SECURE ELECTRONIC COMMERCE OF MULTIMEDIA CONTENTS OVER OPEN NETWORKS

A DISSERTATION
SUBMITTED TO THE DEPARTMENT OF APPLIED MATHEMATICS AND TELEMATICS
AND THE COMMITTEE ON GRADUATE STUDIES
OF UNIVERSITAT POLITÈCNICA DE CATALUNYA
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF DOCTOR

By
Jordi Herrera Joancomartí
May 2000
I certify that I have read this dissertation and that in my opinion it is fully adequate, in scope and quality, as a dissertation for the degree of Doctor.

__________________________
Josep Domingo Ferrer
(Principal Adviser)

I certify that I have read this dissertation and that in my opinion it is fully adequate, in scope and quality, as a dissertation for the degree of Doctor.

__________________________
Miquel Soriano Ibáñez
(Tutor)

Approved by the University Committee on Graduate Studies:
I wish I knew as much as I thought I knew ten years ago.
Preface

This thesis is about secure electronic commerce in a generous understanding of the subject. The study is divided regarding the different points of view of the two main parts of a commerce transaction: the buyer and the merchant.

Payment transaction security is an important issue for both parts, buyer and merchant. This thesis first addresses micropayments (payments of small amount) and presents a new micropayment scheme that offers more flexibility than previous proposals. From a merchant’s point of view, electronic copyright protection is needed to prevent and prosecute piracy of multimedia contents supported by electronic means. Different watermarking and fingerprinting schemes are presented mainly focused on protecting still images. Finally, from a buyer’s side, the huge amount of data open networks offer cannot be analyzed by a single person. Mobile agents are called to play a major role in managing information for electronic commerce. Proposals to increase the security level of mobile agents are also presented in this thesis.
Acknowledgments

I would like to thank all the people who supported and encouraged me during the development of this thesis. My sincerest gratitude goes to:

Josep Domingo-Ferrer for having inspired and advised this thesis; Miquel Soriano for his support and friendliness; Jordi Castellà for his help introducing me in the Java card word and also for his support on the bibliography style of this thesis; Jordi Planes for his company; Francesc Sebé for his comments and assistance in implementing the copyright protection schemes; Pere Canut for his useful comments; Ricardo Sánchez Del Castillo for his advice on the general style of this thesis.

Last but not least, I must thank my mother Joana, my brothers Quim and Joan, and my sister Rosó, for their unconditional support. Most of all, I would like to thank my wife Ríra for more things than I can express and my son Pep just for being already here.
Contents

Preface vii

Acknowledgments ix

1 Introduction 1
   1.1 Situation and objectives .............................. 1
   1.2 Structure of the thesis ............................. 4

2 State of the Art 7
   2.1 Electronic payment systems .......................... 7
      2.1.1 Security requirements ........................... 9
      2.1.2 Classification of electronic payment schemes ....... 9
      2.1.3 Existing solutions for micropayment schemes ........ 11
   2.2 Electronic copyright protection schemes .......... 16
      2.2.1 Properties of electronic copyright protection schemes . 18
      2.2.2 Classification of electronic copyright protection schemes 22
   2.3 Agents: a tool for electronic commerce management. ...... 26
      2.3.1 General security issues for mobile agents .......... 26
      2.3.2 Mobile trade agents ............................ 29

3 Electronic Payments 33
   3.1 Enhanced hash chains .............................. 34
   3.2 Spending programs ............................... 36
      3.2.1 Non-iterative spending programs ................. 39
3.2.2 Iterative spending programs .................................. 41

4 Copyright Protection: Symmetric Schemes ............. 45
   4.1 A robust watermarking scheme .................. 46
     4.1.1 Mark embedding ................................ 46
     4.1.2 Mark reconstruction ............................... 48
   4.2 Properties of the watermarking scheme .............. 49
     4.2.1 Imperceptibility .................................. 50
     4.2.2 Robustness ........................................ 51
     4.2.3 Information rate .................................... 55
     4.2.4 Secret information .................................. 57
     4.2.5 Multiple watermarking .............................. 58
   4.3 Symmetric collusion-secure fingerprinting scheme .... 59

5 Copyright Protection: Asymmetric Schemes ............ 67
   5.1 An asymmetric and anonymous fingerprinting scheme .... 68
     5.1.1 Merchant initialization .......................... 68
     5.1.2 Buyer registration .................................. 69
     5.1.3 Mark embedding .................................... 70
     5.1.4 Mark reconstruction ................................ 72
   5.2 Properties of the fingerprinting scheme ............... 74
     5.2.1 Imperceptibility .................................. 74
     5.2.2 Robustness ........................................ 74
     5.2.3 Information rate .................................... 76
     5.2.4 Secret information .................................. 76
     5.2.5 Security for the buyer .............................. 76
     5.2.6 Anonymity ......................................... 77

6 Agents: A tool for electronic commerce management 79
   6.1 Protecting a mobile agent route .................... 79
     6.1.1 Agent route scheme ................................ 80
     6.1.2 Correctness of the proposal ....................... 82
6.2 Intelligent trade agents ............................................. 83
  6.2.1 Architecture of the new scheme .............................. 84

7 Conclusions ......................................................... 87
  7.1 Concluding remarks ............................................. 87
  7.2 Results of this thesis .......................................... 88
  7.3 Future research ................................................. 90

A Structural Coding ................................................. 93

B Spending Programs Using Java Cards ............................ 99
  B.1 Features of a Java card ........................................ 99
  B.2 Performance criteria .......................................... 101
    B.2.1 Speed and storage ...................................... 101
    B.2.2 Programming environment ............................... 106
    B.2.3 Interim discussion .................................... 109
  B.3 A JavaCard file system ....................................... 110
    B.3.1 Implementation ....................................... 111

C Secure Contract signing .......................................... 115

D Off-line anonymous electronic payment .......................... 119

Bibliography ......................................................... 121
List of Tables

4.1   Average percent of pixels that can be marked in an image. . .  56
4.2   PSNR of the images after re-marking process . . . . . . . . . . .  58

B.1   Java card hardware features . . . . . . . . . . . . . . . . . . . .  102
B.2   Java card software features . . . . . . . . . . . . . . . . . . . . .  103
List of Tables
List of Figures

2.1 Entities and actions of an Electronic Payment System . . . . . 8
2.2 A general mark embedding algorithm . . . . . . . . . . 17
2.3 A general mark reconstruction algorithm . . . . . . . . 17
2.4 A watermarking scheme division depending on the information
needed for the mark reconstruction process . . . . . . . . . 23
2.5 The Van der Merwe-Von Solms ITA scheme . . . . . . . . 31
4.1 Original Lena image (512 × 512) . . . . . . . . . . . . . . . . 51
4.2 A watermarked Lena image with $p = 38dB$ . . . . . . . . 52
4.3 A watermarked Lena image compressed at 30% JPEG . . . 53
4.4 A watermarked Lena image with $q = 5$ . . . . . . . . . . 54
4.5 StirMark cropping attacks on Lena’s watermarked image . . 55
4.6 5 times watermarked image . . . . . . . . . . . . . . . . 59
5.1 Mark embedding protocol . . . . . . . . . . . . . . . . . . 71
6.1 The new anonymous ITA scheme . . . . . . . . . . . . . . 85
B.1 Average response time over 30 runs for each card and com-
mand (milliseconds) . . . . . . . . . . . . . . . . . . . . . . . 104
B.2 File system structure (FD stands for file descriptor) . . . . 112
List of Figures
Chapter 1

Introduction

1.1 Situation and objectives

In the last years, the exponential growth of computer networks has given birth to a new way of doing business, namely electronic commerce. Electronic commerce offers new opportunities to the two main parts involved in an economical transaction: buyers and merchants. For the former ones the improvement can be translated into offer increase, while the main challenge for merchants is the opportunity which a world-wide market offers regarding the large number of possible clients.

In this thesis we focus on security issues of electronic commerce. We deal with the subject from two different points of view: the buyer’s side and the merchant’s side. Such a distinction implies the following division depending on which part is considered:

Electronic payments: This section joins both sides, the merchant and the buyer, since both are involved in the payment process.

Electronic copyright protection: This part refers mainly to the merchant who is the party interested in protecting the intellectual property of the goods sold.

Electronic commerce management: This part refers to the security of
the tools the buyer needs to search the best price for an item, identifying all possible offers, etc. More precisely, we focus our research on mobile agent security for electronic commerce.

A key point in electronic commerce security is electronic payment systems (EPS) where both parts, buyer and merchant, are involved. The evolution of electronic commerce is strongly related to the development of secure EPS, since any commercial activity needs to perform payments in a secure way and with a high confidence on them. The important role that confidence often plays in security issues causes the main electronic payment systems to be developed by companies like VISA or MasterCard. For instance, the SET (Secure Electronic Transactions) electronic payment scheme, developed by both companies together with some other partners, is called to be a standard for EPS. Electronic commerce expansion needs properties that go beyond the possibilities of such a general scheme. For instance, one of the requirements in electronic commerce is to perform payments of small amount. Such property cannot be offered by systems like SET since the intrinsic cost of the payment transactions is sometimes higher than the payment itself. So despite general EPS there are still new opportunities to design EPS, with new flexibility and security requirements. One of our research areas will focus on payments of small amounts, called micropayments.

Although electronic commerce is sometimes identified only with electronic payment systems, there are other aspects in electronic commerce where security is an important issue.

*Pure electronic commerce* includes commerce transactions that can be performed entirely over computer networks: from the order and payment of the product to the product reception itself. Then, pure electronic commerce is only possible when the product sold is in electronic format. One of the properties of those products is that their copy becomes very cheap and such copies are identical. This fact represents a handicap since on one hand it
reduces the production costs for the merchant but on the other hand it facilitates piracy: the buyer can redistribute the product illegally with no quality loss. The problem of intellectual property rights (IPR) protection is thus especially serious for information in electronic format, where it is also known as electronic copyright protection.

Current research in electronic copyright protection is focused on copy prevention rather than on copy protection. Illegal copies should be traced up to the original buyer. To allow the copy to be traced some extra information has to be embedded on the electronic product by the merchant. The embedding process has to fulfill some robustness properties in order to thwart attacks from dishonest buyers.

An important part of our research work is focused on the development of new copyright protection schemes offering new properties.

The last point of our research refers to the management of electronic commerce from the buyer’s point of view. The exponential growth of electronic commerce has created an incredibly large offer of products and services for the users of computer networks. Such a huge amount of information makes it impossible for a single person to analyze all the offers of a existing product on the net and decide which of them fits better her requirements. In other words, the optimal management of electronic commerce is becoming a problem difficult to solve by human means. This human limitation in net search can be solved with the use of an agent, i.e. a program that roams the net looking for items that best satisfy the buyer requirements. This feature of agents raises important security issues which are considered from different points of view in the literature. Since the agent is a program that roams the network, it needs to be executed at different servers which are not necessarily trusted by the agent (or consequently by the agent’s owner). In this research line, we propose some schemes to protect different parts of an agent against malicious hosts.
1.2 Structure of the thesis

This thesis is organized as follows.

Chapter 2 contains the state of the art of the three main areas of this dissertation: electronic payment systems, electronic copyright protection scheme and secure mobile agents. Basic definitions and properties as well as general ideas of the three areas are presented. The main problems are outlined for later discussion. The rest of chapters are original contributions so this second chapter will be often recalled in the rest of the thesis.

Chapter 3 deals with electronic payment systems. In particular, an extension of the micropayment systems based on hash chains is given. Enhanced hash chains are defined and structural coding is recalled to implement the new scheme.

Chapters 4 and 5 present different copyright protection schemes. In Chapter 4 symmetric schemes are presented. More precisely, a watermarking scheme for images is described and an extension of such scheme to a symmetric fingerprinting scheme is given. Some practical results are presented to show the correctness of the schemes.

In Chapter 5 an asymmetric fingerprinting scheme is presented. The scheme is described at a higher level than the ones in Chapter 4 due to the constraints of the properties offered by the scheme.

Chapter 6 presents some results regarding security aspects of mobile agents. In particular, a new method based on enhanced hash chains to protect an agent route is proposed. Furthermore, a generic scheme for an intelligent trade agent that improves previous proposals is given.

The concluding remarks and a summarize of the results presented in this thesis can be found in Chapter 7. Some guidelines for further research are given in this chapter as well.

In Appendix A, the basic definitions and properties of structural coding are recalled.

Appendix B deals with implementation issues of the micropayment scheme presented in Chapter 3 on smart cards. A basic description of a Java card
is given and a comparison between different Java card manufacturers is presented to decide on the suitability of a particular Java card to implement the micropayment system defined in Chapter 3. Finally, a basic file system for a Java card is presented.

Appendix C and D offer some basic cryptographic background needed to understand some proposals of this thesis.
6 Introduction
Chapter 2

State of the Art

In this chapter is examined the state of the art of the main three research lines of this thesis. Such background information will provide the knowledge needed to identify the weaknesses of the existing proposals and their possible solutions.

It is worth stressing the importance of Section 2.2 which offers a general overview and classification of the main copyright protection schemes. Although existing works have collected information about different copyright protections schemes, to our best knowledge this is the first survey that deals together watermarking and fingerprinting schemes, studying both their connections and differences.

2.1 Electronic payment systems

Credit cards appeared during the sixties can be regarded as the first electronic payment system in history. But nowadays, an electronic payment system (EPS) refers to any payment system that can perform payments through computer networks.

Roughly speaking, EPS are just another step on the way of representing an abstract value, using the tools that are becoming more common in the technology age. So EPS can be regarded as normal payment schemes that
have to satisfy new security requirements arising from the nature of the new support: electronic money.

The parties involved in an EPS are basically the same than in normal payment schemes: the buyer and the merchant. But in EPS, two more entities are needed, the issuer and the acquirer that are the banks of buyer and merchant respectively. These new entities play a supporting role in an EPS as shown in Figure 2.1.

Depicted in Figure 2.1 we can see as well different actions and their sequence (follow the numbers) that take place in an e-cash\textsuperscript{1} EPS.

It has to be mentioned that some other parties and actions, like arbiters and disputes, can be needed for global and correct operation of an EPS but the assumption that such disputes will be handled outside the payment system is commonly made.

\footnote{See Subsection 2.1.2 for more details about EPS classification and in particular about the e-cash model.}
2.1.1 Security requirements

Security issues are a major concern in EPS. For instance, the fact that replication of digital information is easy and exact entails a real threat since an EPS pretends to use an electronic piece of information as a coin.

Some of the security requirements depend on the features of each EPS, so the classification of Subsection 2.1.2 is relevant to define security properties of an EPS. Nevertheless, all EPS have to fulfill some basic properties [AJS97] that can be summarized as follows:

**Integrity:** The EPS has to ensure that nobody can use another user’s money without explicit authorization of that user.

**Authorization:** Payments performed using the EPS have to be authorized by the proper user using one of the existing methods for identification. Such identification can be done with passwords or using digital signatures for stronger security.

**Confidentiality:** The EPS transactions have to offer a certain degree of confidentiality about the identities of buyer and merchant, the whole operation contents and the amount of money involved. Confidentiality must exist at least between the parties involved in the transaction but can be reduced if the payment system can afford buyer anonymity.

**Availability:** All entities involved in the EPS should be able to perform payments. Furthermore, payments should be atomic in the sense that they occur entirely or not at all, to avoid inconsistent states.

**Reliability:** The underlying networking services and all software and hardware components are supposed to be sufficiently dependable.

2.1.2 Classification of electronic payment schemes

The EPS can be classified depending on different properties they offer. One possible criterion focus on the moment when the buyer withdraws the money
from her issuer to pay the merchant. Following this criterion EPS can be divided into: prepaid payment systems, pay-now payment systems and pay-after payment systems. In the first category, the buyer withdraws away some money from her issuer account and later on uses such amount to perform payments to a merchant. Such structure is the electronic analog of a cash payment and for that reason also prepaid payment systems are known as electronic cash systems or e-cash for short. Figure 2.1 shows the scheme of an e-cash system. In the pay-now category the buyer’s account is debited the amount of money exactly in the payment protocol. Some debit cards fall into this category. Finally, the third category includes the payment systems used by credit cards. They are pay-after systems since, when the buyer buys something, her issuer credits the amount of the purchase before the buyer’s account is debited.

Another possible division of EPS results from considering the involvement of the issuer during the payment transaction. An EPS is called on-line if the issuer has to take an active role during the payment protocol and is called off-line if no issuer participation is required during payment.

A third distinction can be made between payments that offer buyer anonymity and others such that the identity of the buyer is disclosed. A finer distinction can be performed in anonymous EPS between schemes that can revoke buyer anonymity and those in which buyer identity is completely untraceable.

There are several classifications apart from the ones stated before, but for our purposes we will use the EPS classification that depends on the amount of the payment. Macropayments, or standard payments, are payments of regular to high amount, namely more than 10 euros. Micropayments are payments that deal with amounts of less than 10 euros. Clearly, such a division of EPS is rather arbitrary and fuzzy, but his motivation will become
clear when presenting micropayment applications below.

The most important micropayment systems are: SET [SET], iKP [BGH95], CyberCash [Cyb], DigiCash [Dig] and NetCash [MN93].

Among them, SET (Secure Electronic Transactions) is the most important one and it is supposed to be the standard EPS in the near future. In fact, SET is a set of specifications developed by VISA and MasterCard in co-operation with different computer companies like IBM, Microsoft, Netscape and Verisign. Basically, SET is based on credit cards and electronic certificates and allows secure payment systems through Internet.

The reason to design specific micropayment schemes is that standard electronic payment systems for low-value payments suffer from transaction costs too high when compared with the amount of payments. For instance, in applications like phone call payment, access to non-free web pages, pay-per-view TV, etc., the transaction cost can be higher than the amount to be paid. This problem arises due to complex cryptographic protocols like digital signatures used to achieve a certain security level in the transactions. However, micropayments do not need as much security as speed and simplicity (in terms of computing). For that reason, as it will be shown in next subsection, several micropayment proposals try to replace digital signatures with faster operations.

### 2.1.3 Existing solutions for micropayment schemes

Different micropayment schemes can be found in the literature [GMA+95, FW95, AMS95, HSW96, Ped97, RS95] but none of them has taken a dominant position in the market. The existing proposals use several techniques to obtain a fast and cheap EPS for small amounts. For instance Millicent [GMA+95] is based on keyed hash functions to obtain certain degree of security. MicroMint [RS95] uses collisions of $k$ elements by a hash function to represent a coin. SubScript [FW95] uses techniques similar to the ones used in Millicent.
We will focus our research in micropayments systems that replace digital signatures by one-way hash functions. This is a common choice in micropayment schemes. It will be shown later on that replacing digital signatures with hash functions is useful in other areas of electronic commerce, like secure mobile agents.

A *computational one-way function* is a function that is easy to compute but hard to invert. So given a value $x$ the image value $y = f(x)$ is easy to compute but it is computationally hard to obtain the pre-image $x$ given the value $y$. On the other hand, a *one-way hash function*, sometimes called message digest, is a function that takes a variable-length input string and converts it to a fixed-length (generally smaller) output string. Such functions are widely used in cryptography for digital signatures. But for cryptographic applications, hash functions must be also *collision-free*. A collision-free function is a function for which it is hard to generate two different pre-images $x \neq x'$ with the same image value $f(x) = f(x')$.

Different micropayment systems based on hash functions have been proposed so far. The main advantage of hash functions is the reduction of complexity in the algorithms and protocols of the micropayment systems. As an example of such a reduction, on a typical workstation, it may take half a second to compute an RSA [RSA78] signature; in that period, 100 RSA signatures can be verified (assuming a small public exponent) and, more important, 10,000 hash functions can be computed. So the advantages of dropping digital signatures in favor of hash functions should be clear.

