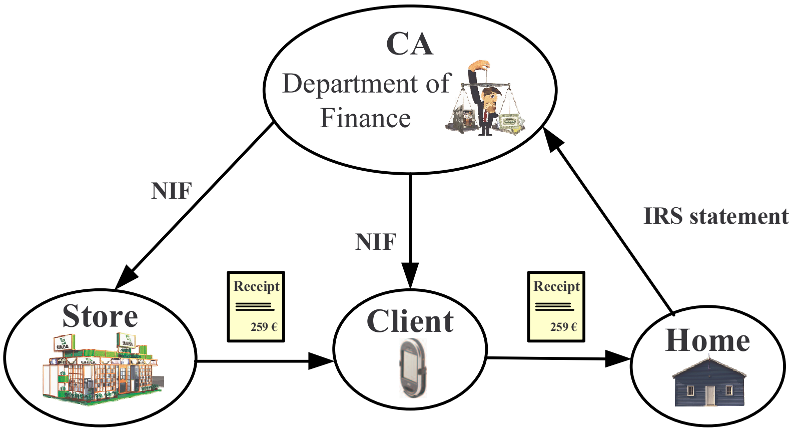

Figure 1.1: System elements' interaction

The elements involved are: the client, the store, the Department of Finance and the client's Home. The store issues a digital receipt to the client. The client must keep the receipt if he wants to use it in the tax statement that must be submitted annually to the Department of Finance. Only the client and the store participate directly in the digital receipt transaction. However, the Department of Finance plays an important role, since it is responsible for the administration of public finance, including the collection and expenditure of the revenues. In the case of digital receipts it also plays an important role in the authenticating the digital receipts.

A system such as this should have the following desirable properties:

- The system should be secure, the information should be sent with confidentiality and integrity,
- Application usability should be intuitive and require minimal human interaction,
- The communication between devices should be fast,
- Both client and seller should be able to:

  - delete receipts,
  - visualize receipts,
  - transfer/download receipts,
  - print receipts,

- The components of the system shouldn't be expensive,
- The receipt must contain all the information required for a valid receipt to be accepted by the client and taxes entities.

1.3 Problem and Goals

The problem that this thesis is trying to solve is the absence of a system such as the one described above to support local digital receipt transactions. Moreover we do not know how to design and develop such a system.

The goal of this thesis is to help understand how to design and build a part of that system - namely the store-client interaction. We focused on this part of the system because:

1. this is the most important part of the system, where receipts are actually exchanged;
2. addressing the whole system would be too big of a task for the limited resources available for this thesis.

This thesis focuses on the communication protocol that supports this interaction. Given that this is a system for local transactions, while focusing on the communications between client and store we will also consider proximity between the client (which is mobile) and the store (which is not mobile), and security issues during the exchange of the digital receipts.

1.4 Thesis Structure

Chapter 2 provides the background required to understand the digital receipt system, including basic concepts on cryptography, the public-key infrastructure on which the system is based, and related work on digital transactions. This chapter also describes the communications and software development technologies that were used in the implementation. *Chapter 3* presents the major contribution of this thesis, an architecture and communications protocol for a digital receipt system for local transactions. This is followed by the implementation of this architecture and protocol in *Chapter 4*, including a description of all the steps of the client and server applications that were implemented. *Chapter 5* provides an analysis of the developed system, focusing mostly on the security measures, and on some implementation alternatives. *Chapter 6* presents the concluding remarks.

Chapter 2

Background

2.1 Cryptography

The goal of an encryption scheme is to achieve confidentiality or secrecy. Which mean that the data sent from one part to another cannot be understood by an unauthorized party.

Cryptography is a security tool which converts a network problem into a key management problem [53]. It secures information by enciphering it in an unreadable format with various cryptographic algorithms, such as RSA.

There are two types of cryptography, symmetric-key and public-key. Symmetric-key cryptosystems use the same key for encryption and decryption of a message which has the main disadvantage of requiring key management to use them securely.

2.1.1 Public-Key Cryptography

Public Key Cryptography is a commonly used cryptographic method, which uses an asymmetric key pair (a public and a private key). The key used for encryption is different from the one used for decryption. Public key cryptography requires the owners to protect their private keys while their public keys are not secret at all and can be made available to the public.

The computation algorithm relating the public key and the private key is designed in such a way that an encrypted message can only be decrypted with the corresponding key of that key pair, and an encrypted message cannot be decrypted with the encryption key. Moreover, it is extremely difficult to deduce the private key if you know the public key.

Public key cryptography is one of the essential components of modern network security for the following reasons [53]:

- It scales better than symmetric cryptography, as each participant needs only one pair of keys to communicate with everyone else (as opposed to sharing a key with each participant).

- It partly solves the key distribution problem associated with symmetric cryptography.
- Public key cryptography, by associating a unique and private key with each participant, provides a way for secure protocols to link actions to individuals, thereby enabling digital signatures and non-repudiation.

In addition to encryption, public-key cryptography can be used to implement digital signature schemes. RSA and DSA are two of the most popular digital signature schemes. Digital signatures are central to the operation of public key infrastructures and many network security schemes (eg, SSL/TLS, many VPNs, etc).

2.1.2 Cryptosystems

Public-key systems, such as Pretty Good Privacy (PGP), are becoming popular for transmitting information via the Internet. They are extremely secure and relatively simple to use. The only difficulty with public-key systems resides in the fact that the recipient's public key to encrypt a message has to be known. A global registry of public keys its required.

Some cryptosystems, such as the RSA encryption scheme, relies on some "difficult" mathematical problems (RSA relies on the difficulty of factoring products of two or more large primes), while Diffie-Hellman and DSA are related to the discrete logarithm problem. More recently, elliptic curve cryptography has developed in which security is based on number theoretic problems involving elliptic curves.. It is easy to see that given limited computing power but enough time, a composite number can always be factored. Such cryptosystems are said to be only computationally secure [55].

Gupta [34] presents the next table that summarizes some of the existing public-key crytosystems and their running time.

Public-key system	Examples	Mathematical Problem	Best known method for solving math problem (running time)
Integer factorization	RSA, Rabin-Williams	Given a number n, find its prime factors	Number field sieve: $\exp[1.923(\log n)^{1/3}(\log\log n)^{2/3}]$ (Sub-exponential)
Discrete logarithm	Diffie-Hellman (DH), DSA, ElGamal	Given a prime n, and numbers g and h, find x such that $h = g^x \bmod n$	Number field sieve: $\exp[1.923(\log n)^{1/3}(\log\log n)^{2/3}]$ (Sub-exponential)
Elliptic curve discrete logarithm	ECDH, ECDSA	Given an elliptic curve E and points P and Q on E, find x such that $Q = xP$	Pollard-rho algorithm: $\sqrt{n}$ (Fully exponential)

Figure 2.1: A comparison of public-key cryptosystems *in "Next generation security for wireless: elliptic curve cryptography"*.

An example is described in [25] of using a public key watermarking algorithm for image verification and authentication. The importance of public key extension is that while a private key is used in watermark insertion, the watermark can be checked using a public key, allowing for any person to perform the integrity check using a public key without the secure exchange of a secret key. The verification is performed using the public key of the owner, which also implies the original ownership of the image. As in public key cryptographic systems, the public key in this watermarking scheme can be published without compromising the security of the system.

For implementation, [25] project uses MD5 as its hash function, and the RSA public key encryption algorithm for encryption and decryption. It also lists the properties of the public key authentication watermark such as: the watermark is invisible; if a watermarked image is cropped or scaled or an image does not contain a watermark, the watermark extraction procedure returns an output that resembles random noise; if one uses the correct public key K in the watermark extraction procedure, one obtains an appropriate watermark.

Another example of using public and private keys is described in [26], which introduces computational security for public-key steganography. A public-key steganography protocol allows two parties, who have never met or exchanged secret data, to send hidden messages over a public channel so that other parts cannot detect that these hidden messages are being sent.

It also presents protocols for public key steganography and steganographic keys exchange that are secure under standard cryptographic assumptions, for example, the Decisional Diffie-Hellman (DDH) assumption. The DDH assumption is a computational hardness assumption about a certain problem involving discrete logarithms in cyclic groups. It is used as the basis to prove the security of many cryptographic protocols.

2.1.2.1 Cryptographic Hash Functions

Cryptographic hash functions (often called message digest functions) do not necessarily use keys, but are a related and important class of cryptographic algorithms. They take input data (often an entire message), and output a short, fixed length hash, and do so as a one-way function. The main idea of the use of a cryptographic hash function is to reduce the problem of protecting a (possibly long) message to the problem of protecting a short imprint of the message (the hash result)[57].

Message authentication codes (MACs) are much like cryptographic hash functions, except that a secret key is used to authenticate the hash value on receipt. These block an attack against plain hash functions.
The protection of the authenticity of information has two aspects: data integrity and data

origin authentication.

- **Data integrity** is the property whereby data has not been altered in an unauthorized manner since the time it was created, transmitted, or stored by an authorized source.

- **Data origin authentication** is a type of authentication whereby a party is corroborated as the (original) source of specified data created at some (typically unspecified) time in the past.

By definition, data origin authentication includes data integrity (information which has been modified effectively has a new source)[57].

2.2 Public Key Infrastructure(PKI)

Public Key Infrastructure (PKI) is a security infrastructure based on certificates and public key cryptography as its security components and a coherent implementation of technical, procedural and legal components aimed at allowing relying parties to trust the binding between a public key and a unique identifier of the owner.[53]

In order to set up a PKI environment, a certificate management system is required and must include creation, distribution, storage, verification and validation of certificates, plus should have the complete functionality to:

- issue certificates,
- receive certificate requests,
- evaluate certificate requests,
- grant/deny/discard certificate requests,
- list pending certificate requests.

For the present system, the trusted third party that authenticates entities and issues digital certificates, CA (Certificate Authority) and possibly also validates them (VA), is the Department of Finance which should be responsible for the certificate management, thus performing the proper diligence to verify that individuals or entities are in fact who they say they are, before issuing a digital certificate. For this process to be secure the CA's public key must be trustworthy and well-known. The success of a PKI implementation depends on the performance of CA to:

- Ensure that the public key that is associated with a private key belongs to the person specified in the public key.
- Establish credibility in order to all parties trust the CA.

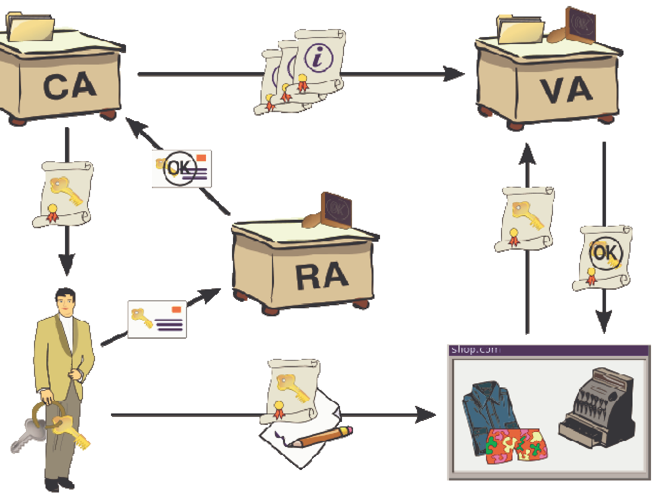

Figure 2.2: PKI [58].

2.2.1 Digital Certificates

Authentication in a PKI is realized through the use of public key certificates. A digital certificate is a collection of data used to verify the identity of the holder or sender of the certificate, where a binding between a public key and a Distinguished Name (DN) is created. The DN should uniquely identify the owner.

The SSL protocol does not depend on a particular format for the public key certificates it exchanges. Practical PKI deployments and implementations, however, depend heavily on the specifics of those certificates. One particular international standard is widely accepted as the appropriate format for public key certificates, the X.509 standard[53]. It contains such information as: version; serial number; certificate issuer; certificate holder; validity period (the certificate is not valid before or after this period); attributes, known as certificate extensions, that contain additional information such as allowable uses for this certificate ; digital signature from the certification authority to ensure that the certificate has not been altered and to indicate the identity of the issuer; public key of the owner of the certificate; message digest algorithm used to create the signature.

There are two ways to steal these types of digital certificates. The first is learning the private key. The second is more complicated, involving the manipulation of a vulnerable certificate ownership protocol. In the second method, an attacker tricks the certificate owner into performing an action that will allow the thief to feign knowledge of the private key.

Field	Description
Version	X.509 public key certificate version number. Three versions exist: 1,2 and 3.
Serial Number	Unique identifier for the certificate within the CA.
Signature	Identifies the algorithm that was used to sign the certificate.
Issuer	The DN of the issuing CA.
Validity	Specifies two dates between which the certificate is considered valid, unless it has been otherwise canceled.
Subject	The DN of the owner of the certificate.
Subject Public Key Info	The public key and algorithm identifier of the owner.
Issuer Unique ID	Optional field assigning a unique ID to the CA issuing the certificate. Not recommended for use in RFC 3280.
Subject Unique ID	Optional field assigning a unique ID to the owner of the certificate. Not recommended for use in RFC 3280.
Extensions	Includes a set of optional extension fields. These can be marked as critical or noncritical. Failure in processing a critical extension forces the examiner to reject the certificate, whereas a noncritical extension can be the skipped, like *Certificate Policies* and *Subject Alternative Name*.
Signature	A signature on all the above fields by using the issuing CA.

Figure 2.3: X509 Version 3 public key certificates [42].

2.2.1.1 Certificate Revocation List (CRL)

[42] A way for a victim to notify all the members of a PKI about a revocation is through a CRL, which is a listing of serial numbers belonging to certificates that have been canceled. X509 public key certificates contain an extension named 'CRL Distribution Point', which points to the location where revocation information for the given certificate can be obtained. CRLs were mainly adopted because in terms of revocation, Java only includes CRL based solutions.

2.3 Digital Transactions

P2P-Paid [3] allows two mobile users to conduct wireless payment trans- actions over Bluetooth. The system uses a 2-dimensional secured protocol, which not only supports peer-to-peer (P2P) payment transactions between two mobile clients but also supports the related secured transactions between the payment server and mobile clients since both mobile devices have already authenticated themselves with the payment system before operating on the payment functions and received the authentication key that will allow them to perform secure communication to one another.

Typical application examples include mobile payments between a TAXI passenger and

TAXI driver, between a merchant in flee market and its customers and payments for parking fees or subways.

P2P-Paid supports Bluetooth device/service discovery, send/request money, and payment management. The mobile payment protocol encapsulates four phases: service request, service discovery (where Bluetooth's SDP provides standard means for a Bluetooth device to query and discover services supported by a peer Bluetooth device), authentication, and transaction.

Bluetooth L2CAP protocol is used to establish a connection and send data between two MIDlet clients (considered as peers). The client communicates with the P2P-Paid server by using HTTP request and response to send or receive messages. It's important to mention that a security header is attached for each message sent across the Internet or directly between Bluetooth devices.

The architecture of P2P-Paid includes mobile client, middleware, P2P-Paid server, and a database server. The technologies and tools used include J2ME, Apache Tomcat Web Server, Java JSP and Servlets, J2ME Wireless Toolkit Emulator, MySQL Server.

This paper follows the same path chosen for the development of our system, i.e it describes architecture, design, payment protocol, and a security strategy.

Mobile Digital Receipts [5] presents a method for exchanging digital receipts between a mobile device and the store terminal and for storing that receipt which is quite similar to our project. After establishing a connection between the merchant's terminal and the customer's device, the system transmits an electronic receipt to the customer's device which provides receipt verification.

This method is not bound to a specific technique or protocol, the authors not wanting in any way to limit the application of such a method. Nonetheless, they acknowledge the existing techniques for controlling the validity of an electronic receipt, such as encryption, check-sum or maintaining a merchant database of electronic receipts.

The mobile devices that are considered in this paper include cellular phones, personal digital assistant (PDA), smart cards, wristwatch or a palmtop personal computer. For providing connection between the devices, the possible techniques include an auto-sensing wireless protocol, a manually initiated wireless protocol, a docking station , and a physical cable between the devices.

The non-limiting philosophy of this paper makes it more an idea than a feasible project, since it considers all technologies and techniques but none in particular.

The thesis reported in [22] presents an auditing system that detects the use of stolen digital certificates. The system uses dual logging via messages sent down side channels to a trusted third party which correlates these messages and automatically detects when an impostor presents a certificate based on the collected evidence. When an owner demonstrates knowledge of a digital certificate to an entity (called a verifier), side channel messages are sent to an entity called the Receipt Resolution Server (RRS). The RRS analyzes these messages for discrepancies and notifies owners if suspicious activity is discovered.

Although this thesis was designed specifically to protect the privacy of digital certificates, it could be adapted to protect other types of privacy as well. For testing, this paper developed a software prototype that integrates the system into TrustBuilder, a middleware trust agent that uses digital certificates and policies to reinforce the trust between strangers. The system developed in thesis [22] focuses on protecting public/private key certificates. In order to do so it identifies the existent types of theft. It also identifies four categories of methods used to prove the security of the digital certificate ownership protocols, including the most popular, BAN logic. BAN logic it's a logic of authentication, it contains no logical inversions, therefore it cannot be used to prove a protocol flawed, but when proof that a protocol is correct cannot be obtained, that protocol deserves to be treated with grave suspicion. .

Another important aspect of this paper is that identifies a wide variety of attacks, such as hijacking, forging and modifying the proof records and receipts.

PAM [4] aims to provide an integrated shopping experience. PAM can be used to automatically collect sensor information, manage user preference and context, connect to servers and peers on the Web, conduct service checking and integration, and execute user transactions. For location tracking of the items - location sensing - PAM uses RFID devices/tags due to the low cost and the device availability. The PAM prototype was built using Dell X50 PDA with a Syscan RFID reader, and .Net framework was the tool to implement Web services.

[2] tries to find a payment scheme(called U-payment) promoting seamlessness and privacy with a consumer device and peer-based information transactions. It propose a business model called U-PR that generates a digital coupon (U-Coupon) and a digital receipt (U-Receipt) from the payment process. The digital coupon and receipt are sent to the customer mobile device. The customer's device is the main element among the U-PR business model and possesses most payment-related information such as Payee Account number, Payer ID and Membership Information..

2.4 Technologies

This section provides a short revision on the communication technologies (such as Bluetooth, HTTPS, etc.) and the development tools (Java, iText, Jalimo, etc.) that were used in the system's implementation.

2.4.1 Communication

2.4.1.1 Bluetooth

According to the official Bluetooth website, "Bluetooth wireless technology is a short-range communications technology intended to replace the cables connecting portable and/or fixed devices while maintaining high levels of security. The key features of Bluetooth technology are robustness, low power, and low cost. Interferences are avoided by using a technique known as spread-spectrum frequency hopping and low power wireless signals, which increase security. The Bluetooth specification defines a uniform structure for a wide range of devices to connect and communicate with each other." Bluetooth's goals expanded to include standardized wireless communications between any electrical devices and created a notion of Personal Area Network.

The minimum number of communicating devices is 2 units, and a connection of this type is referred to as a point-to-point connection. A connection envolving more than 2 units is called a point-to-multipoint connection. Using two or more of these units forms a personal-area network (PAN) or piconet with one master, and up to 7 other units, called slaves.

Follows a resume of the bluetooth technology main advantages:

- Devices use wireless connection
- Inexpensive
- Low Energy Consumption
- Automatic setup
- Instant Personal Area Network (PAN)
- Standardized Protocol
- Low Interference
- Upgradeable
- Share Voice and Data

Bluetooth devices can establish connections through walls, and once that connection is established, other devices do not have to be stationary. This technology is highly suitable for small devices such as cordless phones, headsets for cell phones, etc [39].

The interface of a Bluetooth device consists of the Radio and Baseband layer, and the Link Manager Protocol (LMP). The Radio is responsible for transmitting data via radio waves over the air. The Baseband, which has both software and hardware, controls the radio and is responsible for lower level encryption. The Link Manager Protocol is in charge of link setup and configuration, authentication, control and security. The link manager controls the state of the current Bluetooth device as well as the connection with other Bluetooth devices in the Piconet.

