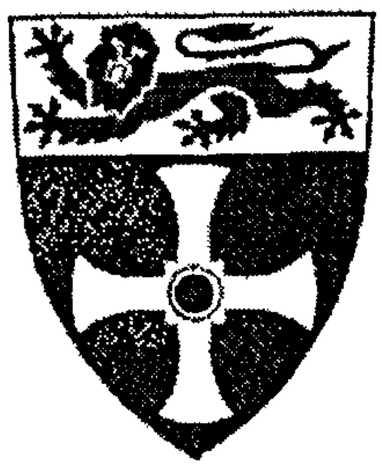
Anonymous and Confidential Communication using PDAs

Carlos Molina–Jiménez

Ph.D. Thesis

UNIVERSITY OF
NEWCASTLE



Department of Computing Science
The University of Newcastle upon Tyne

May 2000

Abstract

Anonymizers based on an intermediate computer (a set of them) located between the sender and the receiver of an e–mail message have been used for several years by senders of e–mail messages who do not wish to disclose their identity to the receivers. The job of the computer in the middle (the mediator) is to receive the message from the sender, delete the sender's address and other personal data from the header of the message, and forward the message to its final destination.

In this paradigm, there are no means to hide the identity of the user from the mediator simple because the message sent arrives in the middle computer, with information that easily leads to the identity of the sender.

The origin of this problem is that the sender uses a computer identified by an IP–address that unambiguously leads to the identity of its user. In fact, the sender discloses his identity to the mediator computer from the very moment he sends his message in the hope that the mediator will protect it.

Because of this, in this paradigm the strength of the system for protecting the identity of the sender depends on the ability and willingness of the mediator to keep the secret.

In this dissertation we propose a novel approach to sending truly anonymous and confidential messages over the Internet which does not depend on a third party. Our idea departs from the mediator approach in that we do not use an IP–addressed computer to send anonymous messages, we use an IP–addressless computer instead, to be specific, we use a Personal Digital Assistant (PDA) which is IP–addresslessly connected to the Internet with the support of a Mobile Support Station (MSS).

The PDA is identified by the MSS by a temporary, non–personal, random identifier (TmpId) which is assigned by the MSS and is valid only for one communication session. Thanks to the use of the TmpId, the sender of the anonymous messages does not need to disclose his identity to the MSS or to anybody else; thus, the strength of the system does not depend on any mediator.

Having observed that a public telephone box provides complete anonymity when operated by coins, we took its functionality as a paradigm for our system. Thus, the main idea of our approach is to make the PDA, the MSS, and the Internet communication infrastructure imitate the work of a public telephone box connected to the telephone network. For this to be possible the PDA user uses anonymous electronic cash to pay for his anonymous message.

To prove the feasibility of our approach and its correctness, the protocol of the proposed system was designed, specified in Promela specification language, and its basic safety properties and proper end–states were validated using the Spin validator.

A mi madre
por haberme enseñado a leer y a escribir
A Cubeito
por haberme enseñado a ver en la obscuridad
A mi tía Maye
por esperarme siempre

A mi maestro
Ramiro Velázquez Bustamantes

Acknowledgements

Financial support for this work was provided by the National Autonomous University of Mexico (UNAM).

I am grateful to the UNAM community, specially to Dr. Jorge Ize and Dr. David Rosenblueth for reading, and evaluating my annual reports and supporting my applications to extend my grant annually. Thanks also to Víctor Germán Sánchez and Sergio Rajsbaum.

I would like to thank Dr. Lindsay Marshall, my supervisor, for his valuable scientific advice and encouragement throughout the course of this work; his support in technical and administrative aspect made my student life in Newcastle a pleasant experience.

I would also like to thank Professor Santosh Shrivastava who patiently followed the slow progress of my research and never lost confidence in my naïve attempts to formulate and present my Thesis Proposal; He never ran out of encouragements even during this most difficult and vulnerable part of my stay in Newcastle.

My internal and external examiners Professor Santosh Shrivastava and Professor Gordon Blair, respectively, read my thesis with critical eye; their comments and suggestions improved the thesis.

I am also sincerely grateful to Dr. Steve Caughey who was alway there not only to read and comment on my Thesis Proposal drafts but also to listen to my vague ideas and to give me feedback. For reading and commenting on one of the earlier drafts of my Thesis Proposal I am indebt with Professor Larry Hughes as well. Richard Achmatowicz read some of the chapters and made helpful comments.

I thank A.M.P. Barcellos, Avelino Francisco Zorzo, Martin Beet and Richard Achmatowicz, my PhD colleagues, for having invited me to their *Fault Tolerance* discussion group. Thanks to them my understanding of fault tolerant systems has improved.

Further thanks go to Shirley Craig, as the librarian for the Computing Science Department, her ability to trace missing references is invaluable and her endless willingness to help will always be remembered.

My English friends Louise Wellington, Nick Brennan, John Sutton, Clare Stubbs, Terry Keane and Pauline Urry read chapters of my thesis and corrected subtle English mistakes. Many thanks to you all.

The pursuit of a PhD is a long and difficult task that involves more than pure academic research; love, encouragement and support from friends also contribute to success. In this respect, I express my gratitude to my international friends. Many thanks to Eligio Hernández García, Celso Mora Nava, Braulio and Flora. I also owe considerable debts of thanks to my friends Mercedes Albors, Emily, Sylvia, Gary, Carla, Jorge, Julio, Eli, Haydi, Janet, Paul, Mario, Mercedes De Grado, Carlos Zamora and John Holland. I also thank Elia Rendón, Liuda Nosova, Serguei Shevchenko, Margarito Vázquez, Juán Manuel Jiménez, and Martín Flores.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Recently, as the Internet and specially the World Wide Web, its most popular application, grows beyond academic and scientific environments to reach the masses (about 100 million hosts located in business and domestic buildings) concern about the use and abuse of the Internet information is growing. This issue is currently the subject of hot debates that involve academic, scientific, business, government, civil, and human rights organizations, and individuals. Several papers, books, and Web pages have been written to discuss this topic which is normally addressed as security, confidentiality, privacy, and anonymity in the Internet [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]. Although there is still disagreement about the terminology, it is generally accepted that security is the concept that encompasses all the others.

Due to space and time constraints in this work we will limit our ambitions to the study of confidentiality and anonymity only, yet the concept of security in the Internet will be always around. For this reason it makes sense to devote a few lines to this issue and to understand why the Internet is considered vulnerable to hackers' attacks.

1.1 Internet security

The design and creation of the Internet dates back to the late 1960s, when its purpose was research. At that time nobody could have predicted what the Internet would look like 30 years later and what applications would be running on top of it. Among other things, no concern was taken to protect Internet information as it relied on the trust, respect, honour, and appropriate behaviour of its users [11]. Consequently, the original TCP/IP protocols have serious security flaws. The most obvious of them have been reported and countermeasures against potential attacks have been suggested [12].

Among the most serious problems concerning Internet security is the lack of protection for Internet communications lines. When a bit of information leaves the sender's computer and travels to its destination it travels through rather exposed channels (open backbones), consequently, the message itself is exposed to all the dangers (both unintentional and malicious) of the outside world. A message should be readable only to its sender and recipient. Yet because the Internet designers did not consider encryption an important part of the Internet protocols, most of information travelling through the Internet is not encrypted and readily available to hackers [13, 14].

Besides the efforts to fix the problems, Internet applications still suffer from security flaws. Further, legal regulations to protect the personal data of Internet users are still in their infancy.

By *Internet information* we have in mind every single bit stored on Internet files or travelling through Internet channels, for example, databases, electronic libraries, payroll and control systems,

electronic transactions, e–mail and so on. On the other hand, *personal data* is a subset of the whole of Internet information and encompasses only data related to individuals as human beings and members of the society, i.e. their personal information, for example, their electronic and post addresses; geographical location of their desktop personal computer, and their current locations; IP address, operating system, browser, and the hardware of their personal computers. Likewise, personal e–mail, shopping preferences, bank card numbers, medical records, etc.

To see how the personal data of an Internet user might be abused, let us discuss how it is exposed in two of the currently most popular Internet application, namely, in the e–mail and the Web systems.

In terms of security the e–mail system represents a high risk; it is very common to receive e–mails from people we have not heard of before and from people we do not want to hear from; without being asked for agreement our e–mail address may appear in an e–mail list or end up in the hands of a friend's friend just because our friend decided to send a group message to everybody in his or her addressbook. It may also travel around the whole world in the headers of a chain letter.

Although most Web surfers ignore it, Web servers gather (for technical and commercial purposes) personal data about their visitors. As explained in [8, 3] due to the nature of the HTTP protocols Web servers create log files where the name of the Web client, its IP address, operating system, and browser are stored as well as the day, month, year, minute and second of the visit; and the name of the files copied.

1.1.1 Wireless computers and their vulnerability

Although radio communication has been around for nearly 100 years, most remote communication is currently performed over wire lines. Tapping a wired communication line is relatively easy, to tap a telephone line for example is it sufficient to locate the line, then find or make a hole in the plastic shield and make direct contact or put in place an induction device for capture; in other cases, a micro–pastille is introduced in the capsule for the receiver handset. Having the line tapped, the conversation can be listened, recorded or broadcast with a quartz transmitter microphone over hundreds of meters. With the advent of wireless communications networks, the problem of information eavesdropping has become more serious since air tapping does not leave behind any trace of the crime as with wireline tapping. Also, to tap a wire the intruder has to make a physical contact with the line which puts him at risk of being discovered, while with wireless tapping the tap can be performed at a distance; for example a conventional analog wireless telephone handset emits over a distance of 200 to 400 meters; to overhear a conversation the only thing an intruder needs is a receiver adapted to the wireless set frequency, for example a scanner able to work at 410 to 520 MHz [15, 14]. However, even with the increased security risks presented by wireless communications the use of wireless computers continues to develop.

Taking into account the latest achievements in microelectronic, computer communication and wireless communication technology one can safely bet on two things. First, that today's poorly integrated wired and wireless communication networks like telephone, Internet, cable–TV, cordless, cellular telephone, mobile data, satellite and wireless LAN networks are going to be interconnected and integrated into a single global, ubiquitous communication network. Second, that cheap pocket size wireless computers, will invade the whole world in the near future, there will be millions of them, nearly everybody will carry one in her pocket and use it as a personal day planner computer and pocket communicator to access the global communication network currently under development [16, 17, 18, 19, 20, 21]. In fact a reasonable number of such computers are already around, the Personal Digital Assistants (PDAs) being the most representative ones[22, 23, 24, 25].

1.2 Protection of identity and other personal data

There are law abiding situations, where it is essential for an individual not to disclose his identity (for a while or forever) after interacting with somebody else, i.e. not to disclose a particular piece of personal data or none at all [26, 27, 28, 29]. Some health services, such as assistance with embarrassing diseases, do not work unless the identity of the caller is protected; in today's societies people need indentity protection to express something that, according to some groups, should not be illegal but it is; for example, in some countries expression of political and religion views are rather limited unless the identity of the speaker is not disclosed.

As explained later in detail, this modus operandi is called *anonymity*. The early Internet did not provide this facility; for example, the recipient of an e–mail message can easily, by reading the e–mail headers, find out the sender's e–mail address; currently only with the help of non–standard e–mail services senders of messages can, with a certain degree of confidence, hide their identity. In the same way a facility to access and publish information anonymously is needed.

The growing interest in personal data protection goes far beyond the academic community —the real pressure to find a solution to the problem comes from the business sector. It has been widely discussed and accepted that security and privacy are the most significant barriers to implementing electronic commerce in the Internet; concern about the collection and use of the personal data of Internet buyers disclosed during transactions has been expressed; experts in the field believe that electronic business in the Internet will not take–off unless personal data of customers is satisfactorily protected [30, 31, 32, 33, 34, 35, 9]. In this direction the anonymity of the buyer and the confidentiality of information she sends and receives during a transaction are two of the most important aspects of personal data protection.

As the above discussed issues become more evident, projects are being launched to research the answer to the questions raised. For instance in the middle 90s Philip Zimmermann wrote the PGP program (Pretty Good Privacy) which allows people to exchange e–mail messages with privacy, authentication and digital signatures [36, 37]. PGP is largely based on public key cryptography.

An early attempt to protect the indentity of the senders of electronic messages over the Internet was the proliferation of free–of–charge electronic remailers in the middle of the 90's [38]. The basic principle of these systems was to hide the identity of the sender with the help of an intermediate computer which receives the e–mail, changes the headers of the message and forwards it to its final destination.

With the advent of the Web, remailers evolved into anonymizers which attempt to protect not only the identity of e-mail senders but also the identity of Web surfers as well. On–going projects in this direction are *The Anonymizer* [39], *The Lucent Personalized Web Assistant* [40], and the *Crowds* project launched by AT&T [41]. More recent and ambitious is the work being carried on by the TRUSTe organization and the W3C consortium under the TRUSTe privacy programme [42] and the Privacy Preferences Project [43], respectively. TRUSTe and P3P are different from the first mentioned project in that they attempt to give a general solution to the issue by the introduction of standard protocols and practices.

1.3 A new approach to protecting identity

As can be seen from section 1.2, much work has been done to protect the personal data of Internet users and much work is still in progress; with different approaches and solutions being suggested. However, despite this massive amount of publication and implementation, the aim of protecting the identity of Internet e–mail senders and Web surfers has not been satisfactorily achieved. As will

be discussed later in detail in chapter 3, the proposed solutions suffer from serious flaws that make them unsuitable for certain applications. The common, and weakest aspect of all the approaches suggested so far is their insistence on trusting a third party, i.e. somebody placed between the sender and the receiver.

In this work, we propose a radically different approach to protecting the identity of the sender of an electronic message over the Internet. We argue that it is not possible to send a truly anonymous message through the Internet from a wired desktop computer unless the sender unconditionally trusts the Internet communication infrastructure (computers, software, and human beings) that work as a third party between him and the receiver. Strictly speaking there are no reasons to trust such a third party since computers and software might succumb to hackers' attacks and human beings might be bribed and forced by legal orders to disclose the vital information for tracing the sender of the supposedly anonymous message.

Trying to send an anonymous message from a wired desktop computer with the help of an anonymizer based on the mediator approach is similar to making an anonymous telephone call from a home telephone using the well–known *caller's number protected* mechanisms. The degree of anonymity of his message heavily depends on the willingness of the communication network owner to keep it, and is reduced to zero if the latter fails to keep his promise. We claim that for an electronic message to be truly anonymous it has to be sent from a wireless IP–addressless computer, just as truly anonymous telephone calls are made from public telephone boxes or mobile phones that can be bought from electronic shops and are operated using pre–paid cards.

Having in mind the arguments discussed above, we propose a novel approach for sending anonymous and confidential messages which is inspired by the functionality of the coin operated public telephone box. A distinguishing characteristic of our approach is that we use a PDA to send the anonymous messages, which is connected to the Internet with the support of a Mobile Support Station (MSS) and without using any IP–address. Instead of using an IP–address to identified itself to the MSS, the PDA uses a temporary non–personal, random identifier (TmpId) which is assigned by the MSS and is valid only for the duration of a single communication session. To pay for the anonymous communication session the PDA uses anonymous electronic cash (e–cash). To prove that our novel approach is feasible, we design the system, specify it in the Promela specification language and validate its basic safety properties and proper end–states, by using the Spin validator.

The reason why we use a PDA as a communicating device is that PDAs fit smoothly in our paradigm and because we believe PDAs (or something similar) will be the most popular computers in the years to come. This is briefly discussed in section 1.1.1 and in detail in chapter 2.8.

The idea of considering a wireless IP–addressless computer as the ideal computer to make an anonymous call might sound rather unusual as normally computers connected to the Internet, both desktop and wireless, are assigned an Internet address which helps identify them within the Internet world and is used by them to send and receive messages [44, 45, 46].

However, it can be argued that a computer might not possess an Internet address and still be able to communicate with other Internet computers. Also, the assignment of an Internet address to a computer is a long and painful process as it involves registration with Internet authorities; this registration might takes days to complete. There are situations where the user does not want to go through a registration process because she does not know how to do it, she wants her computer to work immediately after purchasing it, or because what she has bought is a personal, small and cheap disposable one and it does not make sense to register it today and dispose of it three weeks later.

1.4 Understanding confidentiality, privacy and anonymity

Up until now we have assumed that the reader's understanding of the legal concepts we are talking about matches ours. Frequently this assumption is false. To avoid making this mistake we will stop to define precisely what we have in mind when talking about the core concepts of our work.

1.4.1 Confidentiality

Confidentiality is a complex legal concept whose origins in Great Britain can be traced back at least to 1848 when the *Prince Albert v. Strange* case of breach of confidence was taken to court[47]. Obviously, the definition of confidentiality has changed since it was first defined as *the right to keep secrets*. Because its main concern is information protection its definition has evolved to meet new developments in information technology. The intensive use of computer and communication technology for storing, processing and exchanging enormous amounts of information has changed not only the way confidentiality is protected but its definition as well.

Confidentiality is a requirement aimed at keeping sensitive information stored in any form from being disclosed to an unauthorized recipient.

It is noteworthy that by *any form* we mean any known form of storing information like the human brain, or mechanical, optical, magnetic and electronic media. Also, it is important to mention that a *recipient* may be an individual, a group, an institution or any combination of these.

From the above definition of confidentiality it follows that information of a confidential nature should be kept secret. In fact the essence of confidentiality is to protect secrecy. Although the purists will argue that confidentiality and secrecy are two different concepts[48] we will take them as synonymous on the ground that for the purpose of this work the subtle differences are unimportant. So, for the rest of this work we will use *confidential* and *secret* as interchangeable terms.

Confidentiality exists in everyday life. A society without any guarantee of confidentiality is difficult to think of. Confidentiality is necessary to protect secrets ranging from secrets of state to husband–wife relationships.

It is perfectly understandable that ordinary people would like to keep information about their private life, political preferences, hobbies, spending habits and so on, under confidentiality. On the other hand business people would like to keep information away from the eyes of their competitors; of particular interest to others is information about new technological developments, manufacturing costs, bidding plans and so on.

It follows that any innovative technology aimed at helping people with both everyday and business activities has to guarantee an acceptable level of confidentiality. Computer and communication technology is no exception. Confidentiality is a matter of major concern in the computer and communication world. Particularly interested in this problem are those whose services offer to the user exchange of confidential information over computer networks.

1.4.2 Anonymity

Anonymity is a condition in where an individual, group or institution interacts with others without disclosing his, her, or its identity.

The right to confidentiality protects the individual (group or institution) and gives him the right to decide when and under what circumstances to disclose his identity. Another option is to remain anonymous forever.

Anonymity plays an essential rôle in modern societies. There are many legitimate reasons, ranging from the trivial to business and political, why a law abiding individual, might wish to remain anonymous after interacting with a second party.

- Counselling services for people needing assistance with diseases such as alcoholism, drug addiction, AIDS, and so on are provided under anonymity, otherwise they would not be viable. Not surprisingly these services have been traditionally provided anonymously over telephone lines. The *Alcoholics anonymous helpline* may serve as a good example.

- Freedom of expression is greatly helped by anonymity. To feel free from any repression an individual may wish to express his personal convictions (political or religious views) against his employer or government under anonymity.

- Anonymous advertisements in the Internet are welcome by those who are seeking a new employment or a new partner. In the first case anonymity protects the seeker from jeopardizing his current job. In the second it protects him from bothering his current wife.

- Computerised voting would be useless without a guarantee of anonymity.

1.4.3 Confidentiality and privacy

It is worth noting that confidentiality and privacy are two concepts of major concern to those interested in computer and network security. Confidentiality and privacy protection have been addressed by several authors [49, 27, 28, 29, 50, 26, 10, 51, 52]. Unfortunately, most of them have failed to define clearly these two closely related concepts, to such an extent that they are used synonymously.

According to Munro [48] confidentiality and privacy are distinct concepts and should be carefully distinguished.

Privacy may be defined as a condition in which an individual can determine for himself when, how and to what extent information about his personal life, stored in any medium, is disclosed to others.

From the above definition it follows that the right to privacy concerns the right to protect personal information, i.e. information about personal affairs of individuals. For example, about wife–husband relationship, romantic affairs, sexual preferences, diseases, and so on.

On the other hand, as has been defined in 1.4.1, confidentiality encompasses any information an owner wants to keep secret whether it be about his personal life or not. One of the few author who has paid attention to the difference between confidentiality and privacy is Simson Garfinkel [53].

It follows that the right to confidentiality may apply to information of a private nature but not necessarily.

To make it clearer, it is worth noting that information of private nature may or may not be confidential.

For example, personal information like the age and marital status of an individual are traditionally disclosed to the public. In the same way, at the individual's discretion, information about his sexual preferences may be disclosed to the public. Obviously any personal information disclosed to the public is no longer regarded as a subject of confidence.

The above discussion gives rise to the following conclusion: Although confidentiality and privacy are not the same, they are strongly related. In fact, confidentiality is a wider concept than privacy; the first concept encompasses the latter, consequently, confidentiality is regarded as the first step towards privacy protection. Grounded on this understanding of concepts we have decided to address the issue of information protection from the view of confidentiality rather than privacy. Again, the reader has to keep in mind that most authors do not make a distinction between the two concepts and treat them as synonymous.

1.4.4 Confidentiality in computer networks

Following the Janson [10] approach to confidentiality, we will consider that confidentiality in communication networks encompasses two aspects.

Confidentiality of content of messages The content of a message travelling through the network must be protected against the threat of disclosure to unauthorized individuals.

Traffic confidentiality The origin and the destination of a travelling message must be protected against the threat of unauthorized observers finding out between whom messages are exchanged.

It follows that there are three aspects in this game that may become a target of attack: the source of the message, the destination, and the content of the message. An Internet user may require protection for one only or for any sensible combination of all of them.

Confidentiality in computer networks is not a new topic. Many works have been devoted to it, particularly to confidentiality of the content of messages. This level of confidentiality can usually be achieved by means of cryptographic mechanisms (see chapter 4). The introduction of wireless networks and its applications are demanding reconsideration of the issue about confidentiality not only because this new technology is more vulnerable to intruders (see section 1.1.1), but also because it opens possibilities for new applications not available with previous technologies. Needless to say many of these applications demand confidentiality of the content of messages and traffic confidentiality.

Traffic confidentiality is of major concern, to such an extent that many emerging and potential applications will not find their way into practical use unless users are guaranteed a mechanism to keep their identities secret. Electronic commerce is perhaps one of the most appealing of the applications that are waiting for confidentiality, cashless payment, for example, is heavily dependent on anonymity.

1.5 Summary

The original Internet was designed for use in the academic field where users' behaviour is normally appropriate. Consequently, the original TCP/IP protocols suffer from serious security flaws. As growth of the Internet escalates and it reaches the masses, assumptions about users' appropriate behaviour becomes unrealistic. Thus, countermeasures to prevent or reduce the risk of abusing Internet information must be implemented. Information sent, received and stored on the Internet must be disclosed only to authorized parties. The issue of information protection is strongly related to the concepts of confidentiality, privacy, and anonymity.

Confidentiality is a requirement whose aim is to keep sensitive information (personal, business, medical, etc.) stored in any form, from being disclosed to an unauthorized recipient. Privacy is defined as a condition in which an individual can determine for himself when, how and to what extent, information about his personal life, stored in any medium, is disclosed to others. Anonymity is a condition in where an individual, group or institution interacts with others without disclosing his, her, or its identity.

There are several applications (Alcoholics' Anonymous Internet helpline for example) whose success depends on the provision of a mechanism for sending and receiving confidential and anonymous messages. Fortunately, thanks to cryptographic techniques and the use of mobile devices, it is possible to send confidential and anonymous messages over the Internet.

Chapter 2

Global, ubiquitous communication for the new millennium

2.1 Introduction

Since the invention of the commercial telegraph by Samuel Morse in the late 1830's it has been recognised that remote communication is one of the key factors in the development of modern societies. Nevertheless for many years, due to technological constraints, remote communication has been restricted to the use of stationary terminals (transmitters/receivers), i.e. to the use of equipment that works wire–tethered to a well–known and non–mobile physical location. To this category belong the today's mature and widely deployed wired telephone network and the wired Internet.

It is true that for years airplane pilots have been using radio systems for communication with terrestrial control stations while in the air, however, these kind of systems serve specific purposes and are not available to the general public. Only recently (in the 80s) and thanks to the appearance in the commercial arena, cordless telephones, analogue and digital cellular telephones, mobile data networks, and satellite networks, and wireless LANs, mobile communication is becoming available to everybody.

As can be seen several wired and wireless networks are widespread while others are emerging; unfortunately, they are not well integrated yet. In this chapter we study how these networks are being interconnected to each other (wireless LANs are discussed in section 2.8.7) to form a global ubiquitous communication infrastructure. Also discussed are existing and potential services provided by the interconnected network, and how they are going to be accessed. There are two crucial concepts we will use intensively in this chapter and in the rest of this work, namely user mobility and ubiquity; to avoid any confusion about what we mean, we start this chapter discussing them.

2.2 User mobility and ubiquity

A crucial question in the context of the global ubiquitous communication network is mobility. Mobility is important because it is the basis on which ubiquity is grounded. What follows is a definition and discussion of these two basic concepts.

2.2.1 Aspects of mobility

Following Mohan's [54] approach, mobility involves two aspects: computer mobility and personal mobility.

A *mobile computer* is one that, after being identified by the network, is capable of performing its everyday functions —hopefully without any substantial degradation— independently of the point of connection to the network and regardless of whether its user is static or on the move and inside the area of coverage. For example, e–mail messages can still be delivered to the computer regardless of where it happens to be connected to the network. It is worthwhile pointing out that *connected to the network* does not necessarily imply a wired connection, a mobile computer connects to the network through a wireless interface. In addition mobile computers are normally small and light, therefore, easy to carry.

A closely related concept is that of *personal mobility* which implies that the user is provided with facilities for performing his computation and communication functions independently of both the terminal he uses and the network point of connection. It is based upon a dynamic association between a user and his current terminal (the computer he is using to get into the network).

In the context of this work we assume that, while on the move, a network user always carries his mobile computer with him, consequently, he never uses other computers but his own; on this basis we do not consider aspects of personal mobility.

2.2.2 Ubiquity

One of the main advantages of having a globally integrated network with support for mobile users, is the possibility of ubiquitous communication; meaning that regardless of where geographically (within the area of coverage of the communication network) the communicating parts are located, they can exchange messages, even if the transmitter, the receiver or both are on the move.

The notion of ubiquitous communication has generated a great excitement in the Internet community. Probably the most enthusiastic is the business community (banks and retailers for example) that see ubiquitous communication as an attractive complementing –and even alternative– platform to run their business on.

2.3 Cordless telephone networks

Cordless telephones were introduced in the late 1970s to allow users to roam around their houses while talking over the telephone. The first generation uses analogue technology and is known as the CT1 (Cordless Telephone) standard in the United States. In Europe it is known as the CEPT1 standard because it is supported by the CEPT (Conference for European Post and telecommunications). The system consist of a telephone handset and a base station, both of them equipped with a radio frequency communication interface (see section 2.8.5). The base station is connected to the wired telephone network and serves as a bridge between the cordless handset and the wire telephone network. Also the base station serves only a single telephone number. In the United States the base station transmits in the band of 46.6–47.0 MHz, and the handset in the 49.6–50.00 MHz band. The coverage of the base station is typically 100 to 300 meters [55, 56, 21].

Because mobility is essential in several applications, the CT1 analogue cordless telephones evolved to the second generation cordless telephone standard known as the CT2 standard in 1985 which works in the range of 864–868 Mhz and uses digital technology. This system was introduced in the United Kingdom for residential, business and Telepoint applications. The communication architecture is similar to that of CT1, but users had a wider area of mobility as Telepoint base

stations were deployed in railway stations, airports and shopping centers to provide cordless communication to users. A base station serves several handsets and supports the transmission of data up to 2.4 Kb/s. Unfortunately, user location was not implemented and Telepoint services were limited to outgoing calls only [13, 55, 56, 21]. The system was quickly redesigned.

To replace the CT2 standard the CT3 appeared in 1992. In Europe it is known as the DECT (Digital European Cordless Telecommunications Standard) and is supported by the ETSI (European Telecommunication Standard Institute). The DECT standard is digital and works in the 1880–1990MHz. It was designed to work inside buildings and campuses. In a way it resembles a PBAX (Private Branch Exchange) network but with a wireless interface, hence it is sometimes regarded as a wireless PBAX. Communication support to cordless handsets is provided by a set of base stations which are connected to each other and to the wired telephone network; each base station is responsible for one communication cell. Since the base stations offer user location and handoff, consequently the user has full mobility among the area of coverage. DECT was designed to handle high capacity, it can transmit voice and data up to 1.152 Kbs. Although the mobility it offers is limited to a relatively small area, it is suitable for a great deal of applications; certainly it cannot compete with cellular systems outside its area of coverage but it is cheaper in applications inside buildings. On the other hand the DECT standard specifies the requirements for interconnection with ISDN and GSM networks. In this way a DECT user located inside his company building may route a call through the local DECT network if the recipient is within the area of coverage or through the GSM network if the latter is outside, in both cases dialing the same number. This result in greater economy and efficiency as most of the calls made from a business company target numbers inside the company building. Similarly, the same DECT handset is used a communicator device to gain access to the local DECT network, to ISDN and to GMS, in other words, the DECT handset is a sort of universal communicator.

2.4 The Personal Communication Networks and its evolution

The Personal Communication Network(PCN) is known as the Personal Communication Services (PCS) in North America[57] and is the result of the evolution of a mobile phone system whose first deployments can be traced back to the early 1980s. In order to understand why the PCN is now emerging it is worth going through the different stages of its evolution.

2.4.1 First–generation mobile phone systems

The first–generation of mobile wireless phone systems were analogue and invented by Bell Laboratories about 1982. They were deployed in several countries: in the USA AMPS (Advanced Mobile Phone Systems) standardized was widely used; in England TACS (Total Access Communication System) was deployed; while in Japan the NTT (Nippon Telephone and Telegraph) was used [13]. It is worth mentioning that although the voice channels were analogue, these systems used digital control links between the mobile phone and the base stations.

The only service provided by these system was voice communication transmitted using frequency modulation techniques using two bands of frequencies; one for base station to mobile phone and another for mobile phone to base station transmission. The AMPS used the 870–890 and 825–845 MHz bands, TACS transmitted at 935–960 and 890–915 MHz, and NNT at 870–825 and 925–940 MHz.

In the late 1980s these systems evolved to use digital technology for both control and voice channels; as a result of this, the second–generation of mobile phone systems came to the scene.

2.4.2 Second–generation mobile phone systems

The second–generation of mobile phone systems is characterized by the use of digital technology. They have been in use in several countries since the early 1980s. Currently there are three international standards [13]:

GSM The Global System for Mobile Communications used in European countries.

IS–54 The North American Electronic Industry Association system used in the USA, Canada and Mexico. In contrast with the GSM and Japanese Personal Digital Cellular (PDC) systems which are fully digital, this system is digital–analogue, i.e. it enhances rather than replaces the old AMPS analogue system.

PDC The Personal Digital Cellular system used in Japan.

Second–generation systems offer advanced transmission techniques like speech coding, error correcting channel codes, and bandwidth modulation techniques. As with the first–generation systems, they use one band of frequency for transmission from the base station and another for transmission from the mobile phones. Another goal of these systems was provision of roaming and handoff capabilities across several countries; GSM for example supports roaming and handoff in most European countries; while IS–54 supports roaming and handoff across the three North American countries (USA, Canada and Mexico); unfortunately they only cover a limited region of the world, none of them supports these facilities worldwide; and none of them is recognized as a worldwide standard.

The three standards use TDMA/FDMA transmission techniques but they differ in the transmission bands: 935–960 and 890–915 MHz for the GSM, 869–894 and 824–849 MHz for the IS-54 and 810–826 and 940–956 MHz for the PDC. The PDC has also been assigned the bands 1429–1453 and 1477–1501 MHz for future use [13, 55].

Although these system were mainly designed for voice communication they also offer data communication facilities. For example the European GSM offers a transparent data service at 9.6 kbits/s [58]. The GSM system is considered one of the most advanced of its generation and better documented; hence we will discuss it further.

The Global System for Mobile Communication

The Global System for Mobile Communication (GSM) is the European mobile telephone system. It was developed by the ETSI (European Telecommunication Standard Institute). Having been designed from scratch it is a fully digital transmission system based on cellular infrastructure. The first GSM specification was finished in 1990 and joined by 17 Western European countries.

The current version of the standard is called The Digital Cellular System 1800 (DCS1800) [59] and is considered a successful attempt of of the ETSI toward the standardization of mobile communication systems. It is currently a working technology in use in over 50 countries, both inside and outside Europe (in Africa, Asia, Australia, and New Zealand).

Thanks to a successful standardization, a GSM subscriber can travel to any of the GSM countries with his GSM terminal in his pockets enjoying continuity of communication; regardless of where he goes his GSM terminal responds to the same number with a single bill to be paid at home.

Being a digital system, it belongs to the so called second generation of communication systems together with the Americans IS–54 and Qualcomm CDMA and the Japanese TDMA systems.

The DCS800 provides telecommunication voice, data (9.6 kbit/s) and a message service for delivering messages (up to 160 characters) both to and from a mobile device in a connectionless mode (while the mobile device is unavailable).

GSM radio channel structure

In GSM a geographical region is divided up into cells (1 to several kilometers). Each cell is served by a base station. Mobile terminals communicate with their current base stations through radio frequency waves. For this to happen the GSM has been allocated two frequency bands: 890–915MHz for transmission from mobile terminals to base stations and; 935–960 for transmission from the base stations to the mobile terminals. These bands are divided into 124 pairs of carriers spaced by 200 KHz. For example, the first pair of carriers consists of the frequency channels 890.2 and 935.2 MHz (for communication from the mobile terminal to the base station, and from the base station to the mobile terminal, respectively)[60, 61]. Each cell is assigned from one to 15 pairs of carriers.

The medium access scheme used by the GSM is based on a Time Division Multiple Access (TDMA) protocol. On this account the assigned spectrum to each channel is segmented in eight time slots of 0.577 ms to be shared by eight mobile terminals, this means that eight mobile terminals may be connected to the base station through each channel (one in each slot). Since each time slot is 0.577 ms the duration of a frame is 4.615 ms, consequently, a transmitter may transmit once every 4.615 ms. In terms of bps, each transmitter may send 9600 bps of data[61]. The reader interested in TDMA in wireless LANs should refers to [62].

Identification in the GSM

The Global System for Mobile Communication (GSM) uses smart cards for subscriber identification. The smart card serves as a Subscriber Identity Module (SIM) and is plugged into the GMS device (a mobile or fixed one) to associate the user with the latter [63]. It contains a serial number and the telephone number. The subscriber is identified through the information stored in his smart card. This means that the smart card is strictly personal, while the GSM device is not. Therefore, the subscriber can take his smart card from the GSM device he is using and insert it into another one that offers GSM services (in taxis, and airplanes for example).

When the card is inserted and the GSM device turned on, the system ask for a PIN number. Provided that he types the PIN number that the system expects (in agreement with the information stored in the smart card), the system identifies the current device as being used by a subscriber it recognizes who is then identified by an international subscriber identity.

The international subscriber identity is used for forwarding calls to the subscriber's current geographical location and also for billing him.

Subscribers and users in the GSM

Before going further it is useful mentioning that when it comes to talking about identification in mobile communication systems some authors make a distinction between subscribers and users [64]. In these terms a *subscriber* is an organization or a person that has a contact with the service provider, with a telephone service provider for example, for use of the service. A *user* is the person that uses the service on behalf of the subscriber. It is commonplace that a big company has a contract with the telephone service provider for use of several lines. In this case the company is the subscriber and its employees —the users of the lines— are the users. Needless to say the telephone service provider makes business with the subscriber, it does not care about the users. It is up to the subscriber to set internal policies about the use of the lines. To the telephone service provider this situation looks like a single person, to whom several lines have been assigned.

To make our discussion simple we will assume that each subscriber has only one subscriber identity module. On this account from now on we will not make any distinction between subscribers

and users and will use these two concepts as synonymous.

2.4.3 Third–generation mobile phone systems

As second–generation mobile phone networks are still being deployed a third generation is emerging. This system is a digital one and expected to integrate existing and future wire and wireless phone systems and called the PCN (Personal Communication Network) in Europe and PCS (Personal Communication Systems) in North America. Briefly, the PCN can be described as a system with enhanced capabilities for worldwide ubiquitous multimedia communication.

The PCN

The PCN is aimed not only at supporting the existing services provided by existing second–generation mobile systems but also to provide services not previously implemented [13].

- ubiquitous communication: based on personal and terminal mobility the PCS will provide facilities for communication between two parties *anywhere* at *any time* i.e. regardless of the terminal they use two parties will be able to communicate at any time independently of their geographical location even when one of the or both are on the move.

- single universal phone number: Users will have a mobile handset which will respond to the same number regardless of where in the world the user is located; naturally, users will get a single bill.

- customized set of services: independent of location a user will have the services she is used to.

- high–functionality handset: The mobile user handset is expected to evolve towards a mobile device with multimedia data communication and computation capabilities; among those capabilities are: voice telephony, voice e–mail, fax, video telephony, teleconferences, database access, navigation, location, etc.

As can be appreciated most of the services are wireless version of today's services offered by wired networks (PSTN, Internet) while others come from existing wireless isolated networks like paging systems and GPS (General Positioning System). The PCN is envisioned as a system that will integrate all these wire and wireless networks.

In a system like this it certainly hard to tell whether the handset will be a mobile phone or a mobile computer of the size of a PDA (Personal Digital Assistant). Not surprisingly the PCS is being leveraged by both communication and computer companies among them AT&T, Motorola, IBM, Apple and DEC. It follows that the development of any wire or wireless computer expected to operate beyond 2000 has to match the goals of the PCN. Thanks to this leverage in 1992 the WARC–92 (World Administrative radio Conference 1992) of the ITU identified global bands 1885–2025 MHZ and 1210–2200 MHz for the PCN under the banner of International Mobile Telecommunications–2000 (IMT–2000) —formerly known as Future Public Land Mobile Telecommunication Systems (FPLMTS), those bands include 1980–2010 MHz and 2170–2200 MHz for satellite communications [65]. In response in 1995 the FCC of the USA allocated the band 1700–2300 MHz to the PCN under the name of Personal Communication Systems [61].

The development of a standard for this system is at its early stage, however, more than one proposal has been made [13]:

- Internationally the International Telecommunication Union (ITU) is currently developing standards for wireless personal communications under the name of Universal Personal Telecommunication (UPT). The task has been assigned to the Radiocommunication Sector (ITU–R) —formerly known as the CCIR (Comité Consultatif International des Radio–Communications)— supported by the Telecommunication Standardization Sector (ITU–T) —formerly known as the CCITT (Comité Consultatif International des Télegraphes et Téléphones [65].

- In Europe the ETSI (European Telecommunications Standard Institute) has created a special group to prepare standards for what they call the Universal Mobile Telecommunication System (UMTS).

- In the USA the ATIS (Alliance for Telecommunications Industry Solutions) has assigned the task of developing standards for PCS to its subcommittees T1E1, T1M1, and T1S1. The same task is being done by the committee TR46 of the TIA (Telecommunications Industry Association) and by the committee 802 of the IEEE (Institute of Electrical and Electronics Engineers).