Micropayment systems based on hash functions include NetCard [AMS95], $\mu$-iKP [HSW96], Pedersen [Ped97] and PayWord [RS95]. The principle behind those systems is similar. Let $F$ be a computationally one-way and collision-free hash function. Now the buyer takes a value $X$ that will be the

\footnote{Although sometimes the words collision-free are avoided, all hash functions used in this thesis are also one-way and collision-free.}
root of the chain and computes the sequence $T_n, T_{n-1}, \ldots, T_0$, where

\[
\begin{align*}
T_0 & = F(T_1) \\
T_1 & = F(T_2) \\
\vdots & \quad \vdots \\
T_{n-1} & = F(T_n) \\
T_n & = X
\end{align*}
\] 

(2.1)

The values $T_1, \ldots, T_n$ are called coupons and will be used by the buyer to perform $n$ micropayments to the same merchant. Each coupon has the same fixed value $v$. Before the first micropayment, the buyer sends $T_0$ to the merchant together with the value $v$ in an authenticated manner. The micropayments are thereafter made by successively revealing $T_1, \ldots, T_n$ to the merchant, who can check the validity of $T_i$ by just verifying that $F(T_i) = T_{i-1}$.

**Note 1** In case the buyer wants to spend a total amount of $5v$, it is not necessary to send five coupons; it suffices sending the coupon $T_{n-5}$ to the merchant. But to validate the coupons the merchant has to compute five hash functions to check that $f^5(T_{n-5}) = T_n$.

The security of these systems is based on the one-wayness of hash function as well as the validity of the first coupon $T_0$. Normally, the first coupon is validated using digital signatures. Once the chain is initialized ($T_0$ has been validated by the merchant), an opponent cannot pay using this initialized chain because if he can compute the value $T_1$ such that $F(T_1) = T_0$, that means that he can invert the hash function $F$ which is supposed to be hard. Furthermore, the effort to invert the function $F$ for the particular value $T_0$ will give a profit equal to the value $v$ of the micropayment, which is typically small.

In the way described above, digital signatures have been replaced by hash
functions. Thus a digital signature is not necessary for each individual micropayment, since the hash function used offers enough security. But notice that not all digital signatures have been dropped. It is still necessary to have a digital signature to validate all the hash chain. The advantage is to save $n - 1$ digital signatures of the individual micropayments. Reducing the computation linked to each micropayment clearly reduces the transaction cost.

Some systems, like $\mu$-iKP or NetCard, use standard EPS (iKP and SET respectively) to send to the merchant the first coupon $T_0$ and the value $v$ and, in this way, all the chain is authenticate.

We next mention some differences between the main micropayment systems based on hash functions:

- With NetCard [AMS95] the bank supplies the root $X$ of the hash chain to the buyer. The buyer then computes the chain, signs its last element $T_n$, the total number of elements $n$ and the value of each chain element $v$. These signed values are sent by the buyer to the merchant, who uses the SET [SET] protocol to obtain on-line authorization for the whole chain.

- With $\mu$-iKP [HSW96], the root of the chain is a random value chosen by the buyer and the payment structure is the same as in the iKP [BGH+95] payment system. In other words, the on-line authorization of the chain is performed by authorizing a single iKP payment of regular amount.

- PayWord [RS95] is a credit-based scheme that needs a broker. The buyer establishes an account with the broker who gives her a certificate that contains the buyer identity, the broker identity, the public key of the buyer, an expiration date and some other information. The hash chain is produced by the buyer using a random root. When the buyer wants to make a purchase, she sends to the merchant a commitment to a chain. The commitment includes the merchant's identity,
the broker certificate, the last element of the chain, the current date, the length of the chain and some other information. In this scheme, the broker certificate certifies that the broker will redeem any payment that the buyer makes before the expiration date, and the buyer commitment authorizes the broker to pay the merchant. After that, each micropayment is made by revealing each element of the chain to the merchant.

- Pedersen [Ped97] also iterates a hash function with a random root to obtain a chain of coins but he does not provide much detail related on what kind of system (credit or debit based) he implements nor does he give information about some other security issues.

The use of hash chains implicitly assumes that micropayments take place repeatedly from the same buyer to the same merchant. This stability assumption on buyer-merchant relationship can be relaxed (as it was pointed out in [HSW96]) at the cost of trusting an intermediate broker who maintains stable relationships with several buyers and several merchants: a buyer can send coupons to the broker and the broker is trusted to relay (his own) coupons to the merchant for the same value.

A weak point of the micropayment systems described above is the low flexibility they offer, which results in at least two shortcomings:

- Since all coupons have the same value $v$, the only way to be able to pay any amount is to let $v$ be the minimal value, for instance one cent. But this means that the merchant must verify fifty hash functions to get paid a sum as small as fifty cents. It is true that the buyer just needs to send one hash value (the 50th), but in any case she must store or compute all intermediate hashes.

- Fixed-value coupons do not allow to deal with different currencies.

In Chapter 3 we study these weaknesses and we present a new solution to avoid these problems and to obtain a more flexible micropayment scheme.
2.2 Electronic copyright protection schemes

Computer networks offer a growing number of opportunities due to pure electronic commerce, where the article being sold is in electronic format (music, video, multimedia contents, data, etc.). But in pure electronic commerce, a new security problem arises related to illegal distribution: piracy.

Electronic copyright protection schemes based on the principle of copy prevention have proven ineffective in the last years (see [PAK98]). The current trend for electronic copyright protection is to rely on copy detection. The merchant $M$ selling the piece of information (e.g. image) embeds a mark\(^3\) in the copy sold. The idea is to perform some imperceptible modifications on the product sold to embed the mark. The duplication of the product implies the duplication of the mark.

There are two basic kinds of marks: fingerprints and watermarks. One may think of a fingerprint as a serial number (i.e. something identifying the buyer of the copy) while a watermark is an embedded copyright message similar to the © symbol that appears on books.

The main difference between watermarks and fingerprints is the following. In a watermarking scheme, the hidden message (the mark) is the same for all buyers, but in a fingerprinting scheme such mark depends on the buyer’s identity. This subtle distinction implies important differences between both techniques. While watermarking schemes only allow proofs of ownership of the product, fingerprinting schemes are much more powerful since they allow to identify the illegal redistributor. So fingerprinting deals with problems often known as traitor tracing [CFN94]. Such schemes need extra properties derived from the fact that different buyers get different marks. In particular security against collusions, described in Subsection 2.2.1, is needed.

\(^3\)In this thesis we use the convention that a marked object contains only one mark. Such a mark is a word of $n$ bits all of them distributed inside the marked object.
In spite of above distinction, a general copyright protection scheme (fingerprinting or watermarking) can be described by two algorithms (or protocols): mark embedding and mark reconstruction. In more sophisticated schemes other protocols like register or initialization can be found.

A general mark embedding process is depicted in Figure 2.2. This process takes as inputs the mark $M$, the cover-object $X$, that is the object that has to be marked, and a secret key $K$ used to spread the mark bits inside the object. The output of the marking process is the marked object denoted by $\overline{X}$.

Figure 2.3 shows a general mark reconstruction process in which information about the mark $M$ can be obtained from a marked object $\hat{X}$. 
In such process the inputs are: suspect \textit{marked object} \( \hat{X} \), \textit{secret key} \( K \) used in the embedding algorithm and some additional information. Such additional information depends on the properties of the scheme and it will be discussed in Section 2.2.2. The output is the mark \( M \) or a \textit{true/false} statement, also depending on the scheme used. Notice that the input object \( \hat{X} \) does not need to be the same as the marked one \( \overline{X} \) since the latter could have been altered by an attacker.

### 2.2.1 Properties of electronic copyright protection schemes

The different properties that an electronic copyright protection scheme should offer are detailed below. Some of them depend on the kind of scheme we deal with, watermarking or fingerprinting. When necessary, a distinction will be made.

**Imperceptibility:** The mark performed in the electronic item has to be imperceptible for the buyer, \textit{i.e.} quality cannot be reduced due the mark embedding. Such imperceptibility can be measured depending on the nature of the item. For instance, in image protection, quality of the marked image can be assessed in the same way quality of compressed images is measured [NAdS92].

**Robustness:** This property measures the resistance of the mark against some attacks, which attempt to remove it partially or completely from the item. The final goal of the copyright protection scheme is to maintain the mark in the article until the article has become useless. Unintentional “attacks” or distortions due to errors in the computer network transmission or accidental user manipulation of the item are obviously less dangerous than intentional attacks. For that reason, we focus on the latter. Two different kinds of intentional attacks can be identified:

- \textit{single-user attacks}
- \textit{collusion attacks}
2.2 Electronic copyright protection schemes

Single-user attacks are performed by a single buyer who tries to render the mark useless to disrupt the mark reconstruction process. Such attacks can be of several type. For instance, in image copyright protection, attacks are related to image lossy compression, geometrical distortion (rotation, scaling, shearing, etc.), filter application, and so on. Furthermore, schemes have to be robust against noise inclusion or overmarking. The original marks have to be recognizable in spite of insertion of noise or extra marks in the object. Multiple marking schemes can be obtained as an extension of these kind of robustness systems which offer the possibility of embedding multiple marks on the object that later on can be retrieved without mixing them up.

Both watermarking and fingerprinting schemes have to be robust against single-user attacks.

Collusion attacks are performed by several users who use knowledge of their copies of the same item to remove or alter the marks embedded in them. In particular, schemes robust against collusion attacks are called collusion-secure schemes. Notice that watermarking does not have to face with collusion attacks since all items sold carry the same mark, so all copies are identical and knowledge of several copies does not leak any information. Collusion attacks are important in fingerprinting schemes because each copy sold is different and a coalition of users can detect the position of some mark bits by comparing their copies. In [BS95] the marking assumption is stated. Such assumption says that a bit of the mark embedded in an item can only be detected by comparing different copies from different users. Furthermore, the authors of [BS95] assume that if a bit of the mark cannot be detected, it cannot be removed. This last assumption is rather unrealistic since it is equivalent to say that single-user attacks cannot succeed, which would require a perfectly robust underlying watermarking scheme. For instance, single-user attacks for images described above destroy the mark in several existing watermarking schemes without degrading the
image quality. Although the marking assumption seems too ideal, it can be approximately enforced if a sufficiently robust watermarking scheme is available. This is the situation in Section 4.3 of this thesis.

To summarize, robustness can be renamed as **Security for the merchant**, that is, on finding a redistributed copy, the copyright protection scheme must be able to use the embedded mark to:

- resolve rightful ownership of the product (in a watermarking scheme).
- identify the original buyer of the illegally redistributed copy (in a fingerprinting scheme).

Obviously, robustness is the hardest and most important property in a copyright protection scheme. Furthermore, this property is against other properties like imperceptibility or information rate. A trade-off between them has to be reached.

**Information rate:** This property measures the information amount that can be introduced into the item without affecting its quality. It can be measured as the bit length of the mark embedded.\(^4\) For robustness purposes, the information included in the object may be replicated to obtain some redundancy. If this replication is done accurately, the scheme is called region-based and it is possible to reconstruct the mark only from a piece of the marked object. Of course, this replication reduces the total information rate of the scheme.

**Secret information:** The secret information is the minimum information that has to be kept secret to ensure the robustness of the scheme. It is clear that the cover-object has to be kept secret; otherwise, illegal redistribution of the unmarked cover object could not be detected.

\(^4\)Although there are copyright protection schemes for which the embedded mark belongs to a continuous domain, in this thesis we will focus on the discrete domain, so it makes sense to speak of the bit length of the mark.
Moreover, some other information needed to execute the mark reconstruction process has to be secret. These information depends on the scheme’s classification (see Figure 2.4 for more details) but at least the key $K$ used in the marking process has to be keep secret. Like in cryptosystems, copyright protection schemes must satisfy Kerckhoff’s principle [Ker83], which assumes that the method to encipher (embed) information is public so that security must lie only in the choice of the key\textsuperscript{5}. But in copyright protection schemes the key $K$ is somewhat different from a cryptographic key. In cryptography, the key must be short because it has to be sent from the sender to the receiver; in copyright protection schemes the “sender” and the “receiver” are usually the same party, the merchant, so the key length is not critical because the key does not need to be transmitted. Furthermore, it is more important that, upon seeing a marked item, the merchant can identify the key to be used for mark reconstruction. Of course, different objects can (and probably should) be marked using different keys. Nevertheless, the key is supposed to be random and independent, like in cryptography and it is critical to the security (robustness) of the scheme.

**Security for the buyer:** The buyer has to be sure that she cannot be falsely accused of illegal distribution by a dishonest merchant. That is, the scheme must give a proof of the buyer’s guilt to the merchant and such proof must convince a judge beyond reasonable doubt. As we will explain in the next section, copyright protection schemes that offer such property are also called **asymmetric schemes.** In particular, watermarking schemes are never asymmetric schemes while fingerprinting schemes can be symmetric or asymmetric.

**Anonymity:** The buyer should be able to preserve her identity when she

\textsuperscript{5}"Il faut qu'il n'exige pas le secret, et qu'il puisse sans inconvénient tomber entre les mains de l'ennemi"[Ker83]
buys some article that has been marked. This property does not apply to watermark since the user identity is not included in the marked object. Yet, the merchant may wish to maintain a sales register that records if a particular user is authorized to use such watermarked product. Nevertheless, the anonymity of such register is not a matter related with the copyright protection scheme used.

2.2.2 Classification of electronic copyright protection schemes

Based on some properties described in the above section, the electronic copyright protection schemes that can be found in the literature fall into one of the following groups:

Watermarking schemes

As stated before, watermarking schemes are copyright protection schemes that offer proof of ownership.

Within watermarking schemes, a finer distinction can be made depending on the information needed for mark reconstruction. Obviously, it is always assumed that the object $\hat{X}$ and the key $K$ are inputs of the reconstruction process (see Figure 2.3 for more details) but the need of further information yields the following distinction (similar to the one in [KP99]) graphically expressed in Figure 2.4.

Private watermarking: Schemes that need all possible information as input of the mark reconstruction process. That is $\hat{X}$, the cover object $X$, the secret key $K$, as well the included mark $M$. The mark reconstruction process gives a true or false statement as an output, depending on whether the mark $M$ is the one embedded in $\hat{X}$. Schemes falling in this category are [Car95, CKLS96, KH97, PZ97]

Semi-public watermarking: These schemes do not need as much information as the private ones. For instance, the original object $X$ can be
2. Electronic copyright protection schemes

\[ \text{PUBLIC} \]
\[ \hat{X} \times K \rightarrow M \]

\[ \text{Input: object } X \]
\[ \hat{X} \times K \times X \rightarrow M \]

\[ \text{SEMI-PUBLIC} \]

\[ \hat{X} \times K \rightarrow \{0, 1\} \]

\[ \text{Input: mark } M \]
\[ \hat{X} \times K \times M \rightarrow \{0, 1\} \]

\[ \text{PRIVATE} \]
\[ \hat{X} \times K \times X \times M \rightarrow \{0, 1\} \]

Figure 2.4: A watermarking scheme division depending on the information needed for the mark reconstruction process.

required as input, like in [RP98] or the ones presented in Sections 4.1 and 4.3. On the other hand, it may happen that the original object \( \hat{X} \) is not needed but the embedded mark \( M \) is. Examples of such schemes are [BBCP98, KH98, NO98, NP98, TKS98, vSTO94, WD96]

**Public watermarking:** Schemes in this category only need the suspect marked object \( \hat{X} \) from where the mark has to be extracted and the key \( K \) used in the embedding process. The schemes in [HG98, Kut98, LvdLL97, SZT96, ZK95] fall into this category.

Although watermarking schemes are mainly developed for copy detection, they can also be used for copy protection. An example can be found in [Bel99] where a mark is introduced in a DVD recorded disc. Such a mark specifies a message that can be: "copy once" or "never copy". In case the recording DVD device extracts the first mark, it will allow the original DVD to be copied and it will change the mark to "never copy". If a "never copy" mark is found, the DVD recorder device will not make the copy.
Symmetric fingerprinting schemes

The fingerprinting concept was first proposed by N.R. Wagner in [Wag83]. Later on, when new schemes appeared, the Wagner scheme was classified as a symmetric fingerprinting. Symmetric fingerprinting schemes [BMP86, BS95, DBS+99, Pfi96] are the ones where both the merchant and the buyer know the marked copy. But the buyer does not know where the mark bits are located inside the item. The main problem of such kind of schemes is that they do not offer security for the buyer in the sense described in Section 2.2.1. Since the merchant knows the marked copy, a dishonest merchant could redistribute himself the copy to falsely accuse the buyer. Furthermore, such an argument can be used by a guilty buyer to deny guilt in court. Notice that the above discussion does not apply to watermarking. In these schemes, all marked copies are the same and it is hard to assume that the merchant does not know them. So, neither symmetric nor asymmetric concepts can be applied to watermarking schemes.

Several symmetric fingerprinting schemes use a watermarking scheme as a building block. In fact, some watermarking schemes can be transformed into symmetric fingerprinting schemes by adding the property of security against collusion, but the transformation is not possible for all schemes. In Section 4.3 we discuss the upgrade of a watermarking scheme into a symmetric fingerprinting scheme.

Asymmetric fingerprinting schemes

Asymmetric fingerprinting schemes were first proposed by Pfitzmann and Schunter in [PS96]. The main contribution of such schemes is that they offer security for the buyer. An important difference with symmetric schemes is that the marking algorithm, performed typically by the merchant alone, is transformed into a protocol in which the buyer plays an active role. These schemes have the feature that the merchant does not know the marked copy sold to the buyer, but when the merchant obtains such a copy, he can retrieve the mark that was embedded and thus the buyer’s identity. Since the buyer
is the only one to know the marked copy, she can be accused without any doubt in case this copy is found to have been illegally redistributed.

Different asymmetric fingerprinting proposals can be found in the literature [DF98, DF99, PS96, PW97b]. In [DF98, PS96] the authors use multiparty computation [CDvdG88] to ensure that the merchant cannot obtain knowledge about the marked copy. Since multiparty computation is computationally complex, some proposals are focused on avoiding such a technique. For instance, in [DF99] the cryptographic tool Committed Oblivious Transfer [CvdGT95] is used instead of multiparty computation. In Chapter 5 a new proposal that avoids multiparty computation is presented.

Anonymous fingerprinting schemes

The concept of anonymous fingerprinting was introduced in [PW97a] and can be regarded as an improvement of asymmetric fingerprinting. Anonymous fingerprinting is analogous to the *blind signature* concept defined by D. Chaum in [Cha83]. The buyer of the marked object should keep her anonymity in the same way electronic payments offer anonymity. All proposed anonymous fingerprinting schemes are also asymmetric schemes. For instance, the anonymity is obtained in [PW97a, DF98] through the use of pseudonyms that every user has obtained from a registration authority. On the other hand, in [PS99] a coin-based scheme is used to provide anonymity to the buyer. Redistribution is modeled as double spending in payment. In this way, an illegal copy of the object implies the loss of anonymity.

---

6See Appendix D for more details on blind signatures.
2.3 Agents: a tool for electronic commerce management.

As mentioned in Chapter 1, the huge amount of products that a buyer can find on the Internet cannot be analyzed by a single person. To solve this problem, mobile agents have been developed. Mobile agents are programs that roam the network searching for products that fit best buyer requirements. Such property of roaming the network raises important security issues from different points of view. Since the program roams the network, it has to be executed on different servers. The security issues arising from this fact are recalled in next subsection.

2.3.1 General security issues for mobile agents

Remote execution of mobile agents encounters the following problems derived from the mutual mistrust of the parties:

- how to protect hosts against malicious agents.
- how to protect agents against malicious hosts.

The first problem above is a server security one. For instance, mobile agents that are executed on a server should not access information not intended for them to have or they should not go into an infinite loop and so squander the server resources. In fact, such security problems are similar to the ones caused by a virus. Different proposals are given in the literature to protect hosts against agents. There is some work regarding the restrictions of the code executed in the servers (Java, Software Fault Isolation [WLAG96], Proof-Carrying Code [NL96], operating systems extension mechanisms such as packet filters [MJ93] or type safe languages [BSP+95, HFG+96]).

The problem of protecting agents against malicious hosts has been dealt from different points of view.
In [ST98] a partial solution for that problem is proposed based on computing with encrypted functions, an extension of computing with encrypted data [DF96, DF97, DFHJ98b, DFHJ98c] applied to a general function. The authors identify programs with functions and they propose a solution for polynomial and rational functions.