The process of establishing links in order to transfer data is called a pairing.

While authentication and privacy could be handled at the software protocol layer, they are also provided in the Bluetooth physical layer. A particular connection can be specified to require either one-way, two-way, or no authentication.

For authentication, the verifier (either master or slave) sends a random number to the claimant, which in turn calculates a response, including the secret key. Both devices share the same secret key. If a slave is receiving and sending an authentication request at once, it first accepts the incoming request by an authentication response and then starts its own attempt.

For encryption, a temporary link key must be issued from the master to the whole piconet, while every device has to agree on the use of encryption. Then, the master is configured to only receive encrypted packages. Next, the slave is configured to transmit and receive encrypted packages. All Bluetooth devices implement this physical layer security in the same way. Of course, for highly sensitive applications, it is also recommended that you utilize more advanced algorithms in the network transport or application layer [41].

Bluetooth devices perform a discovery operation to find other devices nearby. The user must choose the proposed recipient, which will require special information.

2.4.1.2 IP over Bluetooth

The Bluetooth wireless technology system contains a set of profiles. A profile defines a selection of messages and procedures (generally termed capabilities) from the Bluetooth SIG specifications.

A profile can be described as a vertical slice through the protocol stack, that is, a set of rules to use the Bluetooth protocol stack in a device. It defines options in each protocol that are mandatory for the profile. Profiles are used to decrease the risk of interoperability problems between different manufacturers' products. Different profiles are defined for different types of services.

Profiles do not (normally) define any additions to the Bluetooth specification, which is why new profiles can be added where desired. The underlying Bluetooth technology is the same, only the specific way that it is used is defined and clarified.

The Public Area Network Profile (PAN) profile allows Bluetooth units to participate in a Personal Area Network. In other words, the PAN profile lays out the rules for carrying

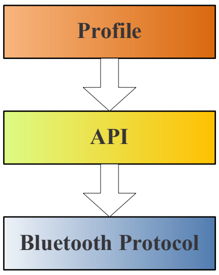

Figure 2.4: Connection between a profile and an API for a Bluetooth protocol [40].

Internet Protocol (IP) traffic across connections, which can be understood as Ethernet emulation over Bluetooth. Ethernet packets are encapsulated into Bluetooth special packets (L2CAP packets) using Bluetooth Network Encapsulation Protocol (BNEP) and are sent over Bluetooth link.

To the devices participating in the ad-hoc network, are assigned one of following roles:

- **Network Access Point (NAP)**, a Bluetooth device that provides some of the features of an Ethernet bridge. The NAP device forwards Ethernet packets between each of the connected Bluetooth devices, refereed to as PAN Users (PANU). In addition, NAP forwards Ethernet packets to/from a different network connected to the NAP via Layer 2 bridging or Layer 3 routing mechanism.
- **Group ad-hoc Network (GN)**, a Bluetooth device acting in the same way as a NAP device with the exception that it does not provide access to any additional networks.
- **PAN User (PANU)**, a Bluetooth device that uses either the NAP or the GN service.

The PANU uses baseband inquiries and the Service Discover Protocol (SDP) to find other devices that provide the NAP or GN service. The NAP or GN acts as a bridge and forwards packets between PAN users, and each of them is treated as if it were a bridge port.

The BNEP allows IP packets to be carried in the payload of L2CAP packets. It attaches a header to the information to identify the packet type. The session management layer (TCP or UDP) comes above the IP layer [40].

Christian Gehrmann [29] describes the usage of the Bluetooth security as well additional security architectures for selected Bluetooth wireless profiles. The architecture presented in [29] contains detailed security recommendations applicable for the different profiles. Some general recommendations apply to all profiles.

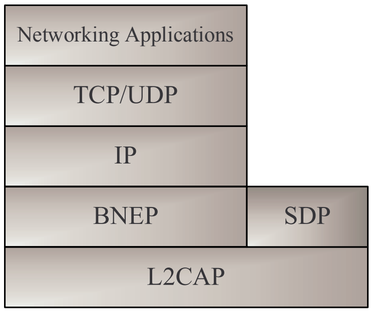

Figure 2.5: The BNEP protocol stack [40].

This paper also describes the air interface for specified services and use cases. Working groups within the Bluetooth SIG define these profiles. The Security Expert Group (BSEG) provides the Bluetooth SIG and associated working groups with expertise regarding all aspects of Bluetooth security.

2.4.1.3 HTTPS

HTTPS is an Internet protocol that provides a SSL layer of security and uses both SSL and HTTP to protect the communication channel between the client and the server on a network.

SSL is designed to make use of TCP as a communication layer to provide a reliable end-to-end secure and authenticated connection between two points over a network (for example, between the service client and the server). SSL can be used for protection of data in transit in situations related to any network service and it is commonly used in HTTP server and client applications.

2.4.1.4 SSL Protocol

The SSL protocol enables server and client authentication, and an encrypted SSL connection between a client a server. The server/client validates the certificate and the public ID of a client/server and verifies that they are issued by a Certificate Authority (CA). When SSL is used, a handshake is performed by the client and the server as shown in Figure 2.6.

The handshake may also include a client certificate to let the server authenticate the client. After the handshake is done, control is passed to the original handler, who now communicates through the encrypted channel. An encrypted SSL connection between a client and a server provides confidentiality to the transmitted data, since encrypted data is sent.

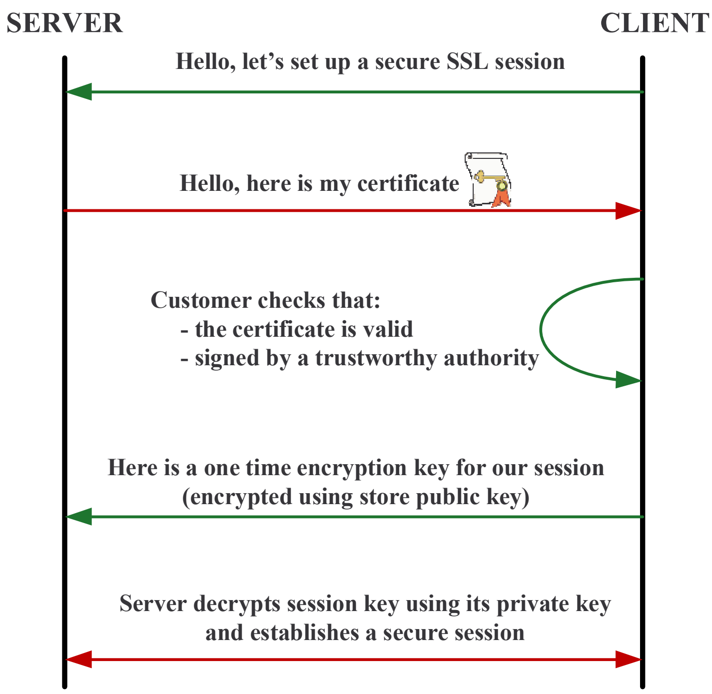

Figure 2.6: SSL handshake protocol [37].

SSL is suitable for wired Internet environment. However, for restrictive devices such as the mobile phone or smart card, the protocol is too heavy for implementation. Moreover, a SSL session can only be reused for connections between the same pair of client-server. If a client tends to communicate with different servers, a new SSL session must be established for each server [37].

RSA (in conjunction with the specified encryption and digest algorithm combinations) is used for both key generation and authentication on SSL connections. This usage is independent of whether a certificate authority (CA) trustpoint is configured [35].

2.4.1.5 OpenSSL

The OpenSSL Project is a collaborative effort to develop a robust, commercial-grade, full-featured, and Open Source toolkit implementing the Secure Sockets Layer (SSL v2/v3) and Transport Layer Security. The core library (written in the C programming language) implements the basic cryptographic functions and provides various utility functions. Wrappers allowing the use of the OpenSSL library in a variety of computer languages are available.

OpenSSL is based on the excellent SSLeay library developed by Eric A. Young and Tim J. Hudson. The OpenSSL toolkit is licensed under a Apache-style licence which basically means that you are free to get and use it for commercial and non-commercial purposes.

2.4.1.6 Apache

Apache HTTP Server (commonly known as Apache) is a web server, which means it is a program that accepts HTTP requests from a client program (usually a web browser) and provides HTTP responses. Apache may serve:

- static content, when the information served in the response consists of files that existed prior to the request in the filesystem,
- dynamic content, when the information consists of data generated by an application called by the web server during the processing of a request.

Serving static content is much faster than serving dynamic content, especially when dynamic content involves database connections. However, and due to the natural evolution of the world wide web, there is a constantly growing need of serving content (text, images, form fields, etc.) that can change in response to different contexts or conditions (time, user input, etc.). Apache supports a set of features, which extends the web servers basic functionality, and that can be integrated as independent modules. These modules can enable the support of different server-side programming languages (like PHP, for example), authentication, SSL, logging, etc.

2.4.1.7 Apache Tomcat

An application server is a sever program in a computer within a distributed network, that executes applications when requested by other clients [52]. It also stores the business logics and the business models classes of applications.

Apache Tomcat is a Servlet container. It is an implementation of the Java Servlet and JavaServer Pages technologies, developed at the Apache Software Foundation (ASF) and it provides an HTTP web server environment for Java code to run. It is commonly used integrated with Apache HTTP server combining the best of both servers.

2.4.2 Development Tools

2.4.2.1 Java

Java is an object-oriented programming language developed by Sun Microsystems and released in 1995. It was developed with the purpose of being a language whose programs are platform independent, meaning they could run on any hardware as well as on any operative system.

Java's goal is to write a program, compile it once, and run it anywhere. To achieve this goal, Java code is compiled to an intermediate code, called 'Java bytecode', which consists on simplified machine instructions of the Java platform. This code will be executed on a virtual machine (Java Virtual Machine), a program written in the machine's native code, which interprets and runs the bytecode.

Being Java an object-oriented programming language, it is essential to comprehend the concept of the word 'object'. An Object is a portion of software with a state of its own and behaviors that change it. An object stores its state in 'fields' (or variables) and exposes its behaviors through 'methods'. Methods operate on the internal state of an object and act as the main mechanism for the communication between objects. Grouping code into individual software objects has as its main advantages [56]:

- Modularity: an object can be developed and maintained independently of any other object
- Data encapsulation: by demanding that the interaction with an object can only be achieved through its methods, the details of its internal implementation remain hidden from the 'outside world'
- Code reutilization: if an object already exists (developed by someone else), it can be used for the development of a new program.
- Simple installation/replacement and debug; if an object turns out to be problematic, it can be easily removed from the application and replaced by another. This situation is similar to the real case of the resolution of mechanical problems. If an auto part is broken, the part is replaced, not the car.

Inheritance is a way of creating a new class (type) by extending another class, i.e. a new class is formed by adding data and/or functionality to an existing class. All but one class in Java, *Object*, are created in this manner. The process is referred to as either the subclassing or extending a class. Inheritance can be viewed as being a *is-a* relationship between classes. As one class has inherited all of the attributes, i.e. data members and methods, of the other class, it is also of that type.

Java only supports "single-inheritance", which means that a class can only inherit from a single superclass.

2.4.2.2 JavaServlet

Servlet is a Java class used to extend the application servers' ability to support request-response communication models. Servlets can respond to all sorts of requests, but are usually used in applications of web servers. Thus, Java Server technology defines a specific class that handles HTTP requests (HttpServlet). A Servlet container is a specialized web server that supports the execution of servlets. It combines a web server's basic functionality with Java Servlet technology's optimizations and specific extensions - such as a Java runtime environment and the ability to translate specific url's into servlet requests.

Individual servlets are registered in a servlet container, giving it information about the functionalities they provide and the URL (or another resource locator) they use to identify themselves. The servlet container controls its servlets lifecycles, follows the set of steps executed when it receives a request:

1. If there's not an instance of the servlet, the web container
    - Loads the servlet class,
    - Creates an instance of the servlet class,
    - Initializes the servlet instance through its 'init' method.
2. Calls the 'service' method, passing a request object and a response object as parameters.

After loading and instantiating the servlet class, and before it is able to respond to requests, the web container needs to initialize the servlet. The service provided by a servlet is implemented in the 'service' method of a GenericServlet, the 'doXXX' methods (where XXX can take the value of 'Get', 'Delete', 'Options', 'Post', 'Put', 'Trace') of a HttpServlet, or any other protocol specific methods defined by a class which implements the servlet interface.

A service method's general behavioral pattern is to extract information from the request (HttpRequest object), access to external resources and build the response (HttpResponse object), based on that information.

2.4.2.3 Eclipse (IDE)

Eclipse is an extensible, open source IDE (integrated development environment). According to [44], the stated goals of Eclipse are "to develop a robust, full-featured, commercial-quality industry platform for the development of highly integrated tools." The Eclipse platform, when combined with the Java Development Tools, offers many of the features expected from a commercial-quality IDE:

- a syntax-highlighting editor,
- incremental code compilation,
- a thread-aware source-level debugger,
- a class navigator,
- a file/project manager.

Despite the large number of standard features, Eclipse is different from traditional IDEs in a number of fundamental ways. The most interesting feature of Eclipse is that it is completely platform- and language-neutral. In addition to the eclectic mix of languages supported by the Eclipse Consortium (Java, C/C++, Cobol), there are also projects underway to add support for languages as diverse as Python, Eiffel, PHP, Ruby, and C# to Eclipse.

2.4.2.4 iText

iText is a library that allows applications to dynamically generate and manipulate PDF files on the fly. iText is built into applications when, for example:

- Due to time or size, the PDF document can't be produced manually.
- The content of a PDF document needs to be calculated or based on user input.
- The content needs to be customized or personalized.
- The PDF content needs to be served in a web environment.
- Documents are to be created in "batch process" mode.

Amongst its possible applications, iText can be used to: serve PDF to a browser, generate dynamic documents from XML files or databases, split, concatenate, and manipulate PDF pages, automate filling out of PDF forms, add digital signatures to a PDF file, etc.

Basically, the iText classes are useful for generating read-only, platform independent documents containing text, lists, tables and images; or for performing specific manipulations on existing PDF documents.

2.4.2.5 Nokia Internet Tablet - Jalimo

The Nokia N800 Internet Tablet is a wireless Internet appliance from Nokia. The N800 is not a phone, but instead allows the user to browse the Internet and communicate using Wi-Fi networks or with mobile phone via Bluetooth. This device enable end users to utilize the power of the Internet and rich applications while on the go.

Like all Nokia Internet Tablets, it runs Internet Tablet OS, which is similar to many handheld operating systems, and provides a "Home" screen, the central point from which all applications and settings are accessed. The Home screen is divided into areas for launching applications, a menu bar, and a large customizable area that can display information such as an RSS reader, Internet radio player, and Google search box for example. Internet Tablet OS is a modified version of Debian GNU/Linux [51].

[59] Maemo provides an open source development platform for Nokia Internet Tablets and other Linux-based devices. It is built from components widely used in open desktop and mobile systems, and it is aimed at enabling applications and innovative technology for mobile handheld devices.

The maemo platform provides developers with a powerful and easy to use development, build, and test environment. It was created with the intention to be open, accessible and useful to all developers wanting to squeeze the possibilities of the mobile desktop and the Internet. The platform is based on the GNU/Linux operating system and the GNOME desktop. Maemo brings developers an easy to use development environment. Its new, optimized, and evolving Hildon UI is customized for the screen size and usage typical for a touch screen enabled handheld device. The development platform is targeted at innovative developers and innovation houses developing applications and new technologies for the mobile space. Utilizing the maemo platform it is easy to mobilize existing desktop solutions. Maemo enables various business models on the top of the platform, including proprietary application distribution. Figure 2.7 shows the architecture overview.

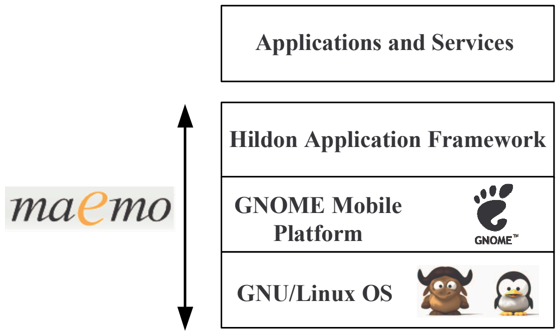

Figure 2.7: Maemo - Architecture overview [51].

The N800 is bundled with several applications including the Mozilla-based MicroB browser, Macromedia Flash, Gizmo, and Skype [5]. It is compatible with any software designed for Internet Tablet OS, and supports most common file formats.

However, Sun has never released a full Java Virtual Machine (JVM) for the N800, what is a problem, because in order to run Java code a JVM is required. Fortunately, a number of open source, alternative JVMs have been written for the system, such as Kaffe, JamVM and CACAO, which vary in quality and efficiency.

Jalimo is a project to maintain a full featured free Java-like stack for mobile Linux-based devices. The aim is to pack and tailor existing JVM-related projects, as well as to develop APIs and tools for easy integration and development for the target platforms.

Jalimo environment is based on CacaoJVM, JamVM, GNU Classpath. It has robust working environments and packages for OpenMoko, Maemo and the Irex Iliad [50].

Chapter 3

Client-Store Communications Architecture

3.1 Introduction

It is assumed that the overall architecture of this system is the one described in section 1.2. The focus of this chapter is on the specification of the interaction between client and store while exchanging digital receipts. We will not go deeper into the other aspects of the basic architecture. Namely, we assume that the following is true:

- The creation of the digital receipt is independent of the payment method and the required data will be available to fill the receipt properly.
- In order for receipts to be accepted as authentic, a means of signature and validation needs to be used. Digital certificates, which are analogous to physical credentials, can be presented as evidence of authenticity.
- A PKI exists that is recognized by the Department of Finance (the sole responsible for the administration of public finances, including the collection and expenditure of the revenues); an authorized CA is able to issue digital certificates.
- Both client and store have already requested and are in possession of a public key certificate issued by the CA.

3.2 Functional Requirements

The requirements posted above refer mainly a transparent system in the user's point of view. However, the requirements of the protocol should be more specific towards security, communication and storage:

- The client device should request its receipt to the server.

- Each device should have a private and a public key, for data encryption and decryption, respectively.
- The client should receive an encrypted receipt that will be decrypted thanks to the seller's device public key held by the client.
- The request-response protocol should allow each device to authenticate the other, after which it is possible to encrypt the transmission for added security.

3.3 Main Elements

The main elements involved in the system's interaction are the store and the client. However, an extra component called the CRL is required to reinsure the existence of a certified store. The CRL is the Certificates Revocation Lists and it provides the final validation of the store. These lists are issued by the same entity that issues the certificates, and because of that, the CRL element could be seen as part of the CA (Certificate Authority).

The following diagram shows the main transactions' flow between the three elements:

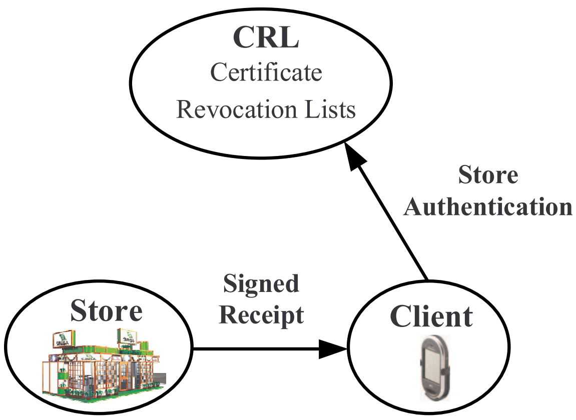

Figure 3.1: Basic Client-Store Architecture

3.4 Communication Protocol

The communication protocol defines the exchange of messages between the three main elements in order for the system to work properly, to be secure, and to provide insurances for outside entities.

The protocol consists of the following steps:

- A secure connection is established between the client and the store.