2.5 Mobile data networks

Mobile cellular phone networks like the European GPS, the North American IS–54, and the Japanese PDC were designed mainly for voice communication [55, 13]. Although they can also transmit data messages at 9.6 Kbps, they have to compete against mobile data networks in this field.

Mobile data networks have been designed specifically to provide data services in urban regions and offer data rates of 8 to 19.2 kbps. They offer wireless data transmission upon which several applications can be built; among the most important are: Internet access, e–mail, remote database and file access, wireless bank card verification, and real time vehicle (taxis, trucks) location.

Currently MOBITEX (developed by Ericsson), ARDIS (developed and run by Motorola), and CDPD (the Cellular Digital Packet Network introduced by IBM) dominate the market.

In an attempt to reduce implementation costs, CDPD shares base stations with the existing analogue AMPS cellular phone network; it has been designed as an overlay to the cellular phone network and uses idle voice channels of the latter [66, 13, 67].

In contrast, MOBITEX and ARDIS have deployed their own dedicated networks using the SMR (Specialized Mobile Radio) frequency near 800–900 MHz [13].

So far, the mobile data network that has been widely accepted all over the world and considered the *de facto* standard is MOBITEX.

2.5.1 Advantages of mobile data networks

In contrast with cellular networks which use circuit–switching mode, mobile data networks are designed for packet switching mode. This approach gives mobile data networks remarkable advantages in data transmission; which explains their wide acceptance in certain applications [68].

- For the transmission of small quantities of data, a mobile data network offers higher performance and lower cost than a cellular network.

- A mobile data network provides its users with store–and–forward capabilities. This helps the mobile terminal save energy since the user may switch off his mobile terminal knowing that any message that might come during the so called saving–power mode will be stored by the

MOBITEX system in a mailbox until he switches his mobile terminal on and connects to the system to open his mailbox. Similarly messages are stored when the user is unreachable (perhaps going through a tunnel).

2.5.2 MOBITEX

In the USA MOBITEX has been deployed in 7700 cities and towns covering over 90 % of the USA business population and about 17600 Km of interstate highways with roaming support across all covered areas; also, MOBITEX networks have been deployed in 16 countries including Canada, United Kingdom, France, Sweden, Finland, Norway, Belgium, the Netherlands and Australia [68]. The radio frequencies used depend on the country. Yet in North America it operates at 900 MHz using a bandwidth of 935 to 940 for downlink channels and 896 to 901 KHz for uplink channels. In other countries it normally operates in the 450 MHz band [68]. Although currently MOBITEX transmits at 8 kbps, it is expected to be increased to 19.2 kbps in the near future [56].

The mobile terminal the user uses to connect to the MOBITEX network consists of a portable computer and a radio modem. Physically the radio modem interfaces with the portable computer through an RS–232 interface at one side and with the MOBITEX network at the other using an air interface protocol. Optionally the portable computer and the radio modem may be implemented in the same physical unit [68].

2.6 Satellite networks

It is believed that personal communications are going to evolve from location–dependent into universally ubiquitous in the first decade of the 21st century [21].

Pocked–sized personal electronic devices with communication and computational power (similar to mobile phone handset and PDAs) will be used to access remote information over a web of wire and wireless networks. It's expected that those devices will be able to send/receive real time multimedia information retrived with the help of software tools similar to current World–Wide–Web browsers.

To be able to provide those services a wire and a wireless communication infrastructure is needed. Besides their current bandwidth limitations the already deployed PSTN and Internet network can serve as wire backbones for many of the potential applications. The issue about a similar worldwide wireless communication infrastructure remains open. It's true that present–time cellular telephone and paging companies offer rather useful services, yet their systems are far from having a worldwide coverage; at the most, they offer continental coverage. Another limitation of these systems is that they are economically attractive for crowded urban areas only; hence they normally do not cover rural communities; nor do they cover deserted areas where occasionally someone might appear to carry on a research, to have a holiday, be lost, or for some other reason.

It seems obvious that a worldwide wireless network infrastructure is needed to complement the wired one and to interconnect the already deployed and emerging wireless systems; also, such a network has to cater for currently uncovered services like worldwide message (paging and telephone, for example) and data services for mobile users. A possible answer to this question is the use of satellite communication systems.

2.6.1 Satellite communications

Thanks to their high location (thousands of kilometres up in the sky) and their wireless communication medium satellites can offer unique features that can certainly complement both wire and wireless terrestrial communications [69, 70, 61].

Wide coverage A single geostationary satellite (see 2.6.2) can cover 1/3 of the earth's surface. In other words, a *constellation* (a group of satellites working for the same purpose) of three of them can cover the whole surface of the earth (the polar regions excluded). Needless to say, communication takes place regardless of the distance and obstacles between the communicating points.

Wide mobility support Worldwide communication is guaranteed for everyone located under the satellite communication umbrella, even for users on the move walking, driving, sailing and flying.

Independence of geographical impediments A satellite communication infrastructure is a suitable solution for hostile terrains (archipelagos for example).

Flexibility Having the satellite in orbit it is relatively easy and quick to deploy a communication network over a wide geographical area and to reconfigure it according to changes in user location and traffic requirements; this facility could be the answer to the problem of casual concentrations of mobile users for short periods of time (at football stadiums for example); moreover, in cases of terrestrial catastrophes when terrestrial networks are normally damaged a satellite link might be of great use.

Broadcast capability A satellite beam is inherently a broadcast medium; for applications of broadcast nature like remote conferences satellite communications might offer advantages over terrestrial ones.

The use of satellite in commercial communications has experienced substantial progress since the *morning bird* (the first commercial communication satellite) was launched by INTELSAL (International Telecommunications Satellite Organization) in 1965 [69, 71]; based on analogue techniques it was capable of carrying a total of 240 telephone circuits or one television channel between Europe and North America. Since then, satellite communications have evolved in different directions; of special interest for the designer of mobile computer applications is the evolution in satellite orbits and in on–board processing power.

2.6.2 Satellite altitudes

The capabilities and limitations of communication satellites heavily depend on the altitude of their orbits; on this basis they are grouped into geostationary and non–stationary satellites.

Geostationary

The first commercial satellites orbit the earth at an altitude of about 36000 Km; at that altitude and being the orbit in the equatorial plane the satellite looks to a terrestrial observer like a motionless point in the sky; i.e. the satellite period is equal to one sideral day; consequently they are known as *geostationary equatorial orbit* (GEO) satellites. Geostationary satellites usually operate at the 4/6 GHz frequency band for some applications, like TV broadcasting, geostationary satellites are attractive because they cover large geographical areas and —thanks to their fixed position relative to their terrestrial stations— they do not need additional tracking equipment; unfortunately due to their high position they suffer for serious drawbacks:

- For transmissions of the order of 19.2 kbps, they require terrestrial stations with high transmitting power (about 1 W) and large antennas (of the order of 1–2.4 m of diameter) [72, 61, 73].

- Rotating in an orbit located in the equatorial plane they cannot cover high latitude regions of the earth.

- Again, due to the altitude it takes a half–second to transmit a data packet between two terrestrial point. such a delay is too long and rather annoying for voice communications [74].

On this account it follows that GEO satellites are unsuitable for personal communication applications to be run in small pocket–size devices like current PDAs and mobile phone handsets which have strong constraints on power consumption and antenna size.

A possible answer to this question is bringing the satellite closer to the terrestrial terminals. This approach is taken in the so called Low Earth Orbit (LEO) satellites and by the Medium Earth Orbits (MEO) satellites.

Non–stationary satellites

LEOs and MEOs orbit the earth at about 200–3000 Km and 18500 Km respectively. Being out of the equatorial orbit both LEOs and MEOs are *non–stationary*. At the price of additional tracking equipment and by deploying them in constellations (dozens of satellites) to cover the whole earth surface they can overcome the problem of propagation delay inherent in GEOs. Also, some of them operate at frequencies between 1 and 3 GHz (at 1.6 GHz for example [1]) making it possible to communicate with cheap (hundreds of dollars) battery–powered handset devices with small antennas (similar to those used by current mobile phone handsets).

2.6.3 Transparent repeaters and on–board processing

The on–board communication subsystem of satellites consists of a number (12 for example) of *transponders* (receiver–to–transmitter). The job of a transponder is to receive the uplink signal sent by the terrestrial station, convert it, and transmit it on the downlink to the terrestrial recipient station. Depending on the converting operation performed on the signal, transponders are divided into *transparent repeaters* and *on–board processing*.

Transparent repeaters

Transparent repeaters are also known as *non–regenerative* and *bent–pipe repeaters* [69]. As their name implies, a transponder of this type is basically a repeater which receives the uplink signal from the emitter terrestrial station (in the uplink frequency), translates it into the down–link frequency (to avoid possible uplink/downlink interference), amplifies it, and sends the amplified signal back to the recipient workstation (one or many). In terms of the OSI reference model, transparent repeaters focus on the physical and data–link layers [73]. The main advantage of transparent repeaters is that they offer a high degree of flexibility when it comes to integrating the satellite into a wide communication network; for example, they are transparent to different modulation methods, whether they be analogue or digital like FM (frequency modulation) and M–PSK (M–Phase Shift Key) respectively [69] (modulation methods are described in [75, 62]); however, they cannot perform any processing on the received signal.

---

[1] The FCC has allocated the so called L–band (1.6465–1.66 GHz for uplink and 1.545–1.5585 GHz for downlink communications [75].

On–board processing satellites

Thanks to technology innovations satellites with on–board processing transponders are becoming popular. Having more sophisticated electronics, these transponders are able to perform operations on the received signal like demodulation, decoding, error correction on bit–streams, recoding, remodulation, and retransmission; also on–board processing may include buffering, compression, transponder and beam switching, routing, intersatellite traffic, and so on [76]; as can be seen, these transponders are suitable for digital communication techniques.

The result of this is that new satellites will support a variety of packet–oriented services similar to those available at terrestrial computers connected to networks. In other words, satellites with on–board processing transponders will perform functions belonging to layers 1, 2 and higher, of the OSI reference model [73]. However, these on–board processing facilities do not come for free. The on–board protocols place a rigid constraints on the terrestrial stations in terms of bit rates, packet formats, communication protocols, and so on; consequently, satellite access is restricted to those terrestrial stations that understand the satellite protocols; these constraints lead to difficulties in integrating satellites into large internetworks. Nevertheless, since digital communication is the communication technology of the future, new satellites are currently being designed with processing transponders on board [73].

Satellite–based personal communication services

Satellite communication systems will certainly play a fundamental rôle as platform for the personal communication services (PCS) of the 21st century; LEOs, MEOs and GEOs are expected to be integrated into the global communication network; hence, work is being carried on to test their suitability, and towards their standarization [77, 20]. Besides the advantages mentioned in section 2.6.1, satellites will not come to conquer the whole communication market; if they dared, they would face strong competition offered by terrestrial communication systems; it is unlikely that satellites will compete favourably against optical fibre networks and cellular phones in big cities since in populated regions these systems offer better parameters in terms of costs and performance. Because of this, it should be understood that satellites will enter the communication business not to compete against but to complement the existing terrestrial communication systems in those regions where the latter are technically or economically unsuitable. At the risk of being proven mistaken by practice; we guess that terrestrial and satellite communication networks will be integrated into a universal one where populated regions will be dominated by terrestrial networks and satellite networks will cover the rest. Additionally satellites may take advantage of their broadcast inherent facility to provide broadcast–natured services in populated regions; emergency and advertising messages, and time setting are just two examples of services suitables for satellite transmission.

Bearing this in mind investors in the PCS are currently designing satellite communication networks of LEOs in the hope of attracting the attention of mobile phones and mobile computer users. Perhaps the best known technology in this direction is the *Iridium* headed by Motorola [78, 61, 73, 79]. The Iridium satellite system has been operational since November 1998 and consists of 66 LEOs with on–board processing functions and is integrated into a constellation to provide worldwide coverage of communication services to mobile users in possession of handset devices with communication and computation abilities. Among the services provided are global voice phone messages, fax and paging [80].

Another promising satellite network offering similar services is the *Teledesic* constellation [73] which will consist of 840 LEOs orbiting the earth at 700 Km of altitude and with powerful on–board processing functions to support packet–switching asynchronous transfer mode communications.

Teledesic will support a wide variety of terrestrial terminals and bit rates ranging from 16 kbps to 2.048 Mbps.

Based on the previous facts, it makes sense to speculate that for PDA (and other personal communicators) to be fully integrated into the ubiquitous universal communication network, they have to be able to interact with both terrestrial and satellite communication networks.

2.7 Integration of wired and wireless networks

In most industrialized countries people are familiar with services provided by the communication networks we have studied.

In the future the number of these communication networks and their services is expected to be even larger. If this is true, we are on the way to ending up with a mess of incompatible networks offering similar services unless some work is conducted toward their integration. The purpose of this integration is to ensure that a user, be he indoors or on the move, is provided with the communication services he demands, no matter what terminal he is using or what computers or networks his information travels through on its way to its final destination.

For this to be possible it is necessary to integrate all the individual networks together into what we foresee as a *global ubiquitous communication network* (we will call it *a global communication network* for short) i.e. a network made up of multiple interconnected local and wide area networks with the already well established wired telephone and Internet networks serving as backbones.

Projects aimed at the integration of these networks are currently under way. The International Telecommunication Union (ITU) is supporting the so-called Future Public Mobile Land Telecommunication System (FPLMTS) project that will provide a world-wide Personal Communication Network (PCN). In Europe The RACE programme was lunched in 1987 and include projects to identify the enabling techniques for what would be the Universal Mobile Telecommunication System (UMTS)[81]; it concluded its activities in 1995; more exactly its activities were continued by the R & D into Advanced Communications Technologies and Services (ACTS) programme [16, 19].

In the USA the Defence Advanced Research Agency (DARPA) initiated the Global Mobile Information System (GloMo) programme in 1994. The GloMo aimes to conduct research on new opportunities for advancing the state of the art in mobile, wireless, multimedia system technologies[20].

The essential goals of FPLMTS[65], UMTS[59], and GloMo[20] are the same. The system everybody has in mind is in fact the **global worldwide universal ubiquitous communication network** expected to be at least partially operational in 2000. In order that this goal be met the following must be achieved:

- Integration of existing wired and wireless networks. The PSTN, ISDN, B-ISDN, Internet and cellular telephone network to mention some of them.
- Deployment of services for delivering voice, video, and data communication between ubiquitous communicating counterparts, be they people or computers. Among these services are[57]:
  - dialogue (eg., speech, video telephony)
  - messaging (email, fax, paging voice)
  - information retrieval (eg., multimedia WWW documents, voice, music, video on demand, newspapers)
  - access to electronic libraries

- When applicable the quality of wireless services should match that of wired networks.
- Support of unlimited mobility for both computers and users.
- Development of new computing techniques supporting mobile computing.

A picture of how the global ubiquitous communication network will probably look in the near future in presented in figure 2.1. In this figure a computer equipped with a wireless antenna and called a mobile support station plays the rôle of a bridge between the wireless PDA and the wired world. This will be explained below in section 2.7.1.

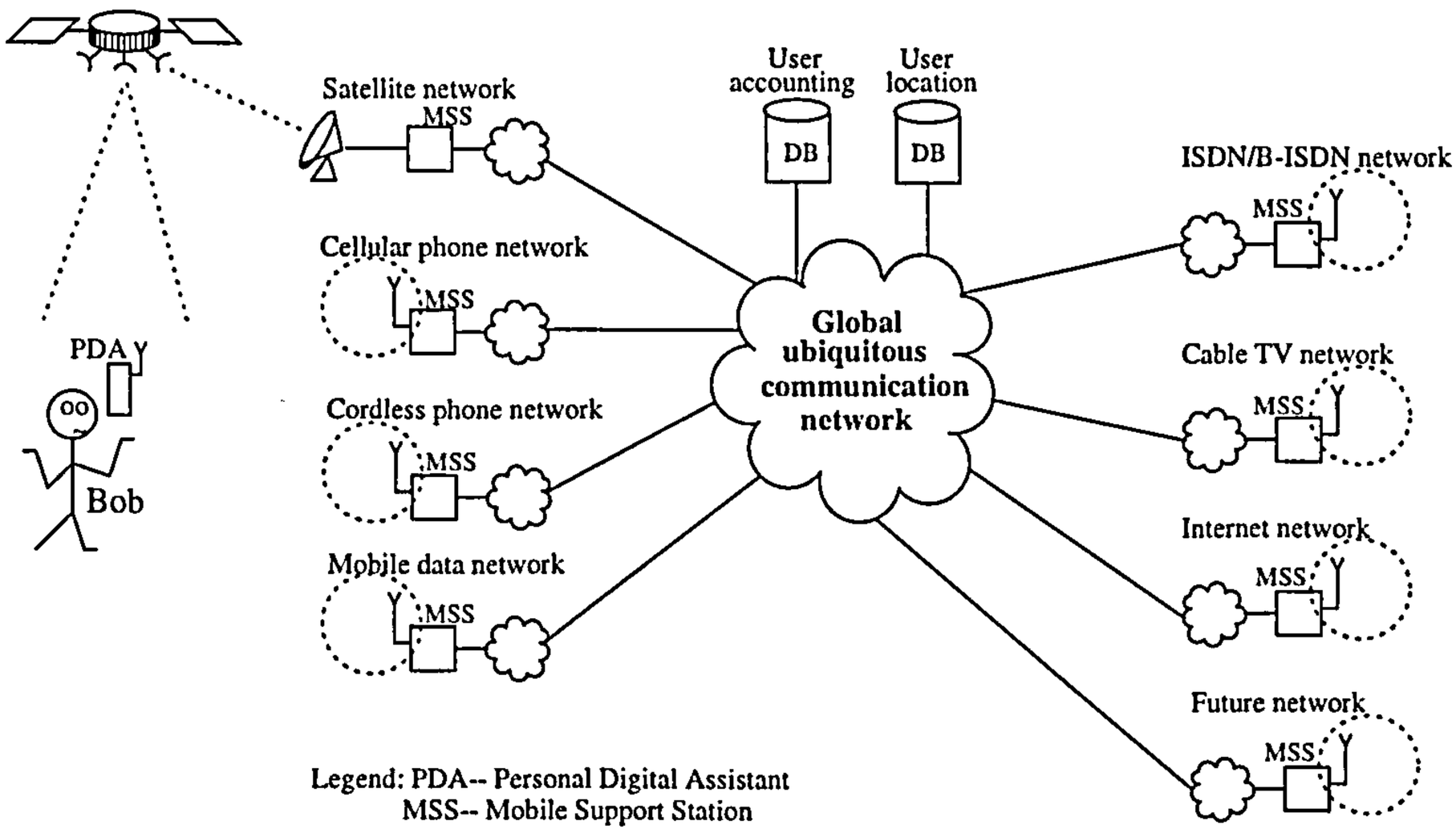


Figure 2.1: The global ubiquitous communication network.

As illustrated in figure 2.2 there are many existing and potential services that make our predictions appealing to both ordinary individuals and business oriented people, an individual will use this global ubiquitous communication network to access several facilities:

- to exchange information (e–mailing) with his wife who is home and with his son who is on his way to the cinema.
- to retrieve information from his office, from a central database for example.
- to access publicly available databases; for example, databases of job vacancies, tourist attractions, etc.
- to access remote available services like bank transactions, Internet shopping, train booking, weather forecasts, financial news, and so on.

2.7.1 WAP protocol

An essential component of the global ubiquitous communication network is the mobile computer represented in figures 2.1 and 2.2 by Bob's PDA. Although in the figure is called a PDA, it can be any electronic device equipped with a wireless antenna to send and receive messages and, in most

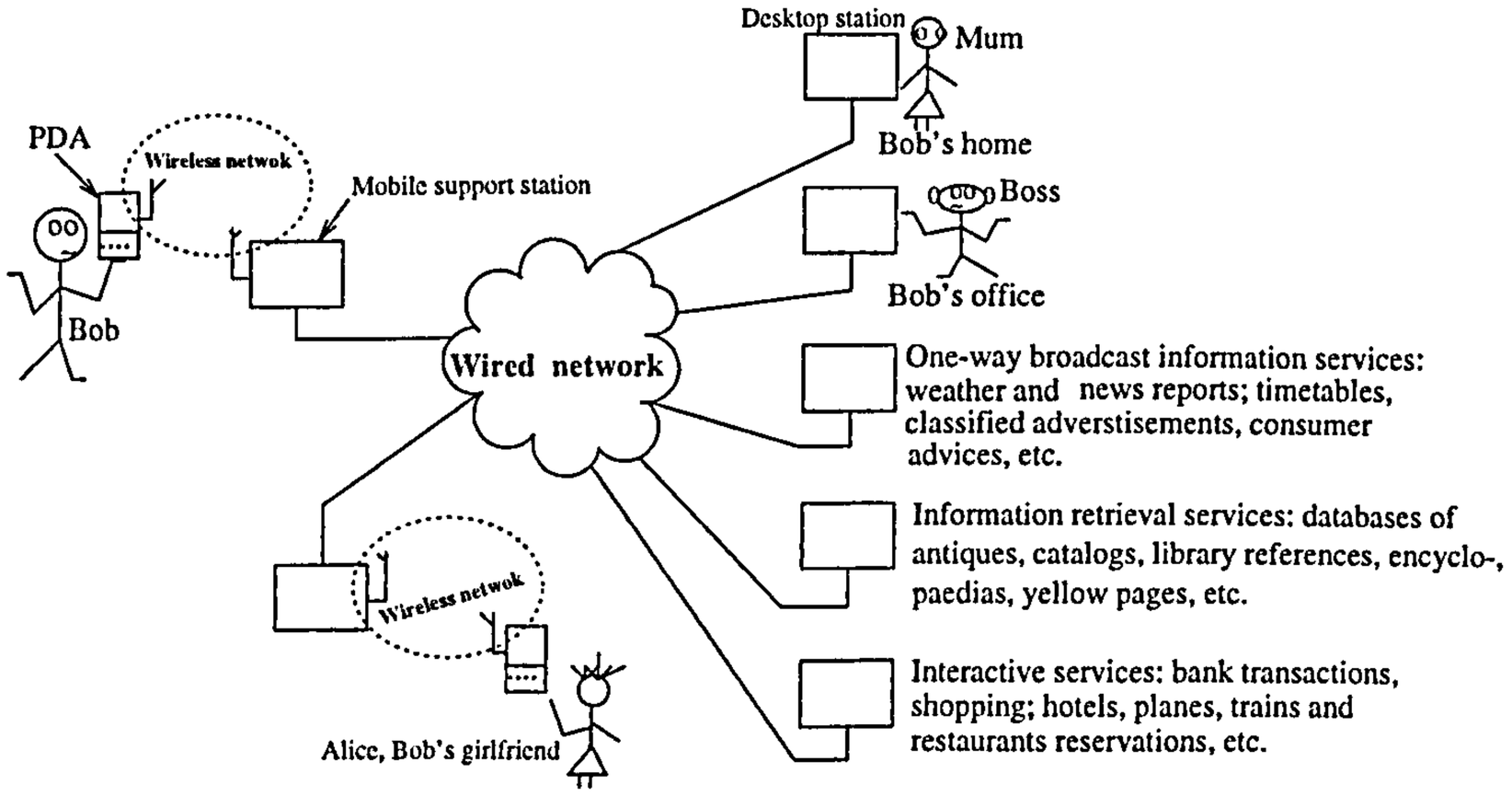


Figure 2.2: The global ubiquitous communication network and its services.

cases, with computational power to process information. Good examples of such electronic device are: PDAs, mobile telephones, pagers, and other handheld communicators.

The question about how a mobile computer like a PDA may be connected to the Internet has been the subject of several papers published in the early 90s [46, 82, 83, 84]. A paradigm that became widely accepted was proposed by Ioannidis [46] in 1991. In this paradigm the mobile support stations depicted in figures 2.1 and 2.2 are ancillary computers whose work is to receive messages originated at PDAs, locate the recipients (one or many) of the message either in the wired or wireless network, and route the message to its final destination. Similarly, a MSS delivers messages addressed to PDAs currently located within its area of coverage. Needless to say such messages may be originated at the MSS itself or come from a remote computer. A critical issue of mobile networking is that of mobile host location and routing of messages to them. This issue has received attention in published papers [85, 86].

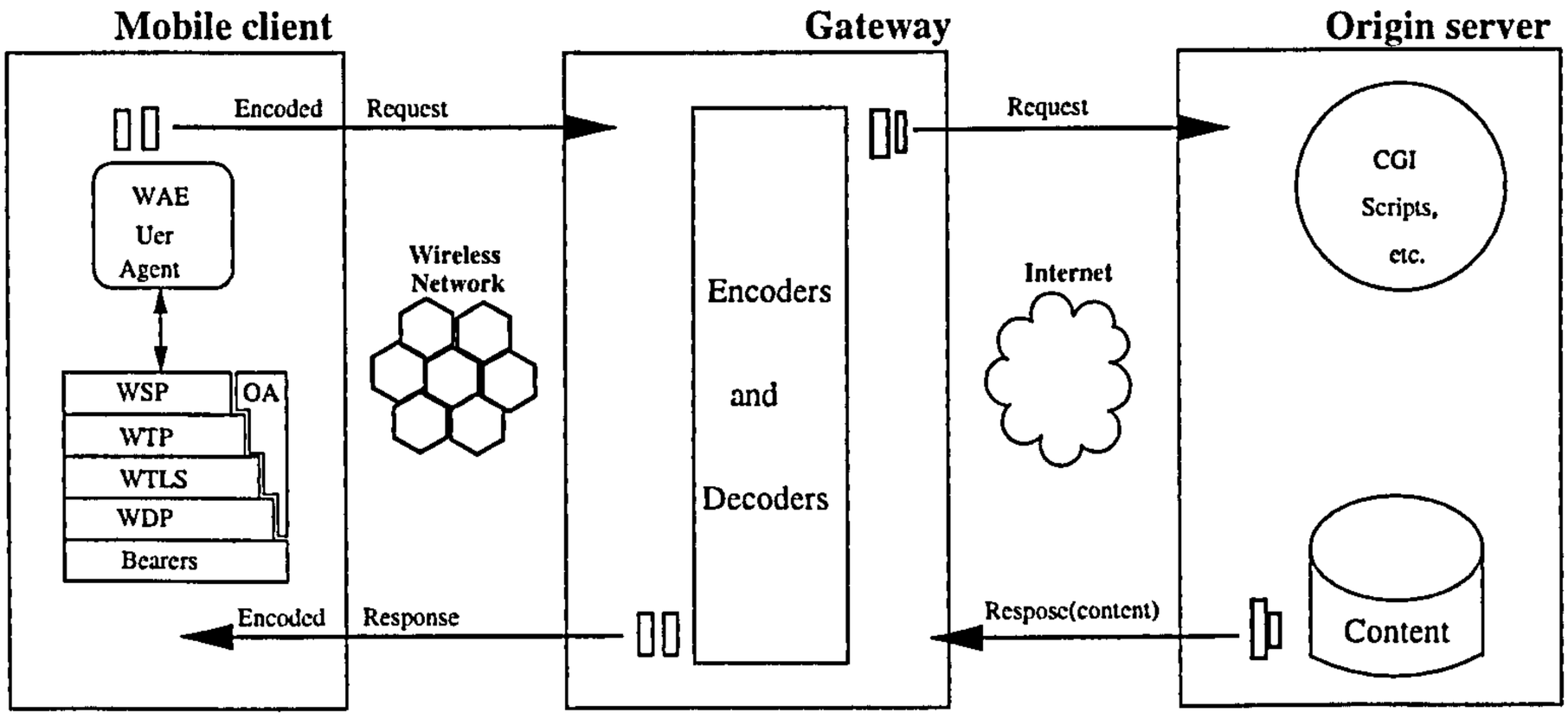
The idea of using MSS to connect wireless devices to the Internet has found an application in the Wireless Application Protocol (WAP) which was proposed by the WAP Forum[2] in April 1998 [87, 88].

As its name and the name of its promoters imply, the WAP protocol is an industrial standard for integrating mobile communicators and the Internet. It primarily aims at providing access to Web information and services to mobile telephone users. Consequently, it is designed to run on top of already deployed wireless transports (called bearers) like GSM, IS-54, PDC, CDPD, MOBITEX, DECT, etc.). The WAP designers decided to include IP as a separate bearer to leave room for integrating wireless devices with any IP–based network, for example a wireless LAN.

Rather than inventing new technology, the designer of the WAP protocol made intensive use of already proven technology, in particular, they based their design on the Web technologies and philosophies. To gain access to the large amount of information stored on Web pages from a device with power, energy, communication bandwidth, and screen–size limitations they proposed a proxy architecture shown in figure 2.3 [89, 90].

The *WAE User Agent* that runs on the wireless client on top of its WAP protocol stack is a micro browser specially designed to run on a small screen and manipulated by a mobile telephone notepad. The *Gateway* is a proxy server that translates requests from the WAP protocol stack to

[2]Founded by Ericsson, Nokia, Motorola, and Unwired Planet in mid–1997.



WAE: Wireless Application Environment
WSP: Wireless Session Protocol
WTP: Wireless Transaction Protocol
WTLS: Wireless Transport Layer Security
WDP. Wireless Datagram Protocol
Bearers: bearer services: short messages,
circuit-switched and packet data,
for example GSM, IS-54, PDC, CDPD,
MOBITEX, DECT, etc
CGI: Common Gateway Interface
OA: Other Services and Applications

Figure 2.3: WAP architecture.

the WWW protocol stack. To reduce the size of data over the network and the size of the data received from a Web server (*Origin Server* in WAP terminology) *Encoders and Decoders* are used in the gateway. The latest news about the WAP protocol and complete specification of each layer of its protocol stack can be found in the Web page Forum [88].

Based on the latest tendencies in wireless communications we are positive that the WAP protocol will be well established in the market in about three to five years time.

If our prognosis about the WAP success and the massive proliferation of PDAs in the future is correct, PDAs and similar devices will serve as the most popular outdoor interface to gain access to the global ubiquitous communication network. For this to be possible, MSS have to be widely geographically available and handy.

- We assume that in the near future the earth will be crowded by thousands of MSS; some of them will belong to private LANs and be located indoors; others will belong to communication providers (WAP bearers for example) and be located outdoors.
- A private MSS will be run by a private company and provide access solely to PDAs belonging to its company.
- A public MSS will be run by a public communication provider and serve any PDA user willing to pay for the communication service.
- Optionally, a private MSS may serve visitors.

If the above assumption proves to be true in the future, a PDA user located inside a building will gain access to network services through his WLAN mobile support stations, if he decides to leave the building then his connection will switch to a public mobile support station, then if he drives his

car to an area of sparse population where no public MSSs are available, his mobile terminal should be able to switch to a satellite network and remain connected to the world.

2.8 Personal Digital Assistants

A Personal Digital Assistant (PDA) is a pocket–size computer belonging to the generation of so–called mobile wireless computers. There is not yet an agreement on terms; other authors call them *portable, nomadic, untethered,* and *roaming computers.* We call them mobile wireless computers to emphasize that they are able to communicate while on the move. For this to be possible, these computer are equipped with a wireless communication interface based on radio frequency or infrared technology.

2.8.1 Technical specifications

There is a growing range of mobile wireless computers, however, among them PDAs are special in that they are cheap and tiny. Their low prices make them accessible to a great number of people. Being tiny means being more portable (than a laptop for example). It is true that a laptop is more powerful, however, computational power is not always a need, while on the move most of the time a computer user needs only to send/receive email; to take notes and telephone numbers at meetings or conferences, and to edit, send or receive short documents. In situations like these a powerful laptop would be a hassle while a PDA a plus. In other words, a PDA is a personal computer for storing personal temporary information and for interfacing to the communication world.

Todays commercial PDAs looks as follows:

- 400–500 USA dollars
- pocket–size dimension
- about 5x2 inches screen
- about 10–30 ounces
- 5–7 Mhz CPU
- no disks
- about 1–2 Mbyte of RAM and slots for SRAM and flash memory cards.
- 10–100 hours of battery life
- radio frequency or Infrared transmitter/receiver

For concrete examples of commercial PDAs refer to [23, 24], where the Magic Link, the Psion 3A, the ZR–5800 and the Z–7000 PDAs are described.

The first five entries in the above list should be self–explanatory. However, the last four probably need further discussion. We will come back to them later, after discussing the most important software component of any computer, the operating system.

2.8.2 Operating system

The functions performed by a mobile operating systems (OS) and a desktop one are rather similar. Yet the environments where they operate are significantly different. The need to economise in storage and electric power consumption makes the design and implementation of an operating system for a mobile device like a PDA a challenging experience.

At the moment, there is not a well–established standard operating system for PDAs, but there are several ones struggling for dominance in the market, the most popular being: Palm Computing's PalmOS, Microsoft's Windows CE, Microware's OS–9 and Symbian's Epoch.

So far the leader is PalmOS which has managed to gain about two–thirds of handheld devices currently on the market [91]. The strongest competitor of PalmOS is Windows CE which is a version of the standard desktop Windows. A promising alternative is Epoch which is being commercialised by Symbian which is a joint venture formed by Ericsson, Motorola, Nokia and Psion [92, 93]. The potential commercial impact of Epoch is rather promising if one takes into account that Ericsson, Motorola and Nokia are the owners of 60% of the worldwide market for GSM–based smart phones and similar communicators.

2.8.3 Storage

Traditionally, thanks to their high storage capacity and low media cost, magnetic disks have been used as the standard read–write permanent (non–volatile) memory in notebooks and bigger computers. However, the mechanical nature of magnetic disks makes them unsuitable for PDAs, even though there are disks of 1.3–in diameter holding 40 Mb available in the market.

- Due to the rotational inertia of the disk–platter, its spindle motor dissipates a great deal of power (2.2 W for a 2.3–in disk) at start–up (acceleration from rest to rated speed).
- Disk access time is low compared against the speeds achieved by non–mechanical components like CPUs.
- Disks are bulky, and sensitive to mechanical disturbances.

Based on the above factors, disks have not been used in PDAs. PDA designers have opted to use semiconductor memories not only for read–only and volatile read–write but for non-volatile read–write.

Because non-volatile read–write memories are a relatively new technology it might be helpful to discuss it further. Currently, they come in two technologies, namely, NV–SRAM and flash:

NV–SRAM A NV–SRAM is a *Non–Volatile* SRAM. A SRAM chip is provided with a battery to preserve the data stored in it after the main energy supply is switched off. In practice, NV–RAM comes in PCMCIA memory cards with a dual battery design (a self–contained battery system) to ensure data integrity and to avoid drain on the computer's main battery. It is worth mentioning that a PCMCIA card is approximately the size of a bank card (85.6 long, 54 mm wide, 3.3 mm heigh and about 40 g); likewise, NV–SRAM cards come in 64Kb up to 4Mb at approximately 35 and 300 USA dollars respectively. Data retention is in the range of one to two years and the backup battery is either disposable or rechargeable; a rechargeable one lasts about 10 years.

Flash Flash memory is made up of solid–state chips that store data that can be read and written during normal operation and do not need backup batteries to keep the data valid when the

main computer power is switched off. In other words, Flash memory is a read–write, non–volatile, low–power memory. Unfortunately, because of the flash technology, flash chips do not support writing of individual addresses. This means that to update the contents of a cell a whole block of cells (around 64 Kb) must be updated (erased and re–written). Also, although the number of times a cell can be read is unlimited, the number of writings is not. When the limit is reached the chip becomes slower to write operations till eventually it refuses to update the required address. However, the existing data remains readable [94]. Flash memories are becoming popular in applications where power is at a premium, and access speed and high density is required. In practice they are used to emulate magnetic disks in portable computers. In the market it is presented as PCMCIA cards of 4 up to 16 Mb at approximately 50 and 157 USA dollars, and with a write cycle of 1 million times.

2.8.4 Power consumption and management

There is no doubt that the number of portable computer–like devices will increase dramatically in the years to come. There will be a great number of them ranging from the familiar notebooks to tiny ones the size of a wrist–watch, all of them powered by batteries. Besides the significant advances in battery technology the time scale for battery improvements is long compared to the doubling time of computer microelectronics [95]. Lifetime of batteries is expected to increase only 20% between 1994 and 2004 [96, 25]. It follows that the success of portable computers, PDAs included, heavily depends on the economy, in terms of energy dissipation, of the hardware of these computers and their applications [25, 97, 22]. According to [98], power dissipation is affected by both hardware and software design.

Hardware reduction of power dissipation can be approached at microelectronic and system levels. In the former case the designer can either reduce the voltage that feeds the chip (from 5.0 V to 3.3 V for example) or decrease the switching frequency of the circuit. In the latter case, the computer works in five different modes of operation: full–on, standby, suspend, hibernation, and off [25].

Power optimization by means of software is a topic still under exploration, yet some techniques have already been proposed; in [98] these techniques are grouped into three categories: minimization of RAM access, optimal selection of and sequencing of machine instructions, and exploitation of low power features of some processors. The reader interested in a real life implementation of these techniques is advised to refer to [99].

2.8.5 Wireless communication interface

PDAs use a wireless communication interface to exchange information with the external world. Through their wireless antenna PDAs send and receive messages to and from other computers in possession of a wireless communication interface. Having a wireless communication interface is necessary but not enough for two computers to talk to each other directly. For this to happens it is essential that the communicating parties follow the same wireless communication protocol. To find somebody who speaks the same communication protocol is a simple task when a standard protocol is widely accepted and deployed. Unfortunately this is not yet the case for wireless LANs. At the time of writing no consensus on wireless protocols exist in the market, different vendors offer different products based on different technologies. However, it seems that two technologies, namely radio frequency and infrared technologies, will dominate the market. The reason for this is that both of them have proved to be the most suitable for transmitting data at high speed in indoor wireless local networks (WLANs).

Radio frequency technology

Radio frequency (RF) systems present four important problems that must be solved, namely frequency allocation, interference, security, and bandwidth.

Frequency allocation Frequencies are regulated by the Federal Communications Commission (FCC). There are not too many frequencies free for developing RF WLAMs. High–speed data communication is a newcomer to the radio spectrum market, so it has to use spectra that other, older applications are not using.

Interference If several WLANs are working in the same building, interference must be avoided. RF signals can penetrate walls, so a data cell is not restricted to a single room, however, this can cause problems if the neighbouring offices have their own networks that perhaps belong to other companies.

Security As RF signals propagate through walls, data security is an important subject to be considered, and so encryption is mandatory to avoid information linkage.

Bandwidth Modern RF technology has managed to transmit data at a rate of 2 Mbps. Unfortunately RF equipment is more expensive than IR equipment.

Infrared Technology

As a medium to short–range, indoor communication, infrared (IR) offers several significant advantages over radio frequency. The behaviour of IR signals is similar to that of visible light. IR signals are absorbed by dark objects, diffusely reflected by light–coloured objects and directionally reflected from shiny surfaces. IR signals penetrate through glass but not through walls or other opaque barriers. In other words, IR signals are restricted to a room. Thus, there is less problem with data eavesdropping and several IR WLANs can be placed in neighbouring offices without interference among them. A frequency assignment plan to avoid crosstalk is not needed, so IR wireless transmission is free from the FCC and other regulations, that means that a virtually unlimited spectral region is available. IR WLANs are recommended for those environments with a high degree of electromagnetic interference.

IR medium is not without drawbacks. The main problem that arises with IR WLANs is how to get enough power to the receivers scattered around the room. The power consumption of IR transmitters can be rather high [100]. Another problem is that in many indoor environments there exists an intense infrared ambient background, arising from sunlight, incandescent lighting, and fluorescent lighting, and shadows from moving people, which induces noise in an infrared receiver.