Another approach to protect agents from malicious hosts is to detect the attacks \textit{a posteriori}. Once the attack is detected, information can be retrieved and used to accuse the malicious host. To obtain this information the host has to perform some extra computations to prove the correctness of agent execution; this type of schemes include cryptography to subsequently prove attacks. Examples of this approach can be found in [Ben97, Vig98]. In [Mea97] an idea in the same direction is proposed: the agent includes some dummy data to detect whether the host wants to perform an attack or not. But a posteriori detection is not safe since an existing host can disappear in seconds from the network. It does not make sense to accuse a server if it does not longer exist.

Some other techniques to protect agents from malicious hosts try to make attacks unlikely by obfuscating the readability of the code of the agent. In this direction we can find [Dye97, Hoh98, Lib93, Sri97].

Another way to protect agents against malicious hosts consists of having them circulate only in trusted execution environments. For instance, in [WSB98] a tamper proof hardware device is proposed to ensure the correctness of the hosts the agent is going to visit. In [Rot98] a solution based on the cooperation of different agents that visit different hosts is proposed.

Other proposals focus on protecting the agent’s route, that is, a predetermined itinerary that is carried by the agent and is to be followed by him.

In [SRM98] a general a concept of an agent route, called itinerary, is given. The itinerary is composed using different itinerary entries. Such different entries, together with the "Exactly-Once" property defined in the same paper, allow specification of flexible agent's travel plans as well as
dynamic adaptation and expansion of the travel plan during the execution of the agent.

The problem of the above scheme, is that the agent route implemented in this way can be attacked by malicious hosts at least in the following two different ways:

- A malicious host can modify the itinerary entries since the itinerary is carried by the agent. Some extra hosts can be added or some can be removed.

- Knowledge of the whole route of the agent can be obtained by any visited host. This knowledge can be used to perform attacks on other servers to deny the service to that agent.

In [BRSR99], some of that problems are also identified and a partial solution is provided using a public key encryption infrastructure and a trusted authority. The problem of protecting the agent route from host addition or removal is not completely solved.

In [WSUK99] nested encryptions and signatures are used to provide increased security to the agent route. The basic idea is to sign and encrypt hosts addresses iteratively to achieve a high security degree. The scheme proposed works as follows:

Let $PK_i$ be the public key of $i$ host and let $S_h(\cdot)$ be the signature function of the agent owner. Then the whole route $r$ is coded in the following way:

$$r = E_{PK_1}[H_2, S_h(H_1, H_2, t, E_{PK_2}[\cdots]), E_{PK_2}[\cdots]]$$

where, for $i = 1$ to $n - 1$,

$$E_{PK_i}[\cdots] = E_{PK_i}[H_{i+1}, S_h(H_i, H_{i+1}, t, E_{PK_{i+1}}[\cdots]), E_{PK_{i+1}}[\cdots]]$$

and $E_{PK_n}[\cdots] = E_{PK_n}[H_0, S_h(H_n, H_0, t)]$. In the above expressions, $H_i$ is the IP-address of the $i$-th host, $H_0$ is the IP-address of the agent owner.
(host which originates the agent) and \( t \) is a timestamp (IP stands for Internet Protocol).

The previous coding enforces the route as follows. The first host \( H_1 \) receives \( r \) and decrypt it using its private key to obtain: \( S_h(H_1, H_2, t, E_{PK_2}[\cdots]) \), \( H_2 \) and \( E_{PK_2}[\cdots] \). Using the public key of the agent owner, \( H_1 \) can verify that the next host address \( H_2 \) and the value of the rest of the route \( E_{PK_2}[\cdots] \) were included in the route by the agent owner. The inclusion of his own address \( H_1 \) in the signature allows to detect that he was also included in the route by the agent owner. The timestamp \( t \) is used to include an expiration date to prevent reuse of older routes of the same agent by malicious hosts. Beyond these validations, \( H_1 \) cannot obtain any other knowledge on the rest of the route since all remaining information is encrypted under the public key \( PK_2 \) of \( H_2 \). Then \( H_1 \) sends the agent to \( H_2 \) together with

\[
r_1 = E_{PK_2}[H_3, S_h(H_2, H_3, t, E_{PK_3}[\cdots]), E_{PK_3}[\cdots]]
\]

The above decryption and verification process is repeated by \( H_2 \) and so on up to \( H_n \). After \( n \) steps, the agent is returned by \( H_n \) to \( H_0 \).

The main problem of the proposal described above is the high cost that nested encryptions and signatures imply (as the authors point out in [WSUK99]). In Chapter 6 a new proposal that offers the same level of security and reduces computational cost is made to code an agent route.

### 2.3.2 Mobile trade agents

The majority of mobile agents are designed to perform actions on their owners’ behalf. When the application field of the mobile agent is electronic commerce, the agent buys (or sometimes sells) products in her owners behalf which means that electronic payment systems are involved in the transactions. This kind of mobile agents are often called Intelligent Trade Agents (ITA), and for these specific agents some security issues arise in addition to the ones recalled in subsection above. A major issue is to protect the
confidential data carried by the agent. These protected data should enable the agent to perform the payment in her owners behalf. Furthermore, since the agents can perform payments, the security requirements described in Section 2.1.1 for electronic payments also apply for ITA.

There are few examples in the literature showing an explicit implementation of an ITA since the topic requires input from both, the agent community and the EPS community. Unfortunately, such joint work is rather scarce so far.

Nevertheless, in [MvS97], a particular ITA scheme based on regular credit card payment is presented. The general scheme described on that paper is depicted in Figure 2.5 and is recalled below.

This scheme solves the issue of agent protection against malicious hosts discussed above by using what they call an Agent Repository (AR). Using a distributed-object technology, the agent physically remains in the protective boundaries of the AR but logically it still roams from one server to the next. In this way, the credit card number of the buyer carried by the agent is stored in a safe place. The description of the scheme can be summarized as follows:

Protocol 1 (Van der Merwe-Von Solms)

1. The buyer identifies herself to the AR in order to instruct the agent.

2. The buyer gives the instructions to the agent regarding the goods she is interested in.

3. The ITA roams to the first server (S1).

4. Once the ITA finds an item that fits the requirements specified in Step 2, it starts the transaction via the Authorization Server (AS), which is typically a financial institution such as a bank.

   (a) The agent sends to the server the credit card number of the buyer encrypted with the public key of the AS, so only the AS can read this information.
(b) The agent sends to the server a description of the items and prices agreed upon with the server. This information is signed with the private key of the agent and encrypted with the public key of the AS, so the information can only be read by the AS and can only be written by the agent.

(c) The server writes a message with the same information agreed upon with the agent. He signs this information with his private key and encrypts the result with the public key of the AS. Then he forwards to the AS the two messages received from the agent together with his own message.

(d) The AS decrypts and authenticates all messages. He checks that both the descriptions of the goods and the prices are the same and in that case, he executes the payment protocol to transfer the money to the sellers account.
5. The agent roams to the next server.

6. The buyer tells the agent to come back to the AR. This can be done automatically after some conditions given by the buyer in Step 2 are met.

7. The agent returns to the AR.

8. The buyer checks what the agent has bought.

The main shortcomings that can be identified in most of the schemes based on credit card payments are the following:

- The agent makes the payment using a conventional credit based payment system. This fact implies:
  - the users’ credit card number has to be included in the agent code. This increases the confidentiality of the data carried by the agent. If this information is stolen, the agent user’s security is compromised.
  - the scheme proposed needs the bank on-line authorization for each payment.
  - user anonymity is not preserved since it can be traced back through the credit card number.

- The agent is assumed to be blindly trusted by the user. This might not be realistic if the agent is a ready-made program written by a third party.

In Chapter 6 a new proposal for an ITA is made. The proposal described uses recent advances in off-line EPS and contract signature protocols to avoid the above shortcomings.
Chapter 3

Electronic Payments

As mentioned in Chapter 2 our research line in electronic payments has been focused on micropayments. More precisely, we are interested in schemes that use hash chains to implement micropayments.

In all micropayment schemes based on hash chains proposed so far the lack of flexibility is the main problem. At least, the following two drawbacks can be identified:

- All coupons of the hash chain must have the same value \( v \), so this value \( v \) determines the minimum amount the micropayment system is able to deal with. In order to pay any amount, \( v \) should be minimal, for example one cent. But this means that the merchant must verify fifty hash functions to get paid a sum as small as fifty cents.

- The fact that \( v \) is fixed also hampers multicurrency micropayments. So fixed-value coupons do not allow to deal with different currencies.

In this Chapter we present a new micropayment scheme [DFHJ99] that offers more flexibility.

In Appendix B the implementation of spending programs using smart cards is discussed. In particular, a comparison between different Java cards is presented with the aim of identifying the best candidate to implement the proposed micropayment scheme.
3.1 Enhanced hash chains

The hash chain idea was first published in [Lam81] applied to password authentication. The main property of hash chains is connectivity between contiguous elements of the chain. Without knowledge of the preceding link, the next one cannot be obtained as long as the hash function used is computational one-way and collision-free. But the implementation of hash chains as described in Chapter 2 presents the problem that no information can be included in each link of the chain. Since each element of the chain is the image of a hash function it cannot include any information because a hash function is almost a random oracle. Thus, information contained in each link of a hash chain has to be agreed in advance and independently of the hash chain itself. For instance, in the micropayment schemes described in Chapter 2 it is agreed that each element of the chain will be a coupon of a fixed value $v$.

One way of adding information to each hash chain link is to define what we call an enhanced hash chain.

**Definition 1 (Enhanced hash chain)** An enhanced hash chain is a chain which preserves the authentication properties of a hash chain and allows each link of the chain to carry some arbitrary information in addition to its authenticity.

There are different ways of obtaining enhanced hash chains. One of them is described in the next proposition.

**Proposition 1** Let $F$ be a one-way hash function. Let $V_i$ the value included in the $i$-th link once a redundancy operation has been applied. Then following expression

\[
T_0 = F(T_1) \oplus V_1 \\
T_1 = F(T_2) \oplus V_2 \\
\vdots \quad \vdots
\]
\[ T_{n-1} = F(T_n) \oplus V_n \]
\[ T_n = F(V_n) \]

represents an enhanced hash chain.

**Proof:** We will show that such a structure preserves the hash chain dependence between every element \( T_i \) and the preceding \( T_{i+1} \). Furthermore, we will show that the value \( V_i \) contained in the \((i-1)\)-th link can include an arbitrary value, for any \( i \).

Given \( T_{i-1} \), \( V_i \) can be retrieved with knowledge of \( T_i \) by applying \( F \) in the following way:

\[ T_{i-1} \oplus F(T_i) = F(T_i) \oplus V_i \oplus F(T_i) = V_i \]

Redundancy in \( V_i \) allows to tell it from junk (integrity check). Since a hash function is approximately a random oracle, \( V_i \) can be viewed as a plaintext encrypted under a random key \( F(T_i) \) using a one-time pad cipher; therefore, without knowledge of \( T_i \), \( V_i \) cannot be recovered. On the other hand, \( T_i \) cannot be obtained from \( T_{i-1} \) because \( F \) is a one-way and collision-free hash function.

Finally, for any choice of \( V_1, \ldots, V_n \), an enhanced hash chain can be built by letting \( T_n := F(V_n) \), and then \( T_{i-1} := F(T_i) \oplus V_i \), for \( i = n \) down to 1. \( \diamond \)

For implementation purposes, we mention that hash functions like MD5 [RD92] or SHA [oST95] are good candidates for \( F \) since they are one-way and collision-free.

The advantages of enhanced hash chains are basically a greater flexibility resulting from the fact that every link includes its own value.

One interesting way of using enhanced hash chains is structural coding [DF91a, DF91b]. The information contained in a link is used to store a program instruction. In this way, the enhanced hash chain described in
Proposition 1 can be used to store a sequential program. To obtain the instruction contained in every link, each element $T_i$, called trace, has to be decoded using the image of the preceding one by the hash function. In this way, the executor of the program can be sure that the program has not been altered and exactly corresponds to the program coded by the programmer, the one that authenticates the first trace, $T_0$. To that end, an authenticator $t_0$ can be given with $T_0$ which is a digital signature of $T_0$ using the private key of the programmer.

Enhanced hash chains described in Definition 1 can only be used for sequential programs. That means programs that are executed sequentially one instruction after the other. For programs with forward and backward branch instructions the enhanced hash chain is not suitable. Structural coding, recalled in Appendix A, deals with all kinds of program instructions extending the hash chain concept further than enhanced hash chains do.

3.2 Spending programs

To provide further flexibility to hash-based micropayments we define a new concept, the spending program, which in turn generalizes enhanced hash chains:

**Definition 2 (Spending program)** A spending program $i_1, \ldots, i_n$ is a program whose instructions $i_k$ are either value instructions, flow-control instructions, input-output instructions or assignment instructions.

**Definition 3 (Value instruction)** A value instruction is one that carries a specific sum of money in a currency specified in the same instruction. When a value instruction of a spending program is retrieved by the merchant, the corresponding sum of money is spent by the buyer.

**Definition 4 (Flow-control instruction)** A flow-control instruction allows to modify the flow of a spending program. Four types of flow-control instructions are used:
1. **Forward unconditional branch**

2. **Forward conditional branch**

3. **Backward unconditional branch**

4. **Backward conditional branch**

If $i_k$ is a branch to instruction $i_j$, “forward” means that $k < j$ and “backward” that $k \geq j$. Backward branches allow instruction blocks to be executed more than once.

Input-output and assignment instructions are analogous to machine language instructions of the same type.

Clearly, a spending program generalizes the hash chain concept for micropayments implemented by equations (2.1), since the value instructions are in fact coupons of arbitrary value. An essential issue is how to encode spending programs such that value instructions in an encoded spending program are as unforgeable as coupons in a hash chain.

As discussed in Section 3.1 structural coding was originally applied to the program integrity problem. The coding is called structural because it depends on the flow-control structure of the program. So we use structural coding to implement spending programs. The following protocol shows all the process to create, encode and use a spending program, and in doing so defines a new electronic micropayment scheme.

**Protocol 2 (Spending program micropayment)**

1. **[Writing]** The buyer writes the (unencoded) spending program $i_1, \cdots, i_n$ following her own taste. The buyer’s computer or smart card can be used to edit the spending program. Alternatively, “ready-made” standard spending programs supplied by the buyer’s bank or other institutions can be used.
2. [Coding] Spending program instructions $i_1, \ldots, i_n$ are encoded by the buyer into a sequence of so-called traces $T_0, T_1, \ldots, T_n$ using the structural coding [DF91a] based on a one-way hash function $F$ such that in general $F(X \oplus Y) \neq F(X) \oplus F(Y)$, where $\oplus$ denotes bitwise exclusive OR (MD5 [RD92] or SHA [oST95] are good candidates for $F$). Coding could also be performed by the buyer’s computer or smart card.

3. [Signature] The first trace $T_0$ is signed by the buyer or the buyer’s card; let $t_0$ be the signed form of $T_0$. The buyer also signs an upper bound $\alpha_{t_0}$ of the amount that can be spent using the spending program that starts with $T_0$.

4. [Initialization] The buyer sends $t_0$ and the signed $\alpha_{t_0}$ to the merchant, who uses a standard payment protocol to obtain on-line authorization for the whole spending program. Note that authorizing an upper bound of the spendable amount does not mean that the buyer is paying anything; payment will be done when value instructions of the spending program are sent by the buyer to the merchant.

5. [Microspending] Before executing an instruction $i_k$, it must be decoded from the corresponding trace $T_k$. Decoding is only successful if no modifications have been done to the program in the traces preceding $T_k$ (run-time integrity property, see Appendix A). In [DF91a], a preprocessor to decode traces was assumed to be pipe-lined and encapsulated with the main processor executing instructions. For spending programs, the procedure is slightly different:

(a) When a value instruction (coupon) is to be paid by the buyer to the merchant, the corresponding trace is sent by the buyer to the merchant, maybe after some flow-control and input-output intermediate traces.

(b) The merchant decodes the instructions in the traces received, to check whether these are valid traces. In particular, if the trace
corresponding to the value instruction is correctly decoded, then payment for that value is accepted by the merchant.

6. [Clearing] The merchant relays the traces received (either one by one or in batches to save transaction costs) to his bank. The bank decodes traces the same way the merchant did, to ensure they are valid (the merchant could try to forge non-existing micropayments). For each correctly decoded instruction value, the bank credits the merchant the corresponding value.

Note 2 The standard payment system referred to at Step 4 could be for example SET or iKP. For instance, to use iKP, $t_0$ and $\alpha_b$ would be placed in the COMMON field defined by that protocol (this is a field containing information shared by all parties involved in the payment, see [BGH+95] for more details). This approach is the same adopted by NetCard and $\mu$-iKP, which rely on SET, respectively iKP, for authorization of hash chains.

3.2.1 Non-iterative spending programs

Since spending programs contain value instructions than can be accepted as payments, an important distinction is whether instructions can be executed more than once.

Definition 5 (Non-iterative spending program) A spending program $i_1, \ldots, i_n$ is called non-iterative if it contains no backward branches, i.e., if no instruction can be executed more than once.

Thus in a non-iterative spending program, value instructions cannot be re-used. This simplifies matters, because the bare knowledge by the merchant of a trace encoding a value instruction is enough for the bank to credit the merchant the corresponding value. We next give an example of sequential spending program with value instructions for several different values and/or currencies. In fact, the example below is the simplest kind of spending program, actually an enhanced hash chain.
Example 1 The structural coding corresponding to a sequential block of \( n \) value instructions with values \( v_1, v_2, \ldots, v_n \) is next given (sequential block means that no flow-control instructions are present). Let \( V_i \) be the amount \( v_i \) with some redundancy (in fact the currency name can be used as redundancy). Let \( F \) be a one-way hash function as specified in Step 2 of Protocol 2 and let \( \oplus \) denote bitwise exclusive OR. Then the buyer computes the trace sequence \( T_n, T_{n-1}, \ldots, T_1, T_0 \) as follows:

\[
\begin{align*}
T_0 &= F(T_1) \oplus V_1 \\
T_1 &= F(T_2) \oplus V_2 \\
& \vdots \\
T_{n-1} &= F(T_n) \oplus V_n \\
T_n &= F(V_n)
\end{align*}
\]

After the above computation, the buyer signs \( T_0 \) to get \( t_0 \). Before the first micropayment, the buyer sends \( t_0 \) and the signed value \( a_t = \sum_{i=1}^{n} v_i \) to the merchant for authorization. Micropayments of values \( v_1, v_2, \ldots, v_n \) can be then made by successively revealing \( T_1, \ldots, T_n \) to the merchant. For instance, when the merchant gets \( T_1 \), then it can retrieve \( v_1 \) by computing \( F(T_1) \oplus T_0 = V_1 \). The same check is performed by the merchant’s bank before crediting \( V_1 \) to the seller’s account. \( \diamond \)

The same structure presented in Example 1 could accommodate multicurrency micropayments. For example, \( v_1 \) could be in euros, \( v_2 \) in dollars, etc. Of course, with a sequential block, this means that euros should be spent first and dollars second, which is rather rigid. Anyway, current micropayment systems offering only fixed-value coupons are worse in that they must rely on an intermediate broker to deal with multiple currencies; this has at least two drawbacks:

- The broker must be trusted
- The broker cannot be expected to do the currency conversion for free
A solution to increase the flexibility offered by the spending program of Example 1 is to use input-output and forward flow-control instructions (forward branches). In this way, the buyer could provide input at transaction-time which would be used to select some value instructions and skip some others. Note that such dynamical selection frees the buyer from knowing before signing the first trace which value instructions he will use. For example, if multi-currency value instructions are being used, the selected currency would be a transaction-time input. Note that skipping value instructions does not mean losing money because money is only spent when the merchant retrieves a value instruction. See Appendix A on how to encode forward branches while ensuring instruction unforgeability.

A second advantage of forward flow-control is that we can include in the spending program value instructions for larger sums (which can be optionally used instead of “micro-value” instructions). In this way, the distinction between micropayments and standard payments becomes rather fuzzy.

### 3.2.2 Iterative spending programs

Iterative spending programs are programs in which some instructions can be executed more than once.

**Definition 6 (Iterative spending program)** A spending program $i_1, \cdots, i_n$ is called iterative if it contains at least a backward branch, i.e. if some instructions can be executed more than once.

Clearly, iterative spending programs add new flexibility advantages:

**Storage reduction** For example, an arbitrary number of value instructions of equal value $v$ can be stored as a two- or three-instruction loop using a backward branch.