- The store's authentication. The store sends its certificate to the client, enabling him to authenticate the connection that's been previously established (by confirming that he is communicating with the correct store).

- The client sends his NIF (tax identification number) to the store, so that the receipt can be created.

- At last, the store sends the receipt (containing the client's and the store's NIF) signed with its private key.

The presented protocol represents the main focus of the system. However, as it was described previously, there is an important step required for the client to be sure of the validity of the store's certificate, which is the communication with the CRL.

The client sends to the CRL server the store's certificate and confirms if the certificate is still valid or if it has been revoked. Figure 3.2 illustrates the communication protocol of the system.

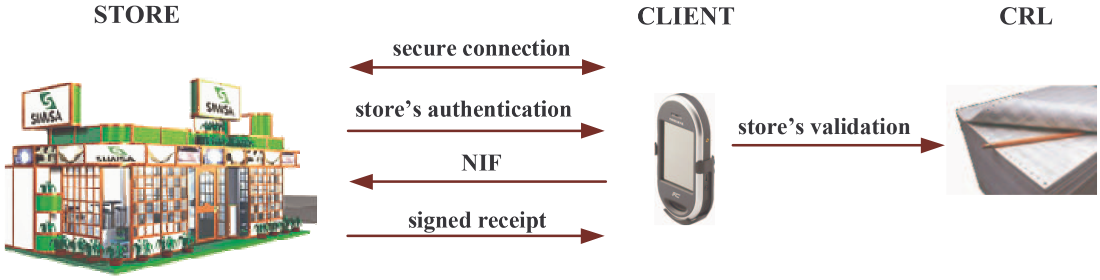

Figure 3.2: Communication Protocol

The establishment of a secure connection is the first requirement to fulfill, since it assures that only these two entities participate in the communication, and no outsiders can listen, replicate or alter the data exchanged between them. One should note that a secure communication guarantees that the connection is done with a certified entity, but it doesn't, however, guarantee that this entity is the desired one.

Thus, the exchange of the store's certificate is necessary to authenticate the store, giving the client the certainty he is sending and receiving data from the entity he is supposed to. The authentication comes from the comparison between the unique identifier in the certificate and the one used to establish the secure connection (or the one who is being broadcasted by the store). Authenticity is guaranteed because the fields of the certificate cannot be changed, unlike the information the store broadcasts. It is important to note, that the sending of the store's certificate only assures security for the client when it is possible to establish a trustful relationship between the information included in the certificate and the one that is used to establish the connection.

The authentication of the store can be done before or after the client sends his personal

NIF. This way security encloses all of the communication system, leaving minimum space for leaks.

3.5 Backstage Procedures and Assumptions

The communication protocol on its own is not sufficient to implement a working system. At this point it is important to refer some of the assumptions that were made in order to define in more detail the communication protocol. Those are:

- The store has a certificate signed and issued by a Certificate Authority (CA). From the PKI system it's known that a CA is required to issue the store certificate that will be used to authenticate the store and validate its signed receipts. The store's certificate associates its unique identifier (such as the store's name or the store's NIF) with a public key. This certificate is certified by the CA and allows the client to assure both the store's authentication and the validation of the receipt signed by it (using the public key included in the store's certificate).

- The client also stores the CA certificate, which makes it possible to establish the trusted chain of the store's certificate. This is the only way for the client to verify the authenticity of the store's certificate and, therefore, the store's competence for signing documents.

Figure 3.3 shows the certificates' workflow.

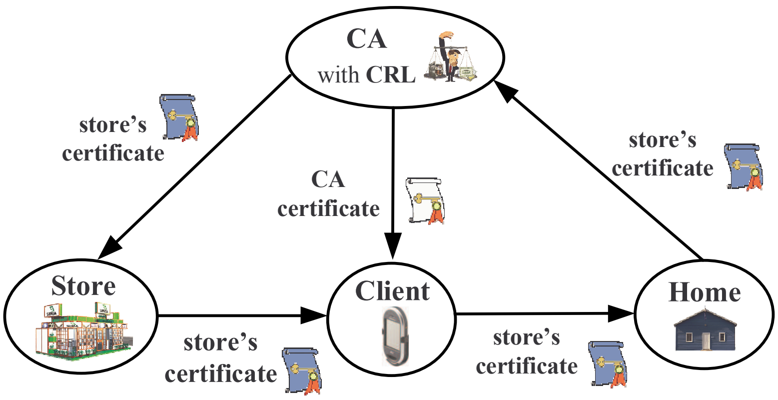

Figure 3.3: Certificates'Workflow

Besides the pre-action where both store (request of its own certificate) and client (request of CA public key)communicate with the CA, backstage procedures include the receipt creation and signature by the store during the communication with the client. The creation of the receipt uses information about the client's purchase and his NIF (which the client sends after the store's authentication). For this architectural design it was assumed that the store's application has full access to all information about the client's purchases (like items, prices, discounts, etc.). The receipt should follow the guidelines imposed by the respective authority (Department of Finance).

3.6 Architectural decisions

At this point some architectural decisions need to be justified, such as the need for the client to send his personal tax number, or the receipt to be signed by the store, when the store's already been authenticated by the client.

The receipts that are exchanged nowadays in stores do not require the client's tax number. The tax number is only needed if the client wishes to declare the purchase to the Finance Services, and in most of the cases the store neither obtains the client's NIF, nor needs it.

The decision to integrate the NIF information as part of the communication protocol was made for two reasons:

- Since the receipt is generated and signed by the store, including the tax number (NIF) will assure that the entity that receives the receipt is the real client, making it impossible for others to use the same receipt as their own.
- Generating a receipt with the client's NIF provides new ways, possibly better ones, for the client to submit his IRS statement.

The reason why the receipt must be signed by the store is easily explained through the following:

- A signature in a document obligates the store to be responsible for the information it contains. This way, even after its authentication, the store needs to include valid information in the receipt.
- A digital signature is a guarantee that the information in the document has not been changed. On top of all the security measures taken before, this is the last one that validates the receipt.
- Taken into account that, ultimately, the receipts generated by the system must be recognized by the Finance Services, this entity needs to be certain that they originated from valid sources. A signed receipt is the way to ensure it.

As seen before, a certificate can be compromised before its expiration date, and the only way for the client to know if that's the case with the store's certificate is to consult the CRL. Thus, the client can reject the store's certificate and its receipt and warn the responsible entities in case he is in the list.

The main goal of this chapter was to present the chosen architecture for the development of the system. The next chapter will show an implementation that aims to validate this architecture.

Chapter 4

Implementation

The purpose of the implementation is to develop an infrastructure that is capable of validating the previous architecture and assess a number of issues such as security.

This chapter describes how the chosen technologies and tools were used to implement the previous architecture. Although the architecture is the core of the thesis and all the work is developed around it, the implementation turned out to be much more complex than anticipated. This section describes in detail the development of the infrastructure throughout its different phases.

4.1 Roles

The implementation is based on a client-server architecture. Thus, the store will have a server, which will be responsible for replying to requests sent through the customer's device. This device will hold an application that will serve as the client, generating these requests. This application must also be able to process the responses obtained from the server. The following image shows the entities that participate in the communication.

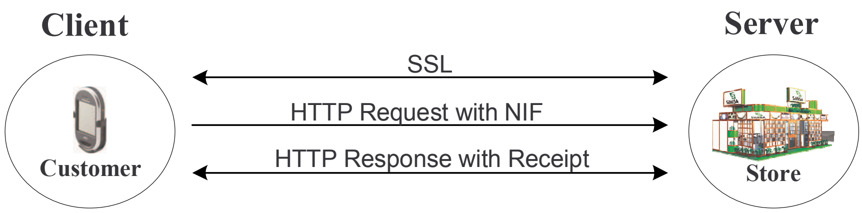

Figure 4.1: Entities' roles.

4.2 Implementation Decisions

4.2.1 Secure Communication

Initially, the application was viewed as a web-application, meaning that a customer (client) would use a web browser in order to communicate with the store (server). Taking this into account, and having the need of a secure communication protocol, the HTTPS protocol was the chosen one. The HTTPS is a secure version of the HTTP protocol, because it uses the SSL protocol to protect the communication channel through which the HTTP messages are sent. The HTTP is a request-response oriented protocol, and it is used by a WWW client (e.g. a browser) to send a request to a Web Server, that responds with a file (usually a web page). Due to it being widely used in internet communications, this was the best approach to initiate the implementation. It was important for both the methodology and the development of the system to follow a simplified track, that could lead to an accurate feedback of the work in progress.

4.2.2 Programming Language

Regarding the choice of the programming language to use in the development of the system, Java was elected. The reasons that led to this choice were [56]:

- Java is platform-independent. Being platform-independent at both the source and binary levels, it has the ability to move easily from one computer system to another, without having to alter the program. This makes it ideal for mobile applications.
- Java is interpreted. To run a program it is required a Java interpreter, Java Virtual Machine (JVM), and the program only needs to be compiled once (which makes it portable), because the bytecode generated by the Java compiler can run on any platform that has a JVM.
- Java is object-oriented. Because it is a programming language that is centered on creating and manipulating objects, it allows the creation of modular programs and reusable code.
- Java is simple. It was designed to be easy to use and therefore is easier to write, compile, debug, and learn than other programming languages.
- Java is secure. It is one of the first programming languages to consider security as part of its design. The compiler, interpreter, and Java-compatible browsers all contain several levels of security measures that are designed to reduce the risk of security compromise, loss of data and program integrity, and damage to system users. This feature was extremely important because, on top of this, there are a number of useful libraries that allow the creation of HTTPS-based applications.

These features made it the best language to use in the development of both applications. Although the server-side application did not require to use the same programming language as the client's application, the fact that the server would need to respond to HTTP requests led to the use of servlets.

Servlets are modules of Java code that run in a server application to answer client requests. They are not tied to a specific client-server protocol but they are most commonly used with HTTP. The decision of using the HTTPS as the communication protocol contributed to the selection of Java Servlet as the server-side technology.

Java servlets are [54]:

- Efficient, with servlets, the JVM stays up, and each request is handled by a lightweight Java thread, not a heavyweight operating system process.
- Convenient, servlets have an extensive infrastructure for automatically parsing and decoding HTML form data, reading and setting HTTP headers, handling cookies, tracking sessions, etc.
- Powerful, servlets can talk directly to the Web server. This simplifies operations that need to look up images and other data stored in standard places. Servlets can also share data among each other, making useful things like database connection pools easy to implement. They can maintain information from request to request, simplifying things like session tracking and caching of previous computations.
- Portable, servlets are written in Java and follow a well-standardized API. Servlets are supported directly or via a plugin on almost every major Web server.
- Inexpensive, there are a number of free or very inexpensive Web servers available that are good for "personal" use or low-volume Web sites. However, with the major exception of Apache, which is free, most commercial-quality Web servers are relatively expensive. Nevertheless, once the Web server is installed, adding servlet support to it (if it doesn't come preconfigured to support servlets) is generally free or cheap.

4.2.3 Digital Certificates

The architecture assumes the Public Key Infrastructure as the foundation of the system. Therefore, digital certification is a crucial requirement.

As previously reviewed, digital certificates are electronic documents which incorporate digital signatures to bind a public key to a variety of identification information. They provide means to authenticate entities and sign documents, making them secure for the receivers. A digital certificate can be seen as an electronic "driver's license" that is used to prove the identity of a client or server. The X.509 standard format for certificates was the one used for this implementation.

An X.509 certificate contains a public key that corresponds to a private key which cannot be feasibly derived from the public key. Proving ownership of this type of digital certificate involves proving knowledge of the associated private key. Therefore, in order to avoid theft of an X.509 certificate one must maintain the secrecy of the private key [22].

Since the protocol will be implemented through HTTPS requests, the common name of the certificates is the server's IP address. This is required because when a request is made to the server (https://192.168.1.101) the certificate used in the SSL handshake (the server's certificate) is the one that has been issued for the same requested IP. The importance of the Common Name in digital certificates will be discussed in Appendix B.

For the development of the system, the digital certificates had to be issued and signed by a fictional entity (CA). In order to simulate the CA, OpenSSL was used to generate and sign the certificates.

OpenSSL is an open source implementation of the SSL and TLS protocols as well as a library that implements the basic cryptographic functions and provides various utility functions. These functions include:

- generation of private keys;
- management of public and private key;
- generation of various types of certificates;
- signing of certificates;

The OpenSSL toolkit is licensed under an Apache-style licence, which means that it can be freely obtained and used for commercial and non-commercial purposes.

When creating certificates and private keys, these can be encoded in different types of files, such as PKCS12, DER or PEM files. Although each of the files contain the same information, the type is restrictive to where they can be used. For example, a DER or PEM file works fine with Apache, but with Tomcat they need to be converted to PKCS12 format.

When an organization requests a digital certificate from a certificate authority, it needs to submit a Certificate Signing Request (CSR). The certificate signing request contains a public key, a common name (e.g., www.example.com) to uniquely identify the site, and locality information to identify the organization.

4.3 Methodology

The methodology used to develop the system for the exchange of receipts between the store and the client's mobile device was defined considering four main steps, which consist of the implementation of the communication protocol for:

- localhost
- remote PC with wireless

- remote Tablet with wireless
- remote Tablet with Bluetooth

The server remains the same for every scenario. Now that the technologies and tools have been chosen, it is possible to redesign the proposed architecture in a more detailed way.

Figure 4.2 is the sequence diagram of the communication protocol used in the implementation.

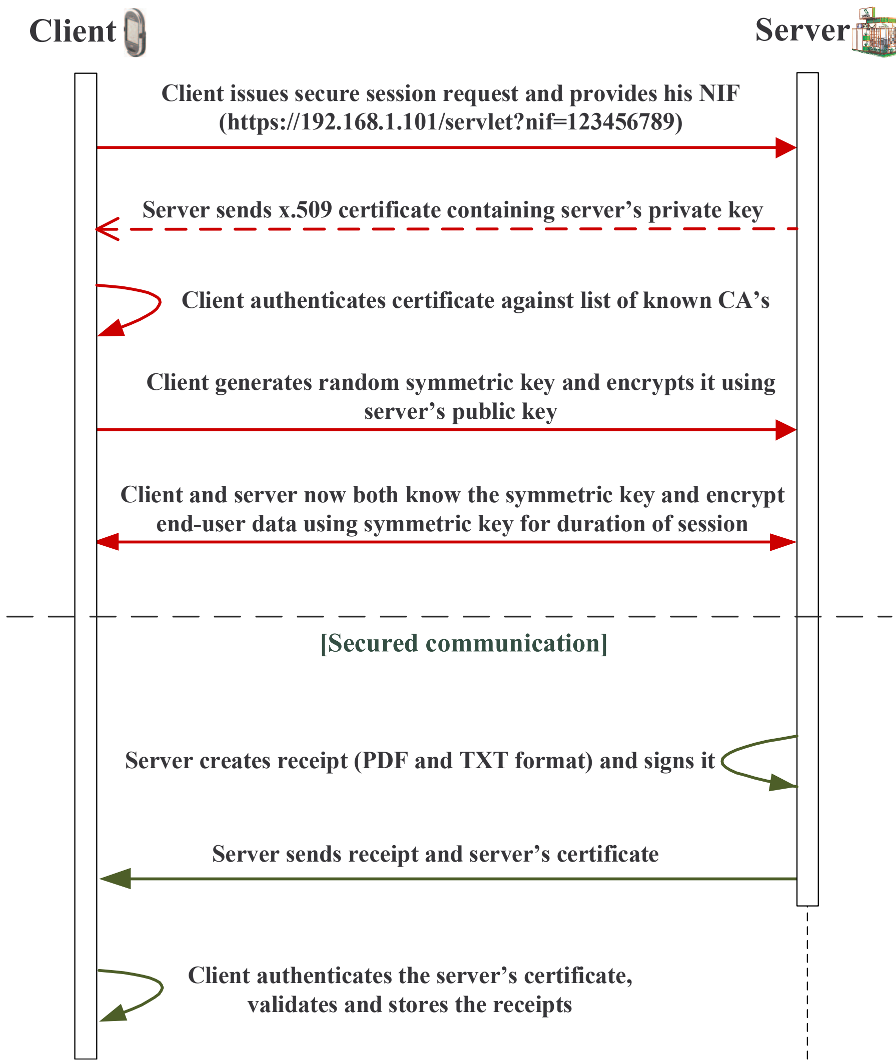

Figure 4.2: Sequence diagram of the protocol.

The next section describes every step of both client and server side implementation, mentioning differences between each scenario when relevant.

4.4 Development

Having no doubts about what technologies and tools to use, it was time to start developing. Based on the previous section, the development was divided in four subsections.

4.4.1 Localhost

The implementation of the localhost communication protocol was divided in two different applications, the server application (Servlet) and the client application (HTTPS client).

4.4.1.1 Server-side

A server is an application or device that performs services for clients as part of a client-server architecture.

The server application is the first item to be addressed for the simple reason that no request can be done if there's not a server to respond. Thus, the next subsection will describe in detail how the server application was developed.

Apache was the HTTP-server selected to install on the server-side. According to [45],"the Apache HTTP Server Project is a collaborative software development effort aimed at creating a robust, commercial-grade, featureful, and freely-available source code implementation of an HTTP (Web) server."

The development of the server application was done in the following steps:

- I. Installation of Apache 2.2;
- II. Generation of digital certificates and the associated private keys;
- III. Configuration of Apache to work with SSL.
- IV. Installation of Tomcat 6.0;
- V. Configuration of Apache to forward the client's requests for servlets to Tomcat;
- VI. Development of the servlet to process the client's requests.

Each of the previous items will be described in detail in the next subsection. The first five items describe the setup of the environment that allowed the developer to start implementing and testing without restrictions from the server's configuration.

4.4.1.2 I. Installation of Apache

The installation of Apache 2.2 did not present any problem, and the package included a lighter version of the openssl, so it was possible to generate the certificates without having to install the complete OpenSSL toolkit.

4.4.1.3 II. Generation of the certificates

Generation of the store's certificate implied the creation of a CA certificate and a private key that would be used to certify other certificates. These were the commands used.

1. Generating the CA's certificate and private key:

» openssl genrsa -des3 -out **ca.key** 4096

» openssl rep -new -x509 -days 365 -key **ca.key** -out **ca.crt**

These commands create a self-signed certificate that will be used to sign other certificates. The openssl utility will prompt for a pass-phrase, which is used to encrypt the contents of the private key. Once the private key is generated, using RSA encryption algorithm, it is placed in the file ca.key.

During the request generation process of the certificate, various pieces of information are gathered, like the Organization Name (Department of Finance in the example) and the Common Name (Finances CA). In this case, the Common Name of the CA certificate is not important, because no request will be made directly to it. The CA certificate was placed in the ca.crt file.

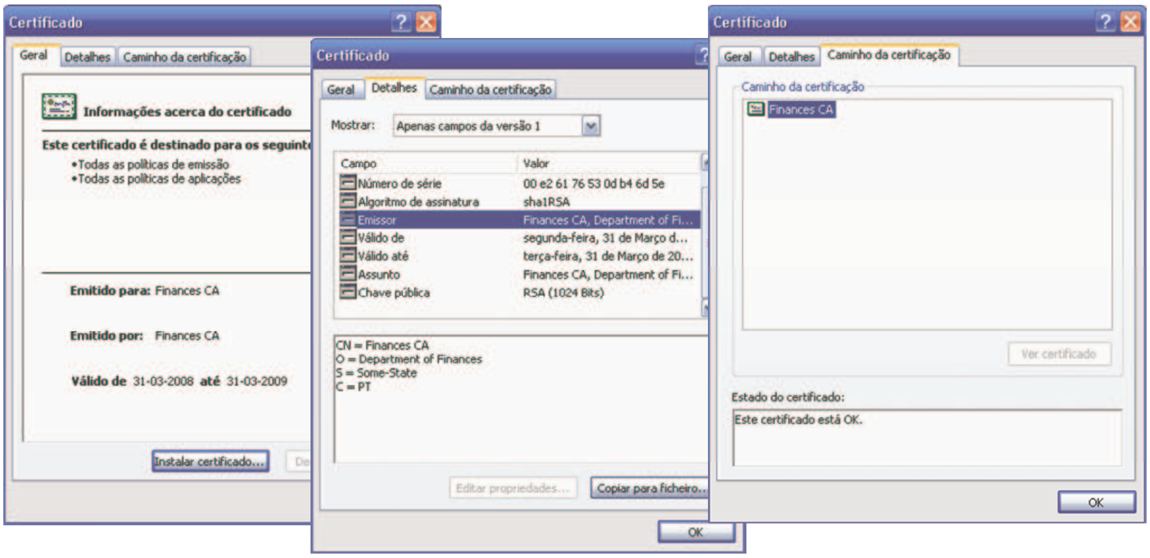

Figure 4.3: The CA's certificate.