Modern IR technology allows build WLANs that transmit at 1 Mps composed of portable base terminals (Palmtops computers for example) served by wired based stations. Small rooms are served by a single base station, while rooms larger than about 10x10 m may require more than one base station.

An example of a wireless network that transmits over an IR medium at 9.6kbps and 19.2kbps is described in [101]. A deep theoretical analysis of wireless LAN systems is given in [102]. The advantages of IR technology over RF are described in [103].

2.8.6 Comparison of infrared and radio communications

A comparison of the most important properties and technical parameters of radio frequency technology against infrared is presented in table 2.1

Property	Radio Frequency	Infrared
recommedable env.	outdoors	indoors
interferences with other WLANs	high	low
security	low	high
dominant noise	interference from other users	ambient light and shadows
bandwidth limitation	regulatory	light–emitting diode power
bandwidth transmission	2 Mbps	1 Mbps
price	high	low

Table 2.1: Comparison of infrared and radio frequency communications

2.8.7 WLANs standards

WLANs have evolved from laboratory implementation to commercial products; at present a great variety of WLANs are offered in the market [56, 104]. Yet none of them has gained the status of *de facto* international standard. Hence, each vendor designs and implements its proprietary standard. Although nearly all the existing WLANs operate at the ISM (Industrial Scientific Medical) frequency bands (2400–2484 MHz anywhere in the world and 902–928 MHz, 2400–2484 MHz, and 5725–5850 MHz in the USA) WLANs from different vendors are incompatible; for example, data rates range from 34.8 kbps to 10 Mbps (and even more) transmitted either over infrared or radio frequency technology. Soncerning WLAN topology, there are those vendors that support or do not support ad–hoc infrastructured networks; also, some of them use the CSMA/CA (Carrier Sense Multiple Access with Collision Avoidance) as access control protocol but others do not.

In order to help the evolution process towards an international standard, international standard organizations are currently working on the specification of what will be an international standard for WLANs. So far it seams that two standards will set order in the market: One of then is the IEEE 802.11 being developed by the (Institute of Electrical and Electronics Engineers) and the other is the HIPERLAN (High Performance Radio Local Area Network) being developed by the ETSI (European Telecommunications Standards Institute).

Although there are some differences between the IEEE 802.11 and the HIPERLAN proposals there are many similarities. The two of them address the issues a user would expect from his wireless WLAN outlined in [105].

- The protocols completely define the physical layer and partly —up to the MAC (Medium Access Control) sublayer— the data link one (see figure 2.4).
- In terms of IEEE 802 standards a WLAN should appear to the logical link control and above protocols just like another 802.x protocol (see figure 2.5).
- As illustrated in figure 2.5 infrared and radio frequency transmission are supported.
- In addition to ordinary data services, time–bounded services for multimedia applications (voice, video, etc.) are available.
- To economize energy in battery–powered mobile terminals sleep mode is considered.

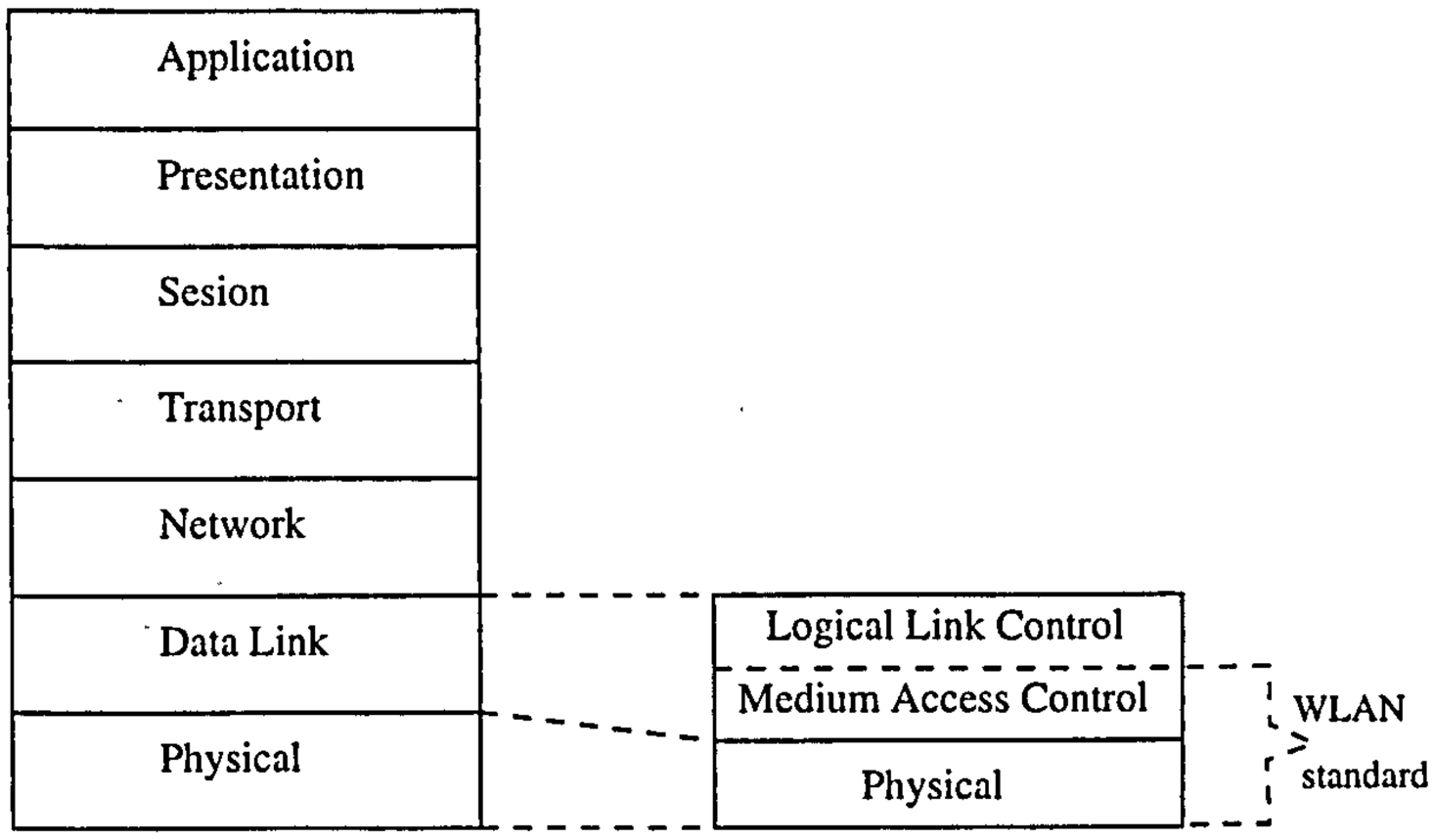


Figure 2.4: WLAN standard and its relationship to the OSI model.

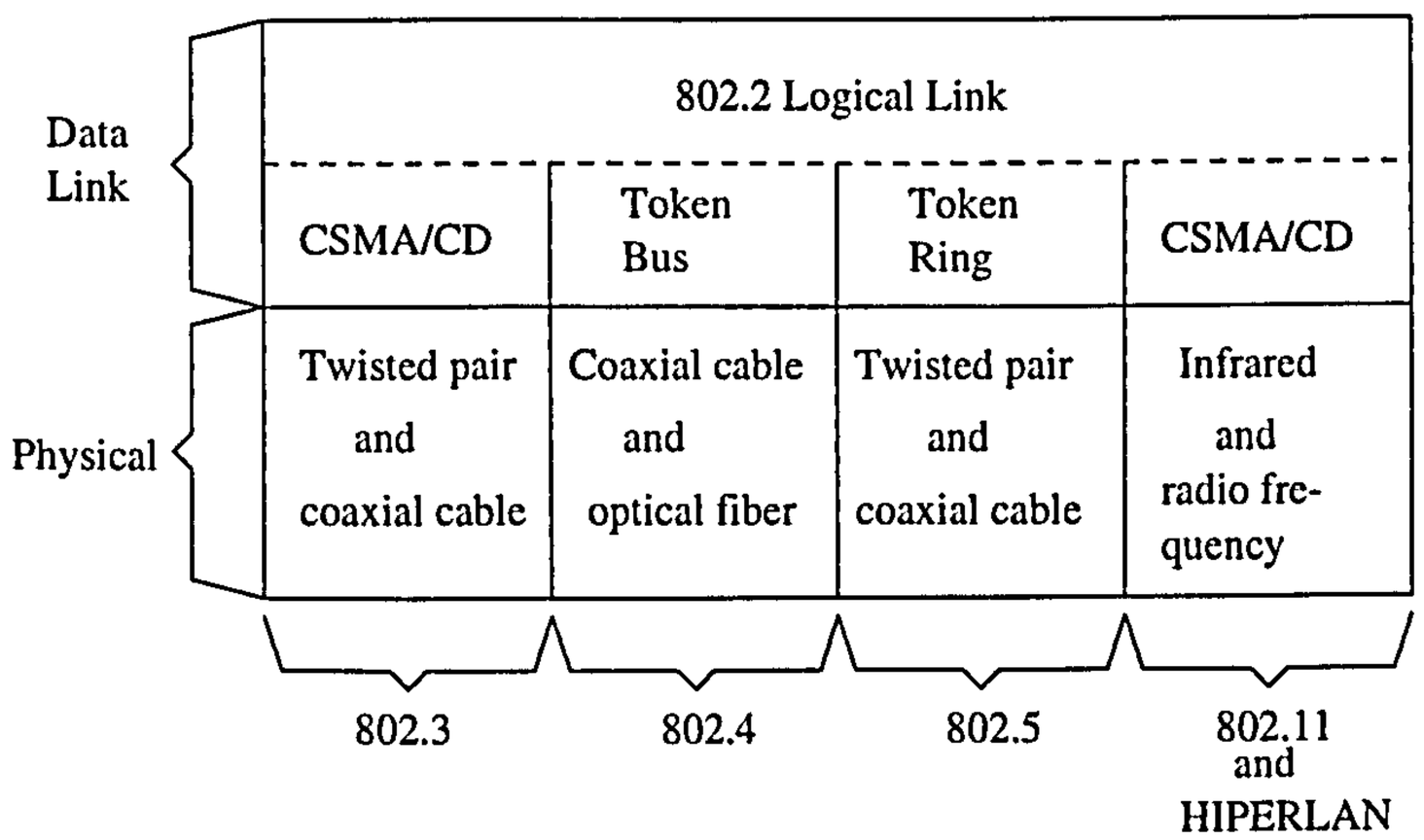


Figure 2.5: WLAN standards.

- Both, WLANs with an infrastructure and ad hoc connections are supported.

However, there are substantial differences between the protocols IEEE 802.11 and HIPERLAN, among the most important are:

- The initial focus of the standard IEEE 802.11 is to operate in the ISM band (2400 to 2483.5 MHz for example) and to provide data rates of 1 to 10 Mbps.
- The HIPERLAN standard is focusing higher data rates. For this purpose, it is expected to operate out of the ISM band, namely at 5150 to 5300 MHz and at 17.1 to 17.3 GHz. Operating in the first band it expects to provide data rates of 1 to 25 Mbps. In the future it might provide data rates comparable to wired ATM networks (100 to 150 Mbps) operating in the second band.
- Support of forwarding mechanisms for ad hoc networks has been envisioned in HIPERLAN but not in IEEE 802.11.
- Generally speaking HIPERLAN basically covers the functionalities of IEEE 802.11.

Additional details about the IEEE 802.11 and HIPERLAN protocols are described in [105, 106] and [107] respectively. A comparative description of them is provided in [102, 55, 56, 21].

2.9 Summary

Currently effort put into network development focuses not only on the development of new network technology (WLANs, home LANs, body LANs, etc.) but on the integration of existing networks (phone, Internet, mobile phone, cordless, satellite and mobile data) as well.

To ensure that the existing network infrastructure is used efficiently, all networks should be reachable regardless of the user's terminal and his geographical location. For this to be possible, existing and future networks must be integrated into a single, global, universal, ubiquitous communication network.

At present, this global network is already partially operational. Thanks to the integration of the Internet and cellular networks, mobile phone users can send/receive e–mail messages from their mobile handsets. It is expected that in the next five years most networks will be integrated; thus, all services offered in the integrated network will be reachable by thousands of users of fixed terminals and by millions of users equipped with mobile, pocket–size and cheap devices with computational and wireless communication power. A good example of these devices is the PDA. Because of its low price, it is expected that in five years time everybody will be in possession of a PDA. PDAs and similar devices will be used to send voice and data and, of course, to retrieve Web pages.

Chapter 3

Anonymity in the World Wide Web

3.1 Introduction

The World Wide Web is one of the most useful applications available in the Internet. But it has been categorized as one of the most potentially dangerous in terms of confidentiality and anonymity protection.

Million of users surf the thousands of Web pages available on hundreds of Web servers every day, however, the vast majority of them are unaware that their actions are being monitored and consequently, their right to confidentiality and anonymity are at risk. How and why Web servers collect personal data from their visitors is discussed in this chapter. Later a review of policies, working software and on–going projects concerning protection of surfers' personal data is presented. Special attention is paid to the limitations of the different proposals.

3.2 Web servers and personal data collection

As explained in [8, 3] due to the nature of the HTTP protocols Web servers normally create log files that record (for technical and commercial purposes) considerable amounts of information that may be examined with the help of standard free software (*getstats* for example) to reveal the identity of users visiting their pages. A typical Web server keeps an *access_log* file which can store the following fields:

- Name or address of the client's host, i.e. the name of the computer where the user is logged in running his browser.
- If supplied by the browser, the login name and the actual name of the user.
- The time (day, month, year, hour, minute, second, and time zone offset) that the transfer was initiated.
- The HTTP command that was executed (*get filename* for example).
- The status code that was returned.
- The number of bytes that were transferred.

Additionally, Web servers normally keep an *agent_log* file which lists the programs that have been used to access the server; from this file the Web server can learn about the operating system and the Window interface of the client and the browser.

As if the *access_log* and *agent_log* files would not be enough to violate the right to anonymity and confidentiality of the users, Web servers keep a *refer_log* file which tells the last place (URL address) the browser previously visited and the URL that it is currently viewing.

Since it is commonplace for users to browse the Web from single–users computers, the above information with or without the user's login and real names, is enough to associate a download with an individual. If the login and real names are not provided by the browser they can still be found out by using the *finger* command provided it has not been disallowed by the system administrator.

Simply put, Web servers, most of the time without the surfer's consent, collect enough information to learn, who is your Internet Service Provider (ISP), where are you, what hardware and software you use, the name of your computer, what sort of information you are after, your login and real name and even your e–mail address. Once the information is in the hard disk of the owner of the Web server, it is no longer under the surfer's control; hence the former might use it at his own discretion. This issue gives room for several questions:

- What information the Web server owner gathers about the Web surfer with and without the surfer's consent?
- What does he or she does with this information?
- Does he have the right to store (how long for?), read, and update the information?
- Who does this information belong to?
- With whom does he share the information?
- Is the Web server owner entitled to use it?
- Can the surfers stop Web servers from gathering information they do not want to give away?
- Should governments appear on the scene to bring order or should we rely on self–regulation leveraged by technology vendors?

So far none of the above questions has been given a definitive answer, the issue is still not well understood and under debate. This list of questions is by no means exhaustive, without any doubt other similar questions not envisaged yet are still to come.

Based on the universally accepted fact that Web surfing puts the right to anonymity and confidentiality of the surfers at high risk, several academic, business, technology, political and government organizations have launched efforts to develop software products, strategies, laws and recommendations to address this issue. The result of this is that recently several publications and implemented software products have appeared in papers and on the Web itself. According to their approach and without taking into account their particularities these proposals may be divided into two groups

- enforcement of regulations
- technical solutions

As can be guessed, the first approach to protecting the right to anonymity and confidentiality of Web surfers is to force things to happen either by ethical rules or legal actions. In its turn, the second approach relies on technology rather than human interaction.

3.3 Enforcement of regulations

The most recent work in this direction is currently being carried on by the World Wide Web Consortium (W3C) and the TRUSTe organization.

It is perhaps worth diverting from the main discussion to mention that the W3C consortium is an international organization founded in October 1994 with the intention to lead the Web to its full potential by developing common protocols that promote its evolution and ensure its interoperability [43, 108]; it is hosted by The Massachusetts Institute of Technology (MIT), Institut National de Recherche en Informatique et en Automatique (INRIA), and the Keio University. Its members are only companies (currently over 230) the size of AT&T, British Telecom, Boeing, Citibank, Hewlett–Packard, IBM, Matsushita, Siemens–Nixdorf, Sun Microsystems, Xerox, Microsoft and Netscape. On the other hand TRUSTe promotes itself as an independent, non–profit privacy initiative created with the intention of accelerating the growth of the Internet Industry by building trust and confidence among Internet users[109]. Currently it has 121 members, among them IBM, Tandem, Lycos, Yahoo, American Online, The Anonymizer and TRUSTe itself.

Currently the W3C consortium is working on the Platform for Privacy Preferences Project (P3P) while TRUSTe is involved on what is known as the privacy programme. The goal of both projects is to develop strategies and software products to force Web servers to observe confidentiality practices; in contrast with technical solution approaches (see 3.4) the P3P project and TRUSTe rely on regulations and their approaches are similar to the one followed by traditional legal systems: there are participants, rules to be observed, a police body, and punishments. By signing the rules, the participants promise not to do something unless they want to be punished; however, if somebody breachs his promise, he can still get away with it so long as neither the victim nor the police detects him. A system like this is corrective rather than preventive, i.e. it reacts only when the damage is already done, certainly, in some cases, the damage can be repaired in others it is too late.

The P3P project and TRUSTe fall into this category of systems and they might complement each other. The main idea of both of them is for a Web server to publish the rules of the game about their privacy policies and for the Web surfer to read them before downloading any page from the Web server and proceed only if the privacy policies offered suit him.

A privacy pledge is expected to contain sentences like the following:

- *This site will NEVER sell or exchange your name, e–mail and post address to anyone*
- *This site does not use cookies to monitor surfers' activities*
- *We do not monitor IP address activity for Web site*
- *This Web server will not share information collected from our site with any other individuals, companies or organizations*
- *Information collected at this site will be shared only with member companies*
- *If you would like to delete the data we have collected about you please contact our Web master, we will delete it in less than 48 hours*

3.3.1 The P3P project

The first public working draft of the P3P protocols was released in May 1998 and the last one in February 2000 [110]. According to its specifications a Web browser is equipped with a facility to set its user preferences over privacy practices; while Web servers come with their own to express

their privacy practices [1, 2, 111]. Both facilities are able to talk and negotiate so that a Web server provides access to its files only after a mutual agreement is reached between itself and the surfer. The Web browser performs negotiation automatically and on the fly; in this manner, if a negotiation with a Web server fails, it moves away and tries another one.

3.3.2 TRUSTe privacy programme

As with the P3P project, the TRUSTe privacy programme encourages Web servers to make their privacy policies available to their surfers; however, instead of relying on Web browsers to find out about the privacy policies of Web servers and negotiating about them, the TRUSTe organization encourage its members to make their privacy policies available at the click of the mouse in Web pages so that the surfer can read them before downloading any page [42, 111]. Members of TRUSTe are easily identified by the TRUSTe logo. The TRUSTe logo is a green rectangle (3x1 cm approximately) with the word *TRUSTe* inside it. By clicking on the logo, the surfer is taken to a privacy pledge text file where she can read all about the risks of downloading further information from the Web server.

3.3.3 Limitations of P3P and TRUSTe

Without any doubts P3P and TRUSTe help to fill the gap in Internet confidentiality and privacy; for applications where legal action may be taken and the damage repaired after a breach of promise they work fine; a Web surfer, for example, might sue a Web server after discovering that the Web server owner has given away the surfer's home address to an ice-cream company. In this case, apart from being annoyed by junk mail advertisement and wasting time in her legal claim nothing serious happens to the surfer. However, there are many applications where the sole intention of downloading information from a server reveals a great deal of information about the surfer; in other cases, taking legal actions does not make sense. For example, for a Web surfer in a country with a military regime it might be compromising to read the privacy pledge of a Web server containing antigovernment information. Similarly, it is unthinkable that a married man will take legal action against a Web dating service after discovering that the Web server has abused his personal data.

In summary, the weak side of P3P and TRUSTe is that the surfer has to give away his personal data before finding out if he has hit the right Web site. Next if he accepts the privacy policies of the Web site, he will give away even more information about himself. Not having the given information under his control the surfer can do nothing but to trust the Web server owner and in cases where legal actions are suitable, hope that he or the police body will detect the crime and punish the offender. Another limitation of this approach is that in case of a legal prosecution, the court members or its equivalent get to know everything about the surfer and his Web preferences, so for the surfer they are just another body to be trusted no wonder the TRUSTe home page bears the TRUSTe logo as well. A serious limitation of the TRUSTe privacy programme is that, at the present stage of development, it expects the surfer to go beyond the green TRUSTe logo to read the acknowledgement and acceptance of term of services file which might contain two or three pages of text full of not easy to understand (for a lay user) law terminology; except for few exceptions, it is unlikely that surfers will go through this painful, and timewasting task; it is clear that something must be done to relieve the surfer from this load, otherwise, the TRUSTe programme will remain of little practical use; perhaps the P3P protocols might be a good complement.

From this discussion it follows that neither P3P nor TRUSTe give a satisfactory answer to the question; it seems that for some applications a different approach is needed.

3.4 Technical solutions

Instead of relying on enforcement of regulations, technical solutions address the question about Web surfing security by relying only on technical strategies. In this approach, instead of giving away data about Web surfers, and believing in promises and police bodies, data is not given away to the Web server, it is hidden behind a wall, consequently the confidentiality and privacy of the surfer depend on the strength of the wall. If the Web server owner cannot see who is behind the wall it cannot know who is visiting his site, neither can he know to whom the requested pages are going. There are several systems available in the Internet that provide this kind of services and are known as *anonymizers*. All of them are based on a mediating computer (one of several) interposed between the sender and the receiver. The computers in the middle are called *mixes* and are there to process the message before it is delivered to another mix in the chain or to the receiver. Obviously, the aim of the process is to hide the sender's identity, her IP–address and e–mail address for example.

3.4.1 The Anonymizer

As advertised in its Web page [39] this system offers a set of services aimed at protecting the anonymity of the user while surfing and publishing Web pages.

Anonymizer Surfing

The anonymizer surfing is aimed at preventing Web servers from learning sensitive data about their visitors. The core of the service is a computer located somewhere in the Internet and called *the anonymizer* which acts as an agent between the Web surfer and the Web server and hides the identity of the former. In fact the anonymizer is a proxy server [112] whose job is to receive requests from the Web surfer's browser, remove sensitive data (what a Web server would like to store in its log files) from them, forward them to the Web server, receive the reply and return them to the Web surfer. To the Web server, requests from a Web surfer coming from his office computer look as though they are coming from the anonymizer.

To use anonymizer surfing the user can go either for free or paid accounts. In the first case the only thing the surfers have to do is to go with their browsers to the service address [113] and type the address of the Web server he wants to visit. After about 30 to 60 seconds of delay the anonymized page appears on screen.

In terms of functionality the paid service is exactly the same as the free one, except that it works without delays and can be instructed not to display advertisements on the anonymized page. Before using it, the surfer must sign up for her account, where apart from $ 15.00 per three months she must give away her e–mail address. In return she receives a user identifier, and a password. Any time she wants to surf anonymously she connects to the service address [114] where she is asked for her user identifier and her password.

Anonymizer servers and network licences

This service works under the same principle as the anonymizer surfing but is intended to protect the anonymity of people belonging to a single organization (Internet Service Provider, Universities, and businesses organizations for example); thus, the anonymizer is a computer, sold by Anonymizer Inc. and under the control of the owner of the organization. Thanks to the anonymizer, the anonymity of Web surfers is protected against the Web servers.

Anonymizer Email

This allows the sending of anonymous e-mail messages. Again, there is a computer, the *e–mail anonymizer*, between the sender and the receiver, which acts as an agent and hides the sender's name and address by changing the original ones for its own. To the receiver, an anonymous e–mail looks as though it is coming from the e–mail anonymizer.

To send anonymous e–mail, connect to the Anonymizer Email page [115], type the destination address, and the body of the message and send it. The anonymizer email hides your address from the receiver. Unfortunately, there is no way to receive a reply.

Anonymous Web publishing

As its name implies, anonymous Web publishing offers Web publishers anonymous Web publishing accounts (also called *Cyberpass accounts*). Any individual interested in publishing anything (political and religion views, personal profiles, and others) on the Web without disclosing her identity may contact Infonex Internet, Inc. (the owner of this service) who for a charge will provide her with an anonymous Web publishing space. For this to be possible, first she submits a Web publisher name and a password to the Infonex's Web server (the one which stores the anonymous pages), and next she sends cash or a money order to Infonex in connection with her Cyberpass account; upon receiving the payment, Infonex activates the account. By means of the *ftp* command the anonymous Web publisher can now upload her Web pages to the Infonex's Web server from her office computer. The readers of the anonymously published Web pages have no clue about the identity of the publishers.

Limitations

The main rôle in the *Anonymizer Surfer*, *Anonymizer servers*, *network licences*, and *Anonymizer Email* services offered by the anonymizer system is played by a third party (the anonymizer Web proxy in the first and second service and the anonymizer remailer in the last) which sits between the surfer and the Web server and works as a middlecomputer between the two interacting parties hiding the identity of the former. The problem with this approach is that the middlecomputer knows everything about the surfer; since in practice this computer is a standard one (a Unix Workstation for example), the information it receives, manipulate and stores is available to its manager. If for some reason (a hacker breaks the root password or the manager is bribed, for example) the middlecomputer fails to hide the secret identity of the surfer, the whole system will collapse. In other words, the third computer is the most important one and the most vulnerable as well. Only users who trust the third party (both computer and manager) will use this system.

In the case of the *Anonymous Web publishing* service the dependence on the third party is even worse as the user now uses the middlecomputer to publish her own information; consequently she is completely exposed to public and government censorship. In case of troubles her one and only hope is the middlecomputer and its manager. As stated by this service's owners [116], "the content of anonymously published pages must be legal in California..." otherwise, they could receive a court order requesting the identity of the anonymous Web publisher, information from access log files (her office computer IP address and timestamps of her *ftp* connections to the middlecomputer) will be revealed. In the same way a million dollars —and even a threat— might be offered to the manager by anybody from the public.

Another limitation of the whole system is that its services were conceived for a free Web, i.e. to surf and publish Web pages free of charge and to anonymously e–mail people or institutions from which no reply is expected, hence no charge is assumed. Although there are many sites in

the Internet that fall into this category, there are many others—those that provide serious and professional services— that do not. Moreover, it is expected that in the future when the Internet become more commercialized, the number of paid sites will increase.

3.4.2 The Lucent Personalized Web Assistant

Homed at [40] and owned by Lucent Technologies, the Lucent Personalized Web Assistant offers Web surfers with a service to prevent their sensitive data being given away by Web browsers and stored in the log files of Web servers. It was designed to serve those Web surfers who for any reason need to surf Web servers which require online registration before one can access their Web pages. In fact the core of the service is a computer called the *LPWA server* which acts as an anonymous proxy server located between the surfer's computer and the Web server she wants to visit. The job of the LPWA server is to hide the identity of the surfer by replacing the real identity of the surfer in the HTTP request with an alias identity computed as a function of a universal password (also called *the secret*), the surfer's e–mail address, and the Internet address of the Web server the surfer intends to visit.

The LPWA server accepts three different configurations. In the so–called *central proxy configuration* the LPWA server is a computer at Lucent Technologies headquarters which works as a central LPWA server for anybody willing to use it. At the other extreme, the LPWA server can be run on a local computer, i.e. on the same one as the browser runs —*local proxy configuration.* Lastly, in a the *firewall proxy configuration* the LPWA server runs inside a corporate Intranet on a firewall computer.

Thanks to the universal password which in fact is a key used as a parameter for a cryptographic function and the surfer's e–mail address, the LPWA computes a different, but consistent, alias identity for each visited Web server. Hence, all responses sent by a Web server to the alias identity are forwarded by the LPWA to the surfer. The LPWA keeps the surfer's data for the duration of a browsing session, consequently, the surfer provides her data only once (when the browser is started) regardless of the number of Web servers she visits during her browsing session. In order to be recognized as the same person by a Web server visited during different browsing sessions, the surfer must start all her browsing sessions with the same universal password. Briefly, the operation of this anonymizer can be described as follows.

1. Configure your browser to use the LPWA as a proxy.

2. Open a browsing session which takes you to LPWA login form.

3. Fill up the login form by providing your universal password and e–mail address.

4. Visit as many Web servers as you want, the LPWA server will hide your real identity by providing the Web servers you visit with alias identities.

5. Log off from your LPWA after visiting your last Web server. If you start another browsing session in the future, make sure you fill up the LPWA login session with the same universal password and e–mail address.

Limitations

Since the LPWA is a system grounded on a proxy approach it suffers from the same limitations as the Anonymizer (see 3.4.1); its security heavily depends on the security of the proxy server.

Currently the central proxy configuration is the only one available. Regardless of their limitations, this configuration together with the *firewall proxy configuration* are the only ones which make sense in terms of anonymous surfing. The *local proxy configuration* is of dubious practical use as in this case there is a direct TCP connection between the surfer's office computer and the Web server, i.e. exactly what we are trying to avoid to stop the Web server identifying the identity of the surfer.

Once again, the service is oriented to helping the surfer visit free Web pages. It does not work when the Web server asks for bank card numbers.

3.4.3 Crowds

Crowds is a system for protecting the privacy of Web surfers. It is currently being developed by AT&T and is available (beta release) in the Internet [41]. The main idea of the system is to blend the Web surfer into a crowd (a group of surfers) so that his requests are hidden among the requests of other members of the crowd. Once the surfer is integrated into the crowd any request made by him is randomly submitted to the Web server or to another member of the crowd; in the latter case the procedure is repeated until eventually the requests is submitted to the Web server; the result of this is that the Web server cannot tell if the party it received the request from is the initiator of the request or just the last member in the chain. Even more, no member in the chain, except for the true initiator, can identify the initiator of the request, since the initiator is indistinguishable from a member that simply forwards a request from another [117].

In the crowd a surfer is represented by a process on her local computer called a *jondo*. A jondo is a process started by the surfer or by the administrator of the surfer's computer which executes the crowd protocol and works (previous browser configuration) as a Web proxy for the local surfer.

When the jondo is started it tries to join a crowd membership list by contacting a process called *the blender*. The blender is run by the crowd administrator somewhere in a computer connected to the Internet. To be accepted as a member of the blender's crowd, a jondo must have an account with the blender, i.e. name and password stored by the blender and verified each time the jondo tries to join the crowd. If the jondo is accepted the blender adds the jondo's IP address, port number and account name to the membership list of jondos and reports the updated list to all the members of the crowd (the new member included). Needless to say, the membership list is updated and reported to the jondos each time a jondo joins or leaves the crowd.

With the membership list of jondos in its hand a jondo is ready to accept requests from the browser it is working for as a Web proxy, and blend the surfer into the crowd by randomly forwarding her requests as explained above. Naturally, Web server replies traverse the same random path of jondos as the requests, but in reverse.

In few lines, the crowd anonymizer can be summarized as follows.

1. Download and install the free beta version of crowd available at [41]. By default, your jondo will join a crowd whose blender is run by AT&T. You can run your own blender as well and instruct your jondo to join it.

2. Run your jondo and open an account with the blender you want to use.

3. Configure your browser so that it uses your running jondo as a Web proxy.

4. Start your browser and wait until a message specifying that you are a member of the crowd arrives.

5. From now on and till the end of your browsing session you can surfer the Web anonymously. Repeat the previous points any time you restart your browser.

Limitations

Crowds overcomes the drawbacks of the proxy oriented systems, however, it suffers from serious limitations as well. Instead of depending on a third party (the middle computer) it depends on the chain of jondos sitting between the surfer and the Web server. Depending on a single party is bad but depending on several of them might be even worse since this means that everybody must work properly.

The updating of the membership list depends on the communication of the blender with the jondos, the longer the membership list the better in terms of anonymity but the more difficult to maintain it up-to-date.

Another problem with the jondo list is that in may include jondos running on different hardware and with different communication links to each other (some of them might have high-speed connections and other modems), to the blender and to the Web server, this implies that the response time of a request no longer depends only on the resources of the initiator only but on the resources of the whole crowd.

The main problem is that once a request is sent to the Web server through a chain of jondos, the reply must follow the same path (in reverse). Because the chain is composed of several computers, one or more of them might fail while the request or the reply is on its way, or any of their owners might decide to leave the membership list or just to break the chain (by intentionally killing his jondo process).

Also it may sound unrealistic but possible that the Web server might offer a thousand dollars to each member of the crowd who proves (showing a copy of the answer or the reply) to be a member of a chain of jondos; this would lead to identifying the jondo that received the request from the initiator of the request and automatically to the initiator itself.

Another problem with the crowd chains is that once a jondo becomes a member of a chain the owner of the jondo becomes involved in a sort of *gossip*; this means that he might be asked to forward a degrading, compromising, or dangerous request; if he forwards it, he becomes a potential initiator of the request. This might encourage a member of the chain to drop the request instead of forwarding it; and discourage him from joining the Crowd in the future.

Another serious limitation of crowds is that its chain of jondos makes it difficult to be used to surf paid Web servers. The breaking of the chain might imply losing the request, the payment for the service or the purchase. In any way the initiator has no way to complain to the Web server unless he is supported by the whole chain of jondos.

3.5 Summary

Current Web protocols do not protect Web surfers' personal data from the Web servers. It is common practice for Web servers to keep records about their visitors. Thanks to the information given away by browsers a Web server can extract the personal data of the surfer, the software and hardware of his computer and the Web pages he is downloading. This information can have commercial, political, or personal value and can be used locally or sold to third parties without the surfer being aware of it.

To alleviate this situation, two approaches have been proposed: enforcement of regulations and technical solutions.

The P3P project and the TRUSTe program are well–known examples of the first approach. The goal of both projects is to develop strategies and software products to force Web servers to observe confidentiality regulations.

Technical solutions are based on the use of anonymizers. To prevent Web servers from extracting sensitive data about their visitors and to help Internet users sending e–mail messages anonymously, anonymizers have been deployed in the Internet. There are several anonymizers available in the Internet (The anonymizer, The lucent personalized Web Assistant, Crowds and other) that offer anonymizing services for free or for a fee. However, all of them are based on the use of mixes, i.e. computers (one or several) interposed between the sender and the receiver. Consequently, the anonymity provided by them is fragile and depends on the ability and willingness of the mixes to keep the secret. Likewise, the degree of anonymity is limited since one of the mixes, at least, will always know the sender's identity. For applications requiring true anonymity, mixes–based anonymizers are unsuitable. Hence, a different approach must be taken.

Chapter 4

Cryptography and message encryption

4.1 Introduction

If Alice and Bob are two people separated by some distance but linked together by a computer communication network, they can send information to each other by means of messages that travel through the network.

The usual assumption made in computer networks is that anybody might read and copy (either accidentally or deliberately) any message that passes between any pair of nodes. In other words, the risk of being overheard or caught when talking over a network is high, and inherent in the system.

Because of this the mere fact that a message is sent by Alice and received by Bob reveals several things about the communicating parties to those who accidentally or maliciously overheard the transmitted message. As discussed later, an intruder might find out about whom Alice is communicating with, the contents of the message, and even change the message, or impersonate Alice or Bob. Most of the time, this is not exactly what the communicating parties want, hence protection against these threats has to be implemented and made available so that a user can use it in accordance with the nature of his messages. In other words, a mechanism is needed to provide the user with the following facilities:

confidentiality the data cannot be read by unintended recipients.

authenticity the data is attributed to the correct originator, who cannot disown it.

anonymity the recipient has no way to identify the sender of the message.

pseudonymity the sender has a way to sign two or more messages with the same pen name.

integrity after being signed, nobody, the receiver included, can alter the contents of the document.

So far the most successful approaches to addressing these issues come from Cryptography. Cryptography is a science with a wide range of topics for study and research with a variety of applications in computer science[118, 119]. A deep discussion of cryptography is beyond the scope of this work, however, a brief introduction to its basic principles is given in this chapter with the intention that it will help to understand the cryptographic techniques used in chapter 5.

4.1.1 Message encryption

Message encryption is the cornerstone of Cryptography and it helps to address the issues mentioned above about the risk of giving away information a communicating party does not want to when sending or receiving messages over insecure channels.

Encryption involves the scrambling of a message by applying a key–driven algorithm to the message, so that it can only be understood after *decrypting* it. The message decryption involves a descrambling process and can be performed by someone who knows both the key and the algorithm. Before going further it is worth noting that some authors prefer the terms *encypherment/decypherment* instead of *encryption/decryption* and that the terms *encoding/decoding* are frequently misused as synonymous with *encryption/decryption.* The difference between them is that encoding/decoding may or may not involve a scrambling process while encryption/decryption always does.

Mathematically an encrypted message is the result of applying to the original message (usually a plaintext message) an encrypting function parametrized by a key.

$$C = E(k_1, M) \tag{4.1}$$

where C is the encrypted message; E is the encrypting function; k_1 is the encrypting key, and M is the plaintext message.

Likewise, the original message can be recovered by applying to the encrypted message a decrypting function parameterized by a decrypting key.

$$M = D(k_2, C) \tag{4.2}$$

where M is the original message (usually a plain text message); D is the decrypting function; and k_2 is the decrypting key.

It then follows that

$$D(k_2, E(k_1, M)) = M \tag{4.3}$$

From equation 4.3 it follows that a pair of keys is involved in the encryption/decryption process. If $k_1 = k_2$ or if one key is easily derived from the other, the cryptosystem is called *symmetric* and k_1 and k_2 are called *symmetric keys*, otherwise it is called *asymmetric* and k_1 and k_2 are called *asymmetric keys* [119].

Since a symmetric key must be kept secret from everybody else except the sender and the receiver of the message, symmetric cryptosystems are also known as *secret–key cryptosystems.* Conversely, asymmetric cryptosystems are called *public–key cryptosystems* because in these cases one of the keys is kept secret by his owner while the other one is known by the public.

Both secret–key and public–key cryptosystems are grounded on the use of the so–called *trapdoor one–way* functions.

A *one–way function* is one that maps a domain into a range such that every function value has one and only one inverse. Also, it should be feasible to compute $f(x)$ for any x in the domain of f while, for almost all y in the range of f it is computationally infeasible to compute $f^{-1}(y)$ even if f is known. A one–way function is called a *trapdoor one way function* if it is feasible to compute $f^{-1}(y)$ given certain additional information. This additional information is the decryption key. Given the decryption key $f^{-1}(y)$ can be computed in polynomial time.

It follows that equation 4.1 is a trapdoor one–way function that can be computed in a polynomial time and equation 4.2 —its inverse— can be computed in polynomial time as well when K_2, the decryption key, is known.

4.1.2 Secret–key cryptosystems

In secret–key cryptosystems the encryption and decryption processes involve the same key, i.e. they are symmetric. A single key, called *the secret key*, is shared and kept secret by the two parts involved in the encryption/decryption process (see figure 4.1).