**Complex spending patterns** Case-like structures can be created in a spending program so that value instructions can be repeatedly selected or skipped as a function of the buyer’s transaction-time input.
Just as clearly, iterative spending programs raise a new security problem. Since value instructions can be re-used, once the merchant has completed a whole loop from an iterative spending program, he could present the value instructions in that loop to his bank an arbitrary number of times pretending to be credited more than he is entitled to.

A way to repair the above security flaw is to require that some additional information should accompany the value instructions in a loop if they are used more than once. Specifically, we propose that a Kerberos-like ticket [NT94] be sent by the buyer to the merchant along with a re-used value instruction. The following protocol can be run in parallel to Protocol 2:

**Protocol 3 (Ticket usage)**

1. **[Initialization]** To use Kerberos tickets, the buyer and the merchant’s bank must share a symmetric encryption key $K$. Such key can be agreed upon by the buyer and the merchant’s bank using their public-key certificates at Step 4 of Protocol 2 (initialization). It should be noted that, since micropayment assumes stability in buyer-merchant relationship, the overhead of a key exchange between the buyer and the merchant’s bank should be affordable.

2. **[Microspending]** Each time a value instruction is re-used (and this happens for instructions in a loop always but the first time), a ticket must be sent by the buyer along with the instruction. The ticket has the form $E_K(T||t)$, where $T$ is the trace corresponding to the value instruction, $t$ is a time stamp, $||$ denotes concatenation and $E_K(\cdot)$ denotes encryption using a symmetric encryption algorithm with key $K$.

3. **[Clearing]** To clear a re-used value instruction, the merchant must send it to his bank along with a fresh ticket, i.e. a ticket that was not used in any previous clearing of the same value instruction. Freshness is ensured by the timestamp $t$ in the ticket. The bank will not accept a re-used value instruction unless it is linked to a fresh ticket.
Note 3 (On clearing tickets) Clearing tickets one by one may cost too much as compared to the value of the associated value instruction. On the other hand, if the merchant chooses to clear tickets by batches, then he incurs in some risk of being paid with invalid tickets (i.e. not being paid at all). Note that tickets cannot be understood by the merchant (K is unknown to him), so the only way for the merchant to check the validity of a ticket is to try to have it cleared. Clearly there is a tradeoff between cost-effectiveness and security. It is up to the merchant’s discretion to find the optimal strategy to solve this (economic) problem.

Note 4 (On coding backward branches) The structural coding used to encode instructions into traces requires that the targets of backward branches be signed (see Appendix A). So each new iteration of a loop requires a run-time signature verification. This may be a computational burden if loops are too short, i.e. if they contain too few value instructions. Again, an economic tradeoff between verification costs and the flexibility offered by short loops is to be solved.
Chapter 4

Copyright Protection: Symmetric Schemes

In Chapter 2 we pointed out the significance of copyright protection for electronic commerce expansion, especially for products supported by digital format like multimedia contents.

In this chapter we focus on symmetric schemes to protect images. Remember that a symmetric copyright protection scheme is one where both the buyer and the merchant know the marked image, but only the merchant knows the exact position of the mark bits inside the image. Although symmetric schemes do not offer security for the buyer (see Section 2.2.1 for a description of properties of a copyright protection scheme), they currently attract the majority of researchers, probably because non-symmetric schemes appeared so far need a computational power too high to be implemented in practical terms.

In the field of image copyright protection different schemes can be found. Some of them are research prototypes [HRP97] but some others [Cor, Lim] are available for commercial use.

We present in this chapter two new copyright protection schemes for images: a robust watermarking scheme and a symmetric fingerprinting scheme which is an improved version of the first one (see Section 2.2.2 for taxonomy
on classification of copyright protection schemes).

4.1 A robust watermarking scheme

This section describes a robust watermarking scheme for images [DFHJS00], which is an improved version of a previous proposal [DFHJ00c]. The key feature of the scheme is the use of the JPEG compression algorithm [Wal91] to detect the pixels of the image that are fit to embed a bit of the mark. This idea was also used in [LPZ99] but our proposal is simpler and more robust.

The scheme can be described in two stages: mark embedding and mark reconstruction. As we mention before, the watermarking scheme presented is symmetric so mark embedding is entirely performed by the merchant $M$, who cannot prove in court that he recovered a mark from a redistributed copy rather than from a legal one ($M$ knows all marked copies he sells from embedding time and could simulate a redistribution).

4.1.1 Mark embedding

For mark embedding, we assume that the image allows sub-perceptual perturbation. The sub-perceptual threshold can be measured using a Peak Signal-to-Noise Ratio (PSNR, [KP99]) which is defined in the following way.

**Definition 7** The Peak Signal-to-Noise Ratio (PSNR) between the original image $X$ and the marked one $\overline{X}$ is defined by:

$$PSNR(X|\overline{X}) = W \cdot H \frac{\max_{w,h} x_{w,h}^2}{\sum_{w,h} (x_{w,h} - \overline{x}_{w,h})^2}$$

where $1 \leq w \leq W$, $1 \leq h \leq H$, $x_{w,k}$ and $\overline{x}_{w,k}$ are the original and marked pixel values, respectively, and $W$ and $H$ are the width and height of the image.

Let $q$ be a JPEG quality level chosen in advance by the merchant $M$; $q$ will be used as an information rate parameter. Also, let $p$ a PSNR parameter
also chosen in advance by the merchant; \( p \) will be used as an imperceptibility parameter, i.e. \( M \) requires that the embedding process does not bring the PSNR below \( p \) dB. Let \( \{s_i\}_{i \geq 1} \) be a random bit sequence generated by a cryptographically sound pseudo-random generator with secret key \( k \) only known to \( M \). From now on, despite notation used in Definition 7, an image \( X \) will be represented as \( X = \{x_i : 1 \leq i \leq n\} \), where \( n \) is the number of pixels and \( x_i \) is the color level of the \( i \)-th pixel. For monochrome images, \( n = W \times H \), where \( W \) and \( H \) are, respectively, the width and height of the image. For RGB color images, \( n = 3(W \times H) \) (there are three matrices rather than one).

**Algorithm 1 (Mark embedding(p,q))**

1. Compress \( X \) using the JPEG algorithm with quality \( q \) as input parameter. Call the bitmap of the resulting compressed image \( X' \). Let \( \delta_i := x_i - x_i' \) be the difference between corresponding pixels in \( X \) and \( X' \). Only positions \( i \) for which \( \delta_i \neq 0 \) will be usable to embed bits of the mark.

2. Call \( \varepsilon \) the mark to be embedded. Encode \( \varepsilon \) using an error-correcting code (ECC) to obtain the encoded mark \( E \); call \( |E| \) the bit-length of \( E \). Replicate the mark \( E \) to obtain a sequence \( E' \) with as many bits as pixels in \( X \) with \( \delta_i \neq 0 \).

3. Let \( j := 0 \). For \( i = 1 \) to \( n \) do:

   (a) If \( \delta_i = 0 \) then \( \overline{x}_i := x_i \).

   (b) If \( \delta_i \neq 0 \) then

      i. Let \( j := j + 1 \). Compute \( s_j' := e_j' \oplus s_j \), where \( e_j' \) is the \( j \)-th bit of \( E' \). The actual bit that will be embedded is \( s_j' \).

      ii. If \( s_j' = 0 \) then compute \( \overline{x}_i := x_i - \delta_i \).

      iii. If \( s_j' = 1 \) then compute \( \overline{x}_i := x_i + \delta_i \).
4. While $PSNR(X|\overline{X}) < p$ do

   (a) Randomly pick an index $i$ such that $1 \leq i \leq n$.
   (b) If $\overline{x}_i - x_i > 3$ then $\overline{x}_i := \overline{x}_i - 1$.
   (c) If $\overline{x}_i - x_i < -3$ then $\overline{x}_i := \overline{x}_i + 1$.

$\overline{X} = \{\overline{x}_i : 1 \leq i \leq n\}$ is the marked image, which yields at least $PSNR(X|\overline{X}) = p$ dB. The use of the value 3 when adjusting the PSNR is empirically justified: this is the minimal magnitude that reasonably survives the attacks considered in the next section. The influence of $q$ on capacity and robustness is discussed in Section 4.2.

### 4.1.2 Mark reconstruction

The scheme proposed here falls into the semi-public category defined in Subsection 2.2.2; for mark reconstruction, knowledge of the original image $X$ and the secret key $k$ is assumed ($k$ is used to regenerate the random sequence $\{s_i\}_{i \geq 1}$) but knowledge of the original mark $\varepsilon$ is not needed (otherwise it would be classified as a private scheme). Note that assuming knowledge of the original image in mark reconstruction is quite realistic in the case of symmetric marking algorithms, where mark reconstruction is performed by and for the merchant $M$.

**Algorithm 2 (Mark reconstruction)**

1. Upon detecting a redistributed item $\hat{X}$, restore it to the bitmap format.

2. Compress the corresponding original image $X$ using JPEG with quality $q$ to obtain $X'$.

3. Let $j := 0$. For $i = 1$ to $n$ do:

   (a) Compute $\delta_i := x_i - x'_i$. 


(b) If \( \hat{x}_i \neq 0 \) then
   
   i. Let \( j := j + 1 \). If \( j > |E| \) then \( j := 1 \).
   
   ii. Compute \( \hat{s}_i := \hat{x}_i - x_i \).
   
   iii. If \( \hat{s}_i = 0 \) then \( \hat{s}_j := \# \), where \( \# \) denotes erasure.
   
   iv. If \( \hat{x}_i \times \hat{x}_i > 0 \) then \( \hat{s}_j := 1 \).
   
   v. If \( \hat{x}_i \times \hat{x}_i < 0 \) then \( \hat{s}_j := 0 \).
   
   vi. If \( \hat{s}_j \neq \# \) then \( \hat{e}_j := \hat{s}_j \odot s_j \); otherwise \( \hat{e}_j := \# \).
   
   vii. If \( \hat{e}_j = 1 \) then \( \text{ones}_j := \text{ones}_j + 1 \).
   
   viii. If \( \hat{e}_j = 0 \) then \( \text{zeroes}_j := \text{zeroes}_j + 1 \).

4. For \( j = 1 \) to \( |E| \) do:

   (a) If \( \text{ones}_j > \text{zeroes}_j \) then \( \hat{e}_j := 1 \), where \( \hat{e}_j \) is the \( j \)-th bit of the recovered mark \( \hat{E} \).

   (b) If \( \text{ones}_j < \text{zeroes}_j \) then \( \hat{e}_j := 0 \).

   (c) If \( \text{ones}_j = \text{zeroes}_j \) then \( \hat{e}_j := \# \).

5. Decode \( \hat{E} \) with the same ECC used for embedding to obtain \( \hat{e} \).

Note that the redistributed \( \hat{X} \) may have width \( \hat{W} \) and height \( \hat{H} \) which differ from \( W \) and \( H \) due to manipulation by the redistributor; this would cause the number of pixels \( \hat{n} \) in \( \hat{X} \) to be different from \( n \). In the next section, we discuss how to deal with attacks altering \( n \).

### 4.2 Properties of the watermarking scheme

In subsection 2.2.1 different properties are defined for a copyright protection scheme. We are going to analyze the ones that refer to watermarking. That means that neither robustness against collusion attacks, nor security for the buyer, nor anonymity will be considered.
To test its properties, the scheme described in the previous section was implemented using a dual binary Hamming code $DH(31,5)$ as ECC (See section 4.3 for a justification of such a choice).

An important consideration to test any image processing algorithm is the images used for the test. It is impossible to get an exhaustive list of classes of pictures and stock photo companies have a lot of difficulties to set up a satisfactory index. However one can at least retain the main themes that are common among these images and that are used very often in the press in order to keep a wide range of kind of pictures: colors, textures, patterns, shapes, lightning. So, our test images were taken from http://www.cl.cam.ac.uk/~fapp2/watermarking/benchmark/image_database.html a commonly used image database for watermarking tests which contains 29 images divided in different groups depending on their characteristics (colors, textures, lines & edges, etc.).

Note that the scheme presented in the previous section is designed to obtain good properties. For that reason, the marking process takes as input parameters provided by the user: the imperceptibility degree $p$ and a parameter $q$ related to the information rate. We also discuss below the implication that a particular choice can have on other properties, particularly on robustness.

### 4.2.1 Imperceptibility

This property is mainly controlled by the $p$ parameter of the marking algorithm that fixes the PSNR value of the watermarked image. The $q$ value determines the number of pixels that are fit to be marked (the lower $q$, the more pixels) but imperceptibility is not altered even if $q$ is low since the correction of the marking process caused by $p$ value still leaves the mark imperceptible.

As an example, Figure 4.1 shows the original Lena image and Figure 4.2 shows Lena marked with a 70-bit mark, $p = 38dB$ and $q = 60\%$.

In addition to PSNR, quality metrics such as the ones in [EF95, NAdS92]
4.2 Properties of the watermarking scheme

![Original Lena image (512 x 512)](image)

Figure 4.1: Original Lena image (512 x 512)

can be used to measure the mark imperceptibility.

4.2.2 Robustness

The base test of the StirMark 3.1 benchmark [KP99, PAK98] was used to evaluate robustness. StirMark is a software developed at Cambridge University that performs different subperceptual image distortions (JPEG compression, geometric transformations, noise addition,....) on a given image. The mark reconstruction algorithm is applied to the modified obtained images so robustness can be tested.

To test robustness property, the following values recommended in [PA99] were taken: $p = 38dB$ and a 70-bit mark $\varepsilon$ (resulting in an encoded $E$ with
$|E| = 434$ bit after a DH(31,5) code is chosen as the ECC in the marking algorithm. The input value of the mark embedding algorithm $q$ has been set at $60\%$. These values have been used in the image shown in Figure 4.2.

On the images from the database mentioned above, the embedded mark survived the following StirMark manipulations$^1$:

1. Color quantization.

2. All low pass filtering manipulations. These include the following linear and non-linear filters:

$^1$The reader is referred to [HS93] for an accurate description on basic concepts of image processing
(a) Gaussian filter (blur).
(b) Median filter (2 × 2, 3 × 3 and 4 × 4).
(c) Frequency mode Laplacian removal [BP98].
(d) Simple sharpening.

3. JPEG compression for qualities 90% down to 30%. In Figure 4.3 a 30% of JPEG compression has been applied. To resist down to 10% it is necessary to embed the mark using $q = 5$. Although using such $q$ value does not allow $p = 38$ to be reached (we get $PSNR = 28$), the image quality is good enough as can be seen on Figure 4.4.

![Figure 4.3: A watermarked Lena image compressed at 30% JPEG](image)

4. Rotations with and without scaling of $-0.25$ up to $0.25$ degrees.
5. Shearing up to 1% in the $x$ and $y$ axes.

6. All StirMark cropping attacks. Figure 4.5 shows an example of two of the StirMark cropping attacks. The mark resists these attacks because it is repeatedly embedded in the image (region-based marking). To resynchronize, the reconstruction algorithm keeps shifting the cropped $\hat{X}$ over $X$ until the the “right” relative position of $\hat{X}$ on $X$ is found. The right position is estimated to be the one that, after running Algorithm 2 on $X$ and the corresponding cropping of $\hat{X}$, yields the minimal number of corrected errors at Step 5.
7. Removal of rows and columns from the marked image \( \hat{X} \) was automatically detected, dealt with, and survived exactly like cropping attacks.

Additional rotation, scaling and shearing StirMark attacks can be detected and undone by \( M \) prior to mark reconstruction by using computer vision techniques to compare with the original image. The really dangerous attacks for the scheme presented here are random geometric distortions and attacks combining several of the aforementioned elementary manipulations.

Note that the region-based marking property offers different possibilities to reconstruct an embedded mark. Only uniform attacks over the image are really dangerous. Local attacks on the image can be detected by dividing the image into different parts, for instance, as small as the one presented in Figure 4.5.

### 4.2.3 Information rate

In Chapter 2 the information rate of a copyright protection scheme is defined as total information amount that can be embedded in the image without altering its imperceptibility. In the scheme presented, the information rate,
measured as the number of bits that can be embedded, is basically determined by the number of pixels that can be marked. This value depends on the $q$ parameter specified in the marking algorithm. Since $q$ determines a JPEG compression level, the total number of different pixels between the original image and the $q\%$ compressed image (those with $\delta_i \neq 0$) depends on the image properties (size, smoothness, etc.). The test performed over all color images from the image database used gives the results shown in Table 4.1. The table presents the percent of the $n$ pixels that can be used to convey a mark bit for each JPEG quality compression value $q$.

<table>
<thead>
<tr>
<th>$q$</th>
<th>% of marking pixels</th>
</tr>
</thead>
<tbody>
<tr>
<td>90</td>
<td>78.89</td>
</tr>
<tr>
<td>80</td>
<td>82.37</td>
</tr>
<tr>
<td>70</td>
<td>84.64</td>
</tr>
<tr>
<td>60</td>
<td>85.44</td>
</tr>
<tr>
<td>50</td>
<td>86.04</td>
</tr>
<tr>
<td>40</td>
<td>87.51</td>
</tr>
<tr>
<td>30</td>
<td>87.73</td>
</tr>
<tr>
<td>20</td>
<td>91.10</td>
</tr>
<tr>
<td>10</td>
<td>90.69</td>
</tr>
</tbody>
</table>

Table 4.1: Average percent of pixels that can be marked in an image.

For example, in the Lena image (size is $512 \times 512$), up to 235,126 pixels can be marked with $q = 60$, which represents 89.69\% of the total number of pixels. Assuming a color image, each pixel can be used to embed 3 bits, one in each RGB component. So Lena allows 705,378 bits to be embedded without affecting the image quality. Of course, embedding such number of bits of net information implies that it is not possible to apply neither ECC nor region-based watermarking, since there are no bits left for redundancy. This fact implies important reductions regarding the robustness of the scheme. Fortunately, it is not necessary to embed a 705 Kbit mark. The use of an ECC reduces such an amount; for instance, using a dual binary Hamming code DH(31,5) as ECC, the amount of bits is reduced by a $5/31$ factor yielding
113,770 bits. This implies that our 70-bit mark has been included 1.625 times in the image to obtain a region based watermarked image.

### 4.2.4 Secret information

We pointed out in Chapter 2 that, in any copyright protection scheme, the original image is supposed to be secret; otherwise redistribution of illegal copies of the original image cannot be detected since they are not marked. Furthermore, in the scheme presented so far even knowledge of the marking algorithm (Kerckhoff’s principle [Ker83]) is not enough to detect mark bits since their location depends on the $q$ value. Moreover, the exact value of each mark bit is determined by the key $K$. Notice also that the value $q$ can be made public because, despite the knowledge of that value, the original image is needed to obtain the positions of the mark bits and this image is assumed to be secret.

In a semi-public watermarking algorithm where the original image is needed during mark reconstruction the need for a secret key is not clear. Since the original image is secret, no one can obtain the mark, even with knowledge of the key: yet notice that it is easier to guess the original image from a marked version of it than to guess a purely random key. Public watermarking algorithms are different because the key is the only information needed to retrieve the mark. So once the key is known, the mark can be extracted from the marked image without the original one. The existence of a secret key $K$ in a semi-public algorithm is mainly justified by fingerprinting applications. If fingerprinting without key is used, an attacker having stolen the original image can, in addition to redistribute this image, embed the mark of any buyer. Then the merchant would wrongly accuse some honest buyer that has been framed by the attacker with the original image.
<table>
<thead>
<tr>
<th>number of re-marked process</th>
<th>PSNR w.r.t original image</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>38</td>
</tr>
<tr>
<td>2</td>
<td>35.1</td>
</tr>
<tr>
<td>3</td>
<td>33.4</td>
</tr>
<tr>
<td>4</td>
<td>32.28</td>
</tr>
<tr>
<td>5</td>
<td>31.37</td>
</tr>
<tr>
<td>6</td>
<td>30.62</td>
</tr>
<tr>
<td>7</td>
<td>30.04</td>
</tr>
<tr>
<td>8</td>
<td>29.5</td>
</tr>
<tr>
<td>9</td>
<td>29.06</td>
</tr>
<tr>
<td>10</td>
<td>28.7</td>
</tr>
</tbody>
</table>

Table 4.2: PSNR of the images after re-marking process

4.2.5 Multiple watermarking

This property\(^2\) is important for environments where different brokers deal with the image and each of them wants to embed his own mark on it.