2. Generating the server's key and certificate

» openssl genrsa -des3 -out **server.key** 4096

» openssl req -new -key **server.key** -out **server.csr**

The Certificate Signing Request is placed in the file server.csr. This file contains the public key, locality information, and a common name to uniquely identify the site. The

Common Name is the most important field of the certificate, because it is through this name that the request is identified. For example, if a client requests an URL to the server, over a secure communication, the server responds with the certificate that has the common name identical to the requested URL (otherwise, it won't recognize it as a valid request). Therefore, for each scenario (localhost, remote PC, etc.) a different server's certificate has to be generated. In this case, the common name of the certificate is 127.0.0.1, which is the localhost address.

3. Signing the server's certificate with the previous created CA's private key

» openssl x509 -req -days 365 -in server.csr -CA **ca.crt** -CAkey **ca.key** -CAcreateserial -out **server.crt**

The output data is saved in the server.crt file, which is the certificate that will be used when the store's authentication is required by the clients.

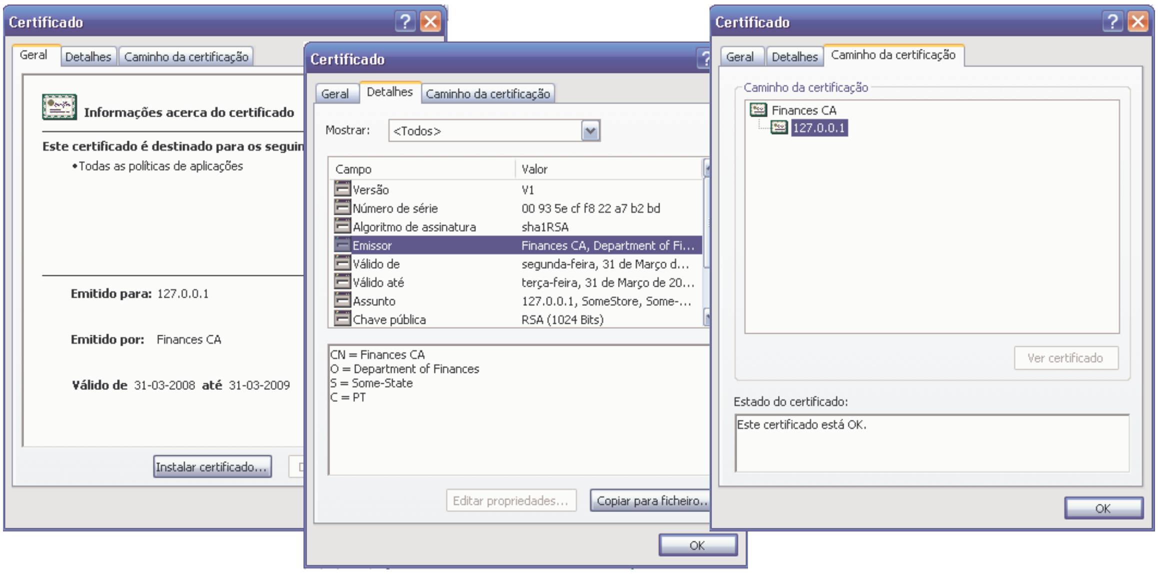

Figure 4.4: The store's certificate.

4. Changing the server.key from secure to insecure

» openssl rsa -in **server.key** -out **server.key.insecure**
» ren server.key server.key.secure
» ren server.key.insecure server.key

This step was only required to avoid Apache to prompt for the private key password every time the web server restarts. For testing purposes, it becomes much easier if the private key is not secure. However, for real applications, the private key should be secured, and the server administrator would need to enter the private key password every time the Apache started.

The server needs to know its certificate and private key location. The next section will describe the adjustments made in the Apache configuration files to implement SSL as part of the communication protocol in the server-side.

4.4.1.4 III. Apache - SSL configuration

The Apache configuration files that required modification were only two: httpd.conf and httpd_ssl.conf. The changes refer to the name of the server, which port to listen and the modules it has to load. The complete updates of the files can be seen in Appendix B.

httpd.conf

Listen **127.0.0.1**:80
ServerName localhost:80

httpd_ssl.conf

Listen **127.0.0.1**:443
ServerName **127.0.0.1**:443

SSLCertificateFile **"C:/Apache2.2/conf/ssl/crt/server.crt"**
SSLCertificateKeyFile **"C:/Apache2.2/conf/ssl/key/server.key"**

The IP of the server was updated, and the paths to both the server's certificate and its private key were included. By default, port 80 is the one the HTTP server is listening to. As HTTPS protocol is the desired protocol, port 443 needs to be used. Port 443 is the number assigned to the SSL application on the server. The *SSLCertificateFile* and *SSLCertificateKeyFile* indicate the location of the the server's certificate and private key, respectively.

One should note, that for future scenarios, new certificates need to be generated, and these files need to be updated, changing the server IP and the paths to the certificate and private key files.

At this point it was possible to test if SSL was working properly in Apache through the browser (simulating a client). When requesting the localhost web page using the "https://" prefix in the URL, the server certificate was sent to the browser, as expected.

Simultaneously, the HTTP client was being developed and tested. The details regarding these events will be described later.

4.4.1.5 IV. Installation of Tomcat

The installation of Apache Tomcat was necessary because Apache HTTP Server can not run Java Servlets, unlike Tomcat. "Apache Tomcat is an implementation of the Java Servlet and JavaServer Pages technologies." [46]. Tomcat has the ability to include SSL support, but its configuration is far more complex than Apache's.

The installation of Tomcat version 6.0 proceeded with minor problems, that were easily resolved.

4.4.1.6 V. Apache - Servlets configuration

In order to have both Apache and Tomcat working, Apache needed to forward the requests for servlets to Tomcat, so that Tomcat would process them itself. After a lot of work and research, the final setup consisted of the following updates.

httpd.conf (Apache)

LoadModule jk_module modules/mod_jk.so

Also, a new configuration file was included in Apache and named tomcat.conf. This file includes the commands for Apache to forward the requests to Tomcat.

The file tomcat.conf can be consulted in Anexo B if desired.

A simple test through the browser proved that a request to Apache calling an example servlet stored in the "C:/Tomcat6.0/webapps/examples/WEB-INF/classes" works properly, and the connection between Apache and Tomcat was implemented correctly.

Now that the setup of the environment was complete, the servlet development could start.

4.4.1.7 VI. Servlet Development

The servlet is the server application. The servlet's development was done in parallel with the client application, so the developer could have constant feedback allowing to immediately correct program flaws, on both ends.

However, to simplify the description of the implementation, the two applications will be described separately, and tests involving both parts will be mentioned as needed.

For the servlet development, Eclipse was the used Integrated Development Environment (IDE). To program the Java servlet, a Java Development Kit (jdk1.6.0_05) had to be installed and an external servlet library imported (servlet-api.jar).

It is important to note that the first step of the communication protocol is independent of the servlet. The establishment of a secure connection using the SSL protocol is done by the Apache server. If the establishment of the connection with the client is successful, the Java servlet will run as supposed to and respond to the client's request. However, it is always the client the one who makes the first request, initiating the message flow.

To clarify which part of the protocol is the servlet's responsibility, Figure 4.5 shows the messages that the servlet sends to the client and the procedures it executes.

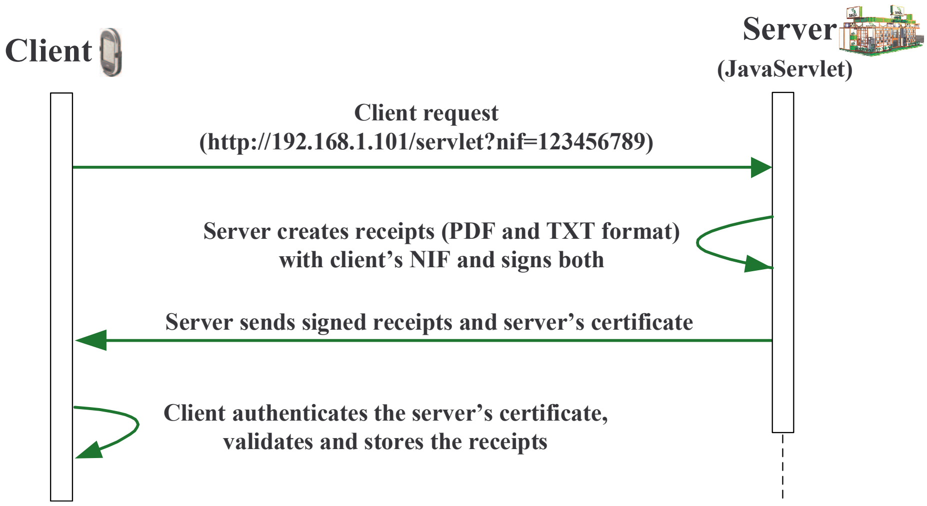

Figure 4.5: Close up to the JavaServlet procedures.

The development of the Servlet was done in six stages related to the creation and sign of the receipt, and will be described next:

- the creation of the PDF receipt - createPDF(),
- the signing of the PDF receipt - signPDF(),
- the sending of the PDF receipt - sendPdf(),
- the creation of the text receipt - createDoc(),
- the signing of the text receipt - signDoc(),
- the sending of all three files: PDF receipt; text receipt; text receipt signature plus the server's certificate.

The format in which the receipt file would be sent to the client was thought in two distinct ways. In a PDF (Portable Document Format) file, or through a data stream that could simultaneously be printed in the user's device screen and saved in a text file. Both ways were implemented.

The previous stages were implemented in 4 classes that constitute the Servlet, which are: Servlet.class, Receipt.class, Sign.class and Send.class.

The following Figure 4.6 represents the classes relationship.

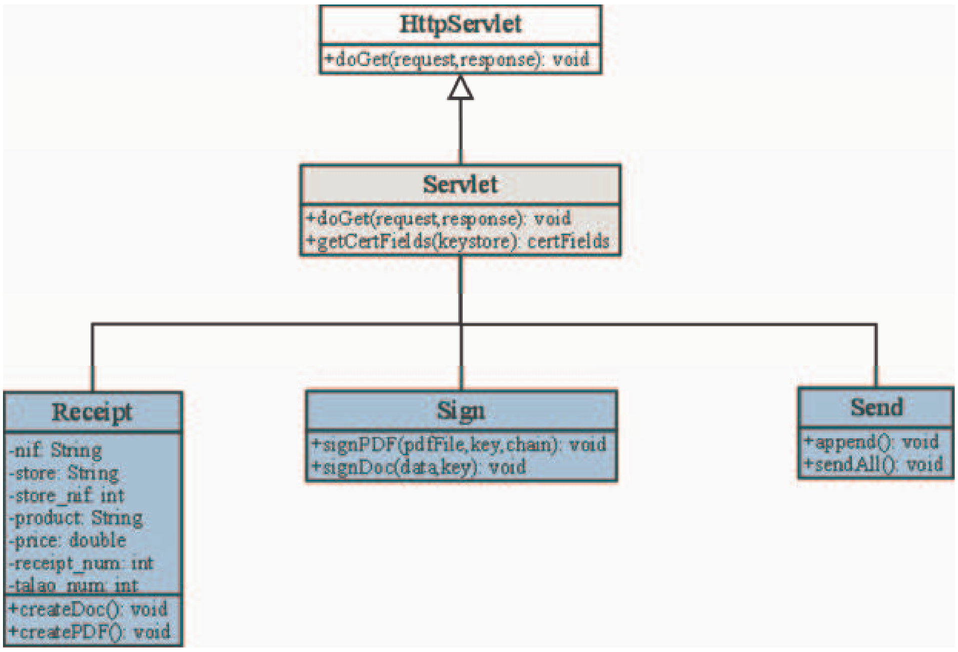

Figure 4.6: The Servlet application classes diagram.

4.4.1.8 Receipt.class

The creation and signature of the receipt is the most important procedure of the Servlet. It is considered that the application has access to all the necessary information to create a valid receipt, according to the requirements imposed by the Department of Finance. However, for testing purposes, the required data for creating a receipt was passed to the responsible method as a parameter.

For the Servlet to create dynamically a receipt (which included the client's NIF) in PDF format, a specific library was required, the iText. "iText is a library that allows the generation of PDF files on the fly. The iText classes are very useful for people who need to generate read-only, platform independent documents containing text, lists, tables and images. This library is especially useful in combination with Java(TM) technology-based Servlets." [47]. The method used for this purpose was named *createPDF()*and was integrated as a method of the class *Receipt.class.*

The creation of the receipt as a "txt" document did not require any external libraries. It was visually done the simpler way possible, so that the user could easily identify the important information of the purchase. The method used for this purpose was named *createDoc()*and was integrated as a method of the class *Receipt.class.*

4.4.1.9 Sign.class

In the process of digitally signing a document, additional information is added to the given document, called a digital signature.


KAMI
Reg. CRC Porto n.7634
Call Center Kami 707 20 34 26

Contribuinte Nr. 503952427

TALAO VENDA : 675643 SQ: 23464

Recibo: 1
873264

Anima Mundi 13.45
TOTAL 13.45

Talão indispensável para troca num prazo de 30 dias. Produto em bom estado e com embalagem original.

Figure 4.7: The content of the server keystore

The digital signature is calculated using the contents of the document and a private key, and it depends on the contents of the document, the algorithm used for signing and the private key used to perform the signing. The digital signature allows the recipient to check the actual origin of the information and its integrity.

The signature of the PDF file, created by the *createPDF()* method was done using the iText tool. This tool uses the private key to sign a document. For that a Java keystore containing both server's private key and certificate is required.

Adding a certificate to a keystore for trusted chain validation does not constitute a problem. However, the scenario changes when it's required to add an existing certificate and the corresponding private key to a Java keystore.

When a Java keystore is created using the keytool command, it is also automatically created a certificate and a private key associated to it. To create a Java keystore with an existing private key is a little more complicated.

The solution to this problem was found in a java program developed and made available by Kieran Shaw [48] named *KeystoreKeyImporter()*. This class can be used for storing the private key and associated certificate, in the PKCS#12 format.

To convert the existing CRT server's certificate and the private key to a PKCS#12 file the following instructions were executed.

» openssl pkcs12 -export -in **server.crt** -inkey **server.key** -out **server.pkcs12** -name server -passout pass:<storepass>

The next instruction import the PKCS#12 file to the server keystore using the *KeystoreKeyImporter()* java program :

» java -classpath . KeystoreKeyImporter **server.pkcs12** <storepass> **server.keystore** <storepass>

The password of the server's keystore and the private key password must be equal,

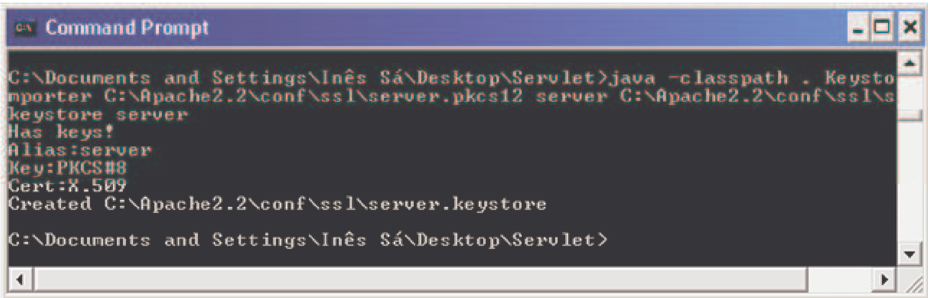

Figure 4.8: Importing the PKCS#12 file to the server keystore

otherwise, the Servlet will throw an exception when trying to get the private key from the server's keystore.

The import was successful and the content of the PKCS#12 file is listed in Figure 4.9. After having both the server's certificate and the private key stored in the server's keystore (the external library, bcprov-jdk14-138.jar, was added to the classpath to fully support the iText tool) a new signed receipt was created.

The signature of the "txt" receipt was obtained through the following steps:

- the calculation of the hash-value (message digest) of the document (with a cryptographic hash function - SHA1),
- the encryption of the message digest with the server's private key.

In the first step of the process, the message digest (a sequence of bits with a fixed length) of the receipt was obtained by applying a cryptographic hashing algorithm (SHA1) to the document.

In the second step of digitally signing the receipt, the hash-value of the document was encrypted with the server's private key (stored in the keystore) and thus an encrypted hash-value, also called digital signature, was obtained.

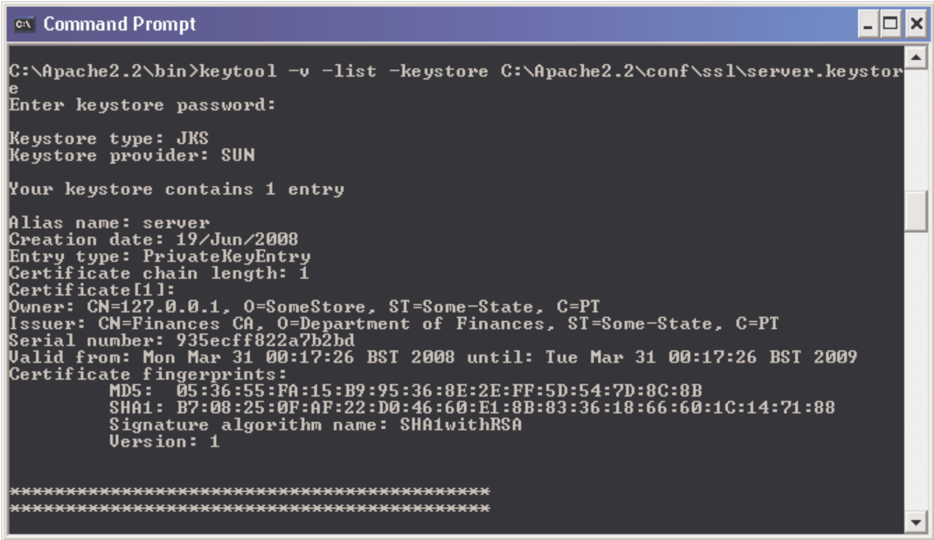

Figure 4.9: The content of the server keystore.

Figure 4.10: The signed receipt.

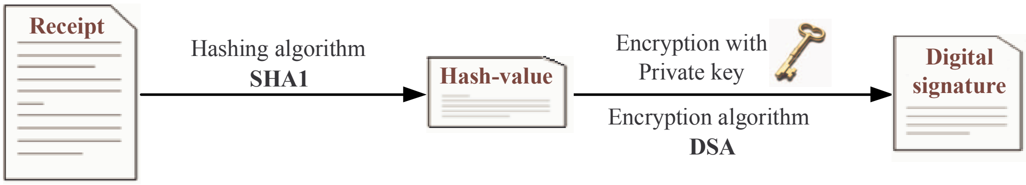

Figure 4.11: Digital signature process.

The digital signature was obtained using the *initSign()* method that takes the private key from the server's keystore, followed by the *update()* method which extracted the message digest of the document, and finally the *sign()* method that encrypted the hash-value returning the signature. This signature was saved into a file, so that it could be attached to the original receipt, and be verified by the client.