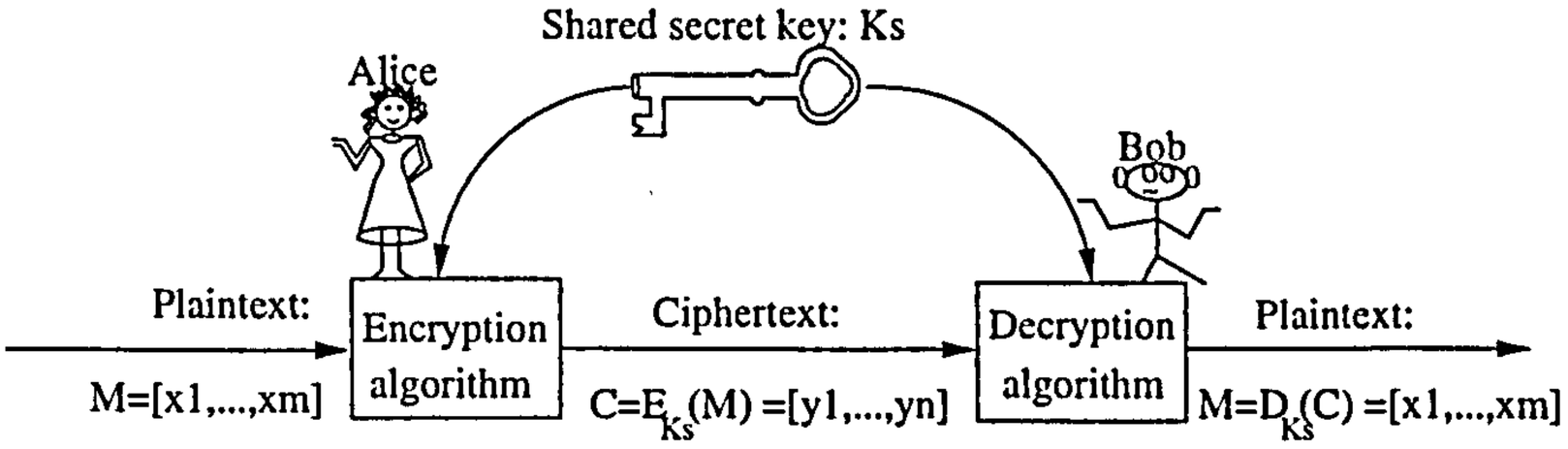


Figure 4.1: Secret–key encryption and decryption.

The encryption and decryption algorithms are normally based on principles of modular arithmetic, in particular on properties of the operation called *modulo 2 addition* or *Exclusive OR* defined in the binary digits a and b as follows:

$$(a+b) \bmod 2 = a \oplus b = \begin{cases} 0 & \text{if } a = b \\ 1 & \text{if } a \neq b \end{cases}$$

It can be proved that if $a \oplus b = c$, then $a = c \oplus b$. In other words, this means that XORing b twice to a restores the original value of a. In terms of Cryptography one can think of a as a plain text message, c as an encrypted message and, b as a secret–key.

The Data Encryption Standard (DES) is the most widely used secret–key encryption system and is based on these properties of XOR arithmetic. It encrypts data in 64–bit blocks into 64–bit blocks of cyphertext under the control of a 56–bit secret–key, which is used to generate a series of other keys to be used during the encryption process. The algorithm is symmetric; the same algorithm and secret–key are used for encryption and decryption; however, during decryption the series of keys generated from the secret one are used in reverse order. Thanks to this symmetry, nothing distinguishes Alice from Bob, either of them can be a sender and a receiver, the same secret–key is used to encrypt messages in both directions.

The DES algorithm is widely discussed in literature [120, 118]. For those interested in exploring the DES weakness and cracking it, the book [121] recently written by the Electronic Frontier Foundation is a good reference.

One of the most serious difficulties with secret–key cryptosystems —DES for instance— is the distribution and sharing of the secret key. Before any secret–key encrypted communication can take place the secret key must be distributed to the sender and the receiver of the encrypted message, i.e. two individuals can exchange secret–key encrypted messages only if they have communicated before. In practice the sender is normally the secret–key generator; hence a private courier or registered mail is used to carry the secret–key from the sender to the receiver. On the other hand, experience shows that keeping a secret is extremely difficult when more than one individual is involved, in secret–key cryptosystems at least two parties (one sender and one receiver) share the secret–key; however, there are situations where more people are entitled to know the secret key; then the chances of losing secrecy increase dramatically.

Besides this notorious drawback secret–key cryptosystems provide for very fast (they are significantly faster than public–key algorithms) and efficient encryption.

4.1.3 Public–key cryptosystems

As was stated in section 4.1.2 two of the major difficulties of secret–key cryptosystems are the distribution and sharing of the secret–key. To attack this problem Diffie and Hellman introduced what they called public–key cryptosystems in 1976 [122]. In public–key cryptosystems encryption/decryption of messages is performed using a pair of asymmetric keys, more exactly a key which is divided into two subkeys (counterparts). The first part of the key, called the *private key*, is known only to the receiver. While the second, called the *public key*, is know to the sender and to anybody else interested in sending messages to the receiver. The pair two keys are mutually dependent, one is useless without the other. The public key is made available to the world by placing it in a public directory, in a sort of public–key directory for example, similar to those used by telephone companies to make telephone numbers available to the public. Thanks to this revolutionary approach Alice and Bob can swap encrypted messages over an insecure computer communication line from the very beginning of their interaction. The need to send or receive a confidential message, from somebody one has not had prior acquaintance with, is a common practice in modern societies, particularly in business.

Public–key cryptosystems are asymmetric. The public key is used to encrypt messages to be sent to the receiver who decrypts the messages using his private key. Data encrypted with the public key can only be decrypted with the corresponding private key. The encryption and decryption algorithms might be different, but it is possible that they are the same.

A simplified diagram of public–key cryptosystem is shown in figure 4.2.

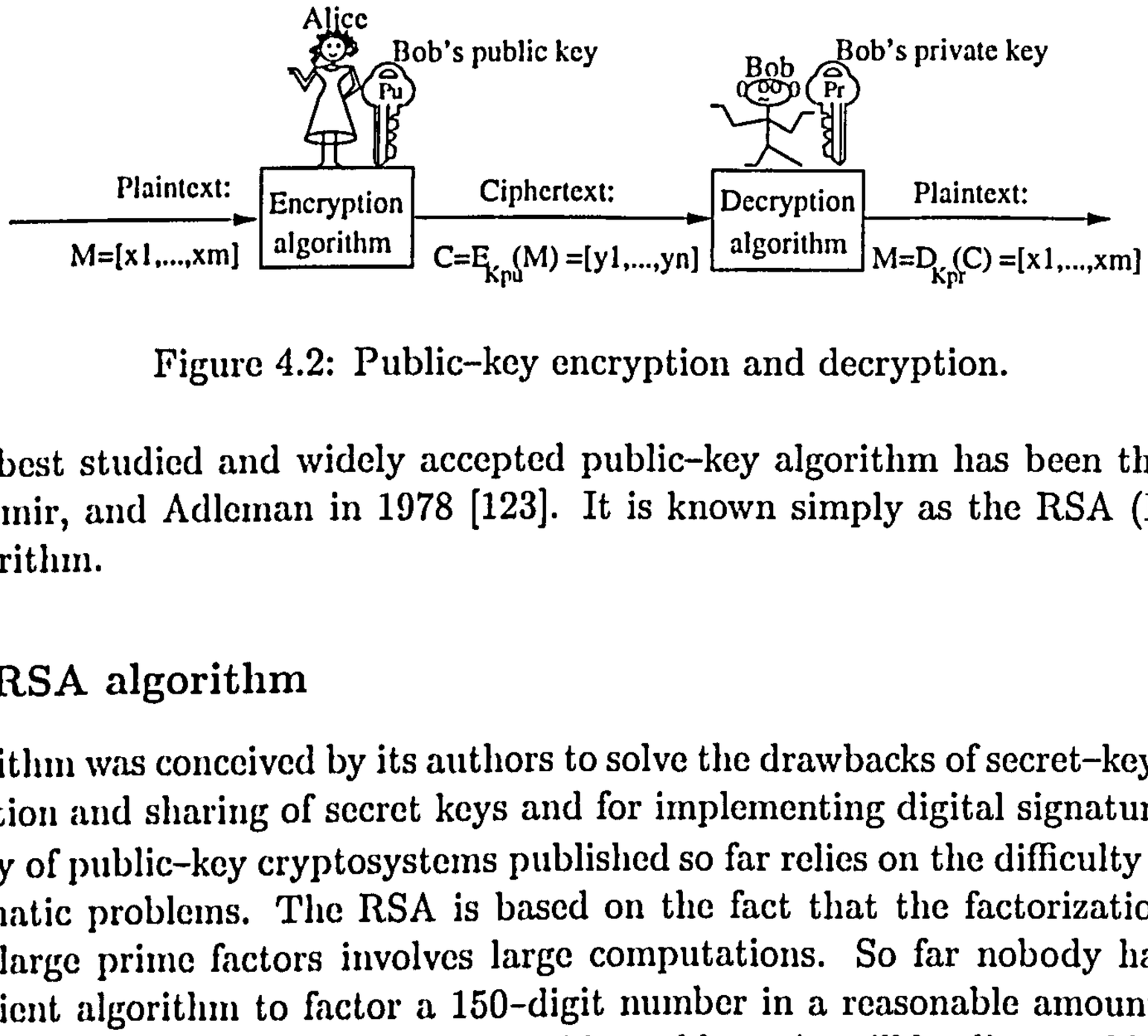


Figure 4.2: Public–key encryption and decryption.

So far the best studied and widely accepted public–key algorithm has been the one proposed by Rivest, Shamir, and Adleman in 1978 [123]. It is known simply as the RSA (Rivest–Shamir–Adleman) algorithm.

4.1.4 The RSA algorithm

The RSA algorithm was conceived by its authors to solve the drawbacks of secret–key cryptosystems in the distribution and sharing of secret keys and for implementing digital signatures [123].

The security of public–key cryptosystems published so far relies on the difficulty of solving well–known mathematic problems. The RSA is based on the fact that the factorization of composite numbers with large prime factors involves large computations. So far nobody has succeeded in finding an efficient algorithm to factor a 150–digit number in a reasonable amount of time; thus, factorization is considered a well–known intractable problem. As will be discussed later, a RSA key is derived from a large composite number, hence, in computational terms, breaking an RSA key is equivalent to finding the factors of a large composite number.

The RSA encryption and decryption algorithms are based on exponentiation in modular arithmetic. Given a plain text message M the encrypting processes mentioned in equation 4.1 and 4.2

and shown in figure 4.2 are mathematically represented by equations 4.4 and 4.5 respectively, where C represents the result of encrypting M.

$$C = M^e \bmod n \tag{4.4}$$

$$M = C^d \bmod n = (M^e \bmod n)^d \bmod n = M^{ed} \bmod n \tag{4.5}$$

As can be appreciated from equations 4.4 and 4.5 both encrypter and decrypter must know the value of n. In addition, the encrypter must know the value of e and the decrypter the value of d. Thus, in terms of public–key cryptosystems, the pair $\{e, n\}$ can be thought of as the public key and the pair $\{n, d\}$ as the private key. e, d, and n are positive integers and are chosen as follows.

Because in practical applications the encrypter and decrypter are geographically separated and linked by a computer communication line it is common to talk of a sender and a receiver in published works or to make it less technical, of Alice and Bob.

The fact that the original plain text M can be recovered at the decrypter's side is based on e and d being multiplicative inverses and can be mathematically proved [123, 124].

In practice the RSA cryptosystem works as follows:

1. The sender generates two large prime numbers p and q and keep both of them private.

2. The sender generates a composite large number $n = pq$.

3. The sender chooses a large random integer d which is relative prime to $(p-1)(q-1)$; i.e. $\gcd(d, (p-1)(q-1)) = 1$; *gcd* means greatest common divisor; d is kept private.

4. The sender computes an integer e which is the multiplicative inverse of d. Thus, e and d satisfies the equation $ed \bmod (p-1)(q-1) = 1$.

5. The pair $\{n, e\}$ is called the public key and is published in a well–known directory of public keys. $\{n, e\}$ is used to encrypt messages (see equation 4.4).

6. The pair $\{n, d\}$ is called the private key. $\{n, d\}$ is used to decrypt messages (see equation 4.5).

Since e cannot be computed without knowing the value of $(p-1)(q-1)$ or what is equivalent, the value of p and q, the security of an RSA cryptosystem depends on the difficulty of factoring n into its two factors p and q. For real life applications it is suggested using 100–bit numbers for p and q so that n results in a 200-bit number. With numbers like these, it would take several million years at a rate of one step per microsecond to factor n using the fastest known algorithms.

Both encryption and decryption in RSA cryptosystems involve raising an integer to an integer power modulo n. These exponential operation can be performed using well–known fast exponentiation algorithms [124]. Similarly, there are well–known algorithms to find large prime numbers which are obviously not based on factorization but on tests for primality [125, 123, 124, 120].

4.1.5 Digital signatures

As was mentioned at the beginning of the discussion of the RSA scheme, the RSA encrypting and decrypting algorithms were conceived for implementing digital signatures which are the basis for authentication of users, and the non–repudiation and integrity of messages. A digital signature is an electronic equivalent to a handwritten one.

To implement a digital signature with the RSA cryptosystem the encrypting and decrypting algorithms must satisfy the following property:

$$(M^d \bmod n)^e \bmod n = M^{de} \bmod n = M \tag{4.6}$$

What equations 4.6 and 4.5 state is that the encrypting and decrypting algorithms are commutative and mutual inverses. Because of this property, the algorithms can be applied in any order, the original plain text M is always recovered. This is what RSA signatures are grounded on, since Alice can sign a message M by decrypting it with her private key $\{n, d\}$ and then send it to Bob who, upon receiving it, encrypts the decrypted message with Alice's public key $\{n, e\}$ to recover M. Bob is sure the message comes from Alice as only Alice knows the pair $\{n, d\}$.

4.1.6 Blind signatures

As with traditional cash money, e–cash money must bear the signature of a financial authority to be accepted as a token in commercial transactions. In the former case, a financial authority creates notes of given values, signs them, and somehow puts them in circulation. The acceptance of the notes by the public is backed up by the signer, whose signature is well–known and in theory impossible to forge. In the latter case, it is the user herself, Alice for example, who creates notes of given value and sends them to a financial authority to sign them; having the signature of the financial authority the note can be used to pay for commodities. In contrast with traditional cash, the physical appearance of a note and a coin are exactly the same in the electronic world, both are represented by electronic bits and are called e–notes and e–coins respectively. Thus, if we ignore the monetary value of an e–coin and e–note which normally is higher in notes, there is no substantial difference between them, consequently we will freely switch between the two terms.

Current implementations represent an e–note as a pair of integers; the first one is the monetary value of the note and the second is an identity string, i.e. a sort of serial number that uniquely identifies the note and prevents users from spending it more than one time.

The problem with bringing or sending an e–note to a bank and requesting a signature is that the bank can keep records that associate Alice with the value and serial number of the e–note; and later when the e–note is brought back to the bank by a merchant, find out when, where, to whom, and how the note was spent, this possibility would reduce the anonymity of the e–note to nothing. Fortunately, several protocols have been devised to deal with the problem, although they differ in details, all of them are centered around what is known as *blind signatures.*

The main idea behind a blind signature is that the signer signs a document with a satisfactory degree of knowledge about the content of the document but without knowing exactly what he is signing. In the e-note case, the banker knows that he is signing a note of £5.00, for example, that belongs to Alice but does not know the identity string. Hence, the banker knows that he has to deduct £5.00 from Alice's account and deposit them into the merchant's account when the latter brings back the e–note. Yet since the name Alice is not written on the e–note and he signs thousand of notes everyday, the banker has no way of associating Alice with the note; the only thing he can learn from the note he is presented with is that it has his signature on it, has not been spent, and that it is worth £5.00. From this information he deduces that its holder is entitled to £5.00 regardless of the origin of the e–note, it might come from Pat, Ted, Katy, Alice or somebody else, in terms of money transfer it does not make any difference to the bank.

As explained in detail by Bruce Schneier [118] the blind signature protocol is based on a technique called *cut–and–choose* which relies on probability to ensure that a blindly signed document contains what its signer expects. The protocol works as follows:

1. Alice makes n copies of the document, an e–note for example, she wants signed.
2. She blinds each of the n copies by multiplying each of them by a different integer called a *blinding factor.*
3. She puts each copy inside an envelope and brings the n envelopes to the signer.
4. The signer chooses and opens $n-1$ documents at random and asks Alice for the blinding factors to verify that each of the chosen $n-1$ documents contain what Alice claims.
5. If the signer is satisfied with the contents of the $n-1$ documents, he signs the only document left unopened by writing on the unopened envelope.
6. Alice goes home and there she removes the blinding factor from the signed document by dividing its contents by the blinding factor.

Mathematically, in an RSA cryptosystem where e is Alice's private key and d the private key of Alice's banker, Alice can make the banker blindly sign a document whose contents is x, as follows:

1. Alice chooses a blinding factor r and blinds x by computing

$$t = xr^e \bmod n \tag{4.7}$$

2. The banker signs the blind document by computing

$$t^d = (xr^e)^d \bmod n \tag{4.8}$$

3. Alice removes the blinding factor by dividing equation 4.8 by r

$$s = \frac{t^d}{r} \bmod n = \frac{(xr^e)^d}{r} \bmod n = \frac{x^d r^{ed}}{r} \bmod n = \frac{x^d r}{r} \bmod n = x^d \bmod n \tag{4.9}$$

4. Alice has her document $s = x^d \bmod n$ signed.

There are a few things that merit additional comments. Since r is random, the banker can never determine x —the contents of the document. If s is an e–note, now Alice has a valuable e–note she can use to anonymously buy a box of chocolates or anything else.

This protocol is a simplified version of the one presented by Schneier [118] where additional data is included in the e–note to discover Alice's identity in case she cheats by spending her e–note more than one time, and to catch the chocolate seller if he intends to present Alice's e–note twice to the banker. An overview of blind signature protocols is presented in [126, 127]; for a detailed discussion we encourage the reader to refer to [118].

4.2 Combination of secret–key and public–key cryptosystems

The drawback of public–key cryptosystems is that they are significantly slower (about 100 to 1000 times) in comparison with secret–key ones [61, 128]. For this reason they are not suitable for encrypting large amounts of data (files and pictures for example). Another drawback of a public key is that since it is managed by key infrastructure its renewal is a comparatively cumbersome process; hence it is meant to be a long term key; incidentally, the more it is used and exposed the

higher the risk of being caught by a meddler. Because of these reasons, experts in Cryptography recommend using a public key as little as possible.

In practice, cryptographic systems are based on a combination of secret and public–key cryptosystems, so they combine the speed of the first with the key management conveniences of the second.

The main idea is to encrypt large amount of data with a secret–key algorithm and then encrypt only the secret key with a public–key one. Assuming that the receiver Bob and the originator Alice of a message are in possession of a pair of keys (private and public keys) the following procedure takes place.

1. Alice has a large message (several Mbytes for example) to send to Bob.
2. Alice generates a secret key.
3. Alice encrypts the secret key with his public key and sends it to Bob.
4. Bob receives and decrypts the secret key using his private key.
5. Alice encrypts a message she wants to send to Bob with the secret key and sends it to Bob.
6. Bob decrypts the message using the secret key.
7. The shared secret key remains valid for the duration of the session.

A well–known implementation that follows this approach is the *Pretty Good Privacy* (PGP) software for secure e–mail[128].

4.2.1 Key management

Key management is the hardest part of any cryptosystem. This comes from the fact that in modern Cryptology the tendency is to open to the public all components of a cryptosystem (encrypting and decrypting algorithms included) but the keys, so they rely on the infeasibility of breaking the system without knowing a key. The result of this is that if the key management fails, the whole cryptosystem is down regardless of how secure the encrypting algorithms is and how long the keys are.

Key management is similar to bank card management. Somebody (a trusted authority) is in charge for issuing cards to users, recognizing the user's handwritten signature, assigning, renewing and cancelling personal identification numbers, and in some cases renewing and cancelling cards.

Key management has to do with a set of key management procedure.

key space management if keys are integer numbers, what is the range of valid keys?

key generation are keys generated from scratch or from previous ones (updating)?

weak key exclusion if there is a key that is easy to guess, it should not be used.

key renewal to reduce the risk of key compromising, keys should expire and be renewed regularly.

key invalidation compromised and stolen keys must be invalidated as soon as possible.

news propagation news about invalid keys and new ones should be spread widely and as soon as possible.

key storage where are secret and private keys to be safely kept? In the owner's brain?, on a piece of papers? on disk?, on a card with read only memory?

key usage policies how many sessions or how long is a key valid? How many keys can a user have? On what computer is a key to be used?

old keys storage expired and cancelled keys cannot be shredded and send to a dustbin. How long must they be kept for and who will store them?

authentication of key holders how does Bob know the holder of Alice's key is the Alice he thinks she is.

key escrow should my neighbour keep a copy of my private key in case I lose it or the government needs it?

As can be seen from the issues raised above, though public–key cryptosystems give an elegant answer to the question about key distribution and sharing in secret–key cryptosystems, management of public keys is not an easy task at all. It encompasses several issues ranging from technical to political ones. To a great extent the standardization of public–key cryptosystems in public networks like the Internet, much depends on the solution to these questions. So far the most difficult ones, due to their political and social implications, have been the last two in the list, namely, user authentication, and key escrow. These two issues have been the objects of discussion in one of the hottest and longest debates ever witnessed in the field of computer science. The debate started in 1993 with the government proposal to introduce the now infamous Clipper chip as a government standard for encrypting unclassified communications [52, 118]; and nobody knows when a consensus is going to be reached. Without any intention of joining this debate, a brief discussion on authentication of key holders and key escrow is presented in the following sections. The reader interested in more details about this debate should refer to [129, 130, 118, 131, 132, 52, 133, 134, 135].

4.2.2 Authentication of key owners

The question to answer here is, when Bob receives Alice's public key from a public key directory or from somebody else, how does he knows that the key is definitely Alice's public key and not somebody's else? It seems logical that Bob will be reluctant to encrypt messages using Alice's public key unless he is completely sure about the authenticity of the key. For this to be true, Bob must receive Alice's public key directly from Alice or indirectly from somebody Bob trusts. In either case the key must be handed in person or sent over a secure medium, an encrypted channel for example or a private courier.

In practice two different approaches have been taken to addressing this issue: the distributed and the centralized authentication of key holders.

Distributed and centralized user authentication

In the distributed approach there is no central key distribution center, there is no need for it since every user generates and distributes his own public key. There is no central key certification authority either. Key certificates are issued by the users themselves by signing each other's public keys [118, 61]. The meaning and contents of key certificates and the rôle key certification authorities play are discussed below in this section.

A well–known example of cryptosystem that follows the distributed approach is the Pretty Good Privacy (PGP)—an e–mail security programme designed by Philip Zimmermann [118] to provide

privacy, user authentication, digital signatures and compression. However, in practice most PGP users rely on trusted PGP Internet servers (see [136] for example) to advertise their public keys and obtain others. It is worth mentioning that the PGP cryptosystem is considered to be a de facto standard among the Internet community.

The centralized approach to key authentication has been supported by the Internet Architecture Board (IAB). It is based on a protocol known as the X.509 where a trusted body, called the *Certification Authority* or just CA [137] assigns a unique name to each key user and issues a digital signed certificate containing the user's public key and user's name and additional personal data (to be discussed below). Certification authorities are also called *trusted third parties* and are organized in a hierarchical tree rooted by the Internet Policy Registration Authority (IPRA) —whom everybody has to trust.

An implementation that supports the centralized key authentication approach is the Privacy–Enhanced Mail (PEM). The PEM is the official Internet standard for private e-mails over the Internet. It provides confidentiality, user authentication, and message integrity [138, 118, 61, 139]. It is worth pointing out that at the moment of this writing, version X.509 v3 is considered the Web standard, and is being used by popular applications like the Netscape Communication Secure Socket Layer (SSL) [53, 140, 141] and PEM [138].

Several schemes for distribution of public–keys are presented by Stallings [142].

Certification authorities and public–key certificates

A *public–key certificate* is a digital document issued and signed by a certification authority that binds a public key to its owner. The certificate attests that a particular public–key belongs to a particular individual. A *Certification Authority* (CA) is a trusted authority that issues public–key certificates and whose digital signature is recognized among the community where the public–key certificate is valid.

A public–key certificate contains data about the public key owner, the key distribution center (if any) that issued the public key and about the key certification authority itself. It also contains control information.

The actual content of the public–key certificate depends on the issuer's taste, but in general it has the following fields [141, 118, 139].

- key owner's data: name, address, age, sex, and so on.
- the public key being certified.
- validity period (interval over which the certificate is valid).
- name of the key distribution center that issued the public key.
- name of the key certification authority.
- control information concerning the certificate (version, serial number, and so on)
- digital signature of the certification authority

The controversial differences between the distributed and centralized approaches become apparent when one takes into account that the distributed approach based purely on trust and it has a flat structure where everybody is equal. It is feasible for friendly communications, say within the academic environment, but infeasible for more sensitive applications where legal issues and money are involved. It seems that for these applications the centralized approach is more realistic. The problem with this is that the body on top of the hierarchy of certificate authorities is given absolute power and is indeed converted into a sort of guard. Critics of the centralized approach have raised the question about who can be trusted to be the maximum trusted entity at the top of the certification chain. Another way of putting it is to say, who will guard the guard himself? The government is the right body for those who trust the government, yet not everybody does.

4.2.3 Key escrow

The central idea of encrypting data is to make it available to the body entitled to read it; normally this person is the owner of the encrypting key. The owner of the data might encrypt all the electronic documents she has collected through her entire life (school homeworks, personal letters, books and poems, address book, jotters, e-mails, medical records, financial records, and other legal documents) and rest assured that nobody else but her can read her files. However, the person in this example would not sleep at all knowing that her life story depends only on the availability of an electronic key that might be lost. Likewise, she might unexpectedly die and leave her family without the key to open relevant information such as her last will and testament for example; a similar problem would face the company where she was employed as she might be the holder of the key for decrypting important information. One might argue that the death of a person is an extreme case, yet the same problems would appear if the person is unconscious or away and not reachable from her company.

It seems that there are personal and business reasons for given a copy of our private key to our neighbour or somebody we trust, i.e. for using a sort of key backup. The idea behind a key back up does not necessarily mean keeping a real copy of the key, a mechanism to recover the lost key would serve the same purpose. In the literature the terms *key recovery, key backup,* and *key archive* are used to refer to the same concept.

At first glance key escrow looks like a sensible service to have, however, the questions about whether it will be voluntary or compulsory may be raised; and about who is honest enough to be trusted as a key escrower.

Under the cover that it needs a key escrow mechanism to catch drug dealers, terrorists and similar criminals, the US government has proposed that key escrow should be compulsory; it claims the right to have access to encrypted information without the knowledge or consent —just after the release of a court order— of the owner of the encrypted data. In this direction, government supporters have proposed several ideas for building key escrow mechanisms [131, 143, 144, 145, 146, 147].

Critics of the government scheme claim that key escrow has serious disadvantages, the main one being that cryptousers have to trust the key escrower and his escrowing procedures. Strong arguments have been made against the government proposal [132, 133, 52, 118] by those who do not trust the government and who are well aware that the security of the key escrow mechanism might be broken and then the whole escrow cryptosystem will fail. They claim that key escrow is a threat to privacy, that it is expensive and difficult to deploy, less secure, and that it puts into risk the acceptance of digital signatures. Unfortunately no acceptable solutions to this issue have been proposed yet.

4.3 Cryptographic co–processors and smart cards

One of the concerns when designing a cryptosystem is the speed loss due to the time wasted during the encryption and decryption of messages sent over the communication lines and encryption and decryption of files before storing them on disk and before bringing them back to main memory. Regardless of the encryption/decryption algorithms used, the speed loss of a system with cryptographic facilities may be significant compared to a plain–text one. Most of the time in a cryptosystem is spent performing modular arithmetic operations. If this is true, one can dramatically reduce this time by using specialized hardware to perform modular arithmetic operations. This hardware is known as a *cryptographic co–processor* and is normally a card attached to the main CPU. A simple way of providing a computer (a PDA for example) with a powerful cryptographic co–processor is the insertion of a smart card equipped with an embedded cryptographic co–processor.

A smart card is a plastic card quite similar in size and shape to the familiar bank card with a magnetic stripe. The main distinction between a smart card and a magnetic stripe one is that the former has an integrated circuit (often called the smart card microcontroller or just controller) inserted in the plastic that provides the card with computation capabilities. Thanks to this new feature it is capable of performing digital signatures, user authentication, message encryption and inside data protection. The physical and electric characteristics of a smart card are specified by the 7816 ISO standard. The smart card controller is a single chip computer without keyboard and display embedded in a piece of plastic. It has, like any other computer, a CPU, memory, and input/output interface. The type of smart card controllers we are interested in here are the cryptology oriented ones which come with an arithmetic coprocessor.

4.4 Summary

Cryptographic techniques have been extensively used to protect both data stored in Internet computers and messages travelling through Internet channels. To prevent un–authorized parties from understanding the content of a message sent from Bob to Alice, the message is encrypted at Bob's computer and decrypted at Alice's computer. A message is encrypted by using an encryption key and decrypted by using a decryption key. In accordance with the keys used, cryptographic systems fall into two classes: secret–key (e.g., DES) and public–key (e.g., RSA).

Secret–key cryptosystems use the same key for encryption and decryption and are also called symmetric cryptosystems. Conversely, in public–key cryptosystems the keys used for encryption and decryption are different. Thus, these cryptosystems are called asymmetric.

One of the main attractions of public key crypto–systems is that they can be used to implement digital signatures. Digital signatures are necessary to sign electronic money and other documents.

Public–key cryptosystems are significantly slower (about 100 to 1000 times) compared to secret–key ones. Because of this, practical cryptosystems use both public and secret keys.

Key management is the hardest part of any cryptosystem. It has been recognised that among the issues that key management has to address, key escrow is the most difficult and controversial.

Chapter 5

A new approach to confidentiality and anonymity protection

5.1 Introduction

Although it is certainly risky to bet on how computer and communication technologies are going to develop in the future, in the previous chapters we dared, based on recent tendencies and on–going research projects, to predict that a global ubiquitous communication network will be deployed in the years to come. We also studied the main components of such a network, identified some of its services and stated that PDAs will be the most popular communication devices used to gain access to these services. Similarly, we introduced the concepts of confidentiality and anonymity and have identified them as one of the main lacks in the current Internet infrastructure and as one of the main issues to be addressed to make the ubiquitous communication network of the future commercially successful. We also explored recently deployed systems aimed at protecting the right to confidentiality and anonymity of Internet users, and identified their limitations. Finally we introduced some concepts of cryptography in the belief that they would serve as a background for the discussion of this chapter.

It is time now to put all the pieces together and present the main object of our research, i.e. our proposal for sending truly anonymous and confidential Internet messages which we believe addresses the flaws identified in the anonymizers reported so far in the literature.

5.2 Design characteristics

Our system is based on end–to–end encryption. In end–to–end encryption systems encryption and decryption are performed by the applications; data never appears in clear at intermediate nodes. Each user of a node has one or more encryption and decryption keys; let us say one key for each application or session. When Alice's application wants to send an encrypted message to Bob, it selects the proper key, encrypts the message body, leaving the message header in clear format, and sends it to Bob through the communication networks. The message travels from node to node till eventually it reaches Bob's computer, only at that end is Alice's message body decrypted.

The only part of Alice's message intermediate nodes between Alice and Bob need to read and understand is the message header; it contains the final destination address and other relevant information; the contents of its body is irrelevant, consequently, intermediate nodes do not need to know about encryption matters between Alice and Bob [148, 120].

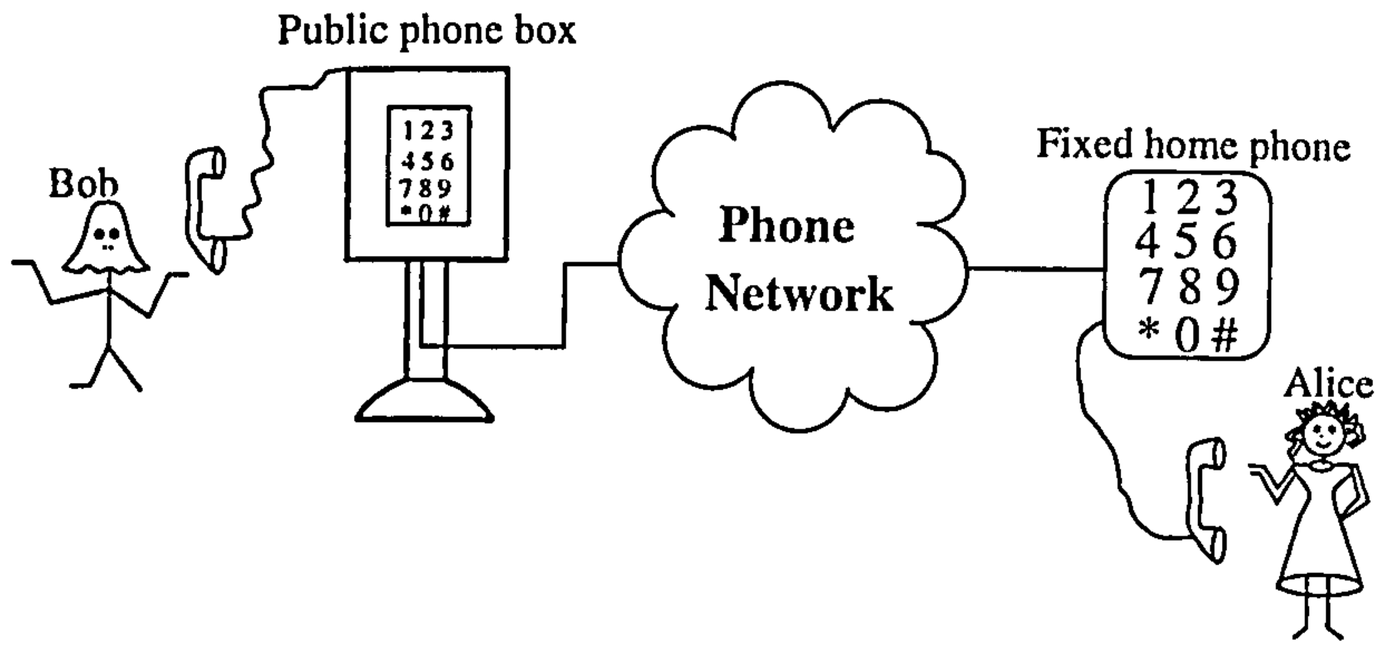


Figure 5.1: An anonymous call made from a public telephone box.

It is widely accepted that for a system to be successfully designed, implemented, understood, debugged, and extended it has to be based on simple ideas. The explanation for this is that simple designs are easier to understand and verify than complex ones. In computer literature this is known as the principle of simplicity and is highly recommended for cryptosystem implementations where a single flaw renders the whole system useless. Hence, to reduce the risk of having a hidden flaw in the design or implementation and to increase the chance that a cryptosystem can be verified and correctly implemented complicated ideas and large codes have to be avoided. Being aware of this, we have made a special effort to present a simple solution to the problem of anonymous and confidential communications and to keep our design as simple as possible under the assumption that it can be easily extended and used in real life applications.

5.3 Anonymous calls from a public telephone box

As was stated in section 1.4.2, anonymity is a necessary condition for several services to work in modern societies; it is used in everyday life intensively, most of the time unconsciously. Yet in certain cases it is used deliberately. In the latter case there are only a few mechanisms providing true anonymous communication, specially when on-line two-way communication is needed. So far, the only one readily available in most urbanized areas is the public telephone box.

As simple as it is, a public telephone box provides for true anonymity when operated by coin, prepaid telephone cards or when the user dials a free of charge number. The only thing a person has to do to initiate an anonymous telephone call is to go to the public telephone box, make the call and fade away from the site immediately, i.e. before the telephone company may trace the call back, find out where the telephone box is located and send someone there to identify the caller. It might also be that all the public telephone boxes the user might use are continuously under surveillance by satellite cameras, so that pictures of all callers are taken to associate them with the anonymous call made at a given time. Although in theory this is possible, currently deployed technology cannot afford it, consequently, we will not consider this as a serious threat in the near future. The use of a public telephone box is illustrated in figure 5.1 where an unknown person makes an anonymous call to Alice.

In addition to its unbreakable anonymity, the public telephone box possesses another essential characteristic that makes it suitable for making an anonymous call; namely its simplicity. The mechanisms operating a public telephone box are rather simple, familiar to everybody who lives in a urbanized area and well understood by both designers and users.

This observation has inspired us to take it as a paradigm for designing a mechanism for making

anonymous calls from PDAs. The basic idea is to equip a PDA with all the necessary components present in a public telephone box that make anonymous calls feasible. How many and what components are they? Fortunately, there are only two of them:

- the public telephone box is a public terminal
- the caller uses an anonymous method of payment to operate it

5.4 Concealment of identity behind a public terminal

Because a public telephone box belongs to a telephone company and not to its users, there is no connection between the number of the public telephone box (its identifier) and the caller. Likewise, since it is there to be used by anybody, a call coming from it, could be initiated by any member of the public. Consequently, the identity of the actual caller is blended into the crowd and automatically hidden. Needless to say, for an anonymous call to be successful the caller has to use a public telephone which is handily available to as many passers–by as possible. In brief, the incognito caller hides himself in the crowd and not from the crowd.

5.5 Anonymous payment

While paying for an anonymous phone call made from a public telephone box the caller uses coins to pay anonymously for his call. Anonymous payments are performed not only in public telephone boxes but in several other situations as well. Because anonymous payment is a crucial concept in our approach to confidentiality and anonymity protection we will devote a considerable amount of space to discussing how it works and what risks it exposes the participating parties to.

5.5.1 Anonymity in cash payments

As metallic coins and prepaid telephone cards are the most common method of payment for operating public telephone box they were mentioned in section 5.3 to pay for an anonymous call. It is time to generalize and discuss what is behind this concepts.

Following Camp's terminology [149] the most common form of money that modern societies use at present time can be classified into two categories: *physical token money* and *notational money.*

physical token money at present time it is represented as bank notes and metallic coins (coins for short) issued by a central financial authority, the Bank of England for example, and recognized as a legal document (at least within the limits of a country) for performing payments. It is also called *cash* and *token currency.*

notational money it is money represented as notations (numbers) in the ledgers of financial institutions such as banks. Notational money does not have a three–dimensional representation unless it is converted into physical token money.

The particularity of cash money is that unlike, cheques and bank cards the name of the payer is never written on it. In addition, when a transaction paid by cash is completed there are no bank or law enforcement records left in the hands of the merchant where the buyer's name appears. Thanks to this practice the buyer's identity is not disclosed. It is worth anticipating that there are some exceptions to this assumption, which are discussed in section 5.5.5.

Take a one pound coin for example (or a one dollar note if you prefer) and use it to pay for a newspaper, the payment is performed under strict anonymity as the coin left in the newspaper seller's hands does not contain any information that can be used to determine its transaction history, nobody can find out what goods were given in exchange for the coin. Obviously, the newsagency is just an example of several business that accept physical token money as a method of payment, many others work in a similar way, supermarkets, book stores, hardware stores, etc.

From the above definition and discussion it follows that the coins we use to operate a public telephone box belong to the physical token money category; consequently, they provide anonymity. In fact bank notes may be used for the same purpose but it is not very practical as bank notes are normally of higher value than the cost of a telephone call.