The proposed algorithm allows multiple marking. The mark reconstruction algorithm works properly and reconstructs the different marks using each time as original image the pre-marked one.

The imperceptibility of the obtained images is still good enough. In Table 4.2 the values of the PSNR of ten successive markings of Lena with \(p = 38 dB\) are given (PSNR are with respect to the original image).

Figure 4.6 shows the exact output of a 5 times marked Lena with a \(PSNR = 31.37\).

It is worth noting that multiple marking does not reduce the robustness of each individual marking.

\(^2\)Although in Chapter 2 this property was included in the robustness category, in this section we consider it separately for clarity.
4.3 Symmetric collusion-secure fingerprinting scheme

Upgrading a watermarking scheme to a symmetric fingerprinting scheme may seem an easy task: the merchant embeds on the image different information for each different buyer. But this simple approach does not always yield a useful symmetric fingerprinting scheme since several problems can be encountered. In particular, if the watermarking scheme is private or a semi-public (needing the mark embedded for the reconstruction process), it cannot be upgraded. The reason is that using such schemes for fingerprinting once a copy is found, the merchant does not know a priori which buyer is the guilty
one and he should try all the buyers’ marks during the mark reconstruction process.

Another problem with this simple upgrade model is the fact that robustness in watermarking does not consider collusion attacks. Since in watermarking schemes all the buyers receive the same copy with the same information embedded, knowledge of more than one copy does not leak information about the mark bits position. For that reason collusion attacks do not make sense in watermarking schemes.

But when a watermarking scheme is upgraded to a symmetric fingerprinting scheme different information is embedded on different image copies. That means different users have different copies and they can compare them to detect a mark bit position (as an example see below in this section Attack 1 for the watermarking scheme defined in Section 4.1).

A possible solution for this problem is to define different bit mark positions for different buyers. In this way such attacks cannot be performed. The problem in that case appears in the mark reconstruction process. Once an image copy is found illegally redistributed, the merchant has to extract the buyer identity. If every buyer identity has been embedded in different positions, the merchant has to look for the mark in different position again for every buyer until the guilty one is found. Such search is impractical if the number of buyers is too large.

Although the upgrade from a watermarking scheme to a symmetric fingerprinting scheme is not as simple as it may seem, different symmetric fingerprinting schemes use watermarking schemes as a building block. If it is assumed that the underlying watermarking scheme is robust then the only remaining issue for the fingerprinting scheme are collusion attacks. In this particular context, the marking assumption [BS95] approximately holds (the more robust the underlying watermarking algorithm, the more exactly the marking assumption holds), so buyers can only detect a specific mark bit if it differs between their copies; otherwise a mark bit cannot be detected and thus it cannot be removed.
So, we will upgrade the watermarking scheme presented in Section 4.1 to a symmetric fingerprinting scheme by relying on the marking assumption. In other words, we will focus only in the collusion attack:

**Attack 1** Two or more different buyers can detect the mark bits by comparison of their copies of the same image, because both the q value that determines the pixels to be marked and the key k that generates the sequence $s_i$ for a given image are the same for all buyers. After detection, there are two attacking strategies.\(^3\)

**Mark bit deletion** The mark bits that differ between buyers can be deleted by simply adding the pixels that embed them. Clearly, if two mark bits differ, then the corresponding pixels in the marked images are $x_i + \delta_i$ and $x_i - \delta_i$. Since $M$ knows that there must be an embedded bit in that position, the probability of correctly restoring a deleted mark bit is $1/2$.

**Mark bit tweaking** The buyer having pixel $x_i + \delta_i$ replaces it with $x_i - \delta_i$ (or vice versa), which tweaks the embedded bit. This attack is worse than deletion, as $M$ has no way to detect that the bit was tweaked.

If the number of bits tweaked (or incorrectly restored) is greater than the maximum number of errors that the ECC can correct, then mark reconstruction will fail.

In what follows, we restrict ourselves to collusions of two buyers; given the danger inherent to piracy, the size of collusions tends to be small, so concentrating on collusions of size two is not unrealistic. In fact, collusions of size two are also dealt in [EC00] but our proposal is more efficient as shown below.

The effectiveness of Attack 1 depends on the distance between codewords of the ECC used in the mark embedding algorithm. The larger the distance,

\[^3\] The random effect of Step 4 in Algorithm 1 is not taken into account since we assume the worst case where the PSNR is correct in the first loop and so no random modification is performed.
the more bits will differ between the codewords of colluders, i.e. the easier to
tweak a codeword bit. On the other hand, the smaller the distance, the less
tolerance capacity will be obtained from the ECC. Thus, it is interesting to use an ECC whose distance between codewords is both lower and
upper-bounded. Dual binary Hamming codes are a natural choice [MS77],
because the distance between any two codewords is fixed: the distance \(d(x, y)\)
between any two codewords \(x, y\) of a dual binary Hamming code of length
\(N = 2^n - 1\) is \(2^{n-1}\).

For simplicity, in what follows we consider the codeword \(E\) that identifies a specific buyer once the ECC has been applied. The subsequent key-
dependent transformation of \(E\) to obtain the marked image is not relevant
for our discussion because the key \(k\) is the same for all buyers of the image.
The ECC used will be a dual binary Hamming code \(H\) with length \(2^n - 1\).
The distance between any two codewords in \(H\) is then \(2^{n-1}\), so that up to
\(2^{n-2} - 1\) errors can be corrected. The following definition introduces some
useful notation.

**Definition 8** Let \(a^1, \cdots, a^l\) be \(l\) codewords of length \(N\) (i.e. \(a^j = a^1_j a^2_j \cdots a^N_j\)).
The \(i\)-th position of the set of codewords \(a^1, \cdots, a^l\) is called invariant position if
\[
a^j_i = a^k_i \quad \forall j, k = 1, \cdots, l
\]
We denote by \(ivt(a^1, \cdots, a^l)\) the set of invariant positions of \(a^1, \cdots, a^l\). Also, we
denote by \(< a^1, \cdots, a^l >\) the set of words that can be generated by taking
as first bit one of \(a^1_1, \cdots, a^l_1\), as second bit one of \(a^1_2, \cdots, a^l_2\), and so on.

Regarding Attack 1, the following lemma holds.

**Lemma 1** Colluders owning \((a^1, \cdots, a^l)\) cannot tweak the bits at positions in \(ivt(a^1, \cdots, a^l)\).

**Proof:** Attack 1 assumes that colluders can tweak a bit only if such a
bit differs in their copies. Since bits at invariant positions have all the same
value, they cannot be changed. \(\diamond\)
The following properties of dual binary Hamming codes are stated for later use.

**Proposition 2** Let $H$ be a dual binary Hamming code with length $2^n - 1$. Then any three codewords of $H$ have exactly $2^{n-2} - 1$ invariant positions.

**Proof:** Let $x, y \in H$ be any pair of codewords. Define $I = \text{ivt}(x, y)$ and $\overline{I}$ the positions not in $I$. We denote by $x|_I$ the bits of the word $x$ in the positions in $I$. Since $d(x, y) = 2^{n-1}$ it follows that $|\text{ivt}(x, y)| = 2^{n-1} - 1$.

By construction, we will prove that given any codeword $z \in H$ and a value $k \leq 2^{n-1} - 1$, if $|\text{ivt}(x, y, z)| = k$ then $k = 2^{n-2} - 1$.

We have

$$d(x|_I, z|_I) = d(y|_I, z|_I) = 2^{n-1} - 1 - k$$

since $x|_I = y|_I$. Now, $x, y$ and $z$ belong to a dual binary Hamming code with length $2^n - 1$, so $d(x, z) = d(y, z) = 2^{n-1}$; therefore,

$$d(x|_\overline{I}, z|_\overline{I}) = d(y|_\overline{I}, z|_\overline{I}) = 2^{n-1} - (2^{n-1} - 1 - k) = k + 1$$

But $d(x|_\overline{I}, y|_\overline{I}) = 2^{n-1}$, so that the only possibility is

$$k + 1 = \frac{2^{n-1}}{2}$$

Otherwise, we would have

$$d(x|_\overline{I}, z|_\overline{I}) \neq d(y|_\overline{I}, z|_\overline{I})$$

since if $x_i \neq y_i$ then $z_i = y_i \Rightarrow z_i \neq x_i$ as we work in a binary domain ($x_i$ denotes the $i$-th bit of $x$). Then

$$k = 2^{n-2} - 1$$

$\diamondsuit$
Proposition 3  Let $x, y \in H$. If $z \in <x, y> \setminus \{x, y\}$, then the correction of $z$ using $H$ cannot yield a codeword different from $x$ and $y$.

Proof: Assume that $z \in <x, y>$ exists such that it decodes into a $z' \in H \setminus \{x, y\}$. Since the code $H$ corrects up to $2^{n-2} - 1$ errors, candidates to be $z$ are words with $\leq 2^{n-2} - 1$ errors. But $|ivt(x, y)| = 2^{n-1} - 1$ and $|ivt(x, y, z)| = 2^{n-2} - 1$, so $z$ has at least $(2^{n-1} - 1) - (2^{n-2} - 1) = 2^{n-2}$ errors, which is more than $2^{n-2} - 1$ (the error-correcting capacity of $H$). ◯

Proposition 4  Let $x, y \in H$. The probability of obtaining a word $z \in <x, y> \setminus H$ such that $z$ does not uniquely decode into a codeword of $H$ is

$$p \leq \left(\frac{1}{2}\right)^{2^{n-1}} \cdot 2^n$$  \hspace{1cm} (4.1)

Proof: The only way that $z$ does not uniquely decode is that it contains exactly $2^{n-2}$ errors.

By Proposition 2, $|ivt(x, y, z)| = 2^{n-2} - 1$. So, from the $2^{n-1} - 1$ invariant positions of $x$ and $y$, only $2^{n-2} - 1$ correct values will remain in the new word $z$ and thus $(2^{n-1} - 1) - (2^{n-2} - 1) = 2^{n-2}$ will be errors.

Thus, the total amount of errors in $z$ must be in $I = ivt(x, y)$, so $z|_I$ has to be exactly equal to the corresponding part of a correct codeword in $<x|_I, y|_I>$. The probability of this event for a particular codeword is

$$\left(\frac{1}{2}\right)^{2^{n-1}}$$

because $d(x|_I, y|_I) = |I| = 2^{n-1}$ so that in every position there is exactly one 1 and one 0. Since the total number of codewords in a dual binary Hamming code is $2^n$ the probability of non-unique decoding is

$$p \leq \left(\frac{1}{2}\right)^{2^{n-1}} \cdot 2^n$$

◊

We are now in a position to state the main theorem of this section:
Theorem 1  The watermarking scheme described in Section 4.1 can be transformed into a fingerprinting scheme secure against collusions of two buyers by taking as ECC in the mark generation algorithm a dual binary Hamming code $H$ of length $N = 2^n - 1$. An innocent buyer will never be declared guilty and the probability that a participant in a two-buyer collusion can be identified can be made arbitrarily close to 1.

Proof:  It follows from Propositions 3 and 4. Proposition 3 guarantees that an innocent buyer will never be declared guilty. The probability of identifying one of two colluders is $1 - p$, where $p$ is defined in Equation (4.1); as $n$ increases, $1 - p$ tends to 1. ◇
Copyright Protection: Symmetric Schemes
Chapter 5

Copyright Protection: Asymmetric Schemes

Asymmetric fingerprinting schemes for electronic copyright protection are called to play a major role in the prosecution of piracy. Although the main research efforts concentrate on symmetric schemes like the ones described in Chapter 4, they present the drawback of the lack of security for the buyer\(^1\). The impossibility to obtain a proof of redistribution to show in a hypothetical trial restricts the usefulness of these schemes. Nevertheless, research on symmetric schemes is indeed unavoidable, as some asymmetric fingerprinting proposals are based on an underlying symmetric scheme.

However, the complexity of known algorithms for asymmetric and anonymous fingerprinting is too high for practical implementation on standard computers. For instance, constructions [DF98, PS96] is that they are based on secure multiparty computation ([CDvdG88]), a theoretical technique rather than an implementable one.

We will describe a new proposal which uses cryptographic tools other than than multiparty computation.

---

\(^1\)See properties of a copyright protection scheme in Chapter 2 Subsection 2.2.1
5.1 An asymmetric and anonymous fingerprinting scheme

In this section, an asymmetric fingerprinting scheme [DFHJ00a] is presented which provides anonymity and has the advantage of avoiding the secure two-party computation needed in previous asymmetric fingerprinting proposals.

The scheme proposed is described at high level, so that no specific assumptions are made for some loose ends because the scheme can be applied to various environments. For instance, we deal with a generic information item without specifying which type of multimedia content we speak about.

The fingerprinting scheme presented in this section offers security for the buyer in the sense described in Section 2.2.1. To obtain this property, the mark embedding algorithm has been transformed into a protocol in which both merchant and buyer take an active part. A new process, merchant initialization, has been introduced to allow the merchant to obtain information subsequently needed in the mark embedding protocol.

The scheme presented also offers anonymity for the buyer; thus a new entity, the registration authority, is needed in the scheme together with a buyer registration protocol whereby the buyer obtains her pseudonym identity.

5.1.1 Merchant initialization

Since the mark embedding process is performed by both the merchant and the buyer, some previous computation has to be done by the merchant to obtain his inputs for the mark embedding protocol. This process is called merchant initialization and is described below.

Let $H(\cdot)$ be a cryptographically strong block hash function, for instance SHA [oST95]. Let $n$, $l$ and $u$ be nonnegative integer security parameters agreed upon by all parties, where $l < u < n$. The value $u$ determines the information rate of the scheme (see subsection 5.2.3).

The merchant $M$ splits the item to be fingerprinted into $n$ disjoint subitems
item_1, item_2, \ldots, item_n of similar length all of which must be concatenated to
reconstruct the original item. In addition, subitems item_1, \ldots, item_u contain
one bit of the mark, i.e. there exist two slightly different versions item_i^0, item_i^1
of each subitem item_i for i = 1 to u where item_i^0 embeds a 0-bit and item_i^1
embeds a 1-bit. To make subitem marking resilient to intentional modifi-
cation, a redundancy scheme may be used (see Subsection 5.2.2 for more
details). Notice that a robust watermarking scheme can be used to embed
the bit into each subitem item_i.

5.1.2 Buyer registration

The scheme presented achieves anonymity through a registration author-
ity. Each buyer receives a pseudonym from the registration authority. The
pseudonym hides the real identity of the buyer. Together with the pseudonym,
the buyer also obtains a certificate proving that the registration authority
knows the real identity behind her pseudonym. The specification of the reg-
istration protocol is described below.

Let p be a large prime such that q = (p - 1)/2 is also prime. Let G be a
group of order p, and let g be a generator of G such that computing discrete
logarithms to the base g is difficult. Assume that both the buyer B and the
registration authority R have ElGamal-like public-key pairs (\cite{ElG85}). The
buyer’s secret key is x_B and her public key is y_B = g^{x_B}. The registration
authority R uses its secret key to issue certificates which can be verified using
R’s public key. The public keys of R and all buyers are assumed to be known
and certified.

Protocol 4 (Buyer registration)

1. R chooses a random nonce x_r \in R \mathbb{Z}_p and sends y_r = g^{x_r} to B.

2. B chooses secret random s_1 and s_2 in \mathbb{Z}_p such that s_1 + s_2 = x_B
(mod p) and sends \( S_1 = y_{r_1}^{s_1} \) and \( S_2 = y_{r_2}^{s_2} \) to \( R \). \( B \) convinces \( R \) in zero-knowledge of possession of \( s_1 \) and \( s_2 \). The proof given in \([CEvdG88]\) to show possession of discrete logarithms may be used here. The buyer \( B \) computes an ElGamal public key \( y_1 = g^{s_1} \ (\text{mod } p) \) and sends it to \( R \). \( y_1 \) will be used by \( B \) as a pseudonym.

3. \( R \) checks that \( S_1 S_2 = y_{r_1}^{s_1} \) and \( y_{r_1}^{s_1} = S_1 \). \( R \) returns to \( B \) a certificate \( \text{Cert}(y_1) \). The certificate states the correctness of \( y_1 \).

By going through the registration procedure above several times, a buyer can obtain several different keys \( y_1 \).

### 5.1.3 Mark embedding

The mark embedding protocol is in some respects similar to secure contract signing (see Appendix C for more details on contract signing protocols). In this protocol, the merchant uses the data obtained during the merchant initialization process described in Subsection 5.1.1. Figure 5.1 depicts the protocol described below.

**Protocol 5 (Mark embedding)**

1. \( B \) sends \( y_1, \text{Cert}(y_1) \) and text to \( M \), where text is a string identifying the purchase. \( B \) computes an ElGamal signature \( \text{sig} \) on text with the secret key \( s_1 \); \( \text{sig} \) is sent to \( M \).

2. \( M \) verifies the certificate on \( y_1 \) and the signature \( \text{sig} \) on text.

3. For \( i = 1 \) to \( l \), \( M \) sends one message out of the two messages \( \text{item}_i^0 \) and \( \text{item}_i^1 \) using the 1-2 provably secure oblivious transfer sketched in \([BPT95]\). If \( \text{item}_i \) is the output of the oblivious transfer, it should be similar to the original meaningful \( \text{item}_i \), so it should be easy for \( B \) to tell it from junk.
4. B computes an ElGamal signature \( \text{sig}(i) \) on \( H(\text{item}(i)) \) using the key \( s_1 \), where
\[
\text{item}(i) = \text{item}_1 \| \text{item}_2 \| \cdots \| \text{item}_l
\] (5.1)

B returns \( H(\text{item}(i)) \) and \( \text{sig}(i) \) to M. B proves to M in zero-knowledge [GMW87] that \( H(\text{item}(i)) \) was correctly computed, i.e., that it was computed based on the outputs \( \text{item}_i \) of the oblivious transfers, for \( i = 1 \) to \( l \). If B fails to return \( H(\text{item}(i)) \) and the zero-knowledge proof, then M quits the fingerprinting protocol.

5. For \( i = l+1 \) to \( u \), M sends one message out of the two messages \( \text{item}_i^0 \) and \( \text{item}_i^1 \) using the 1-2 provably secure oblivious transfer sketched in [BPT95].
6. B computes an ElGamal signature \( \text{sig}_{(u)} \) on \( H(item(u)) \) using the key \( s_1 \), where

\[
item(u) = item_{l+1}||item_{l+2}||\cdots||item_n
\]  

(5.2)

B returns \( H(item(u)) \) and \( \text{sig}_{(u)} \) to M. B proves to M in zero-knowledge [GMW87] that \( H(item(u)) \) was correctly computed, i.e. that it was computed based on the outputs \( item_i \) of the oblivious transfers, for \( i = l+1 \) to \( u \). If B fails to return \( H(item(u)) \) and the zero-knowledge proof, then M quits the fingerprinting protocol.

7. M sends \( item_{u+1}||item_{u+2}||\cdots||item_n \) to B.

If the computational resources of B are a bottleneck (as it may happen when B is a smart card), then a possibility is to suppress Steps 5 and 6 from Protocol 5 and send \( item_{l+1}, \cdots, item_n \) in Step 7. In this case, the wisest choice is probably to take \( l \approx n/2 \).

### 5.1.4 Mark reconstruction

In Chapter 4 we did not worry about the image \( \hat{X} \) used as input to the mark reconstruction algorithm. We implicitly assumed that \( \hat{X} \) was similar to the original \( X \), so that it makes sense to try to recover the mark embedded.

Likewise in the scheme proposed in this chapter, it only makes sense to attempt redistributor identification if the redistributed \( \hat{item} \) is not too different from the original \( item \).

**Definition 9** Let sim be an arbitrary relation where \( \text{sim}(\hat{item}, item) \) means that a redistributed illegal copy \( \hat{item} \) is still so close to \( item \) that the merchant \( M \) wants to identify the original buyer.

If \( \text{sim}(\hat{item}, item) \) holds, then it is reasonable to assume that \( \hat{item} \) contains a substantial number of subitems which are (perhaps modified) copies of \( item_1, \cdots, item_n \), for some fingerprinted version \( item \) of \( item \).