Although server's keystore did not present any problem for signing the PDF receipt, that wasn't the case when trying to encrypt the message digest with the private key stored in it. After a careful analysis of the keystore content and private key parameters, the problem was identified. The certificate's public key was RSA, but the algorithm specified to be used in the encryption of the hash-value was the DSA algorithm.

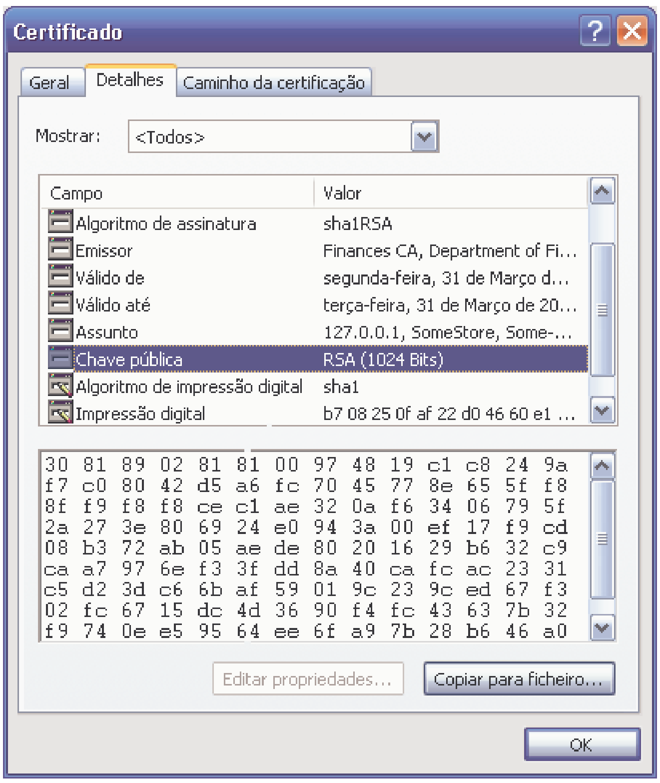

Figure 4.12: Server's certificate with an RSA public key.

The private key in the keystore had to be changed to DSA algorithm, and a new certificate obtained. That was achieved through the following instructions.

» openssl dsaparam -rand -genkey -out **server.key** 1024

» openssl gendsa -des3 -out **server_dsa.key server.key**

» openssl req -new -x509 -days 365 -key **server_dsa.key** -out **server_dsa.crt**

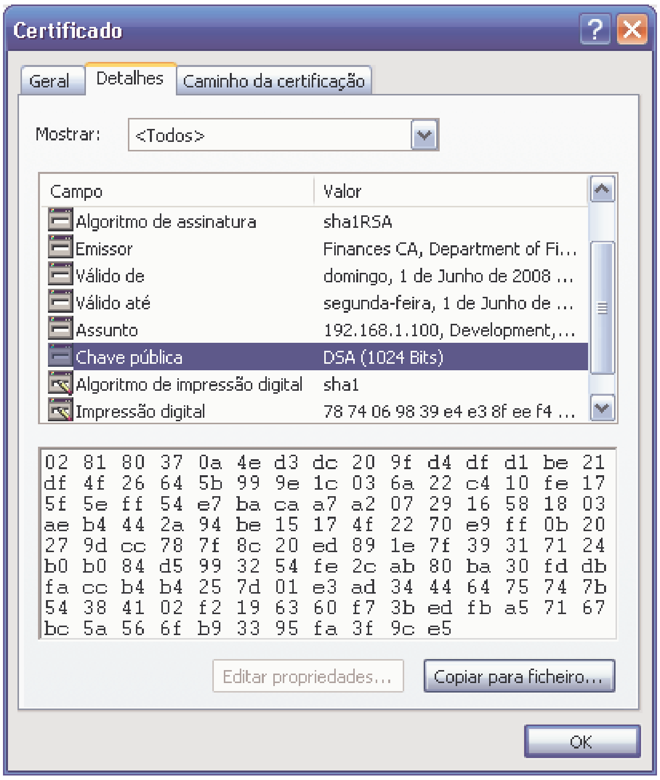

Figure 4.13: Server's certificate with an DSA public key.

Naturally, the new certificate and private key were added to the server's keystore by repeating the earlier steps.

4.4.1.10 Send.class

The final step of the Servlet development was to respond to the client's request with all the data required.

For testing purposes, the *sendPDF()* and *sendDoc* methods were developed, but the final Servlet application uses an integrated version of both methods, the *sendAll()* and also*append()* method. Only them will be described here. These methods integrate the *Send.class*.

The communication protocol defines that the server must send to the client a signed receipt and its own certificate. It was decided to send the server's certificate along with the receipts, this way, only one response would be given to the client. This was done appending all four documents into one: the server's certificate, the PDF and "txt" receipts and the digital signature of the "txt" receipt. Thus, the *append()* method was used to create one single file, and the *sendAll()* to send it as a data stream.

4.4.1.11 Client-side

For the client application, the first thing to be developed was a simple HTTPS Client to communicate with the already working server. This required the access to SSL keys and certificates with Java, which is done through keystores. Before the development of the HTTPS Client, a period of familiarization with the Java keystore concept was needed.

A keystore is a database of keys. Each private key in a keystore has a certificate chain associated to it, which authenticates the corresponding public key. A keystore may also contain certificates from trusted entities.

For the client application, the used Java keystore only needed to store the CA certificate in it. When the server sends its certificate for validation by the client in order to establish the SSL connection, the HTTPS Client will go to its own trusted keystore and verifies if there is a CA certificate to prove the server's certificate trusted chain.

Keytool is a key and a certificate management utility provided by the JDK. It is used to create keystores and to import certificates into them.

The next command shows how Keytool was used to install the CA certificate in the Java trusted keystore, cacerts. This is the default keystore that Java consults when authentication is needed.

» keytool -import -trustcacerts -alias **financesca** -file **ca.crt** -keystore **cacerts**

To verify that the certificate has been correctly imported, the following instruction was executed.

» keytool -list -alias **financesca** -keystore **cacerts** -storepass changeit

As mentioned earlier, the client application was developed simultaneously with the server application, so that tests could be performed. This was accomplished essentially in 5 steps:

- create a secure connection,
- request the receipt from the server,
- receive the PDF receipt and verify its authenticity
- receive the "txt" receipt, its signature and verify the authenticity of the "txt" receipt,
- receive all 4 files, save them separately and verify their authenticity.

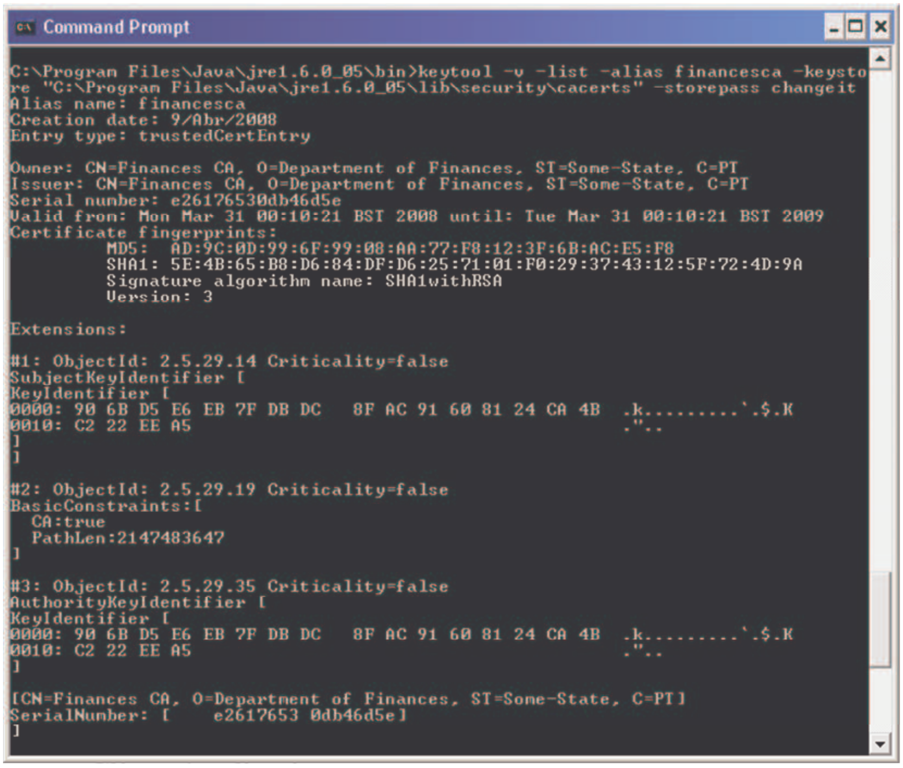

Figure 4.14: Importing a CA Certificate to the Java Keystore.

1. The first thing to incorporate was the SSL handshake. This was done using the classes available in jsse.jar to create an SSLSocket that connected to the desired host.

2. The communication begins with a GET request from the client to the server with the "https://" prefix. If the host in the URL of the GET request matches the Common Name of the server's certificate, the server responds with its own certificate, allowing the client to authenticate it with the CA certificate stored in the client trusted keystore. If the establishment of the connection is successful, the client will get the Tomcat webpage as response.

Regarding the protocol communication, it was decided that the store's authentication would be done after the client sent his NIF. This way, the client only has to make one request to the server, with his NIF as a parameter of the GET request.

After the client sends his request, it awaits for the response from the server. This response comes as a data stream, and the client needs to parse the useful information.

3. As the first receipt to be created in the server-side was a PDF file, the client was prepared to receive a PDF file and save it correctly, so the user could easily identify it as a receipt.

Although a PDF file may represent a smooth transaction from the paper receipts to the user, it presented a limitation, which was the need of a PDF reader in the client-side.

Thus, it became clear that a new alternative had to be presented. This alternative was to give the client application the ability to receive information in a format that allowed

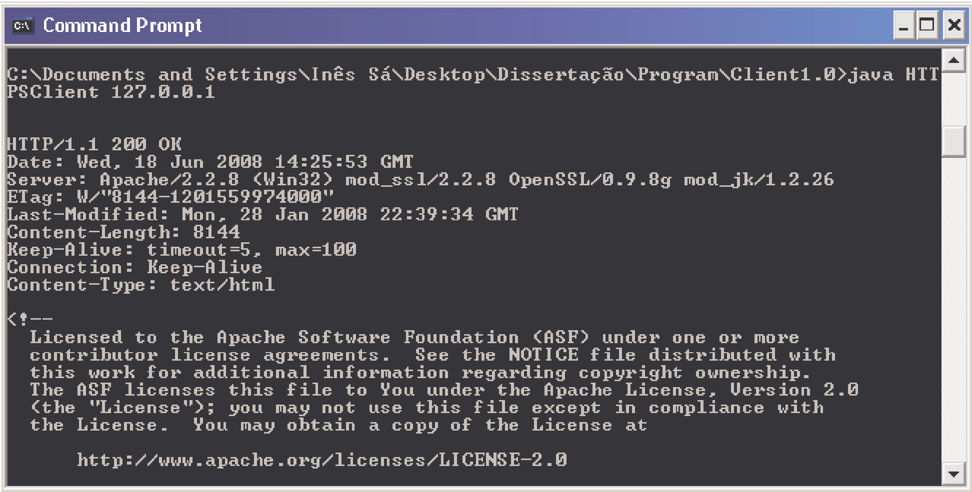

Figure 4.15: First test of HTTPS communication with the server.

it to present the receipt to the user in a simpler way, such as printed information on the terminal, or a text file.

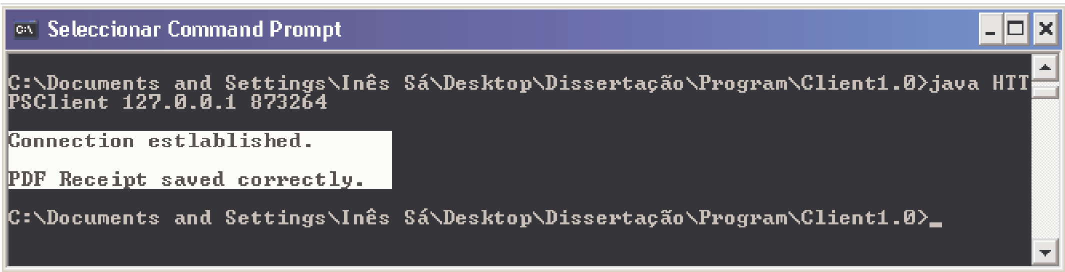

Figure 4.16: The client application running.

After correctly saving the PDF receipt, its authentication was done automatically by the PDF reader, as shown by Figure 4.17. It only required the CA certificate to be installed in the Trusted Root Authority.

4. The reception of the "txt" receipt was identical to the PDF file, the application is modified implementing the authentication of the receipt using the following methods: the *initVerify()* that takes the public key for verification; the *update()* which calculates the message digest from the receipt and the *verify()* that compares the hash-value calculated by its *update()* method and the decrypted message (obtained using the public key of the store's certificate) from the digital signature received. The *verify()* method returns true if the verification is successful (if the signature corresponds to the given message and public key), and false if not.

Figure 4.19 shows the message that is printed for the user, after the "txt" receipt being correctly received and successfully authenticated.


KAMI
Reg. CRC Porto n.7634
Call Center Kami 707 20 34 26

Contribuinte Nr. 503952427

TALAO VENDA : 675643 SQ: 23464

Recibo: 1
873264

Anima Mundi	13.45
TOTAL	**13.45**

Talão indispensável para troca num prazo de 30 dias. Produto em bom estado e com embalagem original.

Documento certificado por 192.168.1.100.
Digitally signed by 192.168.1.100
Date: 2008.06.19 23:11:42 BST
Reason: Loja Kami
Location: Porto

Figure 4.17: The PDF receipt verified.

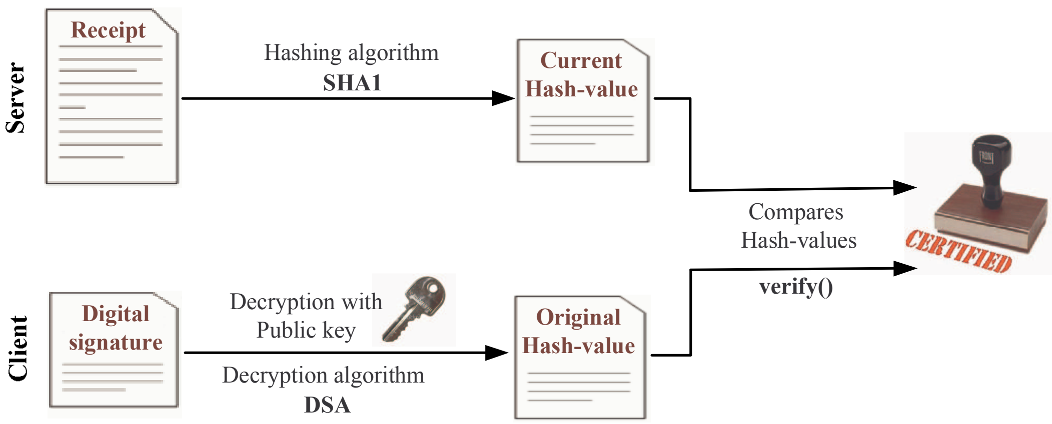

Figure 4.18: The authentication of the receipt.

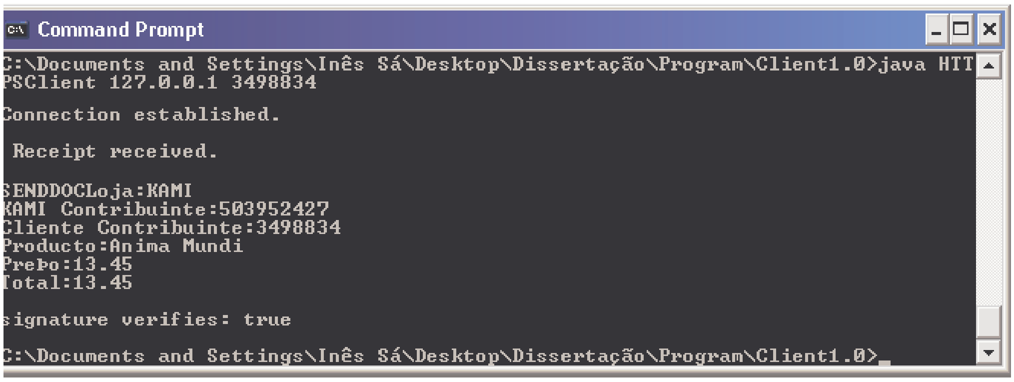

Figure 4.19: The authentication of the receipt.

5. The incorporation of the previous methods for receiving and verifying different documents concluded the localhost client application development.

4.4.2 Remote PC with wireless

The migration of the client application from the localhost to a remote PC with wireless occurred smoothly. The only modification needed for the system to work correctly was to generate a new certificate for the server. It is important to keep in mind that the host requested by the client must be the one for whom the certificate was issued, and because of that, every time the host IP or server name changes, so must the server's certificate.

For the issuing of the new certificate the same CA was used, avoiding to import a new trusted certificate to the client keystore.

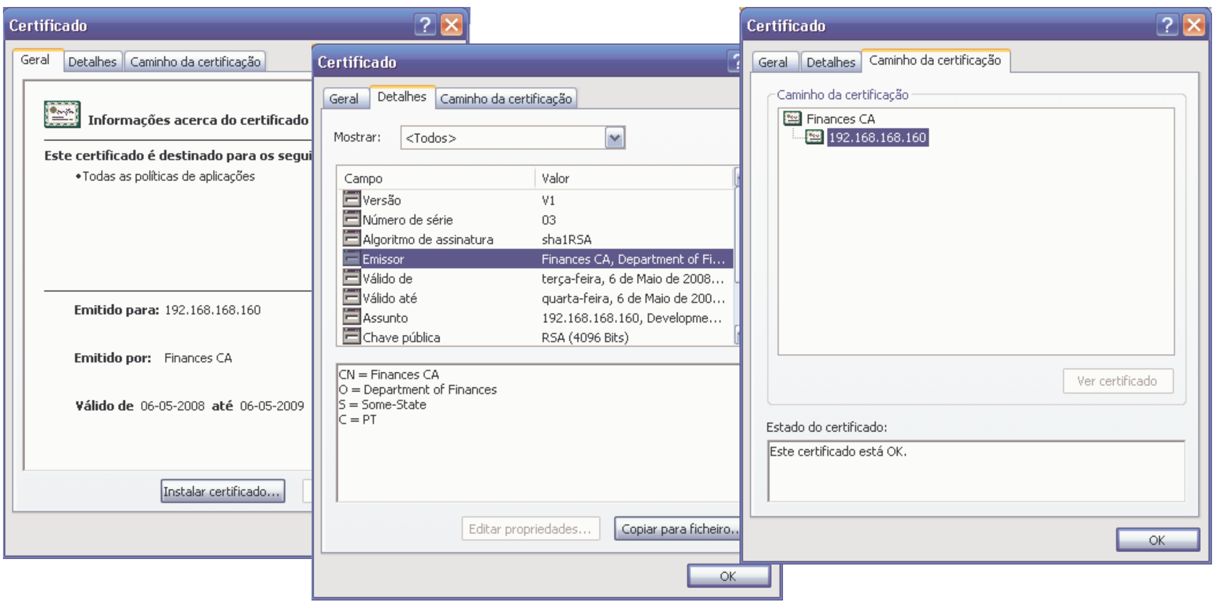

Figure 4.20: Server's certificate for wireless connection.

To optimize the client's application, and make it completely portable, a new keystore was created, containing the CA certificate. This way, when migrating the application to another computer, the trusted keystore can also be taken as part of it, without having to rely on the Java default trusted keystore.

However, this requires that the location of the keystore can be passed as a parameter when running the client's application. Next, it will be shown the creation of a keystore and the new parameters used when running the application.