On the other hand a prepaid telephone card may be regarded as a special kind of physical token money, one that is issued by the telephone company and accepted as a method of payment only within its organization. For the purposes of telephone calls coins and prepaid telephone cards are functionally equivalent. For a person willing to make a telephone call from a public telephone box there is no difference between finding three coins of one pound each and an unused three–pound prepaid telephone card.

Although cash money is widely accepted as an anonymous method of payment in numerous applications, it has serious drawbacks that make it unsuitable in many cases. We will discuss the drawbacks relevant to our system in the following section.

5.5.2 Physical surveillance

The prospect of complete anonymity provided by physical token money is limited by the potential of physical observation i.e. by the possibility of somebody (an accidental or malicious observer) observing revealing details of the transaction. In this case the observer may be anybody located at a privileged location with respect to the place of the transaction. Any shop assistant, bank employee or officer is a potential observer.

This issue is an inherent weakness of physical token money transactions as the buyer has no choice but to expose herself to the public nearby the place of purchase to hand the payment in and to collect her goods. Put in cryptographic terms, in traditional commercial transactions paid by physical token money information travels in *plain text* format and is exposed to the public (to well–located observers); consequently, an observer can learn both about the identity of the buyer and the details of the transaction. Consequently, not only anonymity is at risk but also confidentiality. It seems that if we need a higher degree of anonymity and confidentiality in commercial transactions, a different way of performing commercial transactions has to be devised.

5.5.3 Anonymity from the merchants and the buyer's side

One of the limitations of cash money is that it provides anonymity to the buyer but not to the seller. This is due to the fact that merchants usually give away a printed statement (a receipt) where the name and address of the shop, the date and time, the name of the purchased goods, their quantities and prices appear; hence, the buyer has full evidence in written form of the transaction that might be used at the buyer's discretion, to complain in case of dissatisfaction with the purchase for example.

For a commercial transaction to be truly anonymous from both the seller's and the buyer's side it is necessary that cash is used as a method of payment but it is not sufficient; it is also necessary that the merchant does not hand receipt to the buyer. The problem with this approach is that most of buyers will feel unprotected against potential dishonest retailers. In this scheme the buyer

has no choice but to take the merchant on trust about the quality of his service and goods. It might happen for example that the can delivered by a vending machine is half-empty, that it is not delivered by the machine or even worse, that it does not comply with the *best before date*; in any case, the buyer simply loses his money.

This is not as dramatic as it sounds at first glance, after all, in a similar way, merchants have to trust buyers about the genuineness of the physical token money used as a payment (see section 5.5.4).

Besides this risk, this approach works in traditional commerce and works pretty well for low-value (few pennies to a few pounds) commercial transactions. There are thousands of goods and services that fall in this category (newspapers, video games, vending machines, telephone calls, etc.).

Another way to provide total anonymity is by introducing a third party into the scenario through which the merchant and the buyer interact; although this approach might be suitable for some applications it falls outside our interest as its level of anonymity entirely depends on the trustworthiness of the third party to keep it.

5.5.4 Counterfeits

One of the major problems faced by merchants who accept anonymous payments performed by physical token money is the risk of counterfeit. The merchant is at high risk of receiving counterfeit notes, foreign coins, old notes whose printed-on value has been changed by the issuing financial authority; and old coins and notes declared out of circulation.

While accepting anonymous payments the merchant relies on his ability to detect and refuse fake physical token money by a quick visual inspection. Because most merchants are not experts in forgery detection the acceptance of this method of payment is grounded on trust from the side of the merchant. If he fails to detect a forgery at the moment the transaction takes place, his failure is not compensated. It might be possible to accurately verify coins and notes during transactions, yet this would dramatically slow down the speed of the transaction. After all, receiving counterfeit money is an exception rather than a norm since money forgery is an offense severely punished by governments. In addition, this method of payment is normally used for low-value transactions only. Hence in the case of receiving fake money the impact on the merchant's finances is not significant.

5.5.5 Transaction reporting to governments

In most countries governments enforce policies that require that financial transactions over a certain amount of money are recorded and documented to the government. In the USA for example, to discourage money laundering, the government requires reporting all cash transactions above $ 10 000.00. It follows that cash money can provide anonymity only in low-value transactions. It is likely that when e-commerce becomes widely accepted a similar policy will be enforced by international organizations.

From the above discussion it can be concluded that though cash money suffers from several inconveniences, the degree of anonymity it guarantees is a highly valued property. This is why it is used as a method of payment to pay for anonymous calls in the public telephone box introduced in section 5.3.

In the near future, it is likely that most payments in commercial transactions are going to be performed electronically; if our bet is correct and we agree that anonymous payments are essential

for certain transactions, it seems that the notion of traditional cash has to be extended to its electronic version.

If fact, although still in its infancy, this kind of money is already circulating the Internet and is known as *electronic token money*, *digital cash*, *electronic currency*, *electronic cash* and *e–cash*.

5.6 Anonymity in e-cash payments

E-cash is regarded as an extension of traditional cash, the main idea behind it is to have an electronic equivalent of traditional cash that preserves all the valuable properties of its ancestor, anonymity for instance; and if possible that solve or improve at least some of the drawbacks inherent in traditional cash due to its physical nature; namely, the risk of physical surveillance, anonymity from the merchant's side and risk of counterfeiting.

Although the use of e–cash as a method of payment is not widespread yet in real life, theoretical results converted into working products prove that it is perfectly feasible to do business with support for anonymous e–cash as a method of payment.

A payment system that accepts e-cash as a method of payment is called an anonymous payment system. An *anonymous payment system* brings together a spender, a merchant, and a bank where both the spender and the merchant are account–holders; its aim is to help the spender to pay the merchant for services or goods without disclosing her identity to the latter and without leaving any tracks at the bank to find out how and where the spender spends his money. A well–known implementation of an anonymous payment system is DigiCash [149, 127, 150, 151]. Anonymous payment systems (DigiCash for instance) are based on the use of a technique known as *blind signatures* where the spender uses a sort of electronic notes validated by the blind signature of his bank to pay the merchant (see 4.1.6).

As a side comment it is worth mentioning that e–cash is not the only method of payment proposed for electronic commerce transactions. Other mechanisms are currently being tested in the Internet.

According to the mechanism by which money goes from the buyer to the merchant, the different proposals can be grouped into five broad models: system supporting secure presentation of credit card numbers, e-cash, credit–debit systems, direct transfer, and collection agents [152, 150, 9]. Each of the models has its own advantages and disadvantages and its suitability depends on the specific application, the amount of money to be paid for example, may be determinant; for the particular case of our work, the e–cash model is the most suitable, since it is the only one that protects the anonymity of the payer.

5.6.1 Advantages of e–cash over cash payments

The functional characteristics of e–cash are quite similar to those of its physical equivalent— cash money, the most important to us being the support of anonymity. It is time now to study how e–cash inherits the limitation of cash mentioned in section 5.5.1 and how to reduce their impact.

Due to the fact that Internet commercial transactions can be performed remotely instead of face–to-face, electronic commerce transactions offer a stronger anonymity protection than traditional transactions paid by physical token money. In addition, as the information exchanged between the merchant and the buyer can be easily encrypted confidentiality protection can be improved a great deal. The potential risk of physical surveillance in this case is non–existent.

In cases when the goods (electronic newspapers for example) can be digitized and sent over the communication links the level of anonymity and confidentiality offered by an electronic commerce system is nearly perfect, at least in theory, in practice it is limited by potential flaws in the design of the system and the risk of disclosing its security keys to the wrong person.

If merchants and buyers are happy making business under the anonymity provided by cash money, there are good reasons to think that a similar approach can be used in Internet commerce where anonymity from both the merchant's and buyer's side is a requirement. For this to be possible, e-cash must be used as a method of payment. A mechanism like this may be suitable for implementing Internet vending machines to sell newspapers, documents and other pieces of information.

Taking into account current social behaviour, it seems inevitable that counterfeit money will be present in electronic commerce as well; also it seems that for the sake of anonymity and simplicity it is worth following a similar approach as traditional commerce, i.e. the acceptance of anonymous electronic token money as a method of payment for low–value goods. However, as face–to–face shopping is no longer necessary, the inspection of the electronic money received as payment has to be performed more accurately than in traditional transactions, but at the same time it has to be light enough, at the price of some risk, to keep the efficiency of electronic commerce transactions acceptable.

If by law cash transactions over certain amount of money have to be reported to the government, there are reasons to think that e–cash transactions will fall into the same scheme, except that in this case the government will need to improve its audit methods as e–cash opens new possibilities for making fraudulent transactions appear legal; big transactions, for example might be divided into hundreds of small payments (performed at electronic speed) to comply with the government policies and mislead the government.

5.6.2 DigiCash anonymous payment

Digicash uses public–key cryptography to perform anonymous payments. Thanks to this mechanism it is impossible to link a payment to a payer. However, a payer can prove that he or she did or did not make a particular payment. For anonymous payments to be possible, the DigiCash systems needs the help of a banker who hides the payer's identity by blindly signing e–notes. Obviously, both the payer and merchant are the banker's customers. Briefly the algorithm is as follows[153]:

1. The bank, Bob (the buyer) and the merchant hold a private key and advertise their public counterparts.

2. Messages encrypted with the bank's private key can come only from the bank.

3. Similarly, messages encrypted with Bob's private key can come only from Bob.

4. Again, messages encrypted with the merchant's private key can come only from the merchant.

5. Whenever Bob wants to pay for something, he generates a note for the due amount and blinds the note number with random number (a blinding factor).

6. For the note to be accepted by the merchant it has to be signed by the bank, so Bob sends the note to the bank in order to be blindly signed.

7. Knowing Bob's public key, the bank verifies that the note comes from Bob, signs it and returns it to Bob.

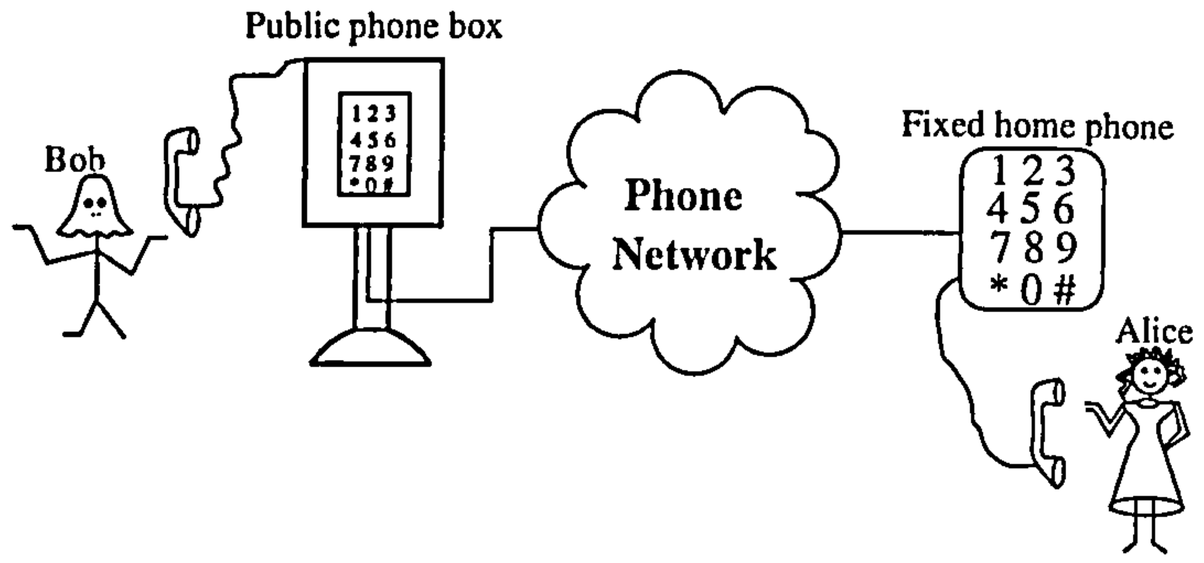


Figure 5.2: Anonymous call made from a public telephone box.

8. An amount of money equivalent to the value of the signed note is transferred from Bob's account into the bank's account.

9. Bob divides out the blinding factor and sends the note to the merchant.

10. Knowing the bank's public key, the merchant verifies that the note has been signed by the bank and accepts it as a payment.

11. Latter on (at the end of the day, for example) the merchant sends the note to the bank.

12. Knowing the merchant's public key the bank verifies that it comes from the merchant, then an amount of money equivalent to the value of the note is transferred from the bank account into the merchant's account.

5.7 The public telephone box paradigm

The hardware components for making an anonymous call from a public telephone box were shown in figure 5.1; to make the discussion that follows easier, the same figure appears in this section as figure 5.2.

As briefly discussed in section 2.7.1 one of the most widely accepted approaches for supporting host mobility in the Internet is the Ioannidis paradigm [46].

In this paradigm a mobile host (while connected to the network) is always located in some cell controlled by a so-called *Mobile Support Station* (MSS). The MSS serves as the current home for the mobile host and is responsible for communication between the mobile host and the rest of the network.

Ioannidis' paradigm assumes that each mobile host owns a home IP address by which it is identified within the Internet domain. This home IP address uniquely identifies the mobile hosts and remains constant regardless of the mobile host physical and logical location. In figure 5.3 for example this is illustrated by Bob's PDA which has been registered to the Internet with the 132.248.51.6 IP address.

By looking at figures 5.2 and 5.3 it is not difficult to realize that there are close similarities between them. For a start, both the Internet and the telephone network are there to transmit messages between the communicating parties; to transmit messages from an anonymous caller to Alice, for instance.

Secondly, one can think of the public telephone box depicted in figure 5.2 as the MSS being used by Bob in figure 5.3; the two of them play play a similar rôle.

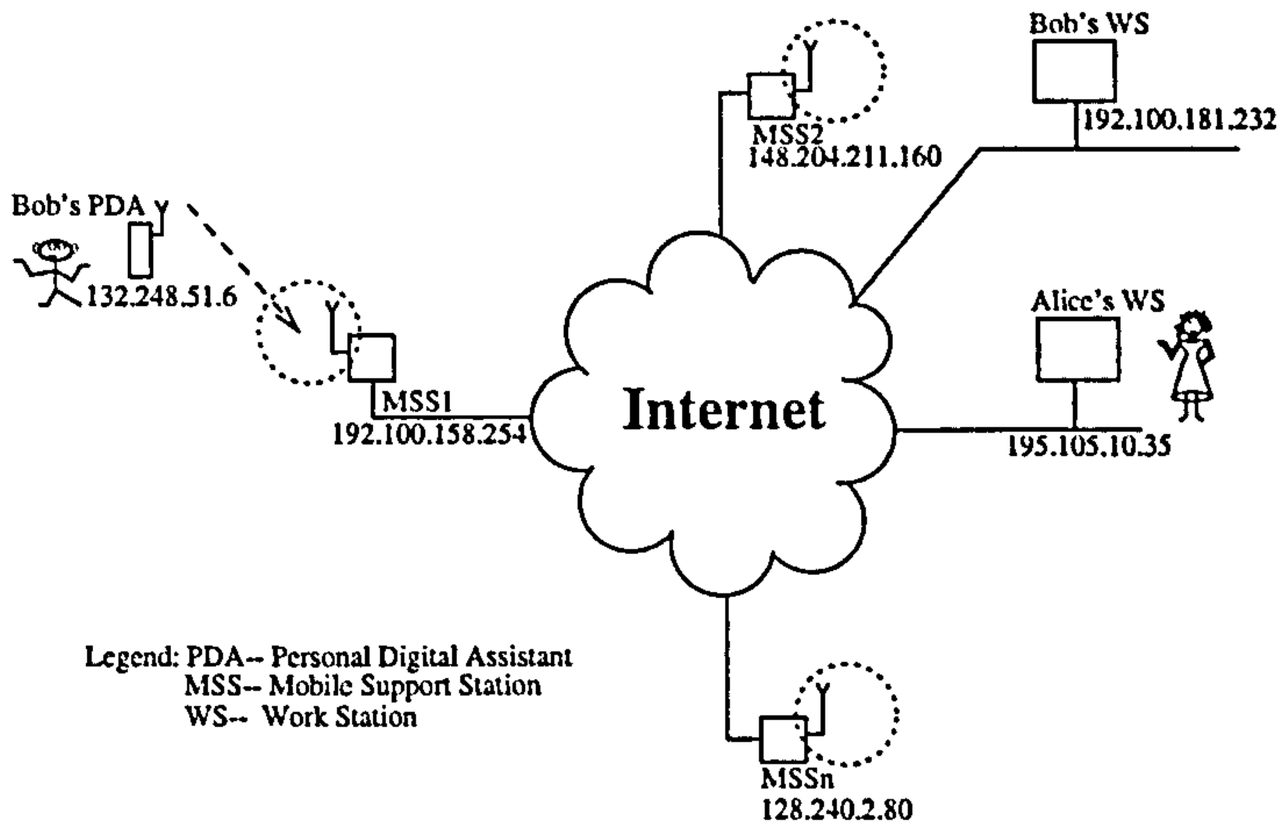


Figure 5.3: Ioannidis paradigm for integrating PDAs to the Internet.

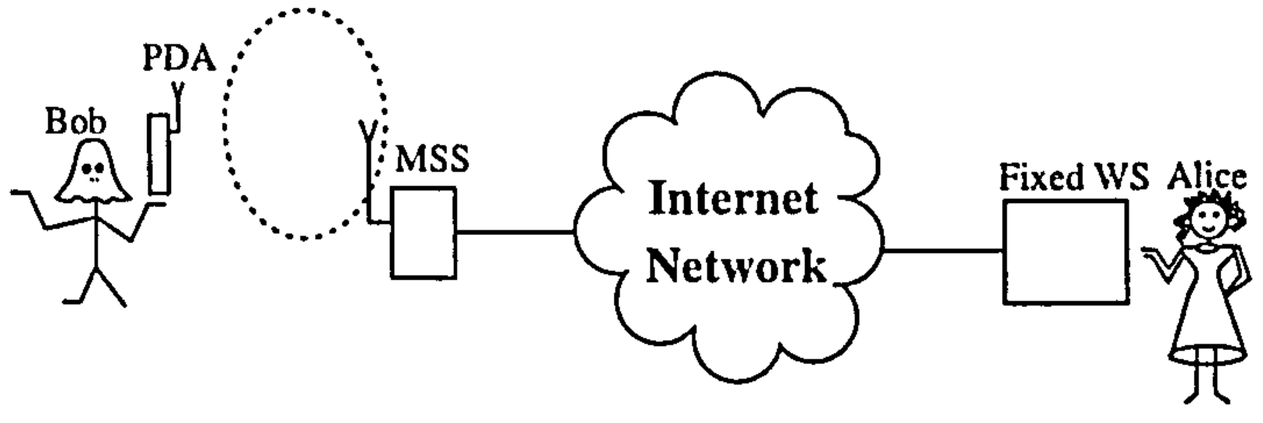


Figure 5.4: Anonymous call from a PDA.

Finally, Alice's fixed home telephone in figure 5.2 is analogous to Alice's WS shown in figure 5.3, they are there to be used by Alice to send and receive calls while she is home.

If this analogy makes sense, then it is reasonable to take figure 5.2 as a paradigm for designing and implementing a system for making anonymous calls like the one shown in figure 5.4.

An anonymous call initiated from a public telephone box is possible only if the caller uses cash to pay anonymously for the service and that caller uses a public terminal rather than a personal one. These two crucial components have to be present in figure 5.4 as well if we want the anonymous calls made from the incognito PDA holder to be truly anonymous.

Suppose that the PDA user in figure 5.4 wants to make an anonymous call and that he has some e–cash in his PDA memory to operate the MSS. The procedure for opening the anonymous session between the PDA and the MSS works as follows:

1. Turn on the PDA.

2. Make contact with the MSS.

3. Slip a couple of e–coins as an advance payment for a communication session of an agreed period.

4. The MSS responds by creating a non–personal random temporary identifier for the incognito PDA user and sends it to him.

5. The PDA and the MSS use the temporary identifier as the PDA's address to talk to each other. Additionally the MSS uses it to time out the PDA session.

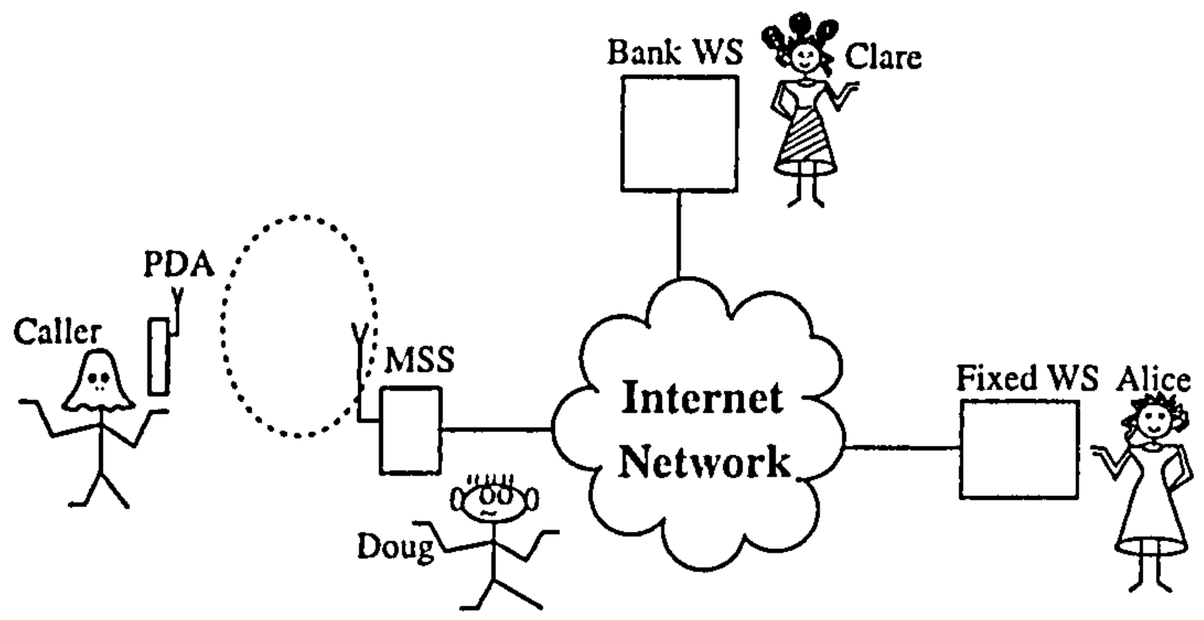


Figure 5.5: Parties involved in an anonymous call from a PDA.

It is now time to study how the anonymous e–cash payment is performed and how the non–personal random temporary identifiers are created.

5.8 E–cash payment for a MSS communication session

While the market for ubiquitous communication services to PDA is still small, it is projected to increase dramatically in the years to come, hence it makes sense to assume that public MSSs will be widely deployed and run by communication companies on a free–for–service basis the same way public telephone boxes are run by telephone companies; if this holds true, it is likely that, as public telephone boxes do, MSS will offer free of charge services, that is, electronic address a PDA user will be allow to communicate with without any charge, example of these addresses might be police, fire brigade, national drugs helpline, and other services equivalent to the usual free–helpline services offered by telephone companies.

In terms of payment, there is nothing to discuss here if the PDA user wants to anonymously call to one of the free–of–charge services offered by the MSS. In this case the MSS owner either does not get paid at all, or gets paid at the end of the month by the government; this is not relevant here; the PDA user can just take it as a free call. However, the matter becomes more complicated when the PDA user wants to call a non–free address since this situation brings into the scene a bank or a similar financial body. The need to address this issue, motivated us to make figure 5.4 slightly more complicated by bringing more players into the scene, the result of this is shown in figure 5.5.

Now we have included Doug, the owner of the MSS and who offers communication services in return for a charge on a pay–for–used–time basis. He is an account holder at Clare's bank and accepts electronic cash as long as it is signed by Clare. Obviously, Clare is a bank owner who is linked to the Internet through her bank workstation. Among other services, Clare offers support for anonymous electronic payments in transactions between her account holders. That is to say, on request she writes blind signatures on electronic notes presented by her account holders as long as the e–notes comply with the blind signature protocol discussed in section 4.1.6.

Now we have all the participants that take part in the opening of an anonymous session with a MSS. Let us now see how they interact.

For simplicity of exposition we assume that the anonymous caller knows what e–notes are accepted by the MSS as payment and the minimum charge for opening an anonymous session.

Assuming that the anonymous caller knows the cost of an anonymous call in his local town seems to be reasonable since we expect that in the future MSS are likely to become an everyday

public facility just as public telephone boxes are at present time. In any case, the anonymous caller can always inquire of the MSS about these details before the anonymous call transaction begins. The inquire process is not an essential part of the main protocol, thus, we will not discuss it here, but later in section 8.

Again, to make a clear distinction between the main protocol for opening an anonymous session and other details, let us assume that prior to opening an anonymous session with the MSS, the anonymous caller ensures that he has e–coins in his PDA memory to pay for the service. To make the e–coins, the anonymous caller creates them and sends them to Clare for a blind signature. The routine for a blind signature is discussed in section 4.1.6. Therefore, we can assume that prior to requesting an anonymous session, the anonymous caller is in possession of enough e–coins to pay for it.

The assumption about the endless supply of e–notes in the memory of the PDA is grounded on the fact that there is no price to pay for keeping e–cash in the memory of the PDA. Furthermore, unlike coins, e–coins are weightless and do not represent any physical burden to carry. After all, should the *endless* supply of e–coins come to an end in the middle of an anonymous session, the PDA user can always divert momentarily from the main protocol and contact his bank for more e–coins; the only thing he has to do is to follow exactly the same routine as he did to obtain the e–coins he followed prior to the anonymous call.

Taking into account the previous discussion we can now summarize how the anonymous caller, let us say Bob, opens an anonymous session with the MSS. The participants in the algorithm are the ones depicted in figure 5.5.

1. Bob, the anonymous caller, turns on his PDA to contact the MSS.

2. Bob sends an e–coin previously blindly signed by Clare to Doug as a payment for the service.

3. Upon receiving the e–coin, Doug forwards it to Clare to verify that the e–coin is valid.

4. Doug waits for Clare's response, if the response is satisfactory, he opens the anonymous session for Bob.

Note that the advantages of e–cash over traditional cash discussed in sections 5.5.1 and 5.6 significantly strengthen the degree of anonymity of the caller.

Firstly, since the PDA and the MSS communicate with each other at a certain distance ranging from several metres to few kilometres, nobody can learn what Bob is doing with his PDA and whom and under what conditions he is communicating with; that is to say, the risks of physical surveillance is no longer a concern.

Secondly, the anonymity of payment is guaranteed by Clare's blind signature.

Thirdly, the risk of money counterfeiting is reduced by taking advantage of the speed of computers to perform thousands of comparisons and looks–up to find out whether an e–coin is a forgery or has already been spent. This risk can be reduced as much as possible by strengthening the validation algorithm, of course at the price of losing speed in the process.

Finally, though it seems obvious, it is worth noting that we expect no government will be interested in receiving reports from commercial transactions involving small amounts of money as payment for communication services requested by a PDA. Hence, it is reasonable to take for granted that Doug will have no problems with the government from accepting anonymous e-cash.

5.9 Mobile hosts without home IP addresses

The traditional Ioannidis paradigm for integrating mobile hosts into the Internet was shown in figure 5.3. This paradigm is grounded on the assumption that when a mobile host, a PDA for example, is away from its home network it still needs to access data (personal files, local data bases, local Web pages, etc.) and services (name, file, Web, mail, billing servers, etc.) normally used on its home network. Consequently, a PDA must be assigned a permanent IP address in its local network which uniquely identifies it and remains constant regardless of current physical and logical location of the PDA.

This assumption is certainly justifiably, however, there are applications where a PDA can work perfectly well without using its home data and services, i.e. without contacting its home network. For simplicity let us assume for the moment that a MSS offers free communication services to PDAs in its area. If this is true, a PDA does not need any support from its home network to download Web pages from a free Web server for example, nor to post information to an e-mail list or listen to the news. The main idea here is that a PDA can still use the Internet without using its home IP address. It might even not have any IP address nor a home network.

In contrast with the Ioannidis paradigm we suggest a new approach to integrating PDAs into the Internet where a PDA does not need a home IP address to send and receive information to and from other PDAs and computers reachable through the Internet.

Our IP addressless approach has two main advantages. First, by not using an IP address we are providing the basis for true anonymity as we will see later on. Second, a PDA is a tiny cheap computer for personal use. There will be millions of them carried by people who would like them to be operational as soon as they buy them. Also, they will be a kind of disposable computers—a user will buy a PDA, use it, and throw it away if it fails or change it for a new one if she is not satisfied with its performance. Because of this, it seems impractical to assign IP addresses to them. What is needed is a simpler way of addressing them.

Our approach is based on the assumption that in the future the world will have many thousands of MSSs. Some of them will belong to private LANs and be located indoors; others will belong to communication providers and be located outdoor and provide communication services to PDA users for a payment just as public telephone boxes do nowadays. PDAs will act as clients (to MSSs) for network services and never as servers.

In this scheme, as illustrated in figure 5.6, a PDA user may have a home workstation (or an office one or both); just as a user of a public telephone box may have an office and home telephone number. PDA users are known to the world by their home workstation address, this is where they normally receive messages; if a message arrives while a PDA user is roaming it is kept unread on a local disk until they come back or retrieve it remotely. Thinking of a home workstation as a home telephone equipped with an answering machine illustrates its rôle.

Whilst on the move, a PDA user can communicate with the world with the help of a MSS. In order to send a message to a computer connected to the Internet or to establish an on–line two–way communication the PDA user must first register with a MSS. After a successful registration the MSS assigns the PDA a temporary, random identification number which is valid for the duration of a session. We will call this number a *TmpId* from now on. Being a random and temporary number, the TmpId is non–personal, as a result it does not lead to the identity of the anonymous caller. The TmpId can be any number, such as a dynamic and temporary IP address assigned by means of the Dynamic Host Configuration Protocol (DHCP) [154] for example.

The combination of the PDA's TmpId and the MSS's Internet address, gives the PDA user a unique non–personal identifier within the Internet, therefore, with the help of this unique identifier she can exchange information with other PDA users and with any computer connected to the

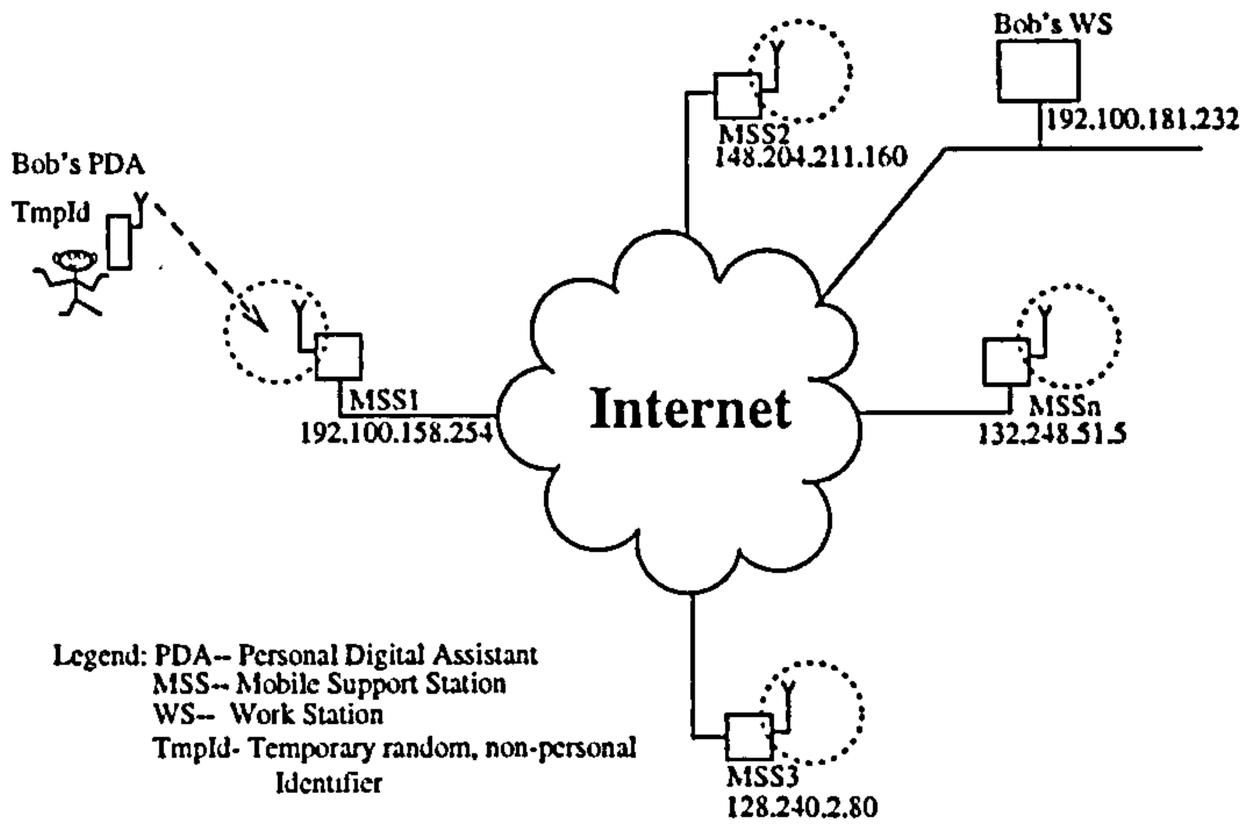


Figure 5.6: A PDA without a home IP address.

Internet.

The MSS serves as a bridge between the PDA and its counterpart. Messages sent to the PDA user are addressed to the MSS which, upon receiving them forwards them to the PDA user identified by her TmpId within the MSS's domain. Needless to say, the MSS will only be able to deliver the message to the PDA user as long as she still is in session and within the area of coverage of the wireless communication device of the MSS.

Once again, there is a close similarity between a PDA user and a user of the telephone system in the sense that while the latter is outdoors and near a public telephone box she can receive calls directly instead of through her answering machine, if she advertises to her potential callers the number of the public telephone box she is near. Similarly, a PDA user can receive messages directly, instead of through her home work station, if she opens a session with a MSS and advertises the Internet address of her current MSS and her TmpId to her potential callers.

If the session time runs out, the PDA user can either pay for an extension of the session or lose the TmpId and stop sending and receiving messages through the MSS. It may also happen that the PDA user moves to another MSS while a session is in progress (hand–off). Although interesting, this question falls in the field of host location and routing strategies [85, 13], and we will not address the issue here.

5.10 An algorithm for anonymous and confidential calls

After learning how to open an anonymous session with a MSS and how a non–personal identifier can be assigned to a PDA it is time to introduce more specific details of the main algorithm to make an anonymous call and to present and discuss it.

5.10.1 Learning the public key of the MSS

So far we have been assuming that Bob, the anonymous caller, is the only PDA user at the MSS and that nobody but him and the MSS can read the contents of the messages sent to and received from the MSS. This might not be true because a MSS is a public facility, hence Bob might clash with Ebe, another PDA user whose PDA might overhear the conversation between Bob's PDA and the MSS. The answer to this problem comes from public–key cryptography. To prevent Ebe from reading the content of the message Bob sends to the MSS asking for an anonymous session, Bob

can encrypt it with the public key of the MSS, so that only the MSS can read it (see section 4.1.3). Yet this approach raises the question about how the PDA obtains the public key of the MSS.

The crucial point here is to ensure that the public key the PDA user uses to encrypt the first message he sends to the MSS belongs to the MSS and not to somebody else, to Ebe the intruder for example, who might be there to intercept the PDA message to impersonate the MSS.

Possible solutions to this question were presented in section 4.2.2, namely the distributed and centralized approaches. It seems that for our system, the centralized approach is simpler than the distributed one; the only thing the MSS has to do is to advertise its public key upon detecting the presence of the PDA within its area of coverage. Naturally, the MSS public key has to come with its public-key certificate signed by a certification authority so that the PDA can verify the signature and be sure of the authenticity of the public key.

Although with some extra hassle, the distributed approach might work as well, the trouble here is that the PDA user might receive from the MSS a public-key vouched by somebody unknown to him, so he might refuse an authentic key advertised by the MSS. A possible solution to this problem is for the MSS to advertise several keys or the same key signed by as many well-known people as possible so that eventually the PDA user receives one that satisfies him.

5.10.2 Session keys

In section 4.2 some reasons not to use public and private keys intensively were pointed out; to comply with experts' recommendations, we use public keys only to exchange session keys, between the PDA and the MSS and between Bob, the anonymous caller, and Alice, the recipient of the anonymous message.

A session key may be used to encrypt one message only or to encrypt messages for the whole session; the more frequently we renew them the more secure is the whole system, unfortunately, one pays for it in terms of speed and complexity. To keep the presentation of our system simple, we opted to go for the latter approach— one key for the whole session.

It is worth recalling that private keys are used to digitally sign messages; in this case their use is unavoidable, hence when a message is digitally signed, the private key of the signer is always involved.

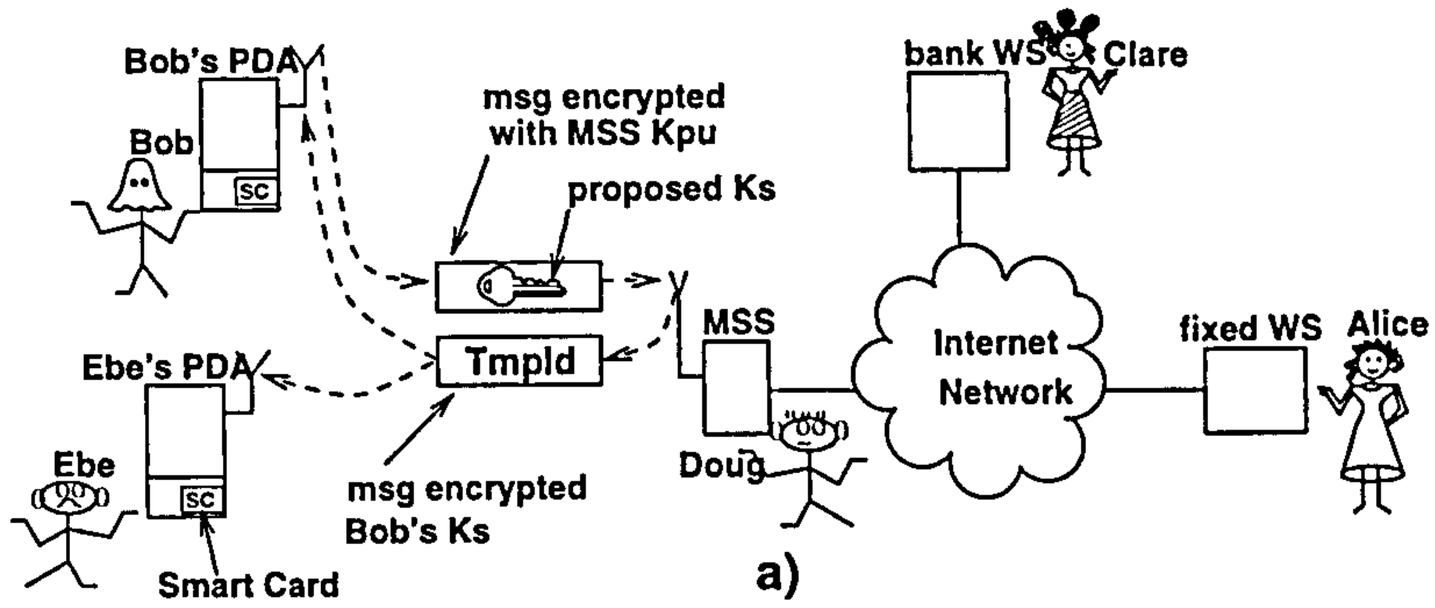
The above procedure seems to work fine, but, it has two serious drawbacks. For a start, it does not protect the content of messages from being heard by unwanted recipients; messages between the PDA and MSS are plain text, that is to say, there is no confidentiality protection.