It must be noted that no redistributor identification can be requested by \( M \) if Protocol 5 is quit before Step 4 is run (\( M \) gets no receipt). In the
following two-party identification protocol between the merchant $M$ and the
registration authority $R$, we will assume that Step 4 of Protocol 5 was run
and that $\text{item}$ contains enough (perhaps modified) copies of subitems among
$\text{item}_1, \ldots, \text{item}_i$ to allow $M$ to reconstruct $\text{item}_{(i)}$:

Protocol 6 (Mark reconstruction)

1. Upon detecting a redistributed $\text{item}$, $M$ determines whether
   $\text{sim}(\text{item}, \text{item})$ holds for some information item on sale. If yes, $M$
   uses the redundancy scheme to recover the bit value that was embed-
   ded in each subitem of $\text{item}$ (notice that $\text{item}$ may not exactly cor-
   respond to any fingerprinted $\text{item}$). If the redundancy scheme is suf-
   ficient and $\text{item}$ contains enough (perhaps modified) subitems among
   $\text{item}_1, \ldots, \text{item}_i$, $M$ can reconstruct in this way the correct finger-
   printed $\text{item}_{(i)}$ from which $\text{item}$ was derived (if a few subitems in $\text{item}_{(i)}$
   had been suppressed or a majority of their marks had been modified in
   $\text{item}$, they can be reconstructed by $M$ using exhaustive search).

2. Once $\text{item}_{(i)}$ has been reconstructed, $M$ computes $H(\text{item}_{(i)})$ and re-
   tirves the corresponding $\text{sig}_{(i)}$, $\text{text}$, $\text{sig}_y$, and $\text{Cert}(y_1)$ from her
   purchase record. Then $M$ sends

   $\text{proof} = [\text{item}_{(i)}, \text{sig}_{(i)}, \text{text}, \text{sig}_y, \text{Cert}(y_1)]$ \hspace{1cm} (5.3)

   to $R$ asking for identification of $B$.

3. $R$ computes $H(\text{item}_{(i)})$ and uses the public key $y_1$ to verify that $\text{sig}_{(i)}$
   is a signature on $H(\text{item}_{(i)})$. If verification fails, this means that either
   $M$ is trying to unjustly accuse a buyer or that the redundancy scheme
   was not sufficient to allow the correct reconstruction of $\text{item}_{(i)}$; in either
   case, identification fails. If verification succeeds, the same key $y_1$ is used
   to verify that $\text{sig}$ is a signature on the text identifying the purchase.
Finally $R$ searches its records to find the buyer $B$ who registered the key $y_1$ and returns the name of $B$ to $M$.

If $\overline{item}$ does not allow reconstruction of $\overline{item}_{(i)}$ and Step 6 of Protocol 5 was run, then reconstruction of $\overline{item}_{(u)}$ (equation 5.2) can be attempted. To do this, take Protocol 6 and replace $l$ by $u$ and $\overline{item}_1, \ldots, \overline{item}_l$ by $\overline{item}_{l+1}, \ldots, \overline{item}_u$. Since usually $u - l > l$, $\overline{item}$ is likely to contain more subitems of $\overline{item}_{(u)}$ than of $\overline{item}_{(l)}$.

**Note 5** If $R$ refuses to collaborate in Protocol 6, its role can be performed by an arbiter except buyer identification. Replace “identify buyer” by “declare $R$ guilty”.

### 5.2 Properties of the fingerprinting scheme

#### 5.2.1 Imperceptibility

The imperceptibility of the marks embedded in the marked object $\overline{item}$ depends on how the merchant embeds the bits in every original subitem $item_i$. This action is performed in the merchant initialization process. As we pointed out before, an underlying watermarking scheme can be used. For instance, if the $item$ is an image, then every subitem $item_i$ can be regarded as a smaller image where the watermarking scheme described in Chapter 4 can be used to embed the bit. In this way, the same imperceptibility level reached for the watermarking scheme is obtained in this asymmetric fingerprinting scheme.

#### 5.2.2 Robustness

Robustness against single user attacks is closely related to the mark embedding process, in the same way that imperceptibility is. Here, since the merchant does not know which is the mark that corresponds to a buyer no ECC can be used to encode the mark. The redundancy mentioned when
describing merchant initialization refers to each bit of information. For instance, a simple redundancy scheme could consist of replicating the bit value embedded in a subitem an odd number \( m > 1 \) of times so that the \( m \)-bit vector \((0, 0, \cdots, 0)\) is embedded in \(\text{item}_0^m\) instead of the bit 0, and the \( m \)-bit vector \((1, 1, \cdots, 1)\) is embedded in \(\text{item}_1^m\) instead of the bit 1. To extract the bit embedded in a redistributed subitem \(\overline{\text{item}}_i\), all \( m \) values are examined by \( M \) and a majority decision is made to determine whether the bit mark is 0 or 1. Use of ECC could be an alternative, but it would be referred to the mark bit embedded in subitem \(\text{item}_i\).

Regarding the description of the mark embedding protocol, the buyer \( B \) can obtain up to \( l \) subitems from \( M \) and then quit Protocol 5 before Step 4. This means that up to \( l \) subitems can be obtained without copyright protection. Thus \( l \) should be small as compared to \( n \). On the other hand, once Step 6 is run, the first \( u \) subitems are copyright protected, but no protection is provided for the last \( n - u \) subitems. Thus \( u \) should not be much smaller than \( n \). However, the last \( n - u \) subitems should contain enough information to deter \( B \) from quitting the Protocol 5 before Step 6, which in the worst case would leave subitems \( l + 1 \) up to \( u \) unprotected.

From the above remarks, a good choice would be to take \( l = 64 \), \( u \) such that \( u - l \geq 64 \), and \( n \) such that the last \( n - u \) subitems contain the minimum amount of information needed to deter a standard buyer \( B \) from quitting Protocol 5 before Step 6.

Robustness against collusion attacks can be assessed using the marking assumption provided that the underlying watermarking scheme is robust. Hence, it is assumed that honest buyers can only locate and delete marks by comparing their copies. In the scheme described above, colluding buyers can delete or alter all marks in the subitem if and only if one of them was given \( \text{item}_0^m \) and the other \( \text{item}_1^m \). To avoid this possibility, some different pairs \( \text{item}_0^m, \text{item}_1^m \) can be generated for each subitem. Using different pairs for different users implies that colluders do not know which subitems embed different bits, because different subitems do not necessary embed different
5.2.3 Information rate

As pointed out above, the fact that this scheme offers security for the buyer prevents the merchant from knowing the embedded mark. Hence, no ECC can be used to encode the mark, so that mark length depends directly on parameter $u$. Thus $u$ is a measure of the information rate of the scheme. Nevertheless, the information rate measured as the information amount that can be embedded in the item without affecting its quality does not depend on $u$ rather than in the underlying watermarking scheme. This information affects the robustness of the scheme rather the net information included in a marked item.

5.2.4 Secret information

Obviously, the original item has to be kept secret for the sake of integrity. The rest of other secret information of the asymmetric and anonymous fingerprinting scheme presented can be divided in two different categories:

- secret information for the basic functionality of mark embedding.
- secret information for the anonymity property.

The first one depends on the underlying watermarking scheme used to embed one bit in every subitem $item_i$. Regarding the secret information needed to satisfy the anonymity property, on one hand, the buyer has to keep his secret key $x_B$, and the secret key $s_1$ corresponding to the $y_1$ pseudonym. On the other hand, the registration authority must guarantee the security of his secret key used to generate a pseudonym certificate $Cert(y_1)$.

5.2.5 Security for the buyer

The fingerprinting scheme proposed offers security for the buyer since it is asymmetric. It should be infeasible for $M$ to accuse a honest buyer in front
of the registration authority. That means the merchant M, should not figure out the values of \(\overline{item}_{[l]}\) and \(\overline{item}_{[u]}\) corresponding to a buyer \(B\), or even to a pseudonym \(y_1\). Thus the sizes \(2^l\) and \(2^{u-l}\) of the spaces where \(\overline{item}_{[l]}\), respectively \(\overline{item}_{[u]}\), take values should be large enough (a good choice could be \(l \geq 64, u - l \geq 64\) which implies \(n \geq 128\)). The following proposition relates the size of those spaces to security:

**Proposition 5** Let \(l\) and \(u\) be the security parameters defined in Protocol 5. Then \(M\) must perform an exhaustive search in a space of size \(\min(2^l, 2^{u-l})\) to obtain the proof stated in equation 5.3.

**Proof:** To obtain the proof of redistribution, \(M\) must figure out either the value \(\overline{item}_{[l]}\) or the value \(\overline{item}_{[u]}\) that corresponds to a fixed \(H(\overline{item}_{[l]})\) or \(H(\overline{item}_{[u]})\) respectively. Since the oblivious transfer [BPT95] is provably secure and \(B\)'s proofs on the correctness of \(H(\overline{item}_{[l]})\) and \(H(\overline{item}_{[u]})\) are zero-knowledge, the only conceivable attack is for \(M\) to start trying all possible values for \(\overline{item}_{[l]}\) or \(\overline{item}_{[u]}\) until one is found such that either \(H(\overline{item}_{[l]})\) is the value contained in \(\overline{sig}_{[l]}\) or \(H(\overline{item}_{[u]})\) is the value contained in \(\overline{sig}_{[u]}\). With subitems having two versions each, \(\overline{item}_{[l]}\) is uniformly and randomly distributed over a set of \(2^l\) different values, and \(\overline{item}_{[u]}\) is uniformly and randomly distributed over a set of \(2^{u-l}\) different values. \(\diamond\)

### 5.2.6 Anonymity

The scheme proposed in this chapter also satisfies the anonymity property. That means that the merchant cannot know the identity of a honest buyer unless he colludes with the registration authority. This result is stated in the next proposition.

**Proposition 6** Protocol 5 provides buyer anonymity for honest buyers in front of malicious merchants.

**Proof:** In the mark embedding protocol, \(M\) sees a pseudonym \(y_1\), which is related to \(y_B\) by the equation \(y_1^{x_r} S_2 = y_B^{x_r}\). Even knowledge of \(\log_y y_r = x_r\)
would not suffice to uniquely determine $y_B$ from $y_1$, since $S_2$ is unknown to $M$. ◦

Furthermore, the buyer registration protocol described in Subsection 5.1.2 offers the proper security level for the private key of the buyer as it is shown below.

**Proposition 7 (Registration security)** Protocol 4 provides buyer authentication without compromising the private key $x_B$ of the buyer.

**Proof:** In registration, $R$ sees $S_1$, $S_2$, $y_1$ and two zero-knowledge proofs. The latter leak no information. Without considering the zero-knowledge proofs, $R$ needs no knowledge of $x_B$ to find values $S'_1$, $S'_2$ and $y'_1$ which are related in the same way as $S_1$, $S_2$ and $y_1$. Take a random $s'_1$, then compute $y'_1 = g^{s'_1}$ and $S'_1 = y_1^{s'_1}$. Finally, $S'_2 = y_{x_B}^{x} / S'_1$.

Now consider the zero-knowledge proofs; imagine that an impersonator not knowing $x_B$ can compute $S_1$, $S_2$ such that he/she can demonstrate possession of $\log y$, $S_1$ and $\log y$, $S_2$ and $S_1S_2 = y_{x_B}^{x}$ holds. Then the impersonator can compute the discrete logarithm $x_B$. In general, if impersonation is feasible, so is computing discrete logarithms. ◦
Chapter 6

Agents: A tool for electronic commerce management

In this chapter we discuss two new approaches to mobile agent security. The first one deals with protection of an agent against malicious hosts. More precisely, we assume that the agents circulate in a trusted execution environment (see Subsection 2.3.1 for more details). This assumption is enforced by defining an agent route where only trusted servers are specified. Such a protected route has to be followed by the agent during his roaming. The new proposal described [DFHJ00b] reduces the computational costs of existing proposals offering the same security degree.

The second proposal [DFHJ98a] to provide more security in a mobile agent environment is focused on trade agents, mobile agents that buy goods on their owners behalf. The scheme presented uses recent advances in off-line electronic payment schemes as well as electronic contract signing to achieve a high security level during the payment transaction.

6.1 Protecting a mobile agent route

In Section 2.3 different ideas to protect agents against malicious hosts have been outlined. The most interesting one, as far as our research is concerned,
is the one that specifies an agent route before the agent starts roaming the network. The existing proposals described in Chapter 2 provide each one different properties. In [SRM98] a flexible agent route is defined but route security is not carefully considered. On the other hand, the scheme proposed in [WSUK99] offers a high security level but, as the authors point out, the computational cost of the proposal is still to high.

The basic idea is to obtain a protected host list that determines the agent route. Such agent route must satisfy the following security properties:

1. It should be infeasible for a host to modify the agent route. In particular, it should not be possible to include or remove host addresses from the agent route without the agent’s owner authorization.

2. Every host should be able to verify that it was included in the route by the agent’s owner.

3. Every host of the route should only see the previous host and the next host of the agent route. That means that the rest of the route information should not be disclosed to a particular host.

4. Every host should be able to verify that the preceding host (the one the agent comes from) is actually the previous host coded in the agent route.

5. A host should not be able to replace the agent route by older routes of the same agent.

6.1.1 Agent route scheme

To reduce the computational burden of previous approaches, we will use the concept of enhanced hash chain presented in Chapter 3 for micropayments. Again, the idea is to replace digital signatures by hash functions because the latter are easier to compute.
Although enhanced hash chains can be applied both to micropayments and to agent routes, there are some differences between both scenarios which require the solution to be specifically tailored for each problem.

The main difference is that, while in the micropayment systems we can identify two entities, buyer and merchant, in agent route model there are \( n + 1 \) entities: one agent and \( n \) different hosts. On the other hand, in a micropayment system, at each step the buyer reveals some information to the merchant and the latter performs a verification using information from previous step. But in an agent route environment, the agent reveals all the information at the same time and all \( n \) hosts have the whole information. So, as it will be shown in the next protocol, different mechanisms are proposed to deal with this new situation.

**Protocol 7 (Agent route)**

1. **[Writing]** The agent owner chooses \( n \) hosts represented by their IP-addresses \( H_1, \ldots, H_n \). Let \( H_0 \) be the address of the agent owner. Let \( PK_i \) be the public key of host \( H_i \).

2. **[Encryption]** Encrypt \( H_{i+1} \) using the key of \( H_i \), that is compute \( U_i = E_{PK_i}(H_{i+1}) \) for \( i = 1, \ldots, n-1 \). For \( i = n \), compute \( U_n = E_{PK_n}(H_0) \). Now transform \( U_i \) into \( V_i \) by adding some redundancy, for \( i = 1, \ldots, n \).

3. **[Coding]** The encrypted IP-addresses \( V_1, \ldots, V_n \) are encoded by the agent owner using an enhanced hash chain to obtain traces \( T_0, \ldots, T_n \) as follows:

\[
\begin{align*}
T_0 &= F(T_1) \oplus V_1 \\
T_1 &= F(T_2) \oplus V_2 \\
&\vdots \\
T_{n-1} &= F(T_n) \oplus V_n \\
T_n &= F(V_n)
\end{align*}
\]
4. **Signature** The agent owner uses her private key to sign the first trace \( T_0 \); let \( t_0 \) to be the signed form of \( T_0 \).

5. **Initialization** The agent is sent to the first host of the route, namely \( H_1 \), together with \( t_0 \) and the vector of encrypted hosts \( (V_1, \cdots, V_n) \).

6. **Operation at host \( H_i \)** \( H_i \) checks that it is included in the route by computing \( T_n, T_{n-1}, \cdots, T_0 \) and checking whether \( t_0 \) is a valid signature for \( T_0 \). Then the agent is run by \( H_i \). When the agent completes his job, \( H_i \) removes the redundancy to \( U_i \) to obtain \( U_i \). Then it decrypts \( U_i = E_{P_{K_i}}(H_{i+1}) \) to obtain the address of the next host. He sends the agent to \( H_{i+1} \), together with \( t_0 \) and the vector \( (V_1, \cdots, V_n) \).

7. **End of route** The route ends at \( H_0 \) (the agent owner).

### 6.1.2 Correctness of the proposal

We next recall the properties stated at the beginning of Section 6.1:

**Property 1** When moving from \( H_i \) to \( H_{i+1} \), the agent can check whether the traces \( T_n, \cdots, T_0 \) computed from \( (V_1, \cdots, V_i, V_{i+1}, \cdots, V_n) \) are such that \( t_0 \) is a valid signature on \( T_0 \). If \( H_{i+1} \) is not the right successor of \( H_i \), the check will fail and the agent will refuse to execute. Thus, the integrity of the route is guaranteed as long as integrity for agent execution is guaranteed. Note that the agent can compute \( V_i \) from \( H_i \) for all \( i \).

**Property 2** An honest host \( H_i \) can recompute \( T_n, \cdots, T_0 \) from the values \( V_1, \cdots, V_i, \cdots, V_n \) received from \( H_{i-1} \). If \( H_i \) was not in the route designed by the agent owner then \( t_0 \) will not be a valid signature on \( T_0 \).

**Property 3** Every host \( H_i \) knows who is the preceding host \( H_{i-1} \), because IP communication between hosts is not anonymous. On the other hand \( H_i \) can decrypt \( V_i \) to obtain \( H_{i+1} \). The rest of host in the route remain encrypted and unknown to \( H_i \).
Property 4 This property is satisfied only if communication between hosts is authenticated. In this case, \( H_i \) can check that the preceding host is actually \( H_{i-1} \). After that \( H_i \) checks whether \( E_{P_{K_{i-1}}}(H_i) = U_{i-1} \) where \( V_{i-1} \) should be such that the traces computed from \( (V_1, \ldots, V_{i-1}, \ldots, V_n) \) yield a \( T_0 \) for which \( t_0 \) is a valid signature.

Property 5 To satisfy this property, a timestamp together with an expiration date should be added to \( T_0 \) before it is signed by the agent owner.

6.2 Intelligent trade agents

In Subsection 2.3.2 an introduction to mobile trade agents was presented. Furthermore, a particular Intelligent Trade Agent (ITA) was described and some weak points were outlined. In this section we propose a new ITA scheme that offers better properties.

Regarding the proposal described in Section 2.3.2, the main weak points that can be improved are:

Trust on the agent The user has to trust the agent since the latter is given the user’s credit card number.

On-line AS For each transaction, the authorization server (AS) has to be on-line to authorize the transaction and this could be a problem in terms of availability, computing time and bandwidth of the connections to the AS.

Lack of anonymity From his knowledge of the buyer’s credit card number and the definition of the objects bought, the AS can link a buyer’s identity with a specific purchase, so anonymity is not preserved.

In the next section we present a new ITA scheme [DFHJ98a] that solves the problems above by combining a secure contract signing protocol (allowing a contract to be signed without the physical presence of the parties) with an electronic payment system allowing off-line anonymous payments. Secure
contract signing is dealt with in Appendix C while Appendix D presents the basic concepts of an anonymous off-line payment.

6.2.1 Architecture of the new scheme

We are now ready to define a new scheme for an intelligent untrusted trade agent. We use the solution presented in [MvS97] to locate the agents, and thus the ITA remains physically in the AR but logically roams the network, thanks to the distributed-object technology. In this scheme, the user does not trust the bank nor the server nor the agent, so we have to introduce an electronic device such a smart card which the user can completely trust and (physically) control. The user uses this card to make all the operations such as withdrawing money from her account or signing a contract with the agent in the way described in Appendix C. In our scheme the whole transaction involves three main steps:

1. The user withdraws money from her bank account using her smart card to execute a protocol such as the one developed in [Bra93].

2. Once the user and the agent have signed a contract, the user sends to the agent the amount of money specified in that contract.

3. The agent pays for the items bought using the money received from the user.

The architecture and the operation of the new scheme are shown in Figure 6.1. The operation of the scheme of Figure 6.1 can be summarized as follows:

Protocol 8 (Anonymous ITA)

1. The buyer withdraws money from her account using the withdrawal protocol of an off-line untraceable payment system such as the one proposed in [Bra93]:


(a) The buyer’s smart card presents to the bank for certification the notes it has minted.

(b) The bank withdraws the amount specified by the notes from the buyer’s account, certifies the notes and sends them back to the user’s smart card.