» keytool -genkey -keystore **keyStore**

» keytool -import -alias **financesca** -file **ca.crt** -keystore **keyStore**

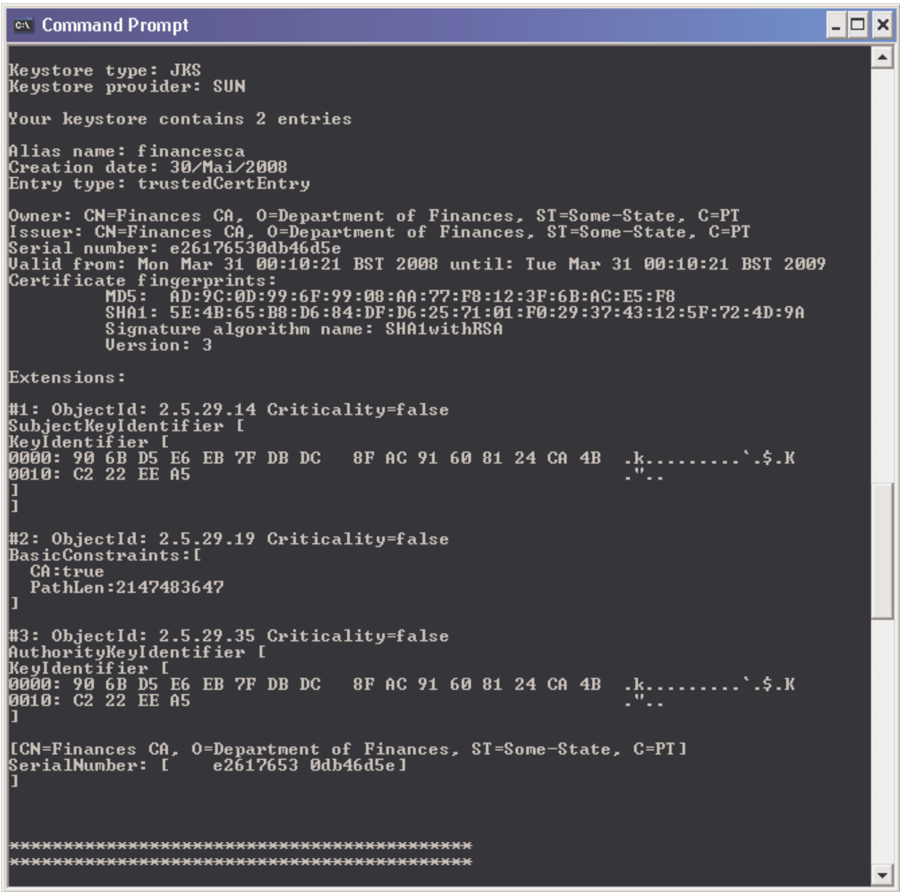

Figure 4.21: Client's keystore with CA's certificate.

When creating a keystore using the Keytool, a private key and a certificate are automatically created and associated to the keystore. To the client, this does not present a problem, unlike adding the server's certificate to the server keystore, for signing purposes (as mentioned before).

»java **-Djavax.net.ssl.trustStore=keyStore -Djavax.net.ssl.trustStorePassword=client** -Djava.protocol.handler.pkgs=com.sun.net.ssl.internal.www.protocol **HTTPSClient.jar**

The "-Djavax" arguments indicate the path to the keystore and its password, respectively. The parameter that follows them defines the protocol handler. This parameter is only needed if the security provider wasn't defined while developing the application. Explicitly using this parameter is useful if the keystore provider needs to be changed.

Sun's standard version of the Java Runtime Environment comes with a default provider, named SUN. Other Java runtime environments may not necessarily supply the SUN provider.

The SUN provider package includes:

- An implementation of the Digital Signature Algorithm (DSA)
- An implementation of the MD5 and SHA-1 message digest algorithms.
- A DSA key pair generator for generating a pair of public and private keys suitable for the DSA algorithm.
- A DSA algorithm parameter generator.
- A DSA algorithm parameter manager.
- A DSA "key factory" providing bi-directional conversions between DSA private and public key objects and their underlying key material.
- A "certificate factory" for X.509 certificates and Certificate Revocation Lists (CRLs).
- A keystore implementation for the proprietary keystore type named "JKS".

This means, that at this point, the application is dependent of the SUN provider to access correctly to the keystore, and to recognize it as a valid file. The tests performed in two different operating systems (Windows and Macintosh) didn't present any problems.

4.4.3 Remote Tablet with wireless

It is important to remind that the purpose of the implementation was to develop a means to validate the architecture, but also that the creation of such an architecture came from the primary goal to develop a mobile mechanism for exchanging receipts in commercial spaces. Thus, it was only natural to develop an application that could run in a mobile device. In order for that to happen, the communication system could either be wireless or Bluetooth (infrared technology was rejected due to its limitations). The Nokia Internet Tablet (N800) was the chosen mobile device.

Although both applications work perfectly in the previous scenarios, the implementation on a Tablet was expected to introduce some complications, because it involves a completely different system.

The first test to be performed with the N800 was through the browser for receiving the signed PDF receipt. The CA certificate had to be installed in the browser for the client to validate the server certificate as a trusted certificate for the SSL connection.

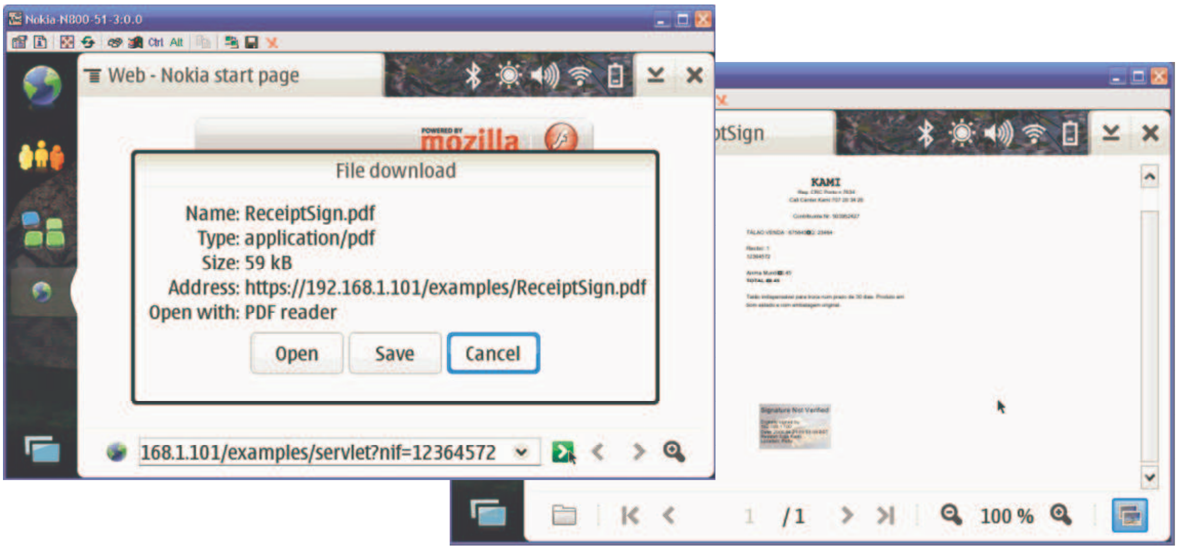

Figure 4.22: N800 receiving PDF receipt through the browser.

This test verified that the Tablet supported SSL protocol through the browser. For the client application to run some software installations were required. The steps that were taken are listed next:

- activation of the red-pill mode
- installation of the becomeroot
- installation
- installation of jalimo (JVM)
- implementation of the HTTPS client

The activation of the red-pill mode (mostly known as "taking the red-pill") is a procedure that activates additional settings such as: to show all installation packages, magic system packages, etc. This was required for the development to proceed (by performing the remaining installations).

The activation of the red-pill mode is achieved by accessing the Menu and selecting the following options:

Application Manager > Tools > Application Catalogue > New

Then, the user inserts "matrix" in the Web address field, presses the Cancel button and chooses the red pill

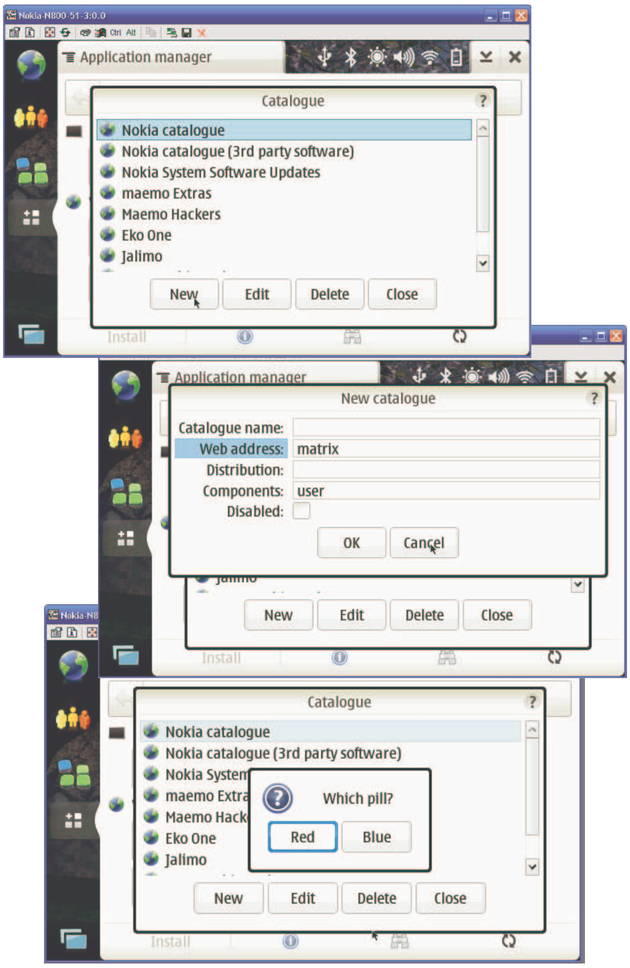

Figure 4.23: The Red-Pill activation.

When installing becomeroot, the osso-xterm was also installed. These packages were crucial for the developer to freely access root (by executing the command **sudo gainroot**) and test the application.

Java programs are run on, and interpreted by the JVM. Rather than running directly on the native operating system, the program is interpreted by the JVM for the native operating system.

Unfortunately, the Nokia Tablet doesn't have a JVM in the operating system. After studying some possible solutions of java virtual machines developed for Linux-based mobile devices, Jalimo seemed like the best option.

Jalimo is a java-like environment for Linux mobile devices. The aim is to pack and tailor existing JVM-related projects, as well as to develop APIs and tools for easy integration and

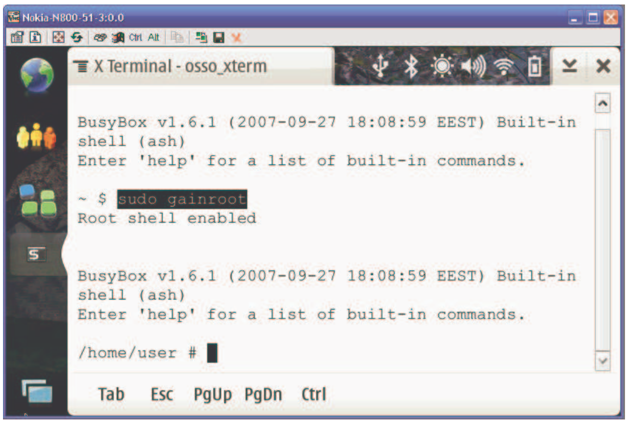

Figure 4.24: N800 X Terminal.

development for the target platforms. Jalimo Java Environment is based on CacaoJVM, JamVM, GNU Classpath. Its primary targets are Maemo and OpenMoko platforms [50].

After testing a simple HTTP application on the Tablet to assure that the Jalimo was working, it was time to run the developed HTTPS client application.

At this point some unexpected problems started to appear. Although every important security libraries and the trusted keystore had been passed to the client device, the SSL handshake could not have been done between the server and the mobile client. After a careful analysis of every printed error and possible cause for the encountered problems, it was considered that the cause for the failure of running the application could reside in the fact the systems had different package providers. As mentioned earlier, the client application was based on the SUN provider, so was the creation of the keystore. One possibility was for the Tablet to have installed the GNU Crypto provider and in that case, the keystore wouldn't be recognized and the SSL handshake would fail. In the brief period of time left to complete the project, the following modification was performed.

A new keystore was created with GNU provider. This was done in a Linux operating system. The keystore location was added to the application and the GNU provider specified in the program, but the problem couldn't be resolved.

There are strong suspicions that the problem is related to the GNU provider not being the provider used for the development of the client application (SSL support is linked to GNU Crypto security providers included in GNU Classpath). However, the combination of the time limitation and the lack of experience in operating with Nokia Internet Tablet (N800), has driven the development to the fulfillment of other system requirements, such as communication through bluetooth technology.

4.4.4 Remote Tablet with bluetooth

Up to this point, all the tests were performed through the 802.11 wireless interface. Although it wasn't possible to run the HTTPS client application in the Tablet (due to setbacks with the SSL protocol that couldn't be resolved in useful time), it was still important to test an HTTP client application through a bluetooth connection in the Tablet. This would not only reduce the client application limitations (by becoming wireless independent) but also increase its perspectives of usability.

The Personal Area Network (PAN) profile provides a means for applications to use IP and other networking protocols over Bluetooth. To have Bluetooth-PAN working in the Tablet, the maemo-pan application was installed. After performing the following instructions of bluetooth configuration in the x-terminal, the application was ready to connect to the server using internet protocol (IP).

```
» sudo gainroot
» pand -Q10
» pand -l
Connect to network "Bluetooth-PAN"
ifconfig bnep0 10.0.0.2
route add default gw 10.0.0.1
```

The first instruction performs a 10sec search for the HCI address of a NAP and then connects itself to it. After the connection is accepted by the server, the second instruction allows to see if the connection has been established and a virtual network interface named bnep0 created. The instruction shows "bnep0 00:16:38:C4:64:23 PANU". In the next step, the client connects to the Bluetooth-PAN network.

The virtual network interfaces (the endpoints of the Bluetooth connection) on both sides needed to be configured. The assigning of IP addresses to the Bluetooth interfaces was straightforward. For testing, it was assumed that the server would have the "10.0.0.1" IP address and the client the "10.0.0.2". The procedures are shown in the next figures.

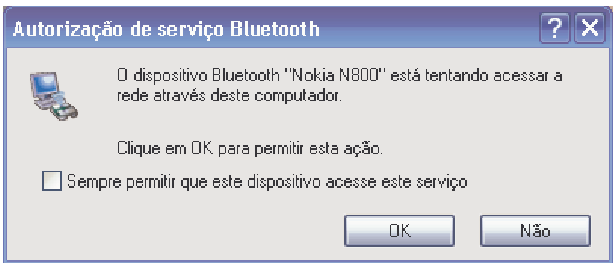

Figure 4.25: Request to server accept Bluetooth connection.

In the server-side, the Bluetooth application of the device already included the PAN profile, thus, the only modification required was to change the IP of the Bluetooth Network. Since the interface IP changed, new certificates had to be generated, and the Apache

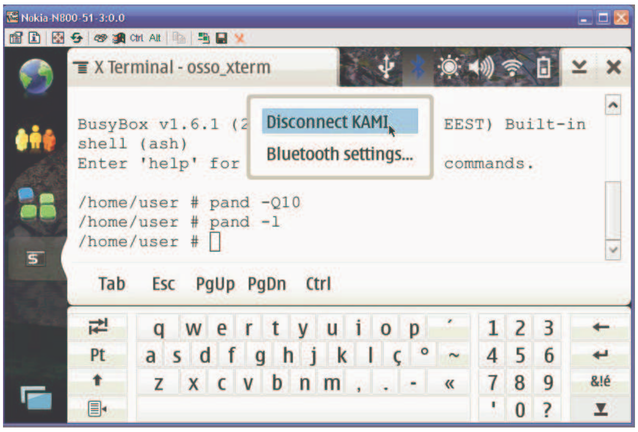

Figure 4.26: N800 connected to KAMI server through Bluetooth.

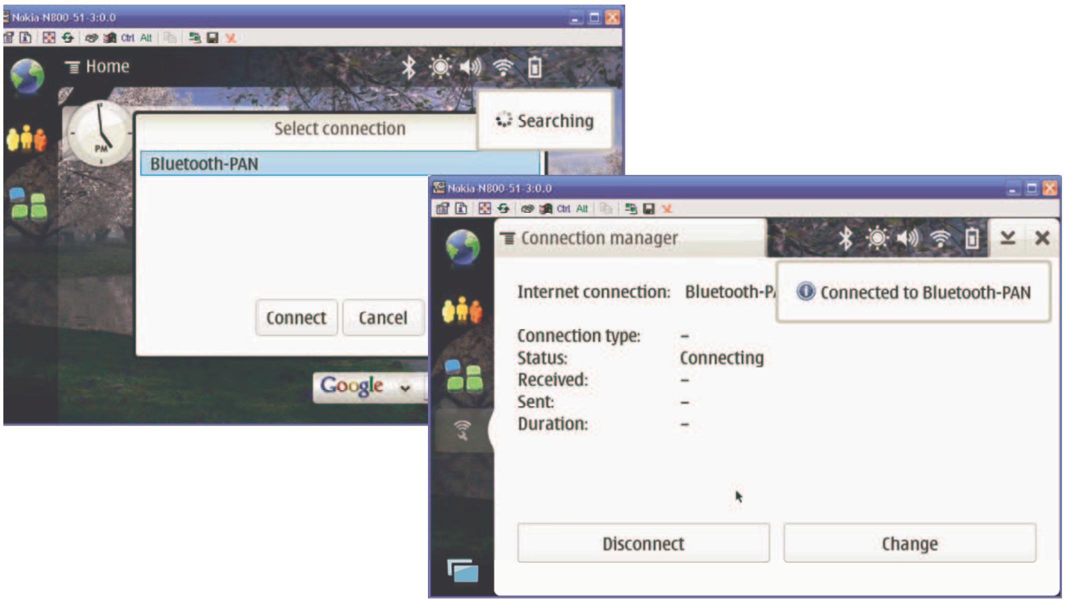

Figure 4.27: The Bluetooth-PAN connecting.

configuration files updated.

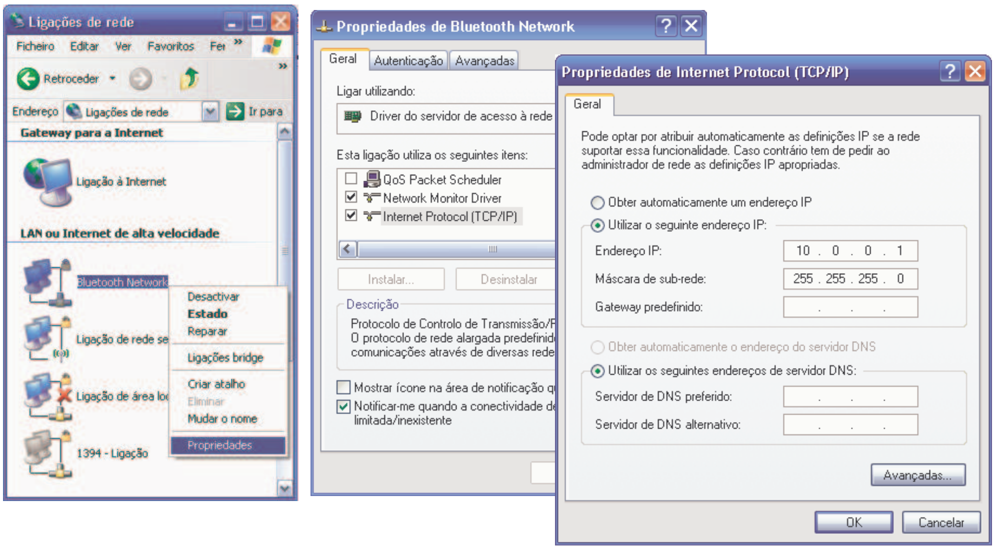

Figure 4.28: Configuration of the Bluetooth interface.