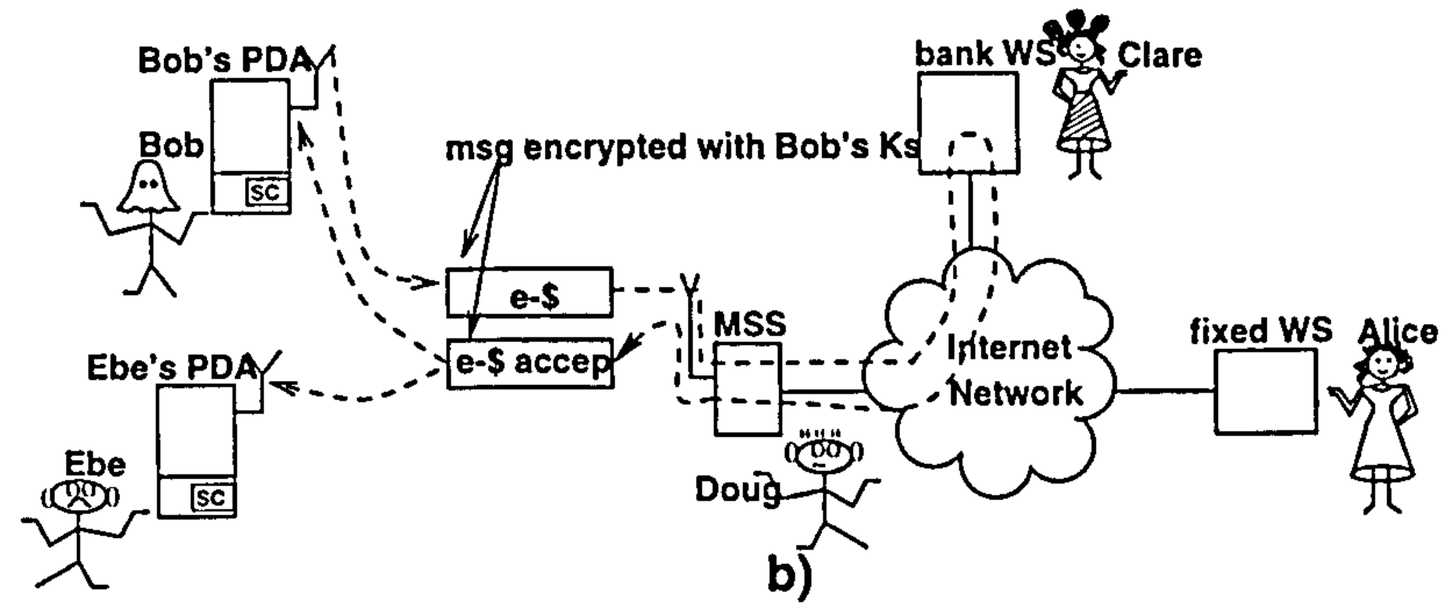
5.10.3 The algorithm

Once again the actors in our algorithm are Bob, a PDA owner wishing to make an anonymous call to Alice; Alice, the recipient of the anonymous call who owns a desktop workstation; Doug, the owner of the MSS; Clare the bank owner; and Ebe, another PDA owner that happened to be at the same MSS as Bob. All these actors and their computers are depicted in figure 5.7. The algorith for sending anonymous and confidential messages using PDAs was first published in [155] and reads as follows:

1. Bob turns on his the PDA.

2. The PDA makes contact with the MSS and learns its public key by listening to its advertisement. It checks for the authenticity of the key by checking the digital signatures attached to the key.





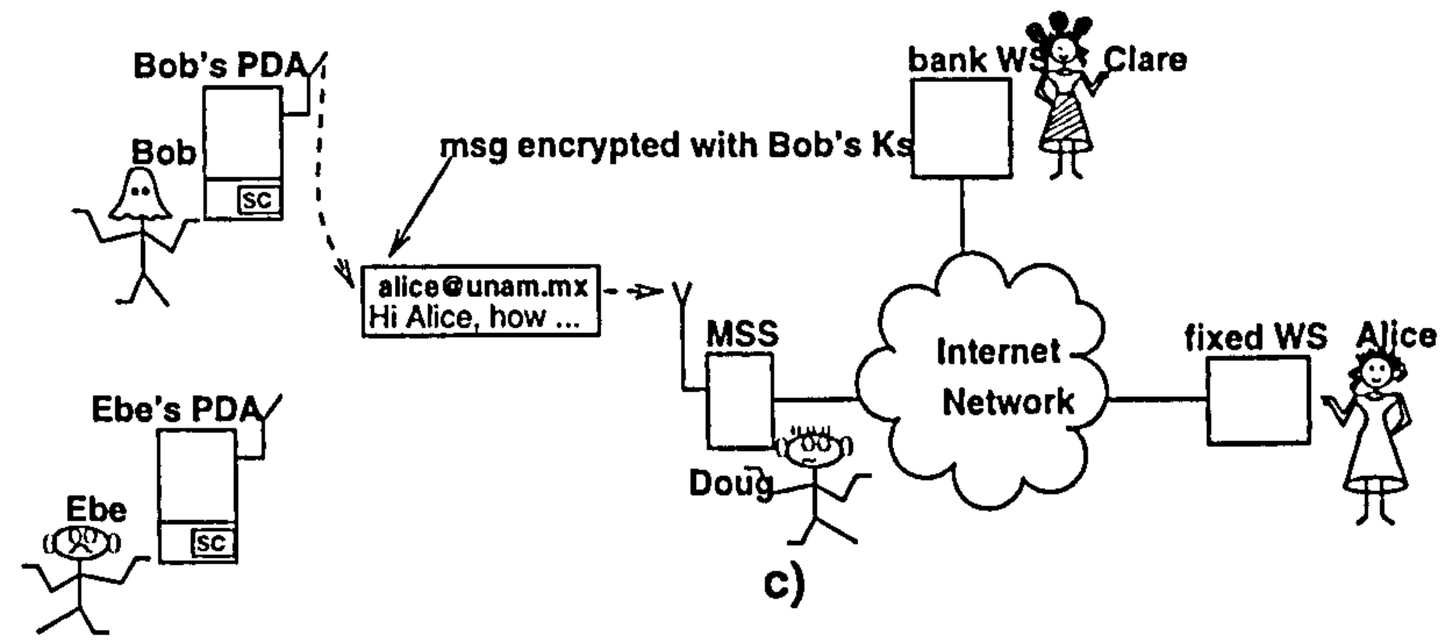


Figure 5.7: Anonymous and confidential call from a PDA.

3. The PDA creates and sends the MSS a session secret key.

4. The MSS checks that the session key suggested by Bob's PDA is not in use by another PDA and Bob's PDA waits for a reply for t units of time. The MSS replies by sending a TmpId only if the key is accepted otherwise if the suggested key is incorrect or already in use it does not reply. The MSS silence is taken by the PDA as an invitation to suggest another session key. To ensure confidentiality, the PDA encrypts the message before sending it using the public key of the MSS. The MSS reply is encrypted with the suggested session key and sent to the air.

5. From now until the end of the session messages exchanged between the PDA and the MSS are encrypted and decrypted with the session key. Since only Bob and the MSS know the session key, only they can make a sense of them, other potential recipients, Ebe for example, ignore them.

6. Bob sends an anonymous e–coin blindly signed by Clare, to Doug to pay for the communication session. Doug contacts Clare to check that the e–coin is valid. The MSS responds with a *EcashAccepted* message if Doug is satisfied or with a *EcashRejected* message in the opposite case.

7. Though not shown in picture 5.7 Bob attachs his TmpId to the messages sent to the MSS, so that the latter can tell where a message is coming from.

8. If Bob wishes to send an anonymous message, to Alice for example, he edits the body of the message, concatenates it with the recipient's address and sends it to the MSS.

9. Upon receiving each message from the PDA the MSS decrypts it, finds out the intended final destination (Alice's address in this case) and forwards it to the specified address. The forwarded message contains Bob's TmpId.

10. To Alice the received message appears as coming from the MSS. To be precise, from someone who identifies himself as TmpId within the MSS area of coverage. However, she has no way of discovering who is the original sender; nevertheless, if she wishes, she can reply by addressing her response to the MSS; the latter will forward the message to Bob.

11. Bob's session ends when he turns off his PDA, leaves his current MSS or his MSS times-out his session.

5.10.4 Discussion of the algorithm

In the algorithm just presented, we made a deliberate effort to keep the discussion as simple as possible at the price of omitting details of secondary importance, it is time now to discuss present these issues to the reader.

Bob hits somebody's session key

One of the crucial steps in the algorithm is that the MSS assigns a TmpId to Bob's PDA without knowing anything about it and about its owner except that it has succeeded in providing a correct session key which is not being used by anybody else. A correct session key is a key that is within the domain of the keys accepted by the MSS and has not been identified as a weak key [118]. If Bob's suggests an incorrect key the MSS does not reply. Nor does it reply if Bob's key is correct

but being used by somebody else, by Frank for example. In this case we say that Frank's session key has been hit and spoilt by Bob. Although the MSS says nothing to Bob, it has bad news to tell to Frank, namely that his current key has been ruined and that he cannot continue his anonymous session till he gets a new session key.

The reason for bothering Frank about this is to stop Bob using this strategy for finding other PDA user's session keys: if Bob knows that he has suggested a correct key and no reply comes Bob knows that he in possession of somebody's else session key and can abuse it unless the MSS informs Frank about it.

Double encryption of Bob's messages

As shown at the bottom of figure 5.7 upon the message received from the PDA, the MSS discovers Alice's address followed by the body of the message Bob wants to transmit to Alice; since both the address and the message are in plain text the MSS can read what Bob's is saying to Alice, consequently, Bob's right to confidentiality is not observed. The only thing the MSS needs to know to convey the message to Alice is Alice's address; if Bob wishes to stop the MSS from reading the body of his message, he may double encrypt it. First he encrypts the body of his message he is sending to Alice using Alice's public key. Secondly, he appends the encrypted text to Alice's address. Finally, he encrypts the result of his concatenation using the session secret key he shares with the MSS.

Again, Bob is faced with the question about how he learns Alice's public key.

Learning the recipient's public key

In a centralized key management system Bob can contact a certification authority to obtain Alice's public key, to save anonymous session time, he should have the key in his PDA memory before opening the anonymous session.

In a distributed key management system learning Alice's key may be slightly more complicated since there is no central authority who knows exactly where Alice's public key is, because of this, it is strongly recommended that Bob is in possession of Alice's public key before opening the anonymous session. Alice's public key can be obtained from a public key directory or Internet public key server as in the PGP system (see section 4.2.2). It is also possible that Bob obtains Alice's public key directly from her or from a common friend. However, this approach puts Bob at risk since Alice might guess who is anonymously e–mailing her by checking to whom she has given away her public key.

Upon receiving the anonymous message Alice decrypts it using her private key, however, she does not know who is writing to her so she cannot encrypt her response with Bob's public key. A simple answer to this is that if Bob is expecting any reply from Alice he must create a secret key and send it to Alice, attached to the end of the body of his message for example. If for some reason Bob does not send to Alice a session key, Alice have no choice but to encrypt her response with the public key of the MSS. However, the confidentiality of her response is broken at the MSS who can read her message before forwarding it to Bob.

E–coin verification

Before opening an anonymous session for Bob, Doug makes sure that the e–coin he has received from Bob is a valid one. Put in other way, Doug has to send the e–coin to Clare for inspection. For the sake of confidentiality Doug has to encrypt his message with Clare public key. Clare responds with a message encrypted with the MSS public key or with a secret session key if Doug cared to

create and send it to her attached to the e–coin. The public keys used here can be obtained the same way Bob obtains Alice's (see the above discussion).

5.10.5 Equipping a PDA with a smart card

Message encryption and decryption as well as blind signatures depend heavily on CPU-consuming arithmetic operations (mainly modular arithmetic). Owing to this, they are normally performed on dedicated hardware known as cryptographic controllers or cryptographic co–processors [156]. To improve the response time of PDAs expected to engage in these cryptographic operations, it seems sensible to provide them with cryptographic co–processors.

A practical solution for providing a PDA with a cryptographic co–processor is the insertion of smart cards into PDAs. A smart card is a tamper–resistant single–chip microcomputer with CPU, I/O operations and memory, inserted in plastic the size and shape of the familiar bank card [157]. Depending on the smart card model it may also contain, in the same chip, a cryptographic co–processor and specialized algorithms to compute and manipulate any necessary keys [158, 156]. More details about smart cards and their computational and storage capabilities were presented in section 4.3.

Note that we are not arguing that the smart card is required for our algorithm to work. Yet it is recommended to increase the speed of the system and strengthen its security [148]. Any cryptographic algorithm can be implemented in both hardware and software, ours is not an exception. However, a hardware implementation is always faster than a software one. To give an example, a software implementation of the DES algorithm is one thousand times slower than an implementation in dedicated hardware. Concerning, security, software implementations are more vulnerable to intruders since, the encryption key might be retrieved from the disk where is it stored or from the main memory during execution time. This is significantly riskier than executing all computational operations inside a tamper–resistant module, where both the cryptographic algorithm and the key are stored.

Another advantage of having a smart card in a PDA is that the former can be used to store the private and public key of the user. So that the former is never disclosed and the latter never forgotten.

The best keys are those generated randomly (by a random key generator for example), yet a truly random key is uneasy to remember, so it ought to be stored in some place other than the owner's brain. This is where a smart card can offer a solution. The PDA user can command his random key generator to generate his pair of private and public key inside the smart card chip, store the former inside the chip, i.e. in the EEPROM and distribute the latter.

The great advantage of this approach is the private key never leaves the smart card chip. It does need to, since cryptographic operations are performed inside the chip, exactly where the private key is.

It is worth observing that a good cryptosystem will provide its users with several keys[118]. For instance, for reasons of security and key management a user should have one private key for encryption and another one for digital signatures. In addition, it is likely that if a person is involved in more that one activity, he or she will be happy to use different digital signatures, one for signing documents at his workplace and another to sign documents at as a member of the antigovernment political party, to give one example. Because of this, it sounds unfeasible for the PDA user to manage his keys with the help of the smart card and not by hand.

5.11 Summary

Mixes–based anonymizers assume that the anonymous message is sent from a computer which uses a permanent and personal IP–address to connect to the Internet.

Anonymous messages sent by using anonymizers based on mixes are not truly anonymous but pseudo–anonymous because one of the mixes, at least, always know the sender's IP–address. Also, the anonymity offered by these anonymizers is fragile since it can always be broken by subversion or conspiracy of all mixes.

Trying to send an anonymous message from an IP–addressed computer is analogous to trying to make an anonymous call from a home telephone line by using the Calling Line Identification Blocking service (the 141 number in the U.K.), the calling number is hidden from the receiver by preceding the dialed number with the digits 141, but it is not hidden from the carrier, neither from anybody who has the means for persuading the carrier to disclose it nor from a miscreant with enough knowledge and resources to break the carrier's computer where the number is stored.

Truly anonymous calls cannot be made from home phone lines but only from public phone boxes operated by coins. The anonymity is guaranteed because the public phone box is a public terminal (not related to the caller) and because the caller uses anonymous payment to pay for the call.

The idea of the public phone box can be implemented by using a PDA bridged to the Internet by a mobile support station. To communicate with the mobile support station the PDA does not use an IP–address but a non–personal, temporary, random identifier (TmpId) assigned by the MSS on a per–communication session basis. The caller pays for the call by anonymous e–cash. Confidential communication is ensured by the use of the public–key and secret–key cryptotechniques.

To relieve the PDA from the burden of cryptographic operations it can be provided with a smart card.

Chapter 6

Protocol specification of the system

6.1 Introduction

This chapter is devoted to the protocol specification of our system, i.e. to the specification of an unambiguous set of rules to be strictly followed by the PDA and MSS during their interaction in order to initiate, maintain, and complete reliable information exchange. These rules govern the format, contents, order, and meaning of the messages exchanged between the PDA and MSS.

According to [159] for a protocol specification to be complete it should explicitly specify five elements, namely: the service provided by the protocol, the assumptions about the environment where the protocol is used, the protocol vocabulary, the format of messages, and procedure rules. Each of these requirements is specified and discussed below.

Our discussion is based on the algorithm introduced in chapter 5 and illustrated in figure 5.7 where a PDA owner called Bob wishes to make an anonymous call to Alice, who owns a desktop workstation; Doug is the owner of the MSS; Clare the owner of a bank where Bob and Doug are account holders; and Ebe, another PDA owner that happened to be at the same MSS as Bob.

6.2 Service specification

The purpose of the protocol is to provide Bob's PDA, with a mechanism to send one or more e–mail messages to Alice's WS and to receive responses to his messages if Alice decides to reply; all of this without Bob disclosing his identity neither to Alice, nor to anybody else.

The protocol protects the contents of messages exchanged between Bob and Alice by encrypting them with a secret session key which is negotiated between the PDA and the MSS through messages encrypted with the public key of the MSS. Bob's PDA learns the public key by listening to the air where the MSS broadcasts its public key periodically.

Messages from Bob to Alice and vice versa are sent through a MSS and from there to a communication network until they reach Alice's WS. The MSS charges the sender of the message for the use of the communication service. The protocol allows Bob to open an anonymous communication session for a certain amount of money, and then it deducts n units of money from the initial payment for each message sent by Bob. Bob is alerted by the MSS m units of money before his credit runs out; if he wishes, he can send additional payments to the MSS. The MSS finishes Bob's session when no money is left. To protect Bob's anonymity, the payment is made by anonymous e–cash. Before Bob's payment is accepted by the MSS, a bank is contacted to verify that the e–coin received from Bob is genuine.

If not a single message is received from Bob after a certain period of time, either no message the

MSS aborts the PDA session under the assumption that PDA has abandoned its communication session without logging out properly or it has crashed.

On the other hand, if no messages can be sent to the MSS (the channel is full) the PDA aborts its session under the assumption that the MSS has aborted the session at the other end.

An unexpected termination of a communication session raises the issue about what happens to the money that Bob has paid for his communication session. As discussed in section 8.8, this is a topic that needs to be studied.

6.3 Assumptions about the environment

The protocol is expected to work in the Internet environment. It is assumed that Bob's PDA, the MSS that supports Bob's wireless communication and Alice's computer are connected to the Internet and support standard TCP/IP protocols [154]. Likewise, we take for granted that the mail server computer and Clare's bank work station which also participate in the system, run TCP/IP protocols and are Internet connected.

A major issue of concern about the working environment of our protocol is the assumption of an open transmission channel. That is to say, all messages that leave a computer to travel to their final destination are at risk of being accidentally or maliciously overheard by anybody equipped with the necessary hardware and software and legally or illegally connected to the channel.

In figure 5.7 the risk of eavesdropping is represented by the presence of Ebe located in the area of coverage of the MSS currently being used by Bob. Yet nothing prevents Ebe from appearing connected to any other part of the network that links Bob and Alice. Similarly nothing stops the evil Ebe from having a gang of bad fellows[1], Mata–Hari and Tom for example, whose job or hobby is to meddle with the network.

To cope with this hostile environment, messages exchanged between Bob's PDA and the MSS are protected by encrypting then with a session secret key known only to Bob and the MSS. At his discretion Bob can either encrypt the messages he sends to Alice with Alice's public key or send plain text. In the former case only Alice can understand the messages, while in the later, everybody can, the MSS included.

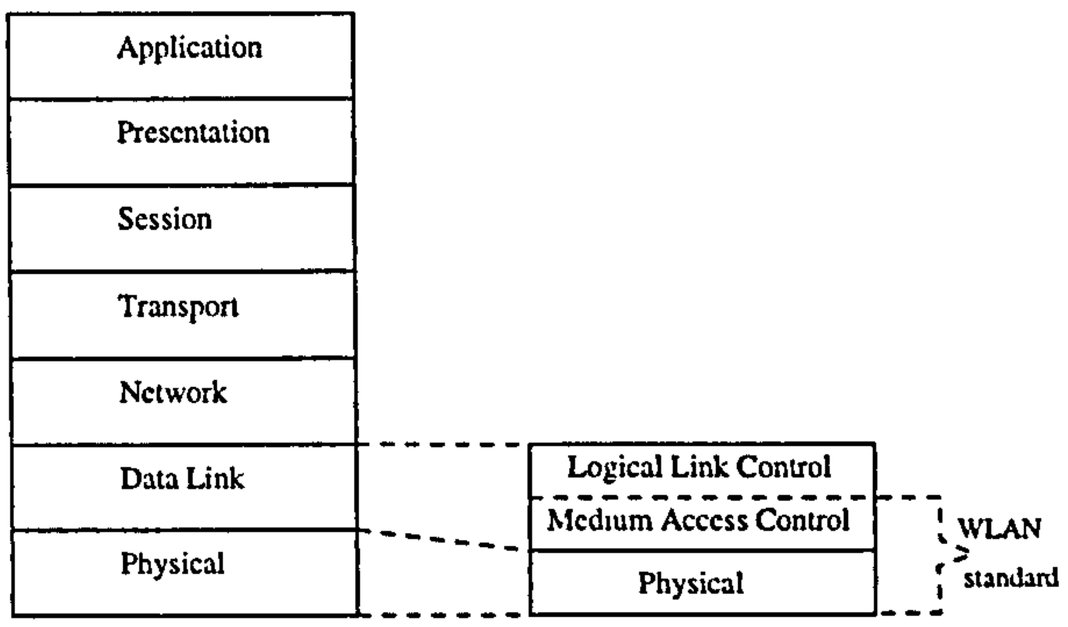
The session key to encrypt messages is negotiated between the PDA and the MSS. For this to be possible the PDA must learn first the public key of the MSS. This is done by the MSS broadcasting its key and the PDA listening for it.

The support of Internet protocols is of crucial importance for us because the protocol we are designing is mounted on top of the services offered by the user datagram protocol (UDP) and the Transmission Control Protocol (TCP).

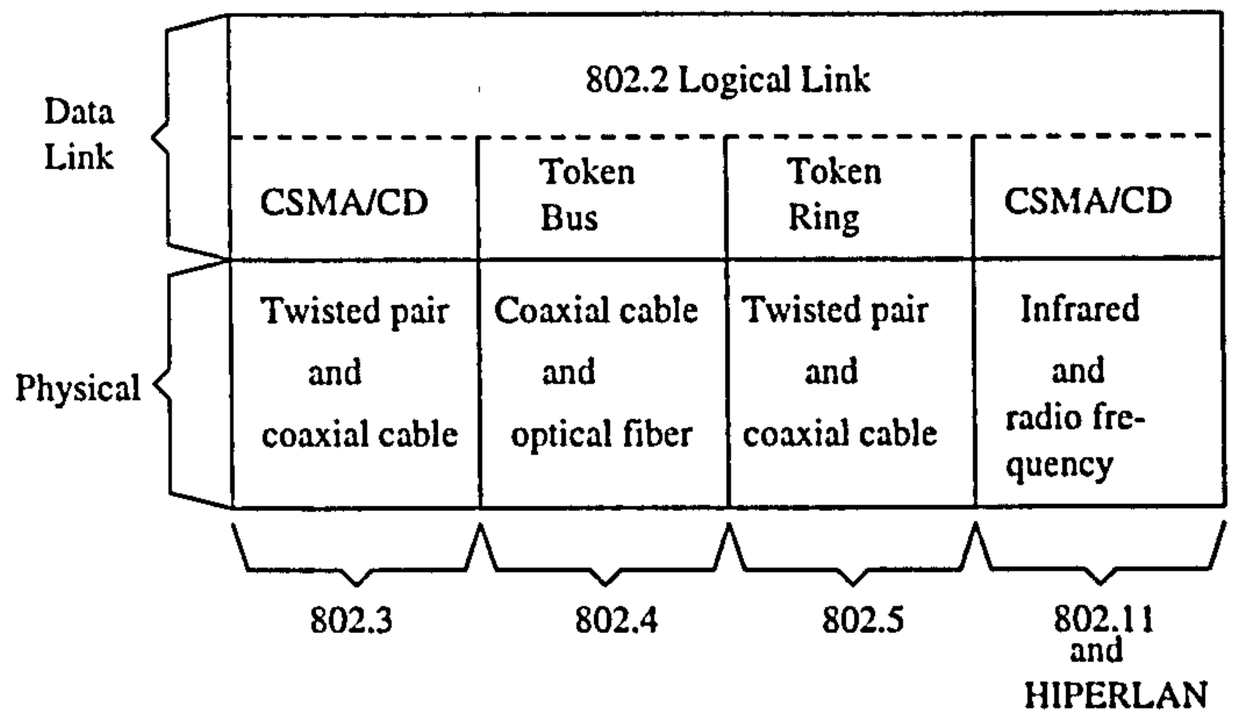
The UDP protocol and in particular its broadcast services are used by the MSS to advertise its public key to all PDA within the area of coverage of the MSS. The fact that the receiver of a datagram message does not need to know the identity of the sender is crucial here since at this stage of the PDA–MSS interaction the PDA is not yet in possession yet of its TmpId. Similarly, the connectionless nature of the UDP and the property that an UDP server can accept messages at a well-known port are used by the PDA to negotiate its first session key (see below) with the MSS.

Once the public key of the MSS is known by the PDA and a session key is agreed upon between

[1]Ebe comes from Ebenezer Scrooge, a mean character in *A Christmas Carol* by Charles Dickens. Mata-hari is probably the most famous female spy who worked as a double agent during the World War I. Tom comes from Tom Castro— the villain of *El Impostor Inverosímil* included in *Historia Universal de la Infamia* by Jorge Luis Borges.



a) Location of WLAN standards in the 7-layer architecture



b) The 802.11 and HIPERLAN WLAN standards compared to the 802.3, 802.4 and 802.5 international LAN standards.

Figure 6.1: Wireless LAN standard protocols.

the former and the latter, the services of the TCP protocol are used to establish a connection-oriented communication between the PDA and the MSS.

It is worth noting that our protocol is independent of the TCP/IP protocol; it can be built on top of any other protocol that offers similar services; we focus on the TCP/IP protocols because so far they are the most widely used.

Bob's PDA communicates over a wireless communication interface, hence to communicate with other computers (both wired and wireless) connected to the Internet, it needs the support of a mobile support station [46] (see sections 2.7.1 and 5.7).

The MSS serves as a bridge between the PDA and the rest of the Internet and forms together with the PDA, a wireless LAN on which TCP/IP protocols are mounted. The bottom layers (Data Link and Physical) of the wireless LAN can be any international standard for wireless LANs, for example, the 802.11 of the IEEE or the HIPERLAND of the ETSI. Both the 802.11 and the HIPERLAND protocol are currently under the study of international standarization bodies, with the intention of adopting them as international standards.

For more information about these protocols refer to section 2.8.7 and references [105, 56, 21, 55, 106, 160, 161]. The place of the wireless standard within the ISO protocol stack is illustrated in figure 6.1.

A MSS serves as the current home for the mobile host and is responsible for two jobs. It receives messages coming from Bob's PDA and routes them to their destination. On the other

hand, assuming that the PDA is switched on, it conveys to Bob's PDA, messages addressed to it. In this case, by messages, we have in mind both control messages exchanged between the communicating computers and e–mail messages exchanged between Bob and Alice.

It is obvious that the problem can be divided into two subproblems or protocols and make it easier to attack. The first protocol has to do with the communication between Bob's PDA and the MSS and the second with the communication between the MSS and Alice's computer. The first part of the problem is the hardest one and the one we will concentrate on. Once Bob's e–mail messages are in the MSS, they are, after undergoing some local processing, delivered to its final destination by means of standard e–mail protocols. Thus, the second part is mainly a problem of routing over the Internet, a topic which has been extensively studied (refer to [154, 61] for example) and not to be addressed in this work.

In this protocol we assume that Alice has a desktop e-mail address (*Alice@ncl.ac.uk* for example) in a desktop computer permanently connected to the Internet. Alice sends and receives e–mail messages in the traditional way, thus, we consider that the receiving peer of the protocol does not need further discussion.

As an aside comment, it is worth mentioning that, apart from having a PDA, Bob can have a desktop e-mail address as well (*Bob@poli.mx* for example) and use it to send and receive non–anonymous e–mail messages. Anybody interested in e–mailing Bob, sends their message to *Bob@poli.mx*, the message travels to Bob's mailbox computer where it is stored for later retrieval or automatically forwarded to Bob's current MSS. While at home or in office, Bob retrieves his messages with the help of his personal workstation, but when he is outdoors he uses his PDA.

6.4 Protocol vocabulary

The protocol vocabulary defines what messages are used to implement the protocol. It is important to mention that at this level of abstraction the focus is on the semantic of the messages rather than on their precise syntax. The inclusion of tiny details here would make the model unnecessarily more complicated and would blur the semantics of the protocol, i.e. the main target at this stage.

6.4.1 Basic components of the protocol

According to the functions performed by the PDA and the MSS, and the messages exchanged between them, the whole protocol can be represented by a set of processes linked by communication channels as illustrated in figure 6.2.

In the figure there is a set of n PDAs currently located within the area of coverage of the MSS and interested in sending anonymous messages. For simplicity at this stage we represent each PDA by a single process, later on (see section 6.4.2) we will see that there are more process involved in each PDA.

There are two permanent processes at the MSS, namely, the *KsTmpIdMan* and the KpuMan, which run permanently in the MSS regardless of whether there is a PDA in the area of coverage of the MSS or not.

The heart of the protocol is the *KsTmpIdMan* process. As its name implies its main task is to take care of the session keys used to encrypt messages exchanged between the PDAs and the MSS. Also, it is in charge of the TmpId assigned to the PDAs. It guarantees that both the session keys and the TmpId are correct and unique. When the *KsTmpIdMan* process is initiated, it comes with well–known bidirectional channels which are used during the negotiation of session keys.

Together with the *KsTmpIdMan* process works the *KpuMan*. The latter is in charge of periodically broadcasting the public key of the MSS to the air (through a well–known broadcast channel)

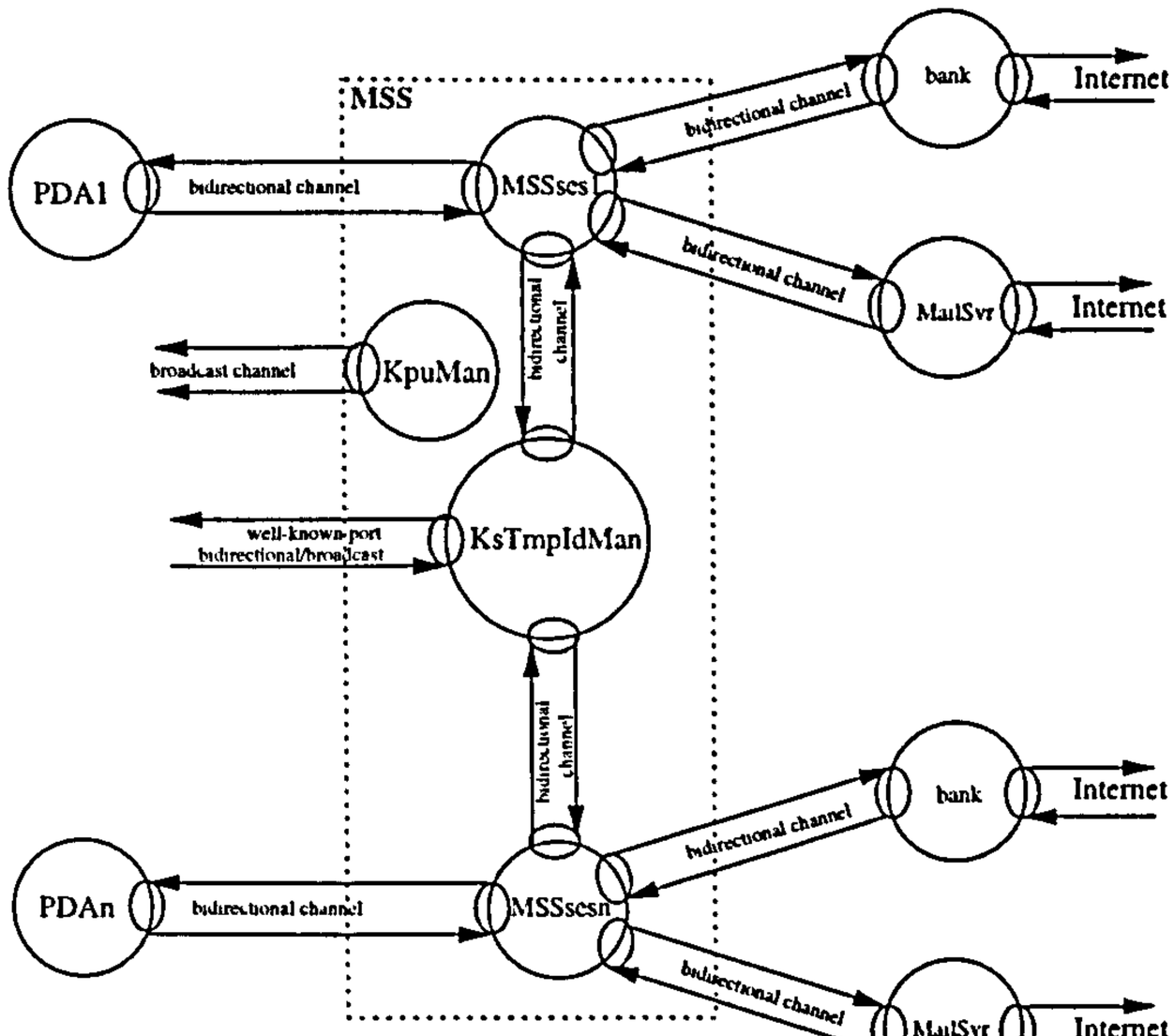


Figure 6.2: Software representation of a MSS serving a set of PDAs.

so that PDAs interested in, can read it.

At a given moment there can be any number of PDAs from none to n. The value of n is limited by the resources (memory, bandwidth, etc.) available in the MSS and depends on the specific hardware where the system is implemented. When a PDA arrives at the MSS it creates its *PDAj* process which will be in charge for communication with the MSS.

Once a PDA opens an anonymous session in the MSS a *MSSsesj* process is created for it. The job of the *MSSsesj* process is to take care of the communication with its corresponding *PDAj* counterpart once a session and a TmpId is assigned to *PDAj*.

A *PDAj* and its corresponding *MSSsesj* process are linked by bidirectional communication channels; so are the *MSSses* and the *KsTmpIdMan* processes.

To do their jobs each *MSSsesj* process is linked by bidirectional communication channels to the *bank* and *MailSvr* processes. The *bank* process is contacted by the *AnoSes* processes through the bidirectional channel that links them any time the latter wants to verify that an e–coin it has received as a payment is genuine. As can be seen from the figure, the *bank* is not in the MSS, it can be in any computer connected to the Internet as long as it is reachable from the *MSSses* processes. The *MailSvr* process serves as a bridge between the *MSSses* process when it comes to exchanging e–mail messages between the MSS and other computers connected to the Internet. The *MailSvr* process is linked to each of the *MSSsesj* processes by a bidirectional channel and is located either inside the MSS or in another computer provided it is reachable from the *MSSses* processes.

6.4.2 Processes and messages

A more detailed representation of the system is shown in figure 6.3, As can be appreciated from the figure, the whole system in composed of nine processes. The PDAuser, PDAses and PDAtcp

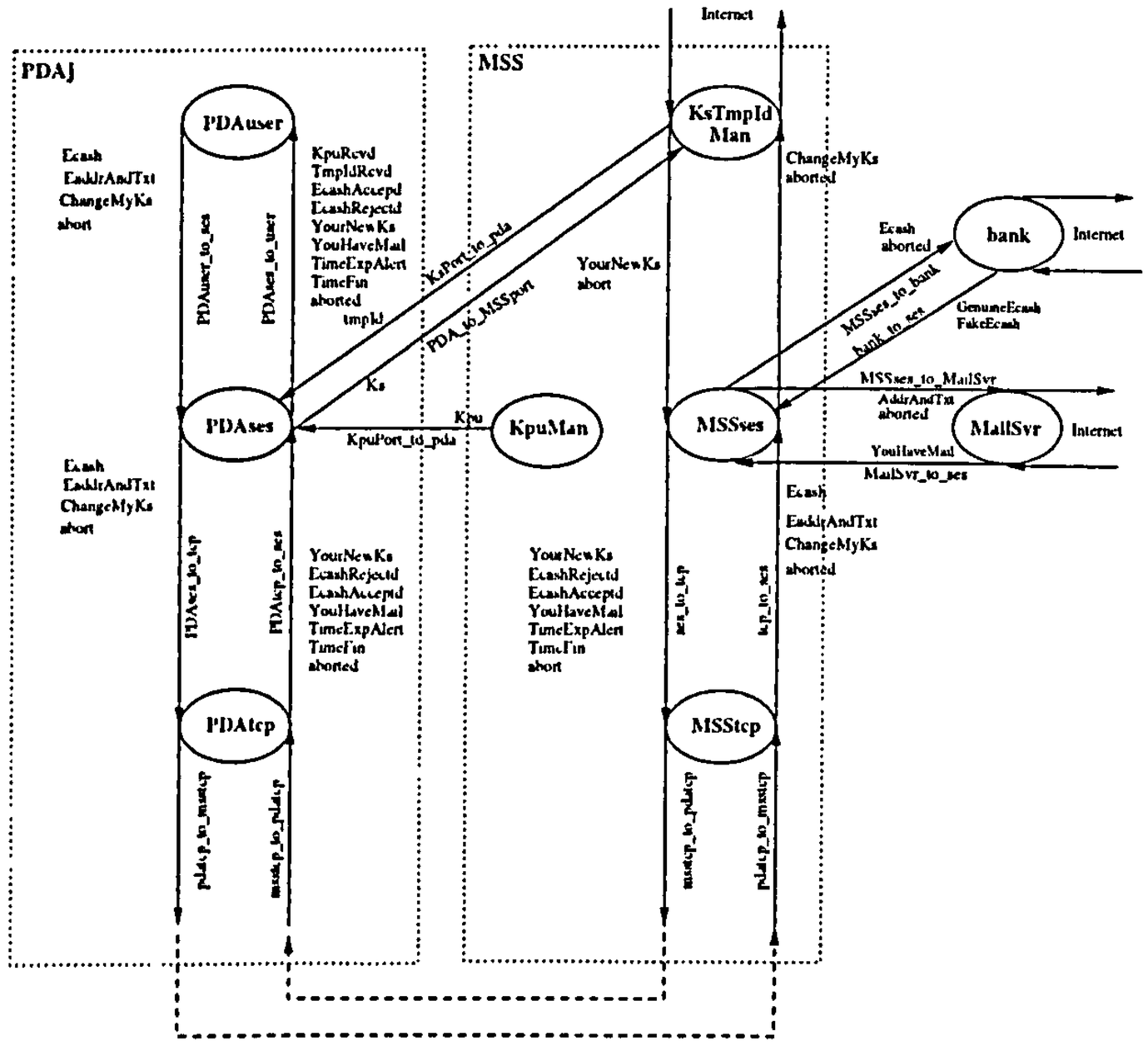


Figure 6.3: Protocol hierarchy.

form a layered hierarchy at the PDA side. So do the KsTmpIdMan, MSSses, and MSStcp at the MSS side. For this reason we use the words process and layer interchangeably. A brief description of each of the nine processes of the system and the meaning of the messages they send each other are presented next. An extended discussion of them is presented in section 6.6.