2. The ITA and the buyer sign a contract using a secure contract signing protocol such as Protocol 10 presented in Appendix C; in that protocol, the buyer’s role is delegated to her smart card. In the contract, the conditions of their agreement (attributes of the goods to buy, amount of money transferred to the agent, etc.) are specified.

3. The buyer sends to the agent the money she has withdrawn at Step 1.

4. The ITA roams to the first server (S1).
5. Once the ITA finds an item that fits the requirements specified in the contract of Step 2, it starts the transaction:

(a) The ITA and the server sign a contract using again a secure contract signing protocol such as Protocol 10 presented in Appendix C. In that contract, they specify the conditions of the transaction (price, no. of items, warranty, commitment to payment, commitment to delivery of goods, etc.).

(b) The ITA sends to the server certified notes received in Step 3 from the buyer to satisfy the amount specified in the contract.

(c) The server sends to the ITA the goods specified in the contract.

6. The agent roams to the next server.

7. The user tells the agent to come back to the AR. This can be done automatically after some conditions given by the user in the contract signed in Step 2 are met or by using a protected route (see section 6.1) in which the AR plays the role of agent owner.

8. The agent returns to the AR.

9. The user checks what the agent has bought.
Chapter 7

Conclusions

7.1 Concluding remarks

In this thesis, we have covered different aspects of the field of electronic commerce, mainly for multimedia and products in electronic media.

The primary concern has been to offer a broad overview of different security issues in electronic commerce from two points of view of the parties acting in an economic transaction: the buyer and the merchant. We have introduced each topic by describing the main existing proposals, their properties and they weaknesses.

Electronic payment schemes and more precisely micropayments have been studied with the aim of obtaining an acceptable level of security and flexibility for both buyer and merchant.

Copyright protection schemes have been studied with the merchant’s interests in mind. The survey presented for copyright protection schemes is relevant in its own right since no general overview exists due to the novelty of the topic. The thesis has covered in detail the two main tools used for copyright protection of electronic goods: watermarking and fingerprinting. Although the merchant requirements are the most important ones in any copyright protection scheme, the security needs of buyers have not been omitted since asymmetric schemes have been also been studied.
Finally, from a buyer’s point of view, the problem of electronic commerce management has been addressed using mobile agents as a tool. Some solutions are given to protect an agent route and to implement an intelligent trade agent. Basic security properties that were already attainable in other environments are now available for mobile agents.

Last but not least, we have to mention that the aim of our research has been to obtain results that can be implemented. Proofs of this claim are the Java card study given in Appendix B for the implementation of spending programs on smart cards and also the implementation of symmetric copyright protection schemes presented in this thesis.

7.2 Results of this thesis

In this thesis different results have been presented.

For micropayments, the new concept of enhanced hash chain has been introduced to expand the properties of hash chains. Also, structural coding has been recalled to create a new tool called spending program. A spending program generalizes even more than enhanced hash chains do the hash chain concept used in several current micropayment schemes. The advantages of such a new tools are twofold. In one hand, the scheme proposed offers more flexibility. Unlike hash chains, spending programs can offer multicurrency support, variable-valued coupons and selective use of such coupons. Case-like structures to choose coupons at transaction-time can be implemented by the buyer or the buyer’s smart card. On the other hand, efficiency has also been improved. With fixed-value coupons, the value of coupons must be the smallest fraction for the buyer to be able to pay any amount. This implies that verifying a lot of hashes is likely to be necessary for most micropayments. Variable-valued coupons permitted by spending programs allow micropayments of arbitrary value to be performed with a single hash verification. In fact, the distinction between micropayments and standard payments fades away with the availability of variable-valued coupons. Furthermore, iterative
spending programs can be considerably more compact than hash chains by allowing coupons to be re-used.

Regarding security, the structural coding mechanism sketched in Appendix A can be used to encode spending programs and obtain a degree of unforgeability similar to that of hash chains. However, we would like to stress that the concept of spending program is independent of the actual coding procedure used, as long as integrity is guaranteed.

The results obtained in copyright protection field are detailed below. In Chapter 2 a general overview of the two main techniques of copyright protection has been presented. This survey is important since it is the first one that deals together watermarking and fingerprinting by studying their connections and differences.

In Chapter 4, we have presented a new robust semi-public watermarking scheme for images that satisfies the main properties of imperceptibility, robustness, information rate and secret information. Also, the scheme presented can be used for multiple watermarking which increases the value of the proposal.

An interesting discussion on the upgradeability of a watermarking scheme to a fingerprinting scheme has also been addressed. The basic properties a watermarking scheme has to satisfy to be suitable for fingerprinting have been pointed out. Such a study has been used to develop a collusion-secure fingerprinting construction based on a robust watermarking algorithm. The problem of two-buyer collusion is solved using dual binary Hamming codes. The constructions for collusion-secure fingerprinting given here are simpler than those in [BS95] and are attractive because they are built on top of a robust watermarking algorithm.

In the spirit of obtaining new copyright protection schemes with new properties, we have presented a protocol suite for asymmetric and anonymous fingerprinting which offers buyer security against fraudulent merchants. The
scheme presented avoids multiparty computation needed in previous proposals. Although it is still a theoretical proposal, general properties of any copyright protection scheme (imperceptibility, robustness, etc.) can be studied assuming the use of a underlying watermarking scheme.

In the mobile agent field, enhanced hash chains, used previously for micropayments, have been applied to securing an agent route. Coding a mobile agent route using enhanced hash chains provides security against possible attacks by malicious hosts. Furthermore, the proposal presented in this thesis reduce the computational cost of previous schemes whilst offering the same security degree.

We have presented also a new scheme for an untrusted and intelligent trade agent system. The system displays clear improvements with respect to previous proposals. First of all, it can deal with untrusted agents thanks to the inclusion of a contract signing protocol between the user and the agent that establishes the terms of the agreement. Another property is that buyer anonymity is preserved even if the bank and the server collude to obtain the user identity. Finally, the presented scheme does not need a third party to be on-line during the payment transaction between agent and server.

7.3 Future research

Our future research will focus mainly on copyright protection schemes. New proposals will be directed to increasing:

- The range of attacks that can be survived by the watermarking algorithm.

- The size of collusions that can be successfully resisted by the fingerprinting algorithm.

Furthermore, other multimedia contents rather than images will be explored, like for instance movies, audio or text (standard formats like postscript
or pdf). For copyright protection of movies, some ideas described in this thesis for still images can directly be applied since some movie formats, such as MPEG, are based on JPEG still images. For audio, similar approaches can be taken by just replacing the JPEG image compressor by MP3 (MPEG I Layer 3 compressor). Another interesting research target is the development of a benchmark for audio, analogous to Stirmark [PAK98] used for images. The availability of such a benchmark will be very useful for the development of robust audio copyright protection techniques.

Finally, new asymmetric schemes for copyright protection will be studied to obtain efficient implementable algorithms.
Conclusions
Appendix A

Structural Coding

We recall in this appendix the basic operating principles of the structural coding for program integrity given in [DF91a].

Let $i_1, \ldots, i_n$ be a program, where $i_k$ is a machine-language instruction (for a spending program, $i_k$ can be either a value, a flow-control or an input-output instruction); $i_n$ is not a branch (it can be set to be an end instruction). A normalized instruction format is defined: the length of all $i_k$ is made equal by appending a known filler and then each $i_k$ is appended a redundancy pattern, whose length depends on the desired security level. Call the normalized program $I_1, \ldots, I_n$.

Let $F$ be a one-way hash function, such that in general $F(X \oplus Y) \neq F(X) \oplus F(Y)$. Now, if $I_1, \ldots, I_n$ is a sequential block (containing no branches) it will be encoded into a trace sequence $T_0, T_1, \ldots, T_n$, where traces are computed in reverse order according to the following equalities:

\[
\begin{align*}
T_0 &= F(T_1) \oplus I_1 \\
T_1 &= F(T_2) \oplus I_2 \\
& \vdots \\
T_{n-1} &= F(T_n) \oplus I_n \\
T_n &= F(I_n)
\end{align*}
\] (A.1)
Now consider branches. If \( I_k \) is a forward unconditional branch to \( I_j \) (i.e. \( k < j \)), it is translated as:

\[
\begin{align*}
\cdots T_{k-1} &= F(T_k) \oplus I_k \\
T_k &= F(T_j) \oplus I_j \\
T_{k+1} &= \cdots \\
&\vdots \\
T_j &= \cdots \tag{A.2}
\end{align*}
\]

When \( I_k \) is a forward conditional branch to instruction \( I_j \), the following traces are computed (also in an index decreasing order):

\[
\begin{align*}
\cdots T_{k-1} &= F(T_k) \oplus I_k \\
T_k &= F(T_{k'}) \oplus F(T_{k+1}) \oplus I_{k+1} \\
T_{k'} &= F(T_j) \oplus I_j \tag{A.3} \\
T_{k+1} &= \cdots \\
&\vdots \\
T_j &= \cdots 
\end{align*}
\]

Backward branches from \( I_k \) to \( I_j \) (i.e. \( k \geq j \)) are slightly more complicated to encode, since the trace \( T_j \) corresponding to \( I_j \) cannot be used to compute \( T_k \) or \( T_{k'} \) as in forward branches, because \( T_j \) follows \( T_k \) in the trace computation and depends on \( T_k \) (reverse trace computation). Let \( \bar{T}(j) \) a one-to-one integer function on \( j \), for example the identity function. A backward unconditional branch at instruction \( I_k \) to instruction \( I_j \) is translated as:

\[
\begin{align*}
\cdots T_{j-1} &= F(\bar{T}_j) \oplus F(T_j) \oplus I_j \\
\bar{T}_j &= F(T_j) \oplus \bar{T}(j)
\end{align*}
\]
\[ T_j = \cdots \]
\[ \vdots \]
\[ T_{k-1} = F(T_k) \oplus I_k \]
\[ T_k = F(\bar{T}(j)) \oplus I_j \]
\[ T_{k+1} = \cdots \]  \hfill (A.4)

Finally, the trace structure for a *backward conditional branch* is as follows:

\[ \cdots T_{j-1} = F(\bar{T}_j) \oplus F(T_j) \oplus I_j \]
\[ \bar{T}_j = F(T_j) \oplus \bar{T}(j) \]
\[ T_j = \cdots \]
\[ \vdots \]
\[ T_{k-1} = F(T_k) \oplus I_k \]
\[ T_k = F(T_{k'}) \oplus F(T_{k+1}) \oplus I_{k+1} \]
\[ T_{k'} = F(\bar{T}(j)) \oplus I_j \]
\[ T_{k+1} = \cdots \]  \hfill (A.5)

In [DF91a], the coding for non-recursive and recursive branch to subroutine is also detailed.

After the previous trace computation, it is up to the protection system (the buyer in the case of a spending program) to endorse the trace sequence with a signature on the first trace \( T_0 \) and on every \( \bar{T}_j \) in a backward branch target. So the protection system replaces \( T_0 \) with \( t_0 = s_{k_{ps}}(T_0) \) and the \( \bar{T}_j \)'s with \( \tilde{t}_j = s_{k_{ps}}(T_j) \), respectively, where \( s_{k_{ps}}(\cdot) \) denotes encryption under the protection system’s private key.

Now the trace sequence \( t_0, T_1, \cdots, T_n \) is fed to a special processor instead of the executable instructions \( i_1, \cdots, i_n \). This special processor contains a preprocessor to decode the above trace structure and retrieving the executable instructions. The following properties hold:
Theorem 2 (Correctness) The program $i_1, \ldots, i_n$ can be retrieved and executed from its corresponding trace sequence $t_0, T_1, \ldots, T_n$.

The proof of Theorem 2 is a construction and it shows that, if the preprocessor is pipelined to the main processor, then there is no significant increase in the execution time. A few examples follow to give a flavour of the construction to decode traces (details can be found in [DF91b]):

- For a sequential instruction block, $i_1$ is retrieved by computing $I_1 = F(T_1) \oplus p_{kps}(t_0)$ (where $p_{kps}(:)$ denotes encryption under the public key of the protection system); the rest of instructions $i_k$ are retrieved by computing

$$I_k = F(T_k) \oplus T_{k-1}$$

While $i_k$ is being retrieved by the preprocessor, $i_{k-1}$ can be executed by the main processor.

- If the instruction $i_k$ is an unconditional forward branch to $i_j$, at cycle $k + 2$, $T_{k+2}$ is read, $T_k$ is decoded by the preprocessor to obtain a valid instruction, and $I_k$ is executed by the processor and recognized as an unconditional forward branch. At cycle $k + 3$, only a read operation on $T_j$ is performed; at cycle $k + 4$ $T_{j+1}$ is read, and $T_k$ is evaluated by the preprocessor to obtain $I_j$

$$F(T_j) \oplus T_k = I_j$$

Here $k + 3$ and $k + 4$ are idle cycles for the processor. From $k + 5$ execution continues in sequence from $I_j$.

- For conditional forward branches, the decoding procedure depends on whether the condition is true or not. In the first case, it is very similar to decoding an unconditional forward branch. If the condition is false, decoding is very similar to the sequential instruction block case.
• For backward branches, decoding proceeds in a similar way, but a few more execution cycles are wasted.

**Theorem 3 (Run-time integrity)** If a program \(i_1, \ldots, i_n\) is stored as \(t_0, T_1, \ldots, T_n\) and is decoded as described above, any instruction substitution, deletion or insertion will be detected at run-time, thus causing the main processor to stop execution before the substituted, deleted or inserted instruction(s). Moreover, only the last five read traces need be kept in the internal secure memory of the processor.

The detailed proof of Theorem 3 can be found in [DF91b]. The key point is that the one-wayness of the function \(F\) which connects traces makes it infeasible to successfully substitute, delete or insert traces ("successfully" means that no valid instructions can be decoded by the preprocessor beyond the point where traces have been substituted, deleted or inserted). It is important to stress that valid trace sequences can only be created by the protection system (the one who can sign with \(sk_{ps}\)).
Appendix B

Spending Programs Using Java Cards

The micropayment scheme presented in Chapter 3 can be implemented using smart cards. In particular, smart cards can be used to implement the buyer’s side of hash-based micropayments [CPDFHJ00] (which include spending programs), so they should implement at least a digital signature algorithm and a hash algorithm. Java cards seem a better choice than classical smart cards and this for two main reasons: easy programming (which results in shorter development times) and smoother compatibility with Web-based e-commerce.

This appendix presents a performance comparison between different Java cards to evaluate their suitability to implement the scheme presented in Chapter 3. Furthermore, a Java card file system is also presented to allow several spending programs to be stored in the buyer’s smart card.

B.1 Features of a Java card

A Java card is the implementation of an interpreter for a subset of the Java programming language on a standard smart card controller. This enables card applications to be written in a modern high-level language. The runtime environment provided by Java cards is optimized for smart cards, with the
goal of bringing many of the benefits of Java software programming to the resource-constrained world of smart cards. The following features of standard Java are not supported in the Java card language:

- Variable types like `long`, `float` or `double`.

- Multidimensional arrays.

- Unicode characters.

- Threads (because the card does not support multitasking). No threads means that synchronized or volatile methods used to control access to shared variables are not supported.

- Garbage collection.

Other differences with standard Java regard the implementation of the Java Virtual Machine (JVM). In a Java card, the JVM is considered as part of the operating system and is placed in read-only memory by the card manufacturer. This can be a problem to achieve smart card upgradability and true multi-functionality, as was pointed in [Mar97]. Furthermore, due to processing and memory constraints, the JVM converter (whose mission is to translate bytecode into compressed code to be loaded on the card) is located outside the card (on the host). Only the other component of the JVM, the Java card run-time environment (JCRE,[Sun99b]), is stored on the card.

From the above considerations, some recommendations for Java card programmers can be stated. Since RAM is mainly used by the Java stack to hold local variables and arguments of methods, method invocations, complex expressions and local variables and arguments should be used sparingly; otherwise the RAM memory becomes full and the system resorts to EEPROM, which is considerably slower. Another issue is that the lack of garbage collection favors the depletion of EEPROM, where objects are stored. A good idea is to minimize garbage by using static or final methods and attributes whenever possible (which increases visibility) and to avoid referencing them only
from local variables (which would make it impossible to reference a method after the local variables disappear).

**B.2  Performance criteria**

In order to carry out a performance comparison between several Java cards to evaluate their suitability to implement the buyer’s part of a micropayment scheme, the following criteria have been taken into account:

**Speed:** Measured as the time interval (in milliseconds) elapsed between command submission to the card by the host and receipt of card response by the host.

**Storage:** Capacity of the card to hold cardlets (Java card applications) in its EEPROM.

**Programming environment:** Development environment supplied by the manufacturer for card programming. A good development environment is crucial since most of smart card application development is done on a standard computer and applets are loaded on the card only at the final stage to test. Therefore, aspects like smart card simulation or graphical interface tools become very important.

In the next subsections, the results of comparing Java cards by three manufacturers are presented. Being very connected criteria, speed and storage have been considered jointly in Subsection B.2.1. To assess the programming environment, Subsection B.2.2 describes how the various steps of application development can be carried out with each card. Subsection B.2.3 contains the conclusions of the comparison.

**B.2.1  Speed and storage**

To implement hash-based micropayments, a smart card is needed which offers enough speed and enough storage to accommodate spending programs (or
hash chains) as well as a cardlet. Three Java cards have been compared, namely GemXpresso (Gemplus,[Gem98]), Odyssey (Bull,[Inc98]) and SmartCafé (Giesecke & Devrient,[Gmb99]). An attempt to test a fourth Java card (Schlumberger’s Cyberflex, [Sch99]) failed because we were unable to obtain a free programming kit from the manufacturer. Due to the lack of commonly accepted benchmarks for smart cards (not to speak of Java cards), synthetic benchmarks have been used. A synthetic benchmark is designed to measure the performance of individual components in a computer system.

Table B.1 lists the main hardware features of the three Java cards being considered. Units are as follows: EEPROM and ROM are in kbytes, RAM is in bytes, clock speed in MHz. GemXpresso EEPROM is physically divided into a 10kB general purpose memory and a 5kB stack; for Odyssey and SmartCafé, the value between parentheses is the amount of free EEPROM once the system is loaded. The “Bits” column indicates the word length of the card processor. The “Arch.” column describes the type of processor architecture. It can be seen that Odyssey is the card with the fastest clock, which will result in a shorter response time (see below).

<table>
<thead>
<tr>
<th>Java card</th>
<th>Company</th>
<th>EEPROM</th>
<th>RAM</th>
<th>ROM</th>
<th>Clock</th>
<th>Bits</th>
<th>Arch.</th>
</tr>
</thead>
<tbody>
<tr>
<td>GemXpresso</td>
<td>Gemplus</td>
<td>10+5</td>
<td>512</td>
<td>8</td>
<td>3-5</td>
<td>32</td>
<td>RISC</td>
</tr>
<tr>
<td>Odyssey</td>
<td>Bull</td>
<td>8 (7)</td>
<td>10</td>
<td>10</td>
<td>8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SmartCafé</td>
<td>G&amp;D</td>
<td>16 (12)</td>
<td>1280</td>
<td>32</td>
<td>1-5-7.5</td>
<td>16</td>
<td></td>
</tr>
</tbody>
</table>

Table B.1: Java card hardware features

Table B.2 compares the considered Java cards from the software standpoint. The cardlet size column refers to the size of the bytecode required to encode the cardlet of the synthetic benchmark described below; a shorter bytecode suggests a more efficient use of available storage, which in turn allows more spending programs and cardlets to be kept on the card. SmartCafé bytecode is shortest because the card offers a native code cryptolibrary. The JC API column lists the Java card Application Program Interface version [Sun99a]; a newer version means that more classes and methods are available, which results in improved bytecode generation.
### Table B.2: Java card software features

<table>
<thead>
<tr>
<th>Java card</th>
<th>cardlet size</th>
<th>JC API</th>
</tr>
</thead>
<tbody>
<tr>
<td>GemXpresso</td>
<td>4539</td>
<td>2.0</td>
</tr>
<tr>
<td>Odyssey</td>
<td>4445</td>
<td>2.1</td>
</tr>
<tr>
<td>SmartCafé</td>
<td>2174</td>
<td>2.1</td>
</tr>
</tbody>
</table>

A synthetic benchmark should consider the four kinds of Application Program Data Unit (APDU) defined in the ISO 7816-4 standard ([GJ98]):

1. No incoming data and no outgoing data.
2. Outgoing data, but no incoming data.
3. Incoming data, but no outgoing data.
4. Incoming and outgoing data.