After having both the client and the server environment prepared to accept IP requests over Bluetooth, two tests were performed. First, the receipt request was sent by the browser using the SSL protocol. Second, the HTTP client application used in the previous scenario (remote tablet with wireless) was tested. Both of them were successful.

The following chapter will contain a discussion of the results and a conclusion that presents what was achieved.

Chapter 5

Analysis

5.1 Introduction

This chapter is structured in four main sections. Section 5.2 has a short review on the security of systems, since security is one of the major concerns regarding our system. Section 5.3 analyzes the security measures that were used in the implementation and their relationship with the fulfillment of the system requirements. Section 5.4 presents alternative technologies and tools that could have been used for the system implementation, and how they could validate/affect the present architecture. Finally, section 5.5 introduces a number of evaluation categories that were considered important to perform a complete system analysis in the future.

5.2 Security

Such a thing as a 100% secure computer system does not exist, and probably never will, but by being security-aware throughout the development process, one may reduce system vulnerabilities. Focusing on a security oriented development, many pitfalls may be avoided.

There are plenty of threats and vulnerabilities that need to be handled, even when developing a small system. Although the system being developed is relatively small, its development needs to observe the same principals the development of a bigger one does. This happens because bigger applications can be exposed to the same threats as the smaller ones.

When implementing a security system, all data networks deal with the following main elements [19]:

- Hardware, including servers, redundant mass storage devices, communication channels and lines, hardware tokens (smart cards) and remotely located devices (e.g., thin clients or Internet appliances) serving as interfaces between users and computers

- Software, including operating systems, database management systems, communication and security application programs
- Data, including databases containing customer - related information.
- Personnel, to act as originators and/or users of the data.

In order to prove the security of a protocol, five steps are suggested by [55]:

1. Specification of the communication protocol
2. Definition of goals, including the goals of the protocol and the adversarial goals.
3. Statement of assumptions, assumptions of the underlying problems, constraints of the user interactions, etc.
4. Description of the protocol
5. Understanding of how the protocol meets its goals regarding security.

The first four steps were done along the thesis, this chapter intends to approach the last one.

5.2.1 Security Analysis of the Implementation

5.2.1.1 Data Validation

Although two systems are interconnected and mutually authenticated, there is no guarantee that input from one of them hasn't been corrupted during transmission. The best strategy for dealing with input and output is to allow only input explicitly defined and drop all other data. Input should, therefore, be assured to be both appropriate and expected [52].

The data input validation is done by the customer, when he receives the receipt. The customer visually confirms the receipt's content, making sure the information in the receipt corresponds exactly to the purchase.

5.2.1.2 Safety Measures

Any sufficiently complex system has failure modes. Failure is unavoidable and should be predicted. However, the security problems related to failure are, in fact, avoidable.

Any security mechanism should be designed in a such a way that if and when it fails, it should fail to a state that rejects all subsequent requests [52].

The system has safety measures that we proceed to explain.

For securing the communication channel between the client and the server, the SSL protocol was used, preventing attackers from listening, copying or altering the information sent. When the client isn't able to authenticate the server in the SSL handshake, the connection isn't established and the client security remains intact.

In case of a communication failure, neither server nor client incur in a security risk. If the receipt wasn't received by the client, a new request has to be made, and therefore, a new SSL connection established. However, the server must have the means to detect that a failure occurred. If the receipt has already been created, the server must delete it and prepare itself to issue a new receipt.

Also, detailed error messages are never exchanged between server and client, but rather stored in a system log. The messages may contain information or weaknesses that can be exploited by an attacker.

Another safety measure of the system is taken when the server's authentication or receipt validation is incorrect (after the client receives the receipt and the server's certificate), in which case, the customer is alerted with a message, so that appropriate actions can be taken.

5.2.1.3 Complexity

Often the most effective security is the simplest one.

Complexity increases the risk of problems concerning security. A complex design is never easy to understand, and therefore, is more likely to include subtle problems. Complex code tends to be harder to maintain as well.[52]

To avoid this kind of trouble, the implementation was as straightforward as possible in the software development process. The number of exchanged messages was kept to a minimum: only one client request is made and one response from the server is given.

5.2.1.4 Trusted Components

Using and reusing trusted components makes sense both from a resource stance and from a security viewpoint.

This was accomplished using the same server's certificate to establish the SSL connection and to validate the signed receipt, which means that the server uses the same private key to encrypt and to sign data. This way, only one certificate is issued by the Department of Finance for each store, and it is used for every security measure that involves the store's authentication.

This measure also provides a means for the customer to be assured of the authentication of the connection, and the certification of the store by the Certificate Authority.

The client's request is made to the server's IP address. By comparing this IP with the one on the server's certificate, the client knows that the receipt he received came from the store he requested. One should note that the authentication performed by the client can be done even if the identifier of the store is not its IP. The only requirement is that the unique identifier of the store used in its certificate(the common name) is the same used to establish a trustful relationship. This way, the client has the guarantee that the received signed receipt, was signed by the same entity with whom he established connection and to which he sent his NIF.

The reuse of trusted components was also done at the software level, since cryptographic libraries of a certified provider (SUN) were used.

The installation of the Apache Tomcat, which is an application server developed by a trusted third party, and tested by multiple users, reenforces the security chain that encloses the system.

Java also supports security. The Java language incorporates features that promote the production of robust systems by controlling access to memory. These features include: strong typing, structured memory access and automatic garbage collection. Security is further enhanced through the provision of support for structured error handling. Whenever an error is detected, an exception is raised. Any exceptions that are not handled within the application, eventually propagate to and are handled by the system's default handler. Class files are checked for integrity prior to their execution and extensive runtime checks are performed as they execute [52].

5.2.1.5 Best practices

Best practices require that regardless of how the certificate status is maintained, it must be checked whenever one wants to rely on a certificate. If not, a revoked certificate may be incorrectly accepted as valid. This means that to use a PKI effectively, one must have access to current CRLs.

The CRL component presented in the architecture is the last stop in the store's authentication. A certificate revocation list (CRL) is a list of certificates (or more specifically, a list of serial numbers for certificates) that have been revoked or are no longer valid, and therefore should not be relied upon. It is important to introduce such an element in the architecture, because a certificate can be revoked before its validity date has expired.

The consultation of these lists was thought to occur after the client receives the server's certificate and his receipt. This allows one process (communication with CRL) to be independent of the other (communication with the server). These lists can either be in a specific consultation server, or be downloaded by the client to its own system. The disadvantages of the CRL scenario presented in the architecture will be discussed in the *"Implementation Alternatives"* section.

5.3 Overall Considerations

Apart from the technical security measures of the implementation mentioned above, there are some considerations that are worth mentioning.

An overall vision on the development of the project shows that different approaches of the initial problem could have led to completely different solutions.

It is important to note that although the final step of the implementation was to test the application in a mobile device, the development of the client application wasn't made having as top priority its implementation on a mobile device. Due to this, the application may not be completely suitable for a mobile environment.

In a subsequent and better version of the client application, J2ME should be used, since it provides a robust and flexible environment for applications running on mobile and other embedded devices such as mobile phones and personal digital assistants (PDAs). Applications based on Java ME are portable across many devices, yet leverage each device's native capabilities. J2ME has been carefully designed to strike a balance between portability and usability. The reason why this technology wasn't used from the beginning of the implementation is because the goal wasn't to provide a full-feature client application but to do it in a integrated test system (communication system) which purpose was to study how a receipt exchange infrastructure works with the architecture purposed in *Chapter 3*.

It is thought that the implementation accomplished its purpose of validating the system's architecture. Although there were some implementation complications, they were not related to the architecture but with the technology decisions made in the development process. The architecture is designed in a fashion that assures both security and authentication in several different ways (secure connection, user confirmation, digital signature, user validation, etc).

Although the architecture should be as independent as possible of the implementation, some of the architectural decisions can't help to impose or limit the available choices for the implementation. For example, the fact that the architecture forces the user to confirm the receipt content makes the use of SmartCard unfeasible, as explained in section 5.4.4.

5.3.1 Digital Certificates

It is important to understand that the creation of a secure system of digital receipts is not only to protect the client from fraudulent behavior of the store, but also to provide means for the Finances Services to have a control mechanism for both client and store's declarations and requests of refundable VAT (value add tax). This subsection intends to provide a short discussion on:

- how the store's authentication and repudiation are done
- what types of frauds can be committed by both store and client
- the relevance and role played by digital certificates and their Common Name.

The digital certificates used in the architecture are essential because they prove the authentication of an entity. The only requirement is for the Common Name (CN) to be unique for each store. The CN binds a certificate to a particular store. If two stores have the same Common Name in different certificates, the user has no way to know if a signed receipt was done by the store he intend it to. As for the Finances Services point of view, the receipt is still valid. The signature of an incorrect entity doesn't mean an uncertified entity.

If the store is using a stolen certificate, the damage can only go so far, because it doesn't possesses the corresponding private key, therefore no signature nor encryption can be performed. Only the announcement of the store as being a certified one can be done.

However, if a store steals both the certificate and the associated private key of another entity, it will be able to perform all the actions of a certified and authenticated entity, when in reality it may not be.

If the store issues a receipt from a real purchase signed by a stolen certificate (which means, the store has made a undeclared transaction, and issued that transaction's receipt in the name of other entity), besides committing a crime, the store is avoiding to pay the corresponding VAT for the transaction. It also means that the client's receipt is not valid if a verification of the purchase is performed by the Finances Services.

Also, a thief store can issue false receipts from transactions that were never performed. This could only be advantageous if both store and client are committed to each other. This way, the client could declare to the Department of Finance purchases that had never been done, receiving the refundable tax that he's entitled by law.

This is where the Common Name plays its role in digital certificates. If the CN of the certificate is, for example, the name of the store, the thief can easily change its own name to be the same in the certificate, leading the client to think that this is a certified entity. Its signed receipts will be valid for the client, but not for the Finances Services. However, if the store's NIF is used as the CN of the certificate, both client and Finances Services can perform a true authentication of the store. The receipt issued by the store has the store's NIF, a simple comparison of both NIFs can verify if the receipt is being signed by the store's certificate or by a stolen one. As for the Department of Finance, if every receipt has the store's NIF that performed the exchange and the certificate that signed it, it is possible to do data crossing and validate every client and store declarations, and fraudulent behavior from both parts becomes a lot more difficult to occur.

In the next section some implementation alternatives will be presented in more detail.

5.4 Implementation Alternatives

The developed system could be supported by tools and technologies other than the ones used in the implementation. Alternative tools will be presented in this section, and a short discussion about each one of them will be made.

5.4.1 Store's Authentication

In the described implementation, the server sends its certificate and the signed receipt at the same time, after the client sends his NIF. This implementation assures the store's authentication, however it doesn't completely secure the client. In fact, the store's authentication is done at the end of all data exchange, and if the receipt or the store's certificate are not valid, the client is alerted but his NIF already has been broadcasted to the server.

The protocol was implemented this way in order to simplify the protocol keeping the number of exchanged messages to a minimum (only one request is sent) and because exposing the client's NIF is considerer a low level risk. If the store's authentication (the validation of the store's certificate) by the client is done before the client sends his NIF, the communication system is enclosed, leaving minimum space for leaks. This is important because this architecture could be used in a similar situation where the information requires maximum confidentiality, in which case the client can be certain of the server's authenticity before sending his confidential information.

5.4.2 Online Certificate Status Protocol

An alternative to using CRLs is the certificate validation protocol known as Online Certificate Status Protocol (OCSP).

OCSP is a scheme for maintaining the security of a server and other network resources. It overcomes the chief limitation of CRL, that updates must be frequently downloaded to keep the list current at the client end.

OCSP has the primary benefit of requiring less network bandwidth, enabling real-time and near real-time status checks for high volume or high value operations. When a user attempts to access a server, OCSP sends a request for the certificate status information. The server sends back a response of "current", "expired," or "unknown." The protocol specifies the syntax for communication between the server (which contains the certificate status) and the client application (which is informed of that status). OCSP allows users with expired certificates a grace period, so they can access servers for a limited time before renewing.

OCSP discloses to the responder that a particular network host used a particular certificate at a particular time. OCSP does not mandate encryption, so this information may also be intercepted by other parties.

5.4.3 Bluetooth

The integration of Bluetooth as a communication technology may alter the architecture.

The SDP (System Discovery Protocol) is used when a bluetooth device searches for other bluetooth devices to connect to. When a device is found, a PAN is created between them, and IP requests are sent to the server that the newly found device represents. In the developed implementation, the web server is configured to respond to client requests to a given IP.

An alternative design could consider the discovery process as part of the architecture, being integrated, for example, as part of the bluetooth secure connection.

5.4.4 SmartCard

SmartCard is an integrated circuit embedded into a plastic card the size of a credit card that has some features through which data can be transferred, stored and processed.

One of the most important qualities of a SmartCard is the possibility to protect data stored on the card against unauthorized access and manipulation. Due to cryptography, encryption and the internal computing power of the smart chip, confidential data can be exchanged and stored on the card in a way that prevents it from being read from the outside. SmartCards are protected against a wide range of security threats, which goes from careless storage of user passwords to sophisticated system hacks.

In addition to information security, smart cards achieve greater physical security of services and equipment, because the card can restrict access to all but the authorized user.

This technology provides means for effecting business transactions in a secure, flexible and standard way with minimal human intervention.

The advantages of SmartCard can be narrowed down to it being:

- highly secure,
- multi-functional,
- multi-services,
- user friendly,
- portable.

In spite of the benefits, SmartCard may not be the best solution for every scenario, some of its disadvantages are:

- small memory and processing capabilities,
- development complexity,
- communication speed (9600 baud),
- form factor (a plastic module as opposed to a card would be even more resilient),
- patent issues.

The existence of strong authentication and non-repudiation procedures along the security schemes, like the Public Key Encryption technique, allows the completion of a transaction between an end-user and a provider without the real time mediation of an authentication server [18].

The use of SmartCard to implement the present architecture was discarded due to its user interaction limitations. The protocol would have to be completely modified, and user visual validation of the receipt and store's certificate would have not been done.

A well known SmartCard application is the Citizen's Card. The citizen's card is a citizenship document that allows the owner to be present himself as a possessor of rights and duties. The advantage of such a document resides in the fact that it combines all the needed information for the public services in an electronic form (taxpayer card, National Health Service user's card, Social Security card and voter's card). It's a smart card with stored digital certificates (for electronic authentication and signature purposes). The chip may also hold the same information as the physical card itself, together with other data such as the holder's address [38].

However, the main disadvantages of smart cards remain, since the user would need a specific reader to interact with the stored information, for deleting receipts or transferring them to his personal home computer.

5.4.5 Distributed computing

Distributed computing could be used as part of the implemented solution for validating the architecture.

Distributed or parallel computing is a method of computer processing in which different parts of a program run simultaneously on two or more computers that are communicating with each other over a network. It requires a program to be segmented into sections that can run simultaneously. It also requires that each segment of the program take into account the different environments on which the different sections will be running.

The main goal of a distributed computing system is to connect users and resources in a transparent, open, and scalable way.

5.4.5.1 Remote Method Invocation (RMI)

RMI is short for Remote Method Invocation, a set of protocols being developed by Sun's JavaSoft division that enables Java objects to communicate remotely with other Java objects, creating distributed computing. RMI is a relatively simple protocol [52].

The Registry is a remote object that defines a useful set of classes. The Registry's main responsibility is to map a name to a remote object. It can be used to register an object to a simple name or to retrieve an object by name.

Java RMI is a simple, easily used mechanism for performing remote calls while hiding many of the lower level networking details. Once a reference to a remote object is obtained a remote method invocation looks exactly like a local method invocation. In fact Java RMI allows to pass full objects as arguments and return values as long as they implement the serializable interface. Java "serialization" allows to pass complex data structures over the network without writing code to parse and reconstruct them. It extends the Java proverb to write once run everywhere. RMI is a Distributed Object Model.

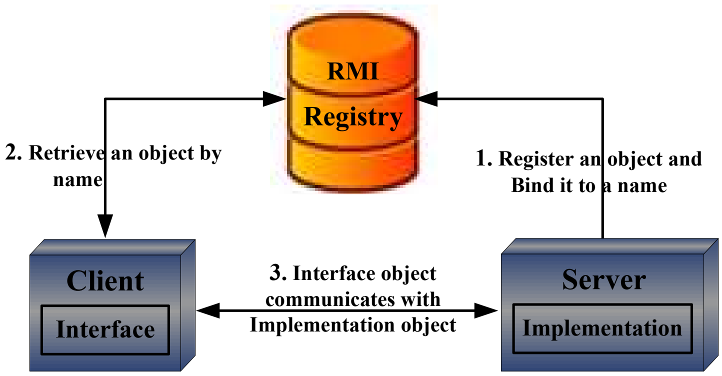

Figure 5.1: RMI Communication Process.

There are four required classes: an interface for the remote object, the remote object (that implements the interface) itself, the RMI server and the RMI client.

The interface is used by both the client and the server. It defines the methods that the remote object must implement that allow other objects to invoke to interact with it. The server creates an instance of the object which implements the interface and registers it with a particular URL in the RMI registry. The RMI client looks up the object on the remote server, casts it to the type of the interface and uses it like a local object.

These classes and interfaces provide a framework for an application to access objects residing on remote hosts by reference (not value, since the name service of the Registry uses URLs to identify remote objects). Once a client obtains a reference to a remote object, it can invoke the methods on these remote objects as if they existed locally. Any modification made to the object through the remote object reference is reflected on the server.

If RMI were used to implement the protocol for digital receipts exchange the secure connection could still be implemented. The version of RMI included in the Java 2 SDK enables the RMI developer to use custom socket factories for RMI-based communication. An application can export a remote object to use an RMI socket factory that creates sockets of the desired type (for example, SSL sockets). Using this technique, The RMI application could use SSL socket communication instead of the default socket communication.

The client receives an object instead of a data stream from the server, which allows him to handle each item separately and to create its own digital receipt. This has the problem of the server not being able to sign the data, unless he signs each of the object's attributes separately, returning each signature as an object as well. The RMI client requires a policy file to connect to the registry and to the HTTP server.

5.5 Issues for further evaluation

In the present days, the issuing of paper receipts no longer makes sense, from an ecologic, logistic and safety point of view. A goal of this thesis was to better understand the requirements and limitations of the development of a system for exchanging digital receipts from local transactions in commercial spaces.

Security has been the main focus of the analysis of this system. Unfortunately, a complete evaluation of the system wasn't possible. Nevertheless, the main evaluation categories were identified for future evaluation when the system is fully deployed, and a small discussion on how each one of them should be approached is presented next.

1. the security of the system against intruders, errors and interferences (malicious or unintentional). Fraudulent or undesired behavior is to be avoided.

2. the quality of service/performance

3. cost, associated not only with the choice of technology but also with the complexity of the algorithm and implemented protocol.