It is shown in the figure that Bob's PDA process is divided into the PDAuser, PDAses and PDAtcp layer which are defined as follows:

PDAuser The PDA user represents the interface between the PDA user and the MSS.

PDAses The PDA session is in charge of managing the anonymous communication session of the PDA user. It is there to hide all details of little interest to the user.

PDAtcp The PDA tcp layer is there to transmit messages from the PDA to the MSS and vice versa. It establishes a connection with its MSS peer and informs the PDAses layer if this connection fails i.e. when it discovers that its MSS peer is not reading a message sent to it.

On the other hand, the anonymous session in the MSS that serves Bob's PDA is divided into the MSSses and the MSStcp layer. The functions performed by these layers are defined as follows:

MSSses The main function of the MSS session layer is to manage the anonymous communication at the MSS side.

MSStcp The MSStcp layer is there to transmit message from the MSS to the PDA and vice versa. It establishes a connection with its peer (PDAtcp) and informs the MSSses layer if this connection fails i.e. if it discovers that its PDA peer is not reading messages sent to it.

It can be seen from the figure that the MSS session layer interacts with two processes, namely, with the *bank* and the *MailSrv*. The description of these processes is as follows:

bank This process runs in a bank WS where after several verification tests it is decided whether an e–coin sent by Bob to the MSS is genuine or a fake.

MailSrv The main function this process performs is to provide standard e–mail protocols to forward Bob's messages to Alice and to receive Alice's responses.

6.4.3 Messages

The layers of the system described in section 6.4 communicate with each other by means of sending and receiving messages. The protocol vocabulary is made up of 18 messages whose semantics are discussed next.

Kpu A message broadcast by the *KpuMan* process. It contains the public key of the MSS.

KpuRcvd A message sent from the PDAses to the PDAuser layer to indicate that the *Kpu* has been received.

Ks A message sent by the PDAses to the *KsTmpIdMan* process. It contains a suggested *Ks*. If *Ks* is accepted by the *KsTmpIdMan* process the latter sends a *TmpId* message and then the accepted *Ks* is used for the anonymous communication session between the PDA and the MSS.

TmpIdRcvd A message sent from the PDAses to the PDAuser layer to indicate that the *TmpId* has been received.

TmpId A message sent by the *MSSses* layer to the *PDAses* to indicate that the suggested session key has been accepted.

Ecash A message sent by the *PDAuser* to the *PDAses* and from there to the PDAtcp until it reaches the *MSSses* layer. The contents of this message is irrelevant to the functionality of the protocol, thus, its is not reflected in the validation model. However, if the protocol is implementated this message contains a certain amount of electronic cash to pay for a new anonymous session or to extend an expiring one. Upon receiving this message, the *MSSses* forwards the e–coin found in it to the *bank*.

GenuineEcash A message sent by the *bank* to the *MSSses* to indicate that the e–coin just received is genuine.

FakeEcash A message sent by the *bank* to the *MSSses* to indicate that the e–coin just received is a fake.

EcashAccepted A message originated at the *MSSses* layer and sent to the *PDAuser* (through the stack of protocols) to indicate that the e–cash the user had sent to the MSS has been accepted as a payment to open an anonymous session or to extend one in progress.

EcashRejected A message originated at the *MSSses* layer and sent to the *PDAuser* (through the stack of protocols) to indicate that the e–cash the user had sent to the MSS has been rejected.

EaddrAndTxt A message originated at the *PDAuser* and containing an e–mail address (Alice's for example) and a text body. It travels from the *PDAuser* through the stack protocol until it reaches the *MailSvr* process from where it is forwarded to its final destination.

YouHaveMail A message originated at the *MailSvr* as a result of receiving Alice's response to one of Bob's messages. This message is sent through the stack protocol until it eventually reaches the *PDAuser* process. Upon the arrival of this message, Bob is signalled so that he can open from his mail box and read it. Next, Bob has the choice of deleting or saving it. How, and when all of this this happens is not a matter of concern to the semantics of the communication protocol.

ChangeMyKs This message is originated at the PDAuser and travels through the stack protocol until it reaches the *KsTmpIdMan*. It indicates that the PDA user wishes to change his current session key, probably after suspecting that his current session key has been compromised.

YourNewKs A message sent by the *KsTmpIdMan* to the *PDAuser* as a response to a *ChangeMyKs* message. It contains a new Ks for the PDA user.

TimeExpAlert This message is originated at the *MSSses* when N units of time before the anonymous session time the PDA user has paid for expires. It travels through the protocol stack till eventually it reaches the PDAuser. It is intended to be an invitation for the user to send more e–cash to the MSS in order to extend his anonymous communication time.

TimeFin The *TimeFin* message is originated at the *MSSses* and sent through the protocol stack until eventually it reaches the *PDAuser*. It is meant to interrupt the communication session immediately after the session time the PDA user has paid for expires. It is an informative message since by the time it reaches the PDA screen, everything related to the anonymous session, both in the MSS and the PDA, is being cleared up.

abort This message can be originated at any of thr processes in the system whenever there is a need to interrupt the anonymous communication session. It is always sent down the protocol stack till it reaches the corresponding tcp layer. From there, it is forwarded to the remote peer; at the same time the message *aborted* is propagated up the protocol stacks at both the PDA and MSS side.

aborted A message originated at the *PDAtcp* and *MSStcp* layer. The layer that receives it, forwards it up the protocol stack and terminates. It causes a cascade abort starting at the lowest layer.

As can be appreciated from the figure, communication between neighbouring layers is achieved by sending/receiving messages through channels; in this way messages sent by the *PDAuser* layer to the *PDAses* layer travel through the *PDAuser_to_ses* channel and messages travelling in the opposite direction go through the *PDAses_to_user* channel. Communication between other layers takes place in a similar way, and is self explanatory by reading the name of the channels.

6.5 Format of messages used

For the purpose of the Promela specification of our system it is enough to represent a message with two fields: *format*= {*control tag, data*} where the first field represents the type of message and the second one the data being transmitted. In a structure–like form, the messages we use look as follows:

```
struct{
 unsigned char type;                /* type of message */
 unsigned char data[DATALENGTH];    /* data            */
}message;
```

Note that to implement a flow control discipline (flow layer) it would be necessary to include two additional fields. One to determine the sequence number of the message and another to implement checksum field to detect possible transmission errors.

6.6 Procedure rules

A widely used and well–known model for protocol specification is the finite state machine [61, 62, 159]. Since its introduction in the early 1950s it has been used for modeling a great number of systems. Its analytical power and the ease with which the model can be loaded into a computer and manipulated automatically with the help of software tools makes this method attractive. Similarly, the graphical nature of this model makes it easy to read and understand the different stages that the protocol goes through during its execution. Unfortunately, this descriptive clarity holds true only for systems with a small number of states. For systems with more than 20 to 30 states and a similar number of transitions, the graphical representation on paper or a computer screen becomes difficult to follow and understand.

For systems that exceed this limit, it is probably a good alternative to describe them in a formal or semiformal verification modeling language or to complement the finite state diagram representation with such code.

While the conversion of a finite state machine into an implementation program is not straightforward, a verification modeling program is. Because of this, it is usually helpful to have more than one representation of a system. On this account, we will present our system in both: its finite state machine and its verification modeling program. We have chosen *Promela* language [159] to represent our system and it will be introduced in section 6.6.2.

6.6.1 Finite state machine

As the finite state machine model—also known as state transition diagrams— is a familiar method, we will not discuss it in depth, but a few words concerning our particular protocol are worth mentioning.

In the finite state machine, a state is represented by a circle or an oval, and a transition by a marked arc. Transition between states take place as a result of an incoming event; for example, when a message is sent, when a message arrives, when a timer goes off, or when an interruption occurs or the user presses a key on her keyboard. A label on the arc represents the name of the event that triggered the finite state machine. In our case most of the incoming events have to do with sending and receiving messages and are labeled by the marks *!* and *?* respectively. In addition we also have events that have nothing to do with sending and receiving messages but with certain local conditions. For examples, the waiting time for something to happens has expired, the user has pressed a key, the results of a computation are ready for use, etc.

Because of their graphic nature, state transition diagrams are easy to read. On the other hand the mathematical theory behind them makes them useful for showing the correct operation of a protocol. However, due to space limitations and the two-dimensional nature of paper, it is not always easy and practicable to show all possible incoming event possibilities including error conditions, local state variables and predicates associated with a protocol. A possible way of simplifying state transition diagrams is by not representing transient states, i.e those that lead to the main ones.

Hence the state transition diagrams we present here, are incomplete specifications. They are meant to help the reader understand the protocol and make the reading of the Promela code easier.

6.6.2 A brief introduction to Promela

Even for relative simple communication system it is certainly difficult to design a correct protocol and even harder is the task of validating the correctness its procedure rules. Because of this the use of verification languages to write the procedures rules and a software tool to verify the correctness of the resulting code, called the *validation model*, is highly recommended at this stage of the development.

So far the most successful software tool used to trace logical design errors in distributed systems and in particular in communication protocols is Spin (Simple Promela INterpreter). Spin is a generic verification system that accepts design specifications written in the verification modeling language called Promela (PROcess MEta LAnguage) [162]. We will discuss here this modeling language and leave the discussion of Spin until section 7.

Promela is Spin's input language and provides a vehicle for making abstractions of protocols so that details that are unrelated to the communication processes are suppressed. A Promela program consists of processes, message channels and variables. The state of the whole system depends on the state of these three components.

Motivated by Promela's power to describe process interactions, we have decided to describe our procedure rules in Promela. Although Promela is a simple language with a C–like syntax, a complete Promela code of a middle (more then 50 lines) or large size program is not easy to follow. To help the reader not to get lost in dozens of Promela lines and to focus his attention on crucial aspects of the protocol only, we use a sort of Pseudo–Promela code to describe our procedure rules here and leave the full Promela code description until section 7. Complete Promela code can triple the size of the pseudocode; hence it is significantly harder to read; moreover, the numerous and tiny details that it includes do not change the main idea of the algorithm in a fundamental way; therefore, we restrict the discussion in this section mainly to the basic ideas of the algorithm.

Before going further, it is worth mentioning that a validation model is a piece of code that describes the procedure rules, i.e. the interaction between processes. Having the code and a simulator to execute it, the verification of the completeness of the protocol and its logical consistency (free from deadlocks for example) is straightforward and furthermore, the implementation of the system follows from converting the Promela code to a high–level one, C or C++ for example. The difference between a Promela version of the protocol and the final high–level language implementation is that the former deliberately abstracts from issues of protocol design, such as message format, neither does it say how a message is to be transmitted, encoded, decoded, stored, etc. Moreover, it does not deal with details irrelevant to processes' interaction such as encryption and decryption of messages and implementation of timers.

The syntax of Promela is described by Holzmann in the appendix C of his book [159], however, to help the reader to understand our Promela code we introduce the basic Promela statements and their semantics here. Promela is a non–deterministic language, it is easy to realize that the semantics of Promela statements comes from Dijkstra's guarded commands introduced in his famous paper [163] and discussed further by Hoare in his classic article [164].

executability In Promela the execution of a statement is conditional on its executability, i.e. at a given moment of time a statement is either executable or blocked depending on the state of a variable or channel. Executability is the basic mean of synchronization; hence, as shown below in *send/receive* statements, input and output through a channel allows the communication between two processes and synchronization as well. For example, the statement

```
if (a == b) a= a + 1 fi
```

either increments the value of a or blocks until the condition (a==b) holds.

send The syntax of the send statement is

```
channel ! msg
```

where channel is the name of a channel and msg is a message.

receive The syntax of the receive statement is

```
channel ? msg
```

where channel is the name of a channel and msg is a message.

separators -> and ; are separators.

skip skip is a null statement. It is always executable and its execution has no effect. It is normally used to satisfy syntax requirements.

goto The goto statement works as the infamous goto of high level languages, it transfers control to any labeled statement. Like the skip statement, goto is always executable. As Promela pays no attention to the problem of programming techniques it lacks most of the constructs for writing a well–structured code, as a result goto is intensively used.

if–fi selection A selection statement begins with if and ends with the keyboard fi and contains a list of one or more options. Every option begins with the flag :: followed by a boolean expression (a *guard*). An option can be executed only if its guard is executable. Only one option from the list is executed. If more than one guard is executable, one of them is selected at random the corresponding option is executed. If all guards are unexecutable, the process blocks until at least one of them becomes executable. In the following example the variable counter is either incremented or decremented depending on the value of a and b

```
if
:: (a == b) -> counter= counter + 1
:: (a != b) -> counter= counter - 1
fi
```

do–od repetition This statements works in a similar way as the if--fi one, but it is repeated until a break statement is encountered or an unconditional goto jump is performed. In the example shown the program loops until either the variable counter is decremented to zero or an error occurs.

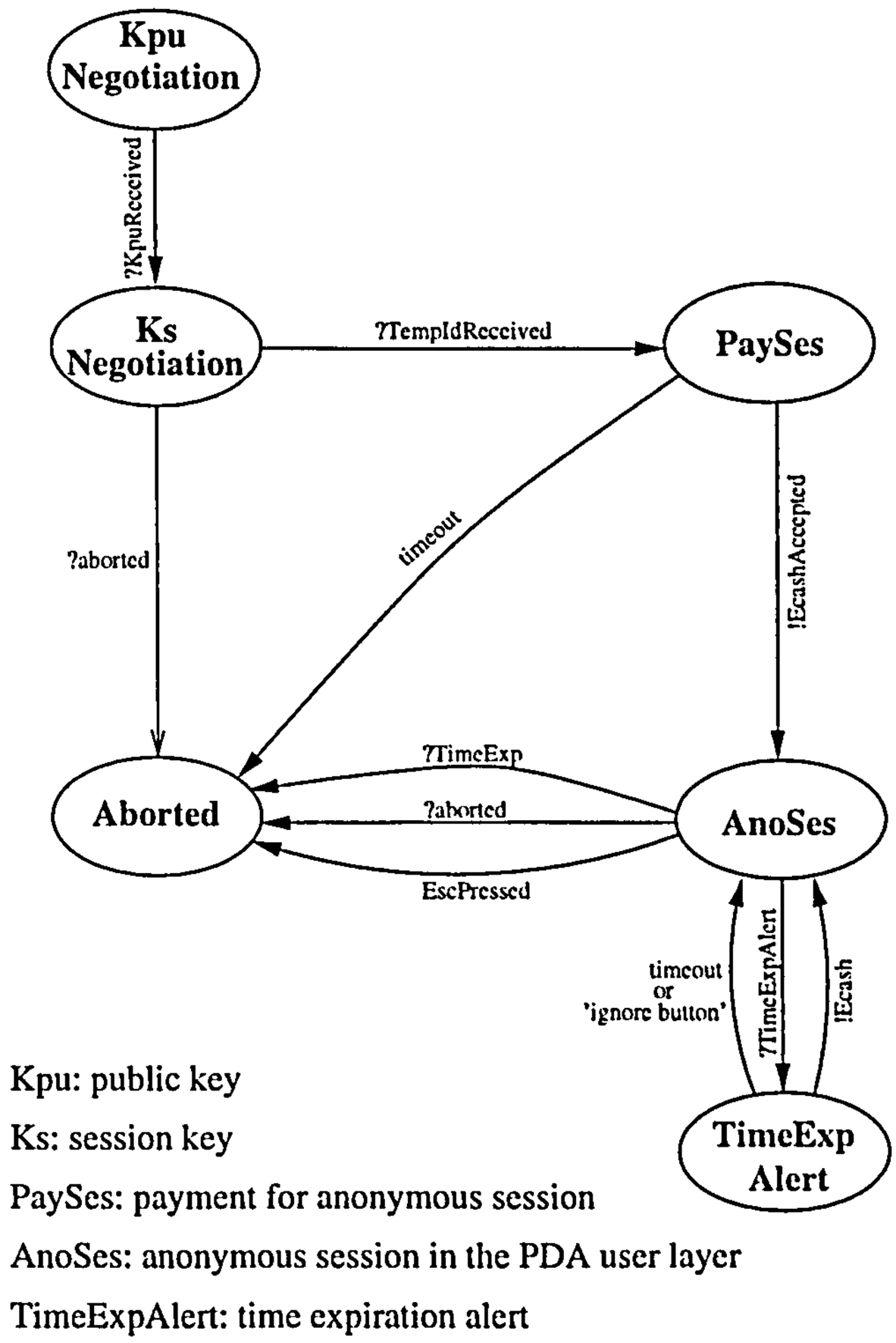
```
do
:: (counter  <  0) -> goto Error
:: (counter  == 0) -> break
:: (counter  >  0) -> counter= counter -1
od
```

timeout This statement represents a condition that becomes true is and only if no other statement in the block of commands is executable. Depending on the timer value, timeout becomes true sooner or later.

Process synchronization

Promela supports synchronous and asynchronous communication between processes. Synchronous communication is achieved by conditioning the send and receive events to the states of the output and input channels. If an input channel is empty, no input messages are available from that queue and the receiving process cannot move to its next state. On the other hand, if an output channel is full, no messages can be sent to that channel and the sender process remains in the same state. In synchronous communication, the move of a process to a new state is conditioned to the state of its communicating peer. To make a move to a new state, both the sender's output queue and the receiver's input queue have to be selected simultaneously. Needless to say, the sender selects the output queue and the receiver the input one. Neither the sender nor the receiver can move to their new states until this match occurs.

In both synchronous and asynchronous communication the message is removed from the input channel by the receiver. Also a message can be read from an input channel for one process only. The argument for this is that most distributed systems can be modelled on this model. Unfortunately this restriction makes group communication (multicast and broadcast) more complicated to model.



Kpu: public key

Ks: session key

PaySes: payment for anonymous session

AnoSes: anonymous session in the PDA user layer

TimeExpAlert: time expiration alert

Figure 6.4: The PDA user layer.

6.6.3 The user layer

The user layer is the interface between the PDA user and the anonymizing system; its work consists in accepting commands from the user and transmitting them to the layer below it, i.e. the presentation layer. In addition, it receives response messages from the presentation layer and displays them on the PDA screen. In fact the user layer can be regarded as an application which runs in the PDA and uses the services offered by the session layer.

Finite state diagram

In accordance with the algorithm presented in section 5.10.3, the first thing that the PDA does upon being switched on is to learn the MSS public key (Kpu). Since the value and the nature of this key is irrelevant to the user, this task is carried out by the session layer. The user layer just waits until the session layer informs it that the Kpu has been learnt.

As illustrated in figure 6.4, when the PDA is activated it enters the *KpuNegotiation* state where it waits for the message *KpuReceived* from the session layer. Such a message indicates that the public key of the MSS has been successfully learnt, hence the finite state diagram can move to the *KsNegotiation* state to negotiate a session key (Ks). During the negotiation of the Ks (see section 6.6.4), the PDA uses the public key of the MSS to encrypt messages sent to the MSS, which contain

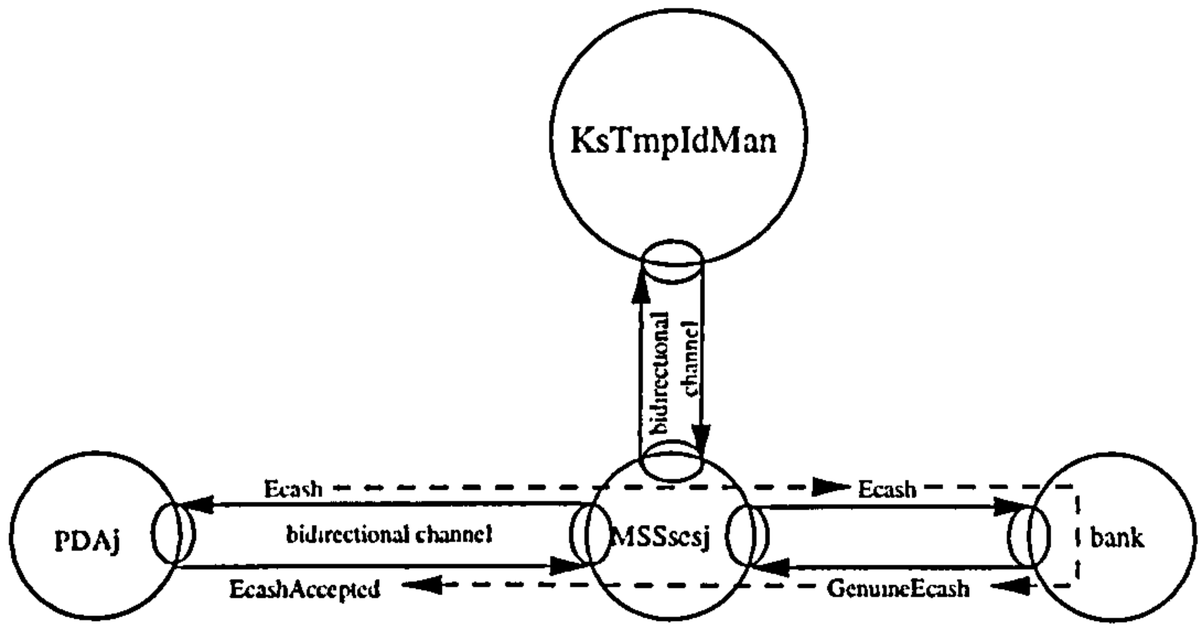


Figure 6.5: Payment for an anonymous call.

candidates for session keys.

The negotiation of the session key is another task irrelevant to the user, therefore, this task is included in the session layer as well. When the session layer has a session key approved by the MSS it sends a *TmpIdReceived* message to the user layer. This event, as depicted in the state transition diagram, transfers the finite state machine from state *KsNegotation* to the *PaySes* one. If, for any reason, the session layer cannot negotiate a Ks key, it sends an *abort* message to the user layer and the finite state diagram moves to the *Aborted* state.

The *PaySes* state is where the PDA user performs his first payment for the anonymous session which credits the PDA user with a certain amount of time of anonymous communication. In order to do that, the PDA user selects an e-note blindly signed by Clare from his PDA memory and sends it to the MSS.

In both the finite state machine and the Promela code, we assume that the PDA user has an endless supply of e-cash in his PDA memory to pay for the anonymous communication session with the MSS. This assumption is grounded on the fact that there is no price to pay for keeping e-cash in the memory of the PDA. Furthermore, unlike coins, e-coins are weightless and do not represent any physical burden to carry.

The e-cash sent by Bob can be accepted or refused as a payment depending on whether the bank declares it genuine or fake. If the e-coin is declared genuine by the bank, it can still be refused by the MSS if its value does not fall within the interval limited by the minimum and maximum amount of money accepted by the MSS as a payment for call. This is why the MSS has to send Bob's e-cash to the bank before accepting it. The path Bob's e-coin follows before it is accepted as a payment by the MSS is shown in figure 6.5. Indepth discussion about the validation of the e-coin and the *bank* is presented later on in section 6.6.7.

If the e-cash sent to the MSS is accepted, the MSS opens an anonymous session for the PDA and replies with a *EcashAccepted* message and the finite state diagram moves to the *AnoSes* state. Otherwise, it replies with a *EcashRejected* message (not shown in the figure). When this message is displayed on the PDA screen the PDA user might wish to select a different e-coin and try again. To avoid the risk of being locked in this state forever, a timeout mechanism is activated after N number of attempts and the finite state diagram moves to the *Aborted* state where everything related to the current session key is cleared up. When everything is cleared up the user can start a new anonymous session by negotiating a new Ks.

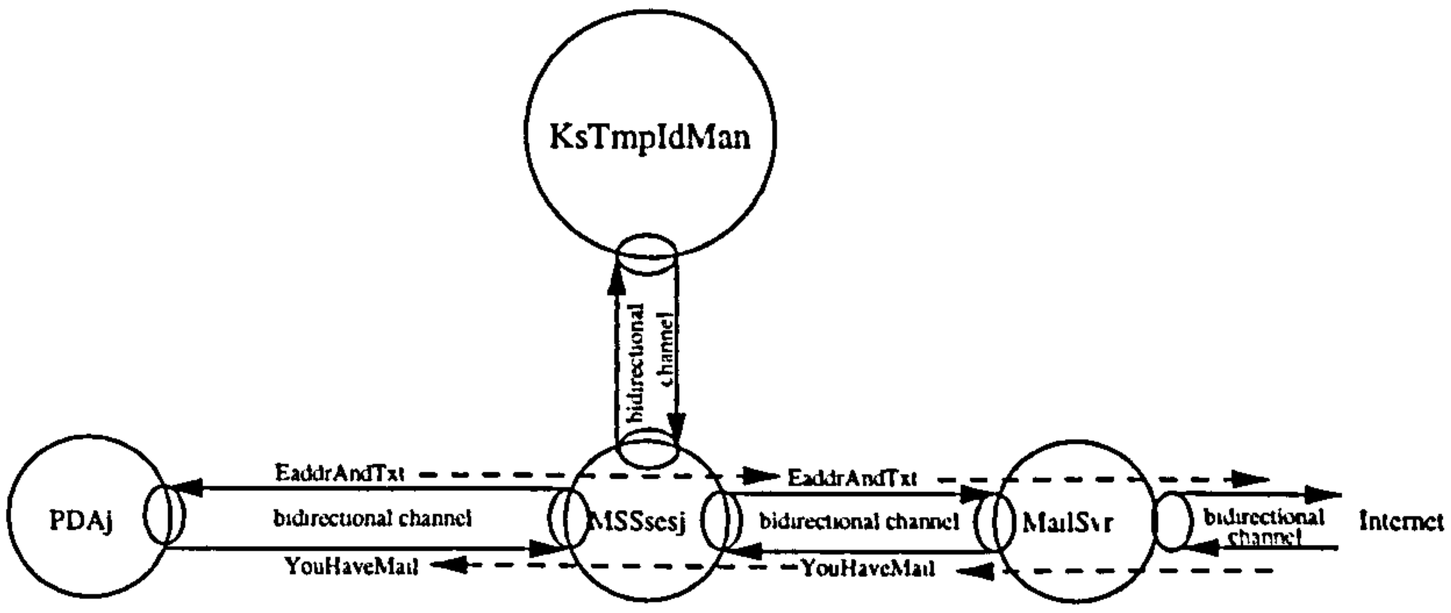


Figure 6.6: Bob's e–mail message to Alice and Alice's reply.

Once the anonymous session is opened with the PDA, the PDA user can send as many anonymous messages as he wants simply by passing down the protocol stack the message *EaddrAndTxt* which contains the recipient's address and the body of the e-mail message. The user is notified of incoming e–mail messages by the message *YouHaveMail* which appears on his PDA screen. The path Bob's e-mail messages and Alice's replies follow before reaching their destination is shown in figure 6.6

After a certain amount of time in an anonymous session two things can happen. Either the user decides to leave his anonymous session or his prepaid time expires. In the first case the PDA user presses his abort key (*EscPressed*); this event takes the finite state diagram to the *Aborted* state. In the second case it receives a warning message from the PDA (*TimeExpAlert*). Upon receiving this message it can ignore it or send more e–cash (*Ecash*) to the MSS to extend the anonymous session. If no more e–cash is sent, the finite state diagram returns to the *AnoSes* state to continue its work until the prepaid time finish and the message *TimeFin* is received which indicates that the prepaid time is over and takes the finite state diagram to the *Aborted* state.

The Promela code of the user layer shows details deliberately omitted from the finite state diagram.

```
/** The user process: it is the anonymous e-mail application running on PDA **/
/**                                                                          **/
/** Communication channels connected to this layer:                          **/
/**                                                                          **/
/**                  ---- PDAuser_to_ses --->                                **/
/**            PDAuser                          PDAses                       **/
/**                  <--- PDAses_to_user ---                                 **/
/**                                                                          **/
proctype PDAuser
{
KpuNegotiation: /* wait till the public key of the MSS is learnt */
 PDAses_to_user ? KpuReceived(Kpu) -> goto KsNegotiation;

KsNegotiation:  /* negotiate a session key */
 Ks= NewKs_Fetched;
 /* encrypt Ks with Kpu */
 do
 :: PDA_to_KsPort ! Ks ->
    if
    :: PDAses_to_user ? TmpIdReceived ->
```

```
                        /* the last send Ks is the session key */
                        goto FirstPayment;
    :: timeout -> Ks= NewKs_Fetched
    if
 od;


FirstPayment: /* first payment for the anonymous session */
/* all messages sent and received to and from the session layer */
 * are encrypted/decrypted with Ks                                */
 do
 :: (i <= MaxNumAttempts) -> /* try MaxNumAttempts times */
      PDAuser_to_ses ! Ecash -> /* e-money to pay for an anonymous session */
      if
      :: PDAses_to_user ? EcashRejected ->
                          Ecash= NewEcash_Fetched /* fetch a different e--coin and try again */
      :: PDAses_to_user ? EcashAccepted -> goto AnoSes
      if
      i++
 :: (i > MaxNumAttempts) -> goto Aborted
 do;


AnoSes:
  do
  :: PDAuser_to_ses ! EaddrAndTxt /* send as many msg as you can before */
                     /* the msg TimeFin arrives */
:: PDAses_to_user ? YouHaveMail
                     /* display the msg on the screen */
:: PDAuser_to_ses ! ChangeMyKs -> PDAses_to_user ? YourNewKs;
                     Ks= YourNewKs; /* update Ks */
:: PDAses_to_user ? TimeAlert->
      if
      :: /* the user wishes to extend his session */
           do
           :: (i =< N); /* try N times to send payment */
              PDAuser_to_ses ! Ecash ->
              if
              :: PDAses_to_user? EcashAccepd -> break
              :: PDAses_to_user ? EcashRejetd ->
                   Ecash= NewEcash_Fetched; /* fetch a different e--coin and try again */
                   i++
              fi
           :: else -> break
           od
        :: skip /* the user doesn't wish to extend his session */
        fi

  ::  PDAses_to_user ? TimeFin -> goto Aborted
  :: PDAses_to_user ? aborted -> goto Aborted
  od;

Aborted:
 /* session has been aborted */
}
```

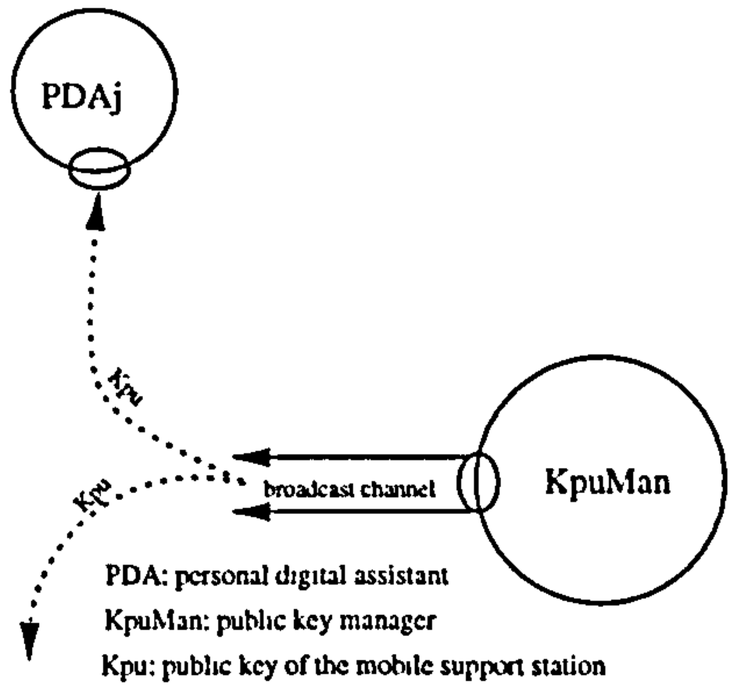


Figure 6.7: The MSS broadcasts its public key.

6.6.4 The PDA session layer

The main function of the PDA session layer is to open and manage the anonymous communication session of the PDA user and to hide all the details irrelevant to the lay PDA user, such as learning the public key of the MSS and the negotiation of the session key and its renewal whenever it is necessary.

The process of obtaining the public key of the MSS is an independent problem easily separated from the main protocol; thus, we present it individually, with the help of figure 6.8 and its corresponding Promela code.

Broadcast the public key of the MSS

To communicate securely with the MSS the PDA needs the MSS public key. Because of this, after being activated by the PDAuser layer, the first job of the PDA session layer is to contact the MSS and learn its public key. The PDA listens to the air until a message containing a *Kpu* is received. This message is broadcast through a broadcast channel that the PDA is listening to as illustrated in figure 6.7.

The Promela code to simulate the *KpuMan* process is presented next. For simplicity, in our simulation we assume that the MSS has only one public key, nevertheless nothing prevents the MSS from having more than one public key and broadcasting them to the air in a polling scheme so that if a PDA fails to verify the authenticity of one of them, it can try the next one.

```
/**
/** The KpuMam process: every t units of time a message that contains the public key  **/
/** of the MSS is broadcast to the air through a well--known port.                    **/
/**                                                                                   **
/**          PDAses                                  KpuMan                           **/
/**                    <--- KpuPort_to_ses ---                                        **/
/**                                                                                   **/
proctype KpuMan
{
 do
 :: (true) ->
      Kpu_val= NewKpu           /* fetch one of your Kpu */
      KpuPort_to_ses ! Kpu_val /* send Kpu to air */
 od
}
```

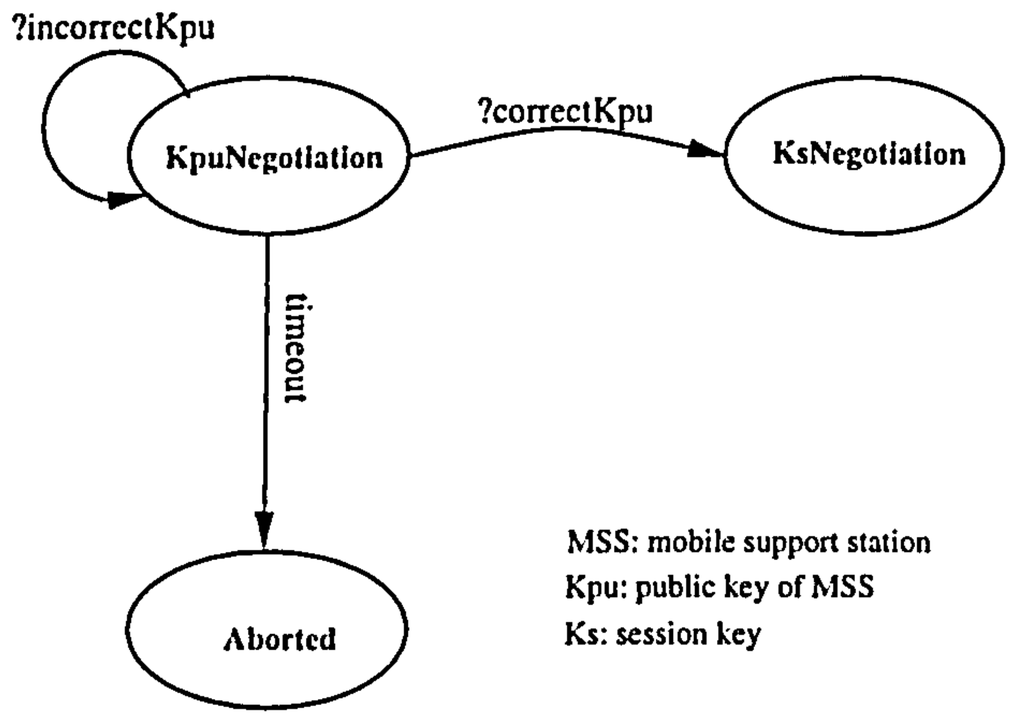


Figure 6.8: Getting the public key of the MSS.

Learning the public key of the MSS

The finite state diagram shown in figure 6.8 shows how the Kpu is learnt by the PDA. The diagram starts when the PDA is within the area of coverage of the MSS. In this state, called *KpuNegotiation*, the PDA reads messages broadcast by the MSS through a well–known port. Upon receiving a message the PDA proceeds to verify the authenticity of the public key of the MSS by checking the signature of the person or institution who vouches for the key. How the authenticity of a public key can be guaranteed is explained in section 5.10.1. If no messages are received within a certain amount of time the diagram moves to the *Aborted* state. This event is an indication for the user to locate a new MSS and start again.

Similarly, the diagram moves to the *Aborted* state if no correct *Kpu* is received after N attempts. If the *Aborted* state is reached the PDA has to try a different MSS and start again from the *KpuNegotiation* state. On the contrary, the diagram moves to the *KsNegotiation* state when a correct *Kpu* is received.

In the finite state machine and Promela code we suggest that the MSS broadcasts its public key periodically. This is only one possible approach, other alternatives are possible, for example the MSS may broadcast its public key only when it detects the presence of a new PDA within its cell. The crucial issue here is that the PDA has to get the public key from the MSS as soon as it enters the MSS cell and before it sends any message that might reveal the identity of its user, to the MSS.

It is conceivable that a PDA may receive *Kpu* messages from more than one MSS at the same time. This is usually the case in urbanised areas where adjacent cells overlap. In case of overlapping, the PDA selects the MSS with the strongest signal.

The Promela code for describing the finite machine of figure 6.8 is presented below.

```
/** GETTING the PUBLIC KEY OF THE MSS: the PDA waits from messages to come          **/
/** from the nearest MSS, containing the public key of the latter. It checks for the **/
/** authenticity of the key until its test is successful or a timeout occurs. In the **/
/** latter case it moves to another MSS and starts the process again.                **/
/** In case of failure the PDA tries one MSS after another until it is turned off by **/
/** its user.                                                                        **/
/**                                                                                  **/
/** Communication channels connected to this layer:                                  **/
/**                                                                                  **/
/**          PDAsos                                KpuMan                            **/
/**                    <--- KpuPort_to_sos ---                                       **/
```

```
/**                                                                    **/
/**                                                                    **/
proctype PDAses
{
KpuNegotiation:
 do
 :: (i <= MAXNUM_ATTEMPTS) ->
    if
    :: MSSbcast_to_PDA ? Kpu -> /* Kpu rcvd: now check for authenticity */
       if
       :: (PuKey == CORRECT) -> goto KsNegotiation
       :: else -> i++
       fi
    :: MSSbcast_to_PDA ? bogus -> i++ /* bogus msg rcvd: ignore it */
    :: timeout -> goto Aborted /* couldn't hear any msg from this MSS */
    fi
 :: else -> goto Aborted /* couldn't get a correct Kpu */
 od;

 KsNegotiation:
   ...
   /* code to negotiate Ks here */


 Aborted:
  ...
  /* clean up */
}
```

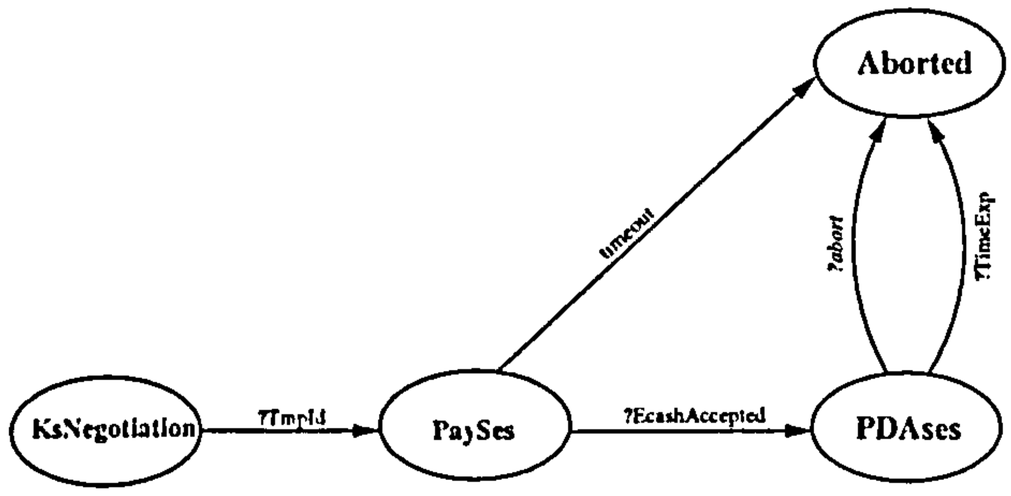


Figure 6.9: The PDA session layer.