In practice, our benchmark tests response times for all APDU types above, except the fourth type, which was not tested due to transport protocol limitations that did not allow data to be simultaneously sent and received. Specifically, the following commands are accepted by the benchmark: **reset** to test the first kind of APDU, **put** to test the second kind, and **get** and **getinput** to test the third kind.

Our synthetic benchmark consists of two parts:

**Cardlet** This is a Java card application which acts as a server by implementing the four commands described above. The cardlet installation process and the size of the corresponding bytecode differ for each card, as reflected in Table B.2.

**Host application** This is a client application written in C which runs on a Linux host and allows to send APDUs to the cardlet to test its four commands and measure the corresponding response time. In order to measure response times in the same conditions for each considered card,
the host application is the same for all cards, which is possible thanks to the PC/SC standard interface [PCS96] between host and cards.

**Reset**

The main purpose of this command is to be able to measure how long does it take for the card to answer a command. This is the simplest benchmark test, as it only implies sending one empty APDU and waiting for its acknowledgement. From Figure B.1, it can be seen that Odyssey is the fastest card, followed by GemXpresso and SmartCafé. There is an explanation for the latter card being slower: since several cardlets can be on a card, the host application must send an APDU to the Java card run-time environment (JCRE) to select a particular cardlet; SmartCafé is the only card in which the JCRE forwards this selection APDU to the selected cardlet, which causes additional overhead.

![Figure B.1: Average response time over 30 runs for each card and command (milliseconds).](image-url)
Put

This command is used by the host application to download spending programs onto the smart card. This actually amounts to sending a byte string to the card. Thus, \texttt{put} causes the host application to send an APDU conveying an $N$-byte string to the Java card, which copies the received string to an internal buffer. In Figure B.1 the response time to a \texttt{put} command with $N = 20$ bytes is given. It must be pointed out that the maximal value allowed for $N$ depends on the Java card considered and varies from $N = 112$ to $N = 128$, although smaller $N$ are recommended to avoid timeouts. Thus, taking into account that hash values or spending program traces are typically 16 bytes long (the output length of MD5) or 20 bytes long (the output of SHA-1), it follows that a spending program cannot be downloaded as a block; a better solution is to use a \texttt{put} command for each trace, so taking $N = 20$ is realistic.

Odyssey is the fastest card for the \texttt{put} command, followed by SmartCafé and GemXpresso. It is somewhat surprising that the card with the most advanced processor is the slowest one, but possible explanations are that the benchmark does not take advantage of the 32-bit architecture and also that GemXpresso uses the JC API 2.0 specification (against JC API 2.1 for the other two cards, [Sun99a]).

Get

This command causes the Java card to compute a hash function and return the answer to the host application. Since using spending programs requires that the smart card compute hash values, the time needed to compute a hash is a sensible thing to measure. The MD5 hash algorithm ([RD92]) has been adapted to fit into a Java card. Adaptation can be done smoothly for Odyssey and GemXpresso, but the lack of the \texttt{integer} type in SmartCafé is a shortcoming. Implementing the \texttt{integer} type for SmartCafé in Java results in a prohibitive response time. Fortunately, SmartCafé is shipped with a basic cryptolibrary including DES and triple DES, RSA and also the
Secure Hash Algorithm (SHA-1). Being implemented in the native code of the processor, the performance of SHA-1 in SmartCafé is not to be compared to the performance of the Java MD5 for the other cards; nevertheless, the only possible alternative was to replace MD5 with native SHA-1 in the implementation of get for SmartCafé. With the above caveats in mind, it can be seen from Figure B.1 that SmartCafé offers the fastest get, followed by Odyssey and GemXpresso.

GetInput

In normal hash chain or spending program operation, hash chains or spending programs may be coded by the card or be downloaded onto the latter by the host application, but in any case they are stored by the card, who releases hash values or traces upon request by the cardholder willing to make a micropayment. The getinput command causes the Java card to return a stored trace, which amounts to returning a byte string. In Figure B.1, it can be seen that Odyssey offers the the fastest getinput, followed by SmartCafé and GemXpresso.

B.2.2 Programming environment

In this subsection, the programming environments of GemXpresso, Odyssey and SmartCafe are compared. The normal steps in developing a Java card application will be considered in turn: Java coding, code verification, bytecode conversion, bytecode download, cardlet test and host application development.

Java coding

Documentation and help tools are fundamental components of a programmer-friendly development environment:

GemXpresso This card is shipped with a detailed manual [Gem98] explaining the principles to develope Java card applications and giving several
examples to illustrate all functionalities offered by the card. The help tool is well organized into several complexity levels: from simple hints for the beginner to code optimization suggestions. Moreover, GemXpresso is the only card among those considered which offers a wizard to help the developer in creating a Java card application. The wizard is based on the Direct Method Invocation (DMI) developed by Gemplus, which relieves the programmer from handling APDUs by allowing her to specify a communication interface (a group of methods) between the host application and the cardlet; APDU communication is left to the DMI. The idea behind the DMI is similar to the principles of the Java Remote Method Invocation (RMI, [GJS97]) and CORBA.

**SmartCafé** The manual for this card [Gmb99] is less complete and assumes some degree of familiarity with Java cards. Still, the documentation contains a rich variety of examples.

**Odyssey** The Odyssey manual [Inc98] is the briefest one and contains only basic information, with a few examples. It is a useful document for experienced programmers.

**Code verification**

For a Java applet to become a cardlet, a verification stage is needed to make sure that the applet complies with the requirements of the Java card Application Programming Interface (JC API). The verification program (verifier) is supplied by the card manufacturer and obviously depends on the JC API version (2.0 or 2.1). The distracting trifle is that the verifier is also manufacturer-dependent, because it also checks card-specific restrictions. For example, there is no guarantee that a cardlet having passed SmartCafé verification for JC API 2.1 will pass Odyssey verification for JC API 2.1 (and conversely). This seriously hampers cardlet portability. It would be nice if verification was divided into two distinct sub-stages: a first one to check compliance with JC API 2.x, and a second one to check card-specific constraints.
It would even be nicer if the second substage could be eliminated.

The Odyssey verifier is the one returning the most information to the programmer, who can see the various steps of verification. GemXpresso and SmartCafé do not offer this feature, but require less parameters to be supplied by the programmer.

**Bytecode conversion**

The converter converts verified code to Java bytecode. Such code is compacted to save Java card storage and may also be signed and encrypted for security purposes. Like for the verifier, there are differences between the converters provided by each manufacturer, which is a hindrance to bytecode portability.

As can be seen from Table B.2, the Odyssey converter yields a shorter bytecode for the benchmark cardlet than the GemXpresso converter. No comparison is possible with SmartCafé, because the get command was implemented differently in the benchmark cardlet for this card (using the native SHA-1 function, as mentioned above).

**Bytecode downloading**

Once the cardlet bytecode has been obtained, it is ready to be downloaded to the Java card or to the card simulator provided by the manufacturer (see discussion on test below). GemXpresso and SmartCafé implement the concept of workspace, which integrates verification, conversion and downloading into a single application. The Odyssey card offers separate applications for each functionality, which is slightly more inconvenient for developers. The workspace tool provided by SmartCafé is especially convenient, as it detects Java code modifications and automatically triggers verification, conversion and downloading.

**Test**

There are basically three ways to test a Java card application:
**Direct test:** It is supported by all considered cards and it consists of sending APDUs to the Java card and checking the correctness of the received responses; debugging erroneous responses can be quite arduous.

**Simulator:** If a simulator is available, it can be used to test the cardlet for correctness prior to downloading it onto the Java card. Testing a cardlet on a simulator is faster and safer than direct test, but does not fully guarantee that the cardlet will run on the Java card (the simulator runs on the host, which has more storage, stack and processing power than a card). Only GemXpresso and SmartCafé provide a simulator.

**Simulator & Debugger:** Simulator test with debugger is only possible with SmartCafé, whose debugger allows to really see what happens when a particular APDU is sent. With the other two cards, a cardlet must be debugged using a standard Java debugger, which is less accurate.

**Host application development**

To build an operational system, a host application must be developed which is able to interact as a client with the cardlet stored on the Java card. Only GemXpresso supplies a client generator that automatically generates the client side and even allows the developer to choose in which language the client code is to be generated (C, C++ and Java are supported). In addition, GemXpresso provides information and examples to help the developer who prefers not to use the client generator. SmartCafé and Odyssey do not offer a client generator, and the latter does not even provide examples nor information to help the developer.

**B.2.3 Interim discussion**

From the speed and storage comparison of Subsection B.2.1, it can be seen that Odyssey is the fastest Java card in most cases, and GemXpresso tends to be the slowest one. However, from the developer's point of view (Subsection B.2.2), Odyssey is slightly outperformed by the other two cards. In
addition, the following features of GemXpresso and SmartCafé should not be overlooked:

**Context restoration** GemXpresso is the only card among those considered which is able to resume execution of the last command when restoring an aborted connection.

**Digital signatures** For spending programs to be accepted by merchants, at least their initial trace must be signed by the Java card on behalf of the cardholder. Implementing in Java a fifth benchmark command `RSASignature` on the considered Java cards was far from straightforward because of lack of storage. Fortunately, one of the considered Java cards offers a native code cryptolibrary including digital signatures (RSA). The average time required by SmartCafé to sign a trace or hash value is 1421 ms.

Given that none of the considered cards clearly outperforms the rest in the above overall comparison, availability of digital signatures in native code was the final argument to adopt SmartCafé for micropayment implementation.

### B.3 A JavaCard file system

It is apparent that, if a Java card is to implement the buyer’s functionality of the micropayment scheme presented in Chapter 3, then it must be able to store and manage spending programs. In normal micropayment operation, spending programs must be kept inside the Java card and can be deleted once they are used up. Unlike hash chains, each spending program implements a different spending pattern (possibly with different currencies, coupon amounts, sequencing, etc.) and it should be possible to store several programs at the same time, which raises the need for an on-card file system. However, the Java card API 2.1 does not offer a file system. We describe below the main implementation guidelines to build a Java card file system allowing dynamical allocation and release of variable-sized memory blocks.
We next state the main requirements for this file system:

A. *Storage reusability.* A shortcoming with Java cards is storage reusability. Even if the memory blocks allocated for a class instance are released, they cannot be reallocated to any new instance. Thus, creating and destroying successive instances eventually uses up all available storage. A Java card file system must find a way to sort out this problem.

B. *Fast data access.* To make up for the smart card limited processing capacity, data access should be as fast as possible. Data structures must be designed to meet this goal.

C. *Optimized storage.* Due to the limited storage capacity of a smart card, the Java card file system must be designed to save as much storage as possible. Therefore, shorter data types should be preferred, and vectors and structures should be kept as small as possible.

### B.3.1 Implementation

The file system consists of three main components (see Figure B.2):

- The *pool of blocks*, which is the collection of memory blocks where data are actually stored.

- The *array of free blocks*, which keeps track of the status of memory blocks (free or used).

- The *file descriptors*, which (like Unix i-nodes) keep track of which blocks are allocated to a given file. A specific feature of spending programs is that they can be stored as a collection of memory blocks of equal size (one program line corresponds to a block). Thus, for groups of contiguous blocks, the file descriptor corresponding to a spending program needs to store only the address of the initial block and the number of blocks used (see Figure B.2). This simplification matches requirement C listed above (optimized storage).
We next describe the Java interface of the proposed Java card file system. To start using the system, the user must create an instance of the FileSystem class. This is the only class instance that is created during the file system lifetime: to sort out storage reusability problems (requirement A listed above), file creation and deletion will be done within the FileSystem class instance and will be managed by the methods of this object. Five methods have been defined to manage files:

**create() returns byte** Create a new file within FileSystem and return the file identifier.

**open( fileId : byte )** Open a file for reading or writing. Before a file is first read from or written to, it must be opened.

**read( fileId : byte, data : array of byte )** Read an array of bytes from a specified file. The array must exist before calling this method.
write( fileId : byte, data : array of byte) Write an array of bytes to a specified file. The file is assumed to exist before this method is called.

erase( fileId : byte ) Remove a file from the FileSystem.

Example 2 Next follows a simple example on how to deal with files:

```
FileSystem fs = new FileSystem();
byte file1, file2;
byte data = new byte[ AnAmountOfBytes ];
byte outputData = new byte[ AnAmountOfBytes * 3 ];
file1 = fs.create();
file2 = fs.create();
fs.open( file1 );
fs.open( file2 );
fs.write( file1, data );
fs.write( file2, data );
fs.write( file1, data );
fs.erase( file2 );
fs.write( file1, data );
fs.open( file1 );
fs.read( file1, outputData );
```

The sequence of file operations above is as follows:

1. Create the FileSystem.
2. Create two files and open them.
3. Write several times into the files.
4. Erase the second file.
5. Write another time into the first file. The FileSystem will probably write the new block on the storage previously released by erasure of the second file.
6. Open the first file and read all the data stored in it.
Appendix C

Secure Contract signing

In any commercial transaction where buyer and merchant can see each other, the exchange of goods is not a problem: the actions of paying and receiving the goods bought (from the buyer’s viewpoint) or getting paid and delivering the goods sold (from the merchant’s viewpoint) can be done almost simultaneously. In electronic commerce, a security problem arises because there is no physical coincidence during the transaction between the buyer and the merchant. None of the parties wants to step forward first since they do not trust each other.

The traditional way of handling that problem is to rely on a trusted third-party (TTP) which is trusted by both buyer and merchant and arbitrates the whole transaction. Such approach presents the problem of adding another party during the transaction which increases communication and bandwidth requirements.

To avoid the need for a TTP, a secure contract signing protocol based on the exchange of secrets can be used. Secure contract signing minimizes the risk incurred by the party who first steps forward.

We next recall a secret exchange protocol described in [EGL85] which can be used as a primitive for secure contract signing. Assume that two parties $A$ and $B$ each have $2n$ secret $m$-bit numbers which they wish to exchange: \( \{a_i, 1 \leq i \leq 2n\} \) for $A$ and \( \{b_i, 1 \leq i \leq 2n\} \) for $B$. 

115
Protocol 9 (Secret exchange protocol)

1. A splits her $2n$ secret numbers in $n$ pairs, for instance $(a_{2j-1}, a_{2j})$ for $j = 1, \cdots, n$. Then, she sends to $B$ one element of each pair using a 1-2 oblivious transfer (e.g., the provably secure oblivious transfer [BPT95] can be used), which means that $B$ receives either $a_{2j-1}$ or $a_{2j}$, for $j = 1, \cdots, n$, but $A$ does not know which elements $B$ received (each element of a pair has $50\%$ probability of being transferred).

2. Simultaneously with Step 1, $B$ does exactly the same with his $2n$ numbers: he splits them in pairs and sends one element of each pair to $A$ using a 1-2 oblivious transfer.

3. $A$ and $B$ send to each other the first bit of all their numbers $a_i$ and $b_i$ for $i = 1, \cdots, 2n$, then the second bit, and so on. If $A$ wants to cheat $B$, she only has a probability $1/2^n$ of success because $B$ has already received $n$ out of the $2n$ secret numbers at Step 1 and $A$ does not know which ones. By symmetry, the same applies if $B$ wants to cheat $A$.

A drawback of protocol [EGL85] is that, if $B$ quits the protocol after $A$ has sent the $k$-th bit of her secret numbers then $B$ has a $2$ to $1$ advantage, since $B$ has $2^{m-k}$ choices to obtain one pair while $A$ has $2^{m-(k-1)}$ (twice as much). This problem was solved by Tedrick in [Ted85] with a modified protocol that significantly minimizes the disadvantage of the party that starts the protocol.

A secure contract signing protocol [EGL85] can be designed by taking a secret exchange protocol such as Protocol 9 as a building block:

Protocol 10 (Secure contract signing)

1. $A$ randomly generates $2n$ DES-like keys $K_i^a$ for $i = 1, \cdots, 2n$, and $n$ pairs of messages $(L_j^a, R_j^a)$ for $j = 1, \cdots, n$. Then she encrypts each
message with a different key: \( P_i^a = E_{K_j^a} (R_i^a) \) for \( j = 1, \ldots, n \) and \( Q_i^a = E_{K_{n+j}}^a \) \( (L_i^a) \) for \( j = 1, \ldots, n \).

2. \( B \) does the same (key and message generation), so that he obtains
\( P_i^b = E_{K_j^b} (R_i^b) \) for \( j = 1, \ldots, n \) and \( Q_i^b = E_{K_{n+j}}^b \) \( (L_i^b) \) for \( j = 1, \ldots, n \).

3. Either of both parties (or both) creates a contract containing the conditions of the transaction and a clause whereby the contract will be considered as signed if and only if \( A \) is able to decrypt \( P_i^b \) and \( Q_i^b \) for some \( 1 \leq j \leq n \), and \( B \) is able to decrypt \( P_i^a \) and \( Q_i^a \) for some \( 1 \leq j \leq n \).

4. \( A \) and \( B \) exchange the 2n secret keys using Protocol 9 (or any secret exchange protocol).
Appendix D

Off-line anonymous electronic payment

In 1982, Chaum [Cha83] presented the concept of blind signature to preserve buyer anonymity in electronic payments by making payments untraceable. Anonymous transactions were further developed in [Cha85]. The idea is that the buyer “mints” her own notes and presents them to the bank, in order for them to be certified and given value. The important thing is that, upon certifying a note, the bank cannot see the note number; in this way, when the note is later spent, the note number cannot be used to trace the identity of the buyer who presented the note for certification to the bank. Chaum’s blind signature protocol [Cha85] is next recalled:

Protocol 11 (Blind signature)

1. The buyer randomly chooses half the digits of a note number \( n \) and repeats these digits to form the note number \( n \) (this repeated halves property will be subsequently used to tell valid note numbers from junk). Then, the buyer picks a random integer \( r \) that will hide the note number to the bank’s eyes. The buyer computes \( x = nr^e \) where \( e \) is the bank’s public key that certifies a certain amount of money, say one euro.
2. The bank withdraws one euro from the buyer’s account and uses its private key $d$ to certify a one euro note, that is

$$y = x^d = ((n^e)^d) = n^{d_e} = n^d$$

Then the bank sends to the buyer the value $y$.

3. The buyer computes $z = y/r = n^d$ to obtain the certified note.

In Chaum’s anonymous payment system, when the buyer wants to spend her one euro note, she sends $z$ to the shop and the shop uses the bank’s one euro public key $e$ to authenticate $z$ by computing $z^e = n$ and checking that $n$ is a valid note number. Later on, the shop sends $z$ to the bank and the bank does the same authentication check and deposits one euro in the shop’s account provided that the note number $n$ has never been used before. The bank records $n$ to guard against double spending.

With the above approach, buyer anonymity is guaranteed but the bank still has to be on-line to prevent double-sending of notes. In [Bra93], an untraceable off-line cash system was presented, also based on blind signatures, which becomes traceable after double-spending. Since double-spenders will be identified later on, the bank does not need to be on-line to check the note numbers of each payment transaction. The payment model of [Bra93] is based on the representation problem in groups of prime order. The basic idea is that during the withdrawal protocol, the buyer embeds her identity into the coin in such a way that nobody can obtain it, provided that computing discrete logarithms is hard. Then, during the payment protocol, the buyer has to answer a time-related challenge from the server in such a way that the answer contains some information about her identity. This information must be such that one instance of it does not reveal anything about the buyer’s identity, whereas knowledge of the answers to two different challenges enables the bank to obtain the buyer’s identity. In this way, double-spending is deterred because it implies loss of anonymity for the double-spender.
Bibliography


[Cyb] Cybercash.

http://www.cybercash.com


[DF99] J. Domingo-Ferrer. Anonymus fingerprinting based on committed oblivious transfer. In H. Imai and Y. Zheng, editors,


[Dig] Digicash.


International Conference on Computer, Communications and Networks (ICCCN’96), pages 603–618, December 1995.


[Sri97] K. B. Sriram. Hashjava - a java applet obfuscator.


Jordi Herrera Joancomartí
April 2000