A summary of the evaluation features is presented in Figure 5.2:

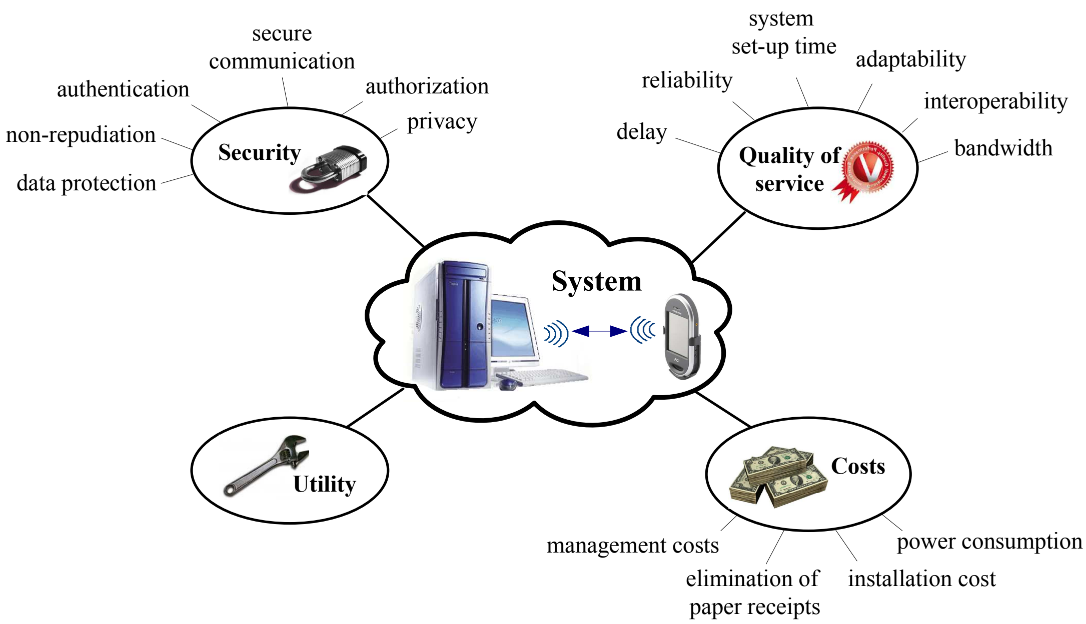

Figure 5.2: Evaluation Categories.

Some of the security services in the architecture were based on public key cryptographic techniques. These techniques require the distribution of key material through trusted channels, and the ability to authenticate material distributed through mistrusted channels. There is the need to secure data not only during the transmission but also while storing it. Data protection is the process of providing data confidentiality and privacy. For example, the authentication method built into Bluetooth authenticates the device and not the device's user or owner, so the user must consider the use of an application-layer security program and make sure that the device itself doesn't fall into the hands of a third party.

There is no known way for a system to be completely secure, but there are procedures to follow in order to guarantee security in some aspects, for example, encryption of the data and authentication of the owner using public and private keys.

5.5.1 Quality of Service (QoS)

The quality of service is equally related both to the protocol and interface layer.

A service typically provides some kind of QoS guarantees, which can be divided into four categories [24].

- Quantitative
- Qualitative
- Relative
- Statistical

There are other QoS parameters that can be identified in a system, such as:

- delay and delay variation - which could have different causes. For example in Bluetooth technology, the primary cause of delay is the amount of bandwidth available.
- bandwidth - the bandwidth requirements may be constant (e.g. peak bandwidth) or variable (e.g. average bandwidth).
- ordering - in the case of Bluetooth the order of data packets is preserved.
- reliability - its improvement reduces the effective bandwidth and increases delay. For data applications, reliability has priority over delay.

Specific to this system, here are some of the aspects that are important to evaluate:

- system set up time,
- power consumption,
- reliability,

- end to end delay - time it takes for a data packet to travel from source to destination,
- interoperability and adaptability.

5.5.2 Costs

Relatively to the cost of the system, the approach may get complicated because there are many factors to consider and it will always depend on which features should be prioritized. The costs have to be estimated considering not only the price of purchase but also the implications that such a system will bring both to buyer and seller.

The costs for the seller can be estimated according to the following factors:

- the price and installation cost of the system,
- the energy spent to supply the system,
- the customer attendance time,
- the quality of the internal management of the receipts and invoices,
- the paper receipts become obsolete and unneeded, therefore making the system user friendly and potentially making customers want to return to the shop,
- the improvement of the store image near the buyers, potentially bringing in more clients.

For the buyer the estimated cost involve other factors such as:

- the price of the device,
- how easy it becomes to exchange an item,
- improving management of personal finances,
- the elimination of paper receipts,
- the possibility of an automatic upload of the receipts when declaring taxes.

5.5.3 User Evaluation

Although a full user evaluation of the system wasn't performed , the following aspects in the evaluation were did consider:

- Data presentation. All the necessary information about the transaction and the authentication is presented to the user in a straightforward way. The status of the connection, the receipt and its authentication are displayed to the user. The interface is simple and clear.

- User operations. The use of the system and user operations are as easy as possible and kept to a minimum number of steps and interactions.

- Warnings and errors. The application provides the necessary warnings and error messages when the user performs operations that are not allowed.

It would be interesting to evaluate the system's utility for sellers and buyers, analyzing the improvement in shopping experience and user acceptance. This evaluation would have to focus on user feedback, meaning that development would have to prioritize the usability of the system (assuming ideal conditions of security for example), rather than the security or cost issues. Evidently, in order to be used, the system would have to be operable, which means implementing a simple protocol and an interface having only the basic functionalities for running the application.

The user's feedback is important for reaching a conclusion on the utility of the system for users, meaning that the human factor plays an important role in the characterization of this field. However, the resources to be allocated to this project are limited and conducting this evaluation using the method before-mentioned may not be feasible. In that case, one would be restricted to analyzing the hypothetical advantages for the users and comparing this system to related ones.

Chapter 6

Conclusion

An architecture for a receipt exchange system was created, based on a communication protocol secured with the help of cryptography and digital certification. The communication protocol occurred between two entities: the server (the entity that generates and sends the digital receipts), and the client (that requests the receipts).

The system was implemented in order to validate the proposed architecture. With this implementation, the main problems that arise in the development of this kind of systems were identified. The main obstacle found in the implementation was to assure the client application's compatibility with the mobile device that was made available for the development of the system. Two applications were developed; one that worked as the server (based on Java Servlet technology), receiving requests for receipts, after which it would generate and send the correspondent receipts, and another (a Java application) that worked as the client and whose purpose was to establish a connection with the server, request the receipt of the commercial transaction and validate the full process, including the server and the receipt.

The implemented protocol is a client-server protocol and functions on HTTP over SSL.

After the analysis of the architecture and its subsequent implementation, it was concluded that they allow to perform digital receipts' transactions between a client and a server in a secure way, while assuring both parties' privacy and the receipts validity and authenticity for future actions that involve their use. Regarding the usability of the system, it was proven to be simple and very straightforward, with little (and easy) interaction with the user. It was also concluded that different implementations could have been used to prove the advantages and disadvantages of the proposed architecture, like for example the use of SmartCard and RMI.

This thesis allowed to understand the extension a system of this kind possesses and all the security issues that arise and need to be solved for its implementation to be feasible. We believe the creation of such a secure architecture serves as a starting point for the analysis of more complex and full-equipped digital receipts' transaction systems.

Personally, this thesis represented an interesting challenge, that allowed me to develop technic skills and to gain experience in the methodology of solving engineering problems in communication systems. This was accomplished by studying tools and technologies that weren't familiar and by interacting with experienced people who have great knowledge in the area.

As future work regarding the implementation of the proposed architecture, it would be interesting to complete the implementation of the system in mobile devices. Also, new implementation approaches could be taken to validate the architecture and to further analyze possible leaks or flaws that went unnoticed. More security measures could be included like the clients' authentication by the server and securing the storage of receipts in both sides. Studies on the performance impact of digital receipts in the clients' point of view and fraud control by the Finance Department could be done. Also, further research could create a cost model of the implementation of such a system.

Appendix A

Apache Configuration files

A.1 httpd.conf

```
Listen 127.0.0.1:80
ServerName localhost:80

<IfModule ssl_module>
    SSLRandomSeed startup builtin
    SSLRandomSeed connect builtin
    Include "C:/Apache2.2/conf/extra/httpd-ssl.conf"
    </IfModule>
```

A.2 httpd_ssl.conf

```
Listen 127.0.0.1:443

<VirtualHost _default_:443>
    <IfModule mod_ssl.c>
        ServerName 127.0.0.1:443
        SSLEngine on
        SSLCipherSuite ALL
        SSLCertificateFile "C:/Apache2.2/conf/ssl/crt/server.crt"
        SSLCertificateKeyFile "C:/Apache2.2/conf/ssl/key/server.key"
    </IfModule>
</VirtualHost>
```

A.3 tomcat.conf

```
JkWorkersFile C:/Apache2.2/conf/workers.properties

<VirtualHost _default_:443>
    <IfModule mod_ssl.c>

        ServerName red5
        DocumentRoot C:/Tomcat6.0/webapps
        JkMount /* worker1

        <Directory "/">
            Options Indexes MultiViews
            AllowOverride None
            Order allow,deny
            Allow from all
        </Directory>

        <Location "/WEB-INF/">
            AllowOverride None
            deny from all
        </Location>

    </IfModule>
</VirtualHost>
```

Appendix B

Server Application Classes

B.1 Receipt.class

```
public class Receipt {

    public Receipt( String nif, String store, int Store_nif, String product, double price ) {
    }
    public void createDoc() throws IOException {
    }
    public void createPDF() throws DocumentException, MalformedURLException,
IOException {
    }

}
```

B.2 Sign.class

```
public class Sign {

    public static void signPDF( File pdfFile, String pdfOutfile, String reason,
                                String location, Image logo, PrivateKey key, Certificate[] chain
                                ) throws IOException, DocumentException {
    }
    public static void signDoc( String data, PrivateKey key ) throws InvalidKeyException,
                            NoSuchAlgorithmException, NoSuchProviderException,
                            SignatureException, IOException {
    }

}
```

B.3 Send.class

```
public class Send {

    public static void append() throws IOException

    public static void sendAll( HttpServletResponse response, String fileName ) throws ServletException, IOException {
    }
    public static void sendPdf( HttpServletResponse response ) throws ServletException, IOException {
    }
    public static void sendDoc( HttpServletResponse response, String file, String sign ) throws ServletException, IOException {
    }
    public static void sendCert( HttpServletResponse response, String cert ) throws ServletException, IOException {
    }
    private static void copyFile( String srcFile, String dstFile, String type ) throws IOException {
    }
    private static void appendTitle( String file, String title ) throws IOException {
    }
}
```

B.4 certFields.class

```
public class certFields {

    public certFields(PrivateKey key, Certificate[] chain){
    }
}
```

B.5 Servlet.class

```
public class Servlet extends HttpServlet {

    public void doGet ( HttpServletRequest request, HttpServletResponse response ) throws
ServletException, IOException {

        crtFields=getCertFields( keystore );

        receipt.createDoc();
        Sign.signDoc( "Receipt.txt", crtFields.key );

        receipt.createPDF();
        Sign.signPDF( pdfFile, pdfOutfile, logo, crtFields.key, crtFields.chain );

        Send.append();
        Send.sendAll( response, "File" );

        /* Send.sendCert( response, "server_dsa.crt" );
        Send.sendDoc( response, "Receipt.txt", "signature" );
        Send.sendPdf(response);* \
        }

    private certFields getCertFields( File keystore ) throws KeyStoreException, NoSuchAlgo-
rithmException, CertificateException, IOException, UnrecoverableKeyException {
    }
}
```

Bibliography

[1] Pereira, M. Teixeira, "Porquê E-commerce?". http://www.dalera.com/cache/imagens/XPQnHCP8s7885Bu21XzJtmwZKU.pdf

[2] Kyong Jun Lee, Jeong-In Ju, Jeong Mu Jeong, "A Payment & Receipt Business Model in U-Commerce Environment", ACM International Conference Proceeding Series, Vol. 156, ACM 2006.

[3] Jerry Gao, Krishnaveni Edunuru, Jacky Cai, Simon Shim "P2P-Paid: A Peer-to-Peer Wireless Payment System", Proceedings of the 2005 Second IEEE International Workshop on Mobile Commerce and Services, 2005 IEEE. http://ieeexplore.ieee.org/iel5/10558/33407/01581582.pdf?arnumber=1581582

[4] Kwei-Jay Lin, Tao Yu, Chia-Yen Shih, "The Design of A Personal and Intelligent Pervasive-Commerce System Architecture", Proceedings of the 2005 Second IEEE International Workshop on Mobile Commerce and Services, 2005 IEEE. http://ieeexplore.ieee.org/iel5/10558/33407/01581589.pdf?arnumber=1581589

[5] Carl S. Marshall, Adam T. Lake, Rosa J. Thanasohon, "Mobile Digital Receipts", US 2003/0055733 A1, 2003. http://www.freepatentsonline.com/20030055733.pdf

[6] C. Giles, JP Leon, "System and Methods for Digital Receipts", US 6,996,546 B1, 2006. http://www.freepatentsonline.com/6996546.pdf

[7] Mark S. Hoffman, US 6,543,683 B2, 2003. http://www.freepatentsonline.com/6543683.pdf

[8] Emerald Insight Staff(CB), "E-Commerce", Bradford, UK: Emerald Group Publishing Limited, 2005. http://site.ebrary.com/lib/feup/Doc?id=10149880

[9] N. Ntarmos, P. Triantafillou, "SeA1:Managing Accesses and Data in Peer-to-Peer Sharing Network", Proceedings of the Fourth International Conference on Peer-to-Peer Computing, 2004 IEEE. http://ieeexplore.ieee.org/iel5/9261/29415/01334938.pdf?arnumber=1334938

[10] Lim, Ee-Peng(Editor), "Advances in mobile commerce technologies.", Hershey, PA, USA: Idea Group Inc., 2003. p iv. http://site.ebrary.com/lib/feup/Doc?id=10032060

[11] Hu, Wen Chen(Editor), " Advances in Security and Payment Methods for Mobile Commerce", Hershey, PA, USA: Idea Group Publishing, 2004. p v. http://site.ebrary.com/lib/feup/Doc?id=10069708

[12] Gailey, Jeannine Hall(Editor), "Understanding Web Services Specifications and the WSE", Chapter 5. Web Service Security, Microsoft Press, January 07, 2004.

[13] Eisl, Florian(Editor),"Smart Card Security Services for an Open Application Environment used in Mobile Phones", Lund, 2004.
http://www.iicm.tugraz.at/thesis/feisl_magisterarbeit.pdf

[14] Ray-McGovern Technical Consultants(Editor),"Smart Card Security User Group, Smart Card Protection ProfileSmart Card Security User Group, Smart Card Protection Profile", Version 3.0, 2001.
http://www.bsi.de/cc/pplist/scsugpp.pdf

[15] CESG(Editor),"An HMG Public Key Infrastructure to Support Authentication", Crown Copyright, February 1997.
http://archive.opengroup.org/public/tech/security/pki/cki/mem15_10.pdf

[16] Chadwick, David(Editor),"Smart Cards Aren't Always the Smart Choice", Ron Vetter,University of North Carolina at Wilmington, December 1999.
http://ieeexplore.ieee.org/iel5/2/17554/00809256.pdf

[17] HID(Editor),"Smart Cards for Access Control, Advantages and Technology Choices", HID Corporation, 2005.
http://www.hidcorp.com/pdfs/HID_wp_smartcardAC.pdf

[18] Pikrammenos, Giannis A.(Editor),"Smart Card Secure Environments for Mobile Privacy and Security".
http://www.pampas.eu.org/Position_Papers/NTUA.pdf

[19] http://www.smartcardbasics.com

[20] http://www.openmoko.com/index.html

[21] http://developers.sun.com/mobility/midp/articles/bluetooth2/

[22] Seeley, Nathaniel(Editor),"Digital Receipts: A System to Detect the Compromise of Digital Certificates", 2006.
http://contentdm.lib.byu.edu/ETD/image/etd1590.pdf

[23] Suvak, Dave(Editor),"IrDA and Bluetooth: A Complementary Comparison", Extended Systems Inc., 2000.
http://www.smallformfactors.com/pdfs/Counterpoint.Fall00.pdf

[24] Martin van der Zee, Geert Heijenk(Editor),"Quality of Service in Bluetooth Networking, Part I", 10/0362-FCP NB 102 88 Uen, 2001.
http://www.ub.utwente.nl/webdocs/ctit/1/0000004a.pdf

[25] Wong, Ping Wah(Editor), "A Public Key Watermark for Image Verification and Authentication".
http://ieeexplore.ieee.org/iel4/5852/15617/00723526.pdf?arnumber=723526

[26] Luis von Ahn, Nicholas J. Hopper(Editor), "Public-Key Steganography", Carnegie Mellon University.
http://www.cs.cmu.edu/ biglou/pubkeystego.pdf

[27] http://publib.boulder.ibm.com

[28] Huomo, Heikki(Editor), "Method and Mobile Terminal device including SmartCard Module and nearfield communications means", 2006.
http://www.wipo.int/ipdl/IPDL-IMAGES/PCT-PAGES/2006/372006/06095212/06095212.pdf

[29] Gehrmann, Christian(Editor), "Bluetooth Security White Paper", Bluetooth SIG, Inc., USA, 2002.
http://grouper.ieee.org/groups/1451/5/Comparison%20of%20PHY/Bluetoo th_24Security_Paper.pdf

[30] CESG(Editor),"An HMG Public Key Infrastructure to Support Authentication", 1997. http://archive.opengroup.org/public/tech/security/pki/cki/mem15_10.pdf

[31] Oliver Falke, Enrico Rukzio, Ulrich Dietz, Paul Holleis, Albrecht Schmidt(Editors),"Mobile Services for Near Field Communication", Embedded Interaction Group, Media Informatics Group, University of Munich. http://www.medien.ifi.lmu.de/pubdb/publications/pub/falke2007mobileServicesTR/falke2007mobileServicesTR.pdf

[32] Kiessling, Johan, Arwald, Jan (Editor),"Method and apparatus for executing secure data transfer in a wireless network", WO 01/28155 A1, 2001.http://www.wipo.int/ipdl/IPDL-IMAGES/PCT-PAGES/2001/162001/01028155/01028155.pdf

[33] http://www.asb.co.nz/section787.asp

[34] Vipul Gupta, Douglas Stebila, Stephen Fung (Editor), "Speeding up Secure Web Transactions Using Elliptic Curve Cryptography", Hans Eberle Sun Microsystems, Inc.

[35] http://cisco.com/en/US/docs/ios/12_2t/12_2t15/feature/guide/ftsslsht.html

[36] http://www.openssl.org/support/faq.html#MISC1

[37] Tin-Wo Cheung and Samuel T. Chanson, "Design and Implementation of a PKI-based End-to-End Secure Infrastructure for Mobile E-Commerce", World Wide Web Journal, Volume 4, No. 4, pp. 235-254, 2001. http://www.springerlink.com/content/3ev4e3vjfyuyqh9v/fulltext.pdf

[38] http://www.cartaodocidadao.pt

[39] Phillips, John, "Bluetooth".

[40] http://www.kjhole.com/Standards/BT/BT-PDF/Bluetooth12alt.pdf

[41] David Blankenbeckler, "An introduction to Bluetooth". http://www.wirelessdevnet.com/channels/bluetooth/features/bluetooth.html

[42] Lars-Helge Netland, "Java Security - an Infrastructure for Secure Client-Server Communication", June 2005 http://www.nowires.org

[43] http://www.networkworld.com/news/64452_05-17-1999.html

[44] http://www.onjava.com/pub/a/onjava/2002/12/11/eclipse.html

[45] http://httpd.apache.org/

[46] http://tomcat.apache.org/

[47] http://www.lowagie.com/iText/

[48] http://blogs.warwick.ac.uk/kieranshaw/entry/creating_a_java/

[49] http://www.cs.hope.edu/ jipping/javan800.pdf

[50] https://wiki.evolvis.org/jalimo/index.php/Main_Page

[51] http://nds1.nokia.com/phones/files/guides/Nokia_N800_UG_en.pdf

[52] Julie-Marie Foss, Nina Ingvaldsen, "Web Application Security", Trondheim, 2005.

[53] Jing Zhang, "Flexible Certificate Management for Secure HTTPS Client/Server Communication", 2005

[54] http://www.apl.jhu.edu/ hall/java/Servlet-Tutorial/Servlet-Tutorial-Intro.html

[55] Hao-Hsien Bobby Wang, "Desired Features and Design Methodologies of Secure Authenticated Key Exchange Protocols in the Public-Key Infrastructure Setting", Canada, 2004.

[56] http://arizonacommunity.com/articles/java_32001.shtml

[57] http://www.cosic.esat.kuleuven.be/publications/thesis-16.pdf

[58] http://en.wikipedia.org/wiki/Public_key_infrastructure

[59] http://maemo.org/