The main protocol

The finite state diagram that describes the session layer of the PDA is shown in figure 6.9. To keep the diagram readable, non-relevant details such as temporary states are not shown in the figure. However, they can be found in the Promela code presented at the end of this section.

Upon obtaining a correct public key from the MSS, the session layer moves to the *KsNegotiation* state where it negotiates a session key with the MSS. Let us assume that PDAj is trying to open an anonymous session.

PDAj creates a Ks, encrypts it using the Kpu, sends it to the MSS for approval and waits while listening to the air (a well-known port). The MSS checks that the Ks suggested by PDAj is correct and not in use. If so, it creates a TmpId for PDAj, encrypts it using the Ks, and sends it to PDAj as a reply in a *TmpId* message. If the Ks suggested by PDAj is incorrect, the MSS does not reply. If it is correct but has been assigned to an existing PDA, say PDAi, the MSS does not reply to PDAj and additionally asks PDAi to change its Ks. After t units of silence, PDAj can try again. Once a suggested Ks is approved, it is used to encrypt and decrypt messages exchanged between PDAj and the MSS until either the end of the session or until the Ks has to be renewed. Messages encrypted with Ks can be overheard by other PDAs but they will be ignored as only PDAj can decrypt and make sense of them. Having negotiated the Ks the finite state diagram moves to the *PaySes* state.

The procedure for performing the first payment for opening an anonymous session is started when the *PDAses* layer receives the *Ecash* message from the *PDAuser* layer. The *PDAses* layer just passes *Ecash* messages coming from the *PDAuser* layer to the *PDAtcp* layer until either the payment is successful or the intention aborted. In the former case, the finite state diagram moves to the *PDAses* state; this means that an anonymous communication session has been opened for the PDA and a certain amount of communication time is credited for the PDA user. In the later case, the finite state diagram moves to the *Aborted* state.

Once the anonymous session is opened, Bob has the right to send anonymous messages to Alice and to receive Alice's responses. E-mail messages addressed to Alice come from the *PDAuser* layer in the form of *EaddrAndTxt*, i.e in a message that contains Alice's address and the body of the message. Once this message is received, the *PDAses* layer encrypts it with the current session key and forwards it to the MSS through the *PDAtcp* layer. Conversely, when a message addressed to Bob (*YouHaveMail*) arrives through the *PDAtcp* layer, it is decrypted with the session key and forwarded up to the *PDAuser* layer.

Once a session key is accepted by the MSS it becomes the session key used by both the PDA

and the MSS to encrypt and decrypt messages till the session finishes or till Bob decides to change it for a new one or till the MSS orders the PDA to change it. If for some reason Bob suspects that his current session key has been compromised he can instruct his PDA to renew it. To do this the *PDAuser* layer passes down to the *PDAses* the *ChangeMyKs* message. This message eventually reaches the *KsTmpIdMan* who tries to find a new Ks for the PDA, if it does, it sends the message *YourNewKs* to the PDA, if for any reason (no more Ks available for example) the *KsTmpIdMan* cannot send the PDA a new Ks, the message *aborted* is sent instead. Also it might happen that the current session key of PDAj is hit by another PDA. The *KsTmpIdMan* responds to that by sending the *YourNewKs* message which contains a new Ks to PDAj.

It might happen that while being in an anonymous mail session, the message *TimeExpAlert* is received from the MSS to indicate that the prepaid anonymous time is just about to finish. This message is passed to the PDA user layer to indicate that Bob has to send more e–cash if he wishes to extend his anonymous session. Bob might either ignore the *TimeExpAlert* message or reply by passing some e–cash (*Ecash* message) to the session layer. As usual, the MSS responds either with a *EcashAccepted* or *EcashRejected* message. The former message indicates that the anonymous message has been extended in the MSS and the latter indicates that it has not. If Bob does not extend his anonymous session time, eventually his anonymous session will finish, this is indicated by the arrival of the *TimeFin* message from the MSS. Upon receiving this message the PDA session layer cleans up everything and goes to the *Aborted* state. Naturally, Bob can finish his anonymous session before any *TimeExp* message arrives, to do this he presses the *Esc* key on his keyboard; as a result of this action the *abort* message is propagated to all layers in the protocol and the finite state diagram moves to the *Aborted* state. The Promela code for the user layer is presented below.

```
/** PDA ses layer: negotiates the opening of an anonymous session and if successful    **/
/** it manages it. With the Ks it encrypts msg before sending them to the tcp layer   **/
/** and  decrypts them before passing them to the user layer.                          **/
/**                                                                                   **/
/** Communication channels connected to this layer:                                    **/
/**                                                                                   **/
/**                                                                                   **/
/**                                        <--- KpuPort_to_PDA  ----                  **/
/**                                                                  KpuMan           **/
/**                                                                                   **/
/**                                                                                   **/
/**          ---- PDAuser_to_ses --->        ---- PDA_to_KsPort  ---->                **/
/**  PDAuser                           PDAses                          KsTmpIdMan     **/
/**          <--- PDAses_to_user ----        <--- KsPort_to_pda  -----                **/
/**                                                                                   **/
/**                                                                                   **/
/**                                        ---- PDAses_to_tcp  ---->                  **/
/**                                                                  PDAtcp           **/
/**                                        <--- PDAtcp_to_ses  -----                  **/
/**                                                                                   **/
/**                                                                                   **/
proctype PDAses
{

KpuNegotiation: /* learnt the MSS Kpu */
...   ...   ...

KsNegotiation:  /* negotiate a session key */
 /* create a random session secret key (Ks), encrypt it with the public    */
 /* key of the MSS and send it to the MSS. Either succeed or give up after */
 /* NUMATTEMPTS times                                                      */
```

```
do
:: (i <= MAXNUM_Ks_ATTEMPTS) ->
   Ks= NewKs_Fetched;
   /** encrypt Ks with Kpu here **/
   PDA_to_KsPort ! Ks;
   if
   :: KsPort_to_PDA ? Ks(TmpId) ->
                      /* the last send Ks is the session key */
                      PDAses_to_user ! TmpIdReceived
                      goto PaySes;
   :: timeout -> Ks= NewKs_Fetched; i++
   fi
:: else -> PDAses_to_user ! abort; goto abort
od;


PaySes:
 /** All messages sent to the flow control layer are previously     **/
 /** encrypted with Ks. All messages received from the flow control **/
 /** layer are decrypted with Ks before passing them  to the user   **/
 /** layer.                                                         **/
 PDAuser_to_ses ? Ecash -> PDAses_to_tcp ! Ecash; /* wait here till Ecash comes */
 do
 :: PDAtcp_to_ses ? EcashAccepted -> PDAses_to_user ! EcashAccepted;
                                     goto AnoSes
 :: PDAtcp_to_ses ? EcashRejected -> PDAses_to_user ! EcashRejected
                                     goto PaySes /* try again */
 :: timeout -> goto Aborted
 od;


AnoSes:
 do /* loop till anonymous session time expires */
 :: PDAuser_to_ses ? EaddrAndTxt  -> PDAses_to_tcp  ! EaddrAndTxt

 :: PDAtcp_to_ses ? YouHaveMail   -> PDAses_to_user ! YouHaveMail

 :: PDAuser_to_ses ? ChangeMyKs   -> PDAses_to_tcp  ! ChangeMyKs

 :: PDAtcp_to_ses ? YourNewKs     -> Ks= YourNewKs  /* update current Ks */

 :: PDAuser_to_ses ? Ecash        -> PDAses_to_tcp  ! Ecash /* Ecash >= 0 */

 :: PDAtcp_to_ses ? EcashAccepted -> PDAses_to_user ! EcashAccepted

 :: PDAtcp_to_ses ? EcashRejected -> PDAses_to_user ! EcashRejected

 :: PDAtcp_to_ses ? TimeExpAlert  -> PDAses_to_user ! TimeExpAlert

 :: PDAtcp_to_ses ? TimeExp       -> PDAses_to_user ! TimeExp; goto Aborted

 :: PDAuser_to_ses ? abort        -> PDAses_to_tcp  ! abort

 :: PDAtcp_to_ses ? aborted       -> PDAses_to_user ! aborted -> goto Aborted
 od;


Aborted:
 skip /* clean everything up */
```

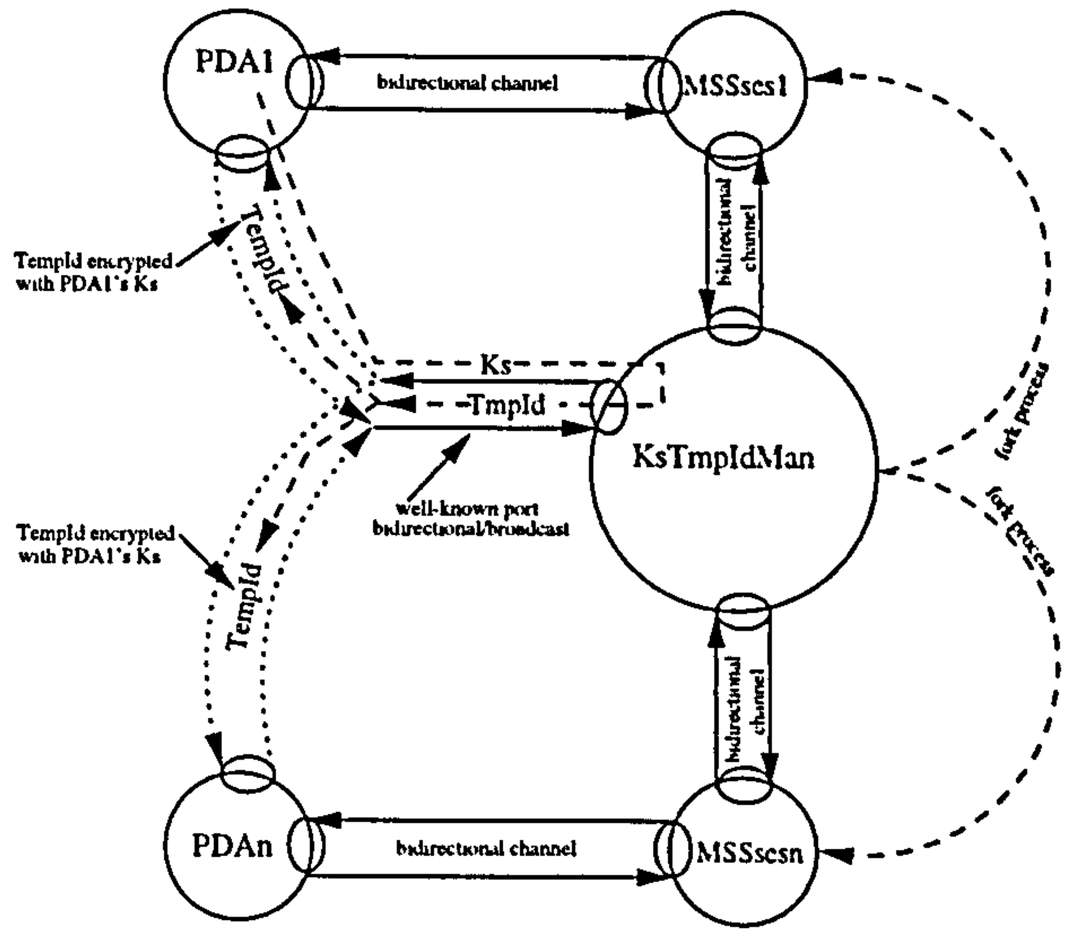


Figure 6.10: The KsTmpIdMan process manages the Ks and TmpId of PDAs.

}

6.6.5 The session keys and TmpId manager

The session keys and TmpIds manager (KsTmpIdMan) is the main process in the protocol, it runs in the MSS and as its name suggests is in charge of managing the session keys used by PDAs and the temporary identifiers assigned to the PDAs. As shown in figure 6.10, it is linked to *MSSses* and *PDA* processes by way of bidirectional channels.

Basically, the *KsTmpIdMan* process is there to perform two crucial functions. First, it guarantees that the session keys used by PDAs are correct and unique. Secondly, it assigns a unique TmpId to each PDA which has succeeded in suggesting the right session key.

To explain how the *KsTmpIdMan* process works let us assume that *PDAi* is owned by Bob. After obtaining the public key of the MSS (see section 6.6.4) Bob's PDA proceeds to negotiate a session key with the MSS. *PDA1* sends its session key proposal to well-known port of the MSS. When a session key proposed by *PDA1* is approved by the key manager, the latter forks a process (*MSSses1*) to be in charge of the anonymous session for *PDA1*. All the details (the number of communication channels to talk to for example) about the *MSSses1* process together with the TmpId assigned to *PDA1* are sent to *PDA1* in the *TmpId* message which is encrypted with Bob's session key and broadcast to the air.

As can be appreciated from the finite state diagram shown in figure 6.11, the *KsTmpIdMan* process spends most of its time in the *WaitingForKs* state listening for *Ks* and *ChangeMyKs* messages to arrive. A *Ks* message contains a suggested session key and comes from a newly arrived PDA that wants to negotiate a session key, it is addressed to a well-known port of the *KsTmpIdMan* process.

When this message arrives, the finite state diagram moves to the *KsValidation* state where it is decided whether the proposed *Ks* is to be accepted or refused.

If the suggested session key is correct, the finite state diagram moves to the *TmpIdSelection* where a TmpId is selected for the PDA. It moves to the *WaitingForKs* state after the message *TmpId* is sent to the PDA.

Conversely, if the suggested Ks is incorrect the *KsTmpIdMan* process does not reply to the PDA and the finite state diagram moves to *WatingForKs* state.

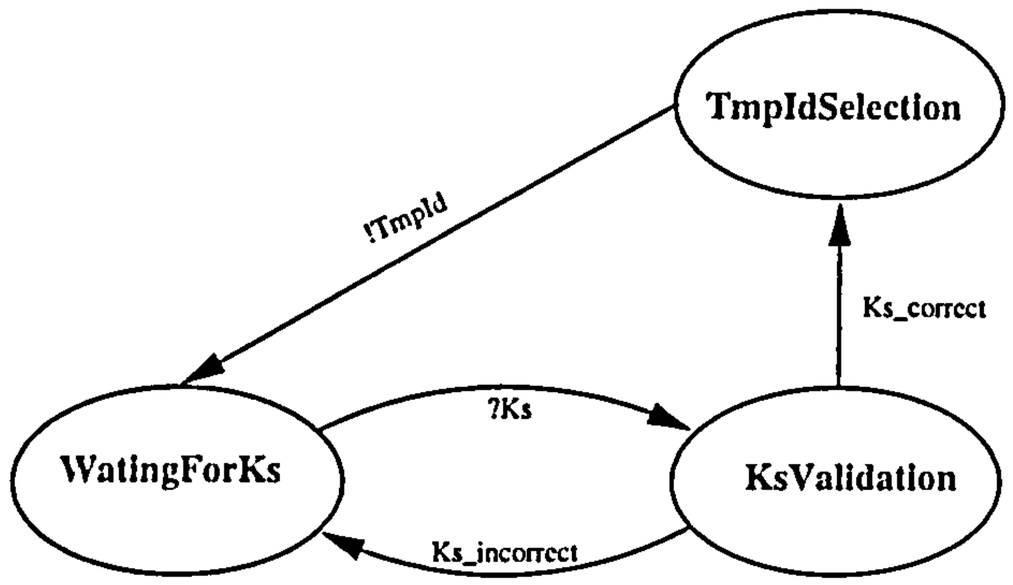


Figure 6.11: Finite state diagram of the KsTmpIdMan process.

Once an *MSSses1* process is assigned to *PDA1*, the bidirectional broadcast channel is not longer needed. This channel is used temporarily by the *PDA1* process. This is why it is represented by a dashed line.

Figure 6.10 shows that the *KsTmpIdMan* process and the *MSSses1* process are linked together by a bidirectional channel. It is through this bidirectional channel that the *KsTmpIdMan* and *MSSses1* processes inform each other about any problem with session keys. It is conceivable that the session key being used by *PDA1* is hit by a session key proposal coming from *PDA2*. If this happens the *KsTmpIdMan* sends a *YourNewKs* message to *PDA1*, which contains a new *Ks* for it. Also, it is possible that the PDA1's user decides to renew his current *Ks*, if so, the *ChangeMyKs* is received from *PDA1*; upon receiving this message, the *MSSses1* process informs the *KsTmpIdMan* about it. Eventually the *KsTmpIdMan* creates a new session key for *PDA1*, puts it in the *YourNewKs* message and sends it to *PDA1*.

```
/** The KsTmpIdMan process: its main task is to ensure that session keys and     **/
/** TmpId assigned to PDA are correct and unique. It is in charge for receiving  **/
/** session key suggestions from newly arrived PDA. It verifies the suitability  **/
/** of the suggested key and if satisfied, forks a child process (MSSses) to ser-**/
/** ve the PDA in its anonymous session. Once a session key is accepted it re-   **/
/** plies to the PDA by sending a TmpId.                                         **/
/** It receives requests from MSSses to change their current session keys. If a  **/
/** session key currently in use is hit by another PDA proposal, the KsTmpIdMan  **/
/** asks the unlucky PDA to change its session key.                              **/
/**                                                                              **/
/**                  ----KsPort_to_pda         --->                              **/
/**                                                 PDA                          **/
/**                  <---PDA_to_KsPort         ----                              **/
/**   KsTmpIdMan                                                                 **/
/**                  ----KsTmpIdMan_to_ses     --->                              **/
/**                                                  MSSses                      **/
/**                  <---MSSses_to_KsTmpIdMan ----                              **/
/**                                                                              **/
/**                                                                              **/
/**                                                                              **/
proctype KsTmpIdMan
{

WaitingForKs:
do
 :: PDAi_to_KsPort ? Ks ->  /* received from PDAi */
    if
    :: (Ks == OK)     ->
       /* create TmpId */
       run MSSses(Ks, TmpId, ... )  /* fork a child process for PDAi*/
       KsTmpIdMan_to_PDAi ! TmpId

    :: (Ks == INUSE) ->        /* Ks in use by PDAj */
       /* send nothing to PDAi */
       /** find a new Ks for PDAj */
       KsTmpIdMan_to_PDAj ! YourNewKs
    fi

 :: PDAj_to_KsTmpId ? ChangeMyKs /* PDAj asking its current Ks to be changed */
    /** find a new Ks for PDAj */
    KsTmpId_to_PDAj ! New_Ks_for_PDAj
od;
```

6.6.6 The anonymous session

It was shown in figure 6.10 that each PDA is served by an anonymous session process created by the *KsTmpIdMan* process when the PDA succeeds in opening an anonymous session. To each *PDAj* process corresponds one *MSSsesj* process. To serve its PDA the *MSSsesj* interacts with other processes through bidirectional channels as illustrated in figure 6.12.

The work of each of the *MSSses* processes can be described with the help of the finite state diagram shown in figure 6.13.

The finite state diagram starts in the *WaitingForEcash* state where it waits until money from the PDA comes. This is represented by the arrival of the *Ecash* message which moves the finite state diagram to the state *PaySes* where the very first payment for opening an anonymous session is performed. Upon receiving the e-coin from Bob, the session layer verifies that the money received is suitable to pay for an anonymous session. If it is suitable it sends the *EcashAccepted* message to

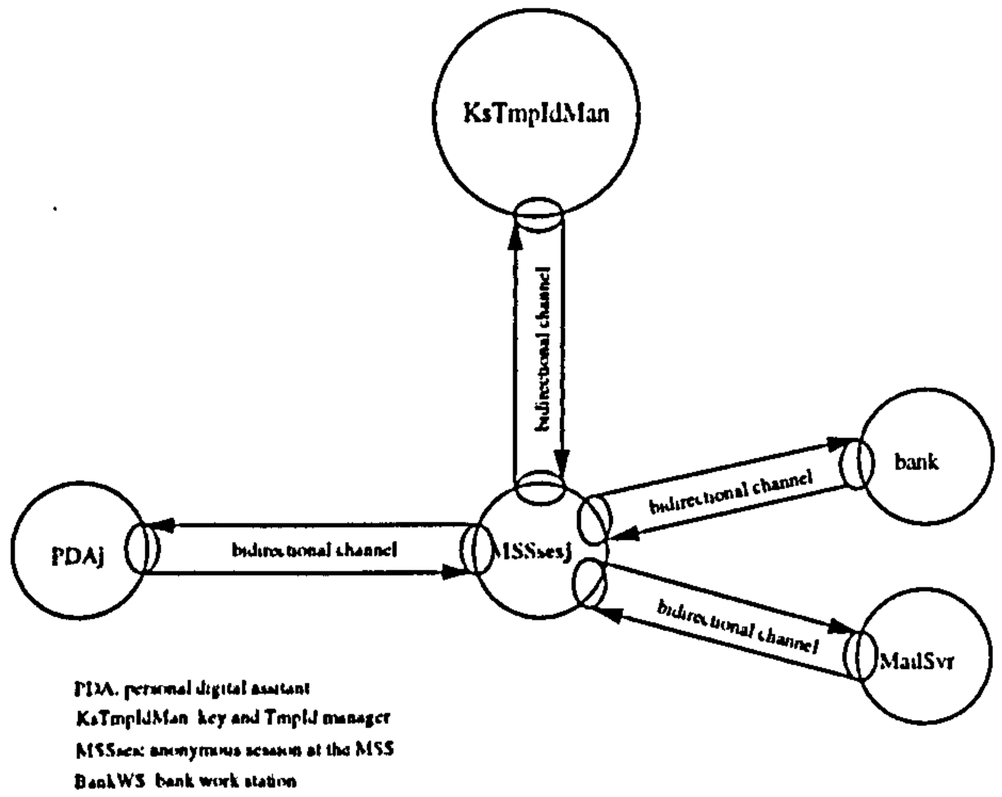


Figure 6.12: The anonymous session process and its connections to other processes

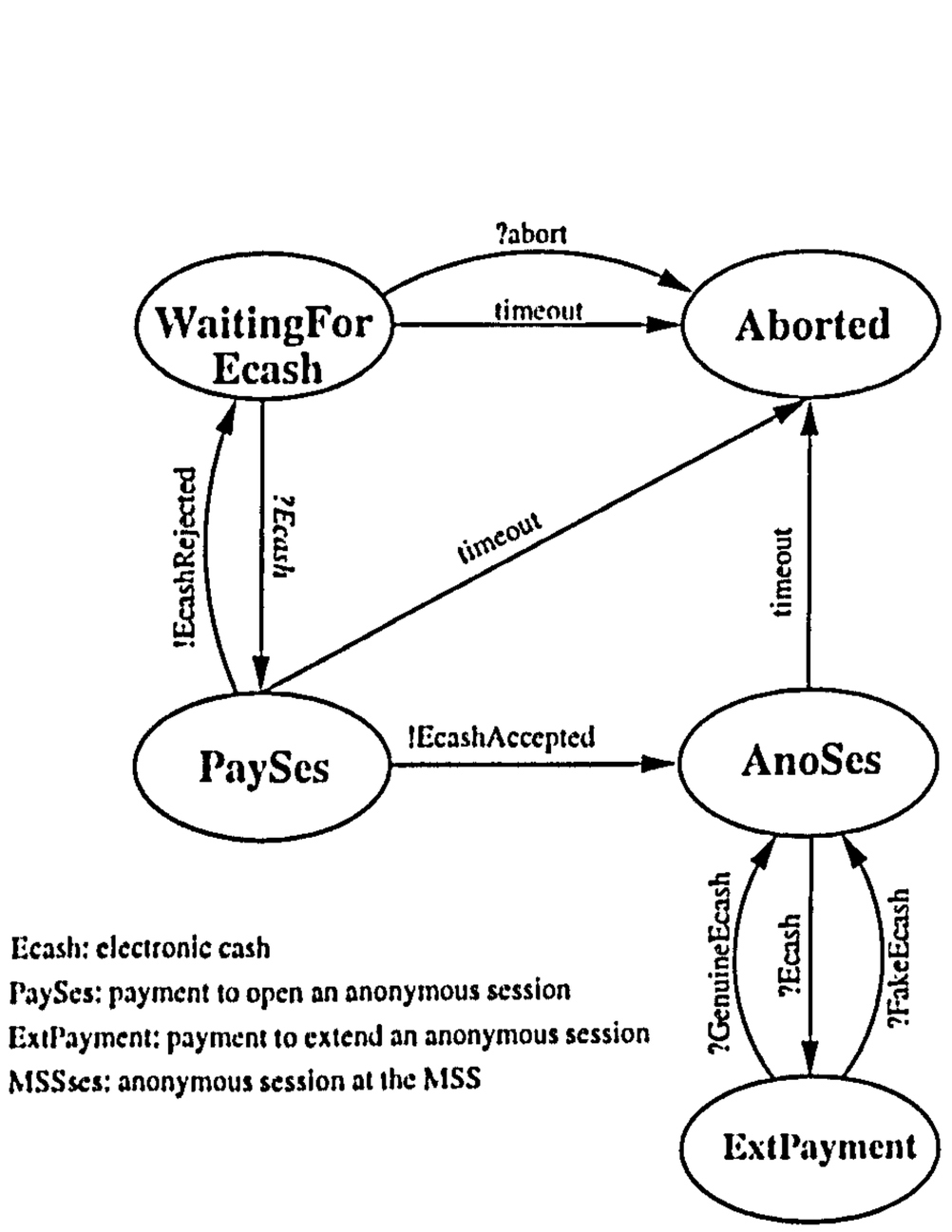


Figure 6.13: The session layer of the MSS.

the PDA and moves to the *MSSses* state; this means that an anonymous session is now opened for Bob.

An e–coin is considered suitable as a payment for an anonymous session when it matches two conditions. First, it has to be a genuine e–coin (not a fake, already spent or out of circulation) in a currency —pound sterling for example— accepted by Doug. Second, it has to be at least equal to the minimum payment that Doug, the owner of the MSS, charges for an anonymous session. Doug can easily verify the second condition by himself, however, he needs the help of Clare —the bank owner— to verify the genuineness of the e–coin. Upon receiving the e–coin, Doug forwards it to Clare through the bidirectional channel that links Bob's *MSSses* and the *bank* process, and waits for a response telling him either that the e–coin is genuine or fake. Based on Clare's verdict, Doug accepts or rejects the e–coin. This is explained in more detail later on in section 6.6.7.

If the e–coin is rejected, the *MSSses* sends an *EcashRejected* message to Bob and the finite state diagram moves to the *WaitingForEcash* state. If no suitable e–coins are received before a timeout runs out everything concerning the PDA (its session key and TmpId for instance) is cleared up in the *MSSses* and in the *KsTmpIdMan* process as well. If Bob wishes to persist with his anonymous session, he has to start again from the very beginning with the negotiation of a session key and a TmpId.

The *MSSses* state is where the session layer spends most of its time. It stays there as long as there is credit for Bob. It leaves this state permanently when a timeout mechanism goes off to indicate that the prepaid communication time has expired, then the finite state diagram moves to the *Aborted* state. Being in the *MSSses* state, the MSS session layer is ready to receive e–mail messages from Bob in the form of *EaddrAndTxt* messages. When a message like that is received, the *MSSses* decrypts it using the current session key, and forwards it to the *MailSvr* process. On the other hand, it receives Alice's replies in the form of *YouHaveMail* messages coming from the *MailSvr*, encrypts them using the current session key, and forwards them to Bob by sending them down through the protocol stack. More details about how the *MailSvr* works are in section 6.6.8.

Naturally, we assume that the *MSSses* process can convert Bob's messages into a standard e–mail format and send them to the Internet using standard e–mail protocols.

Although it is not shown in the finite state diagram, it may happen that the time Bob is credited for his first payment is not long enough for Bob to send and receive his messages. To warn Bob about the expiration of his prepaid credit the *TimeExp* is sent to him. From now on he has n seconds to send more e–cash to the MSS unless he wants his anonymous session to be finished abruptly.

Once again, as in the payment to open the anonymous session, any e–coin sent by Bob to the MSS is subject to a test of acceptance. If the e–coin is accepted, Bob's communication time is incremented according to the value of the e–coin and the message *EcashAccepted* is sent back. On the contrary, if the e–coin fails the acceptance test the message *EcashRejected* is sent and the time for Bob's anonymous session is left unchanged.

The session key Bob negotiates with the *KsTmpIdMan* process does not necessarily last until the end of the anonymous session. It can be changed in the middle of the anonymous session. There are two situations that leads to the change of the current session key.

First, because Bob suspects that his current session key has been compromised, Bob may instruct his PDA to negotiate a new session key. As illustrated in figure 6.14 in this case the initiative to change the current session key comes from Bob's PDA which sends the *ChangeMyKs* message to the *MSSses* process which forwards the message to the *KsTmpIdMan* process. The *KsTmpIdMan* finds a new session key for Bob and replies with a *YourNewKs* message. If for any reason (no more session keys are available for example) the *KsTmpIdMan* sends an *abort* message to the *MSSses* from where this message is propagated down the stack protocol until eventually it

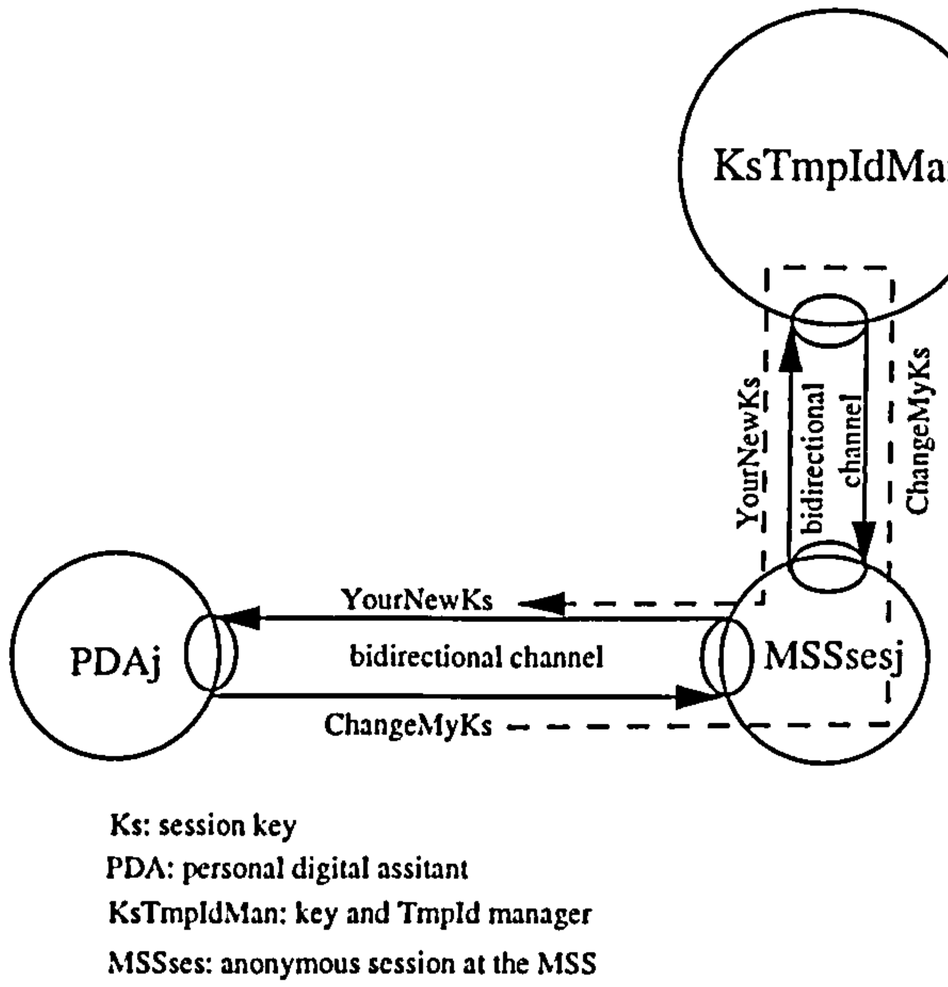


Figure 6.14: Change of session key initiated at PDA.

reaches the *PDAuser* layer. The result of this is that all layers involved in Bob's PDA communication are aborted.

Secondly, an order to Bob's PDA to change its current session key can be received from the *KsTmpIdMan* process in the form of *YourNewKs* message as illustrated in figure 6.15. This message is sent when the *KsTmpIdMan* process detects that Bob's session key has been hit by somebody else. It might be that Bob's session key is hit and that the *KsTmpIdMan* cannot find a new session key for him. In such a case, the *abort* message is propagated to abort Bob's communication session as explained above.

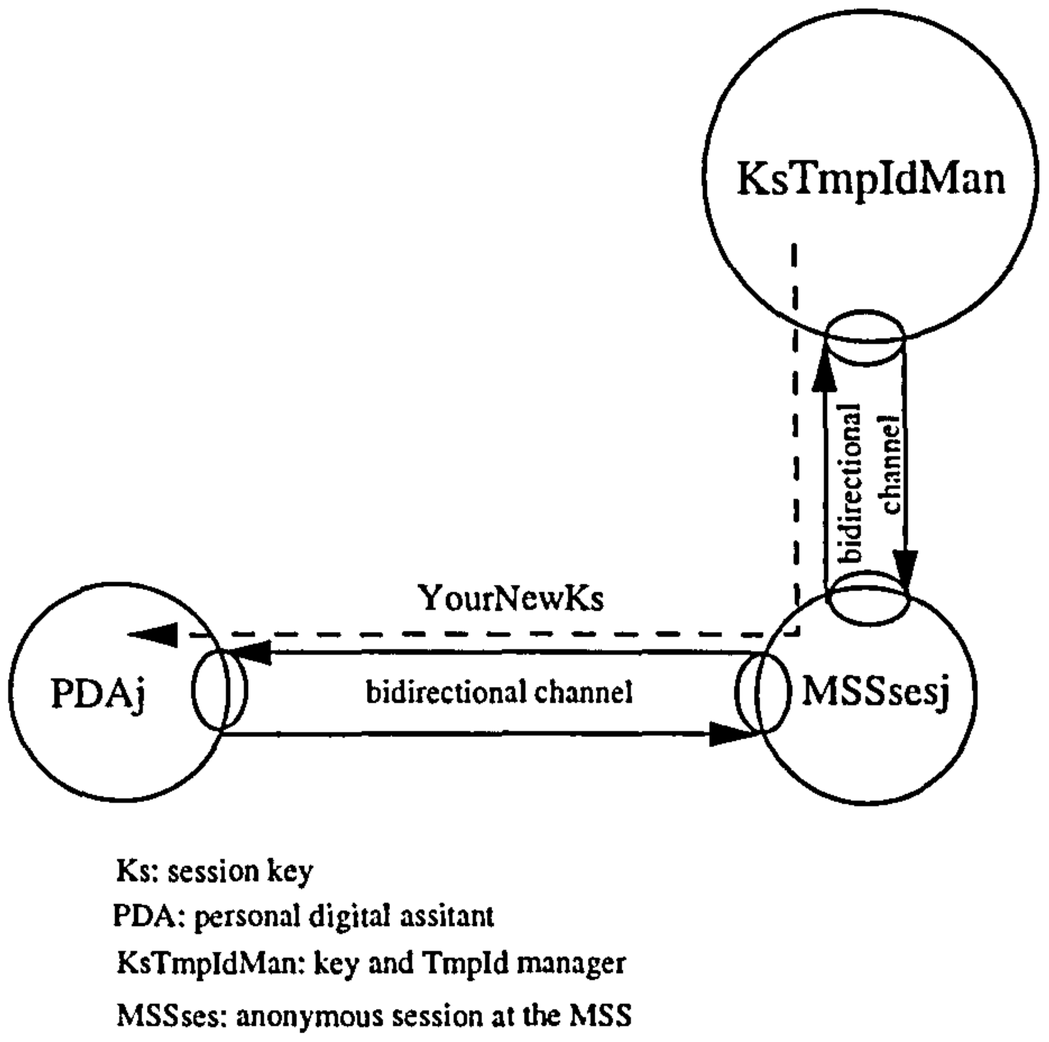
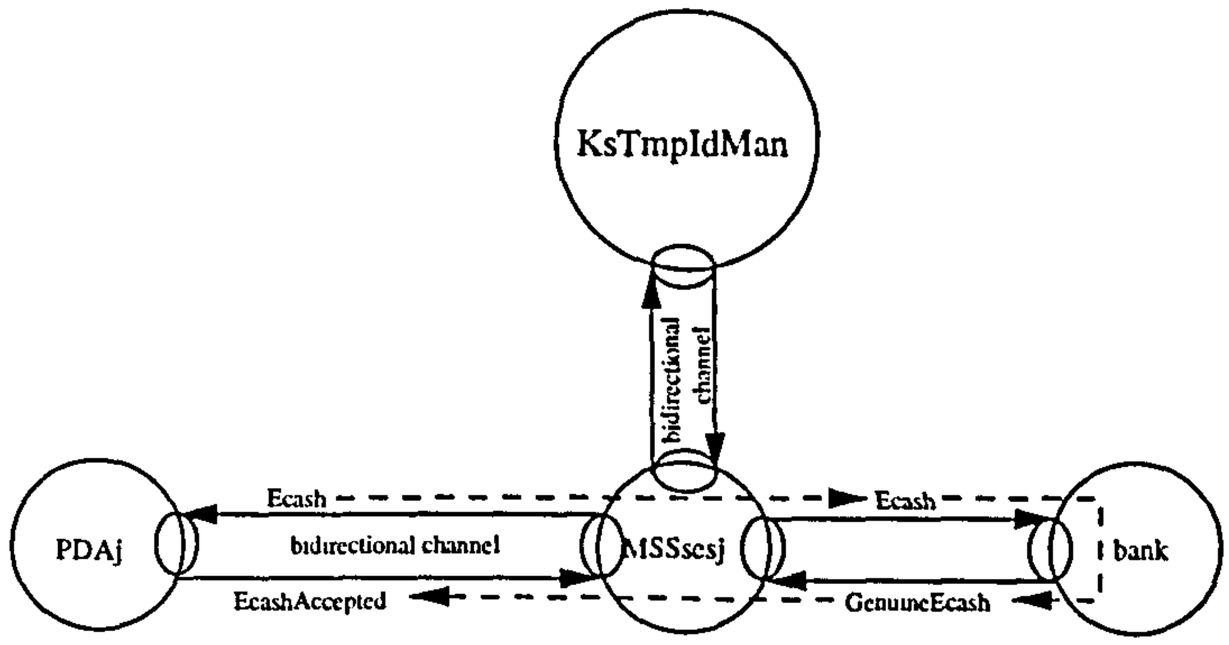


Figure 6.15: Change of session key initiated at key and TmpId manager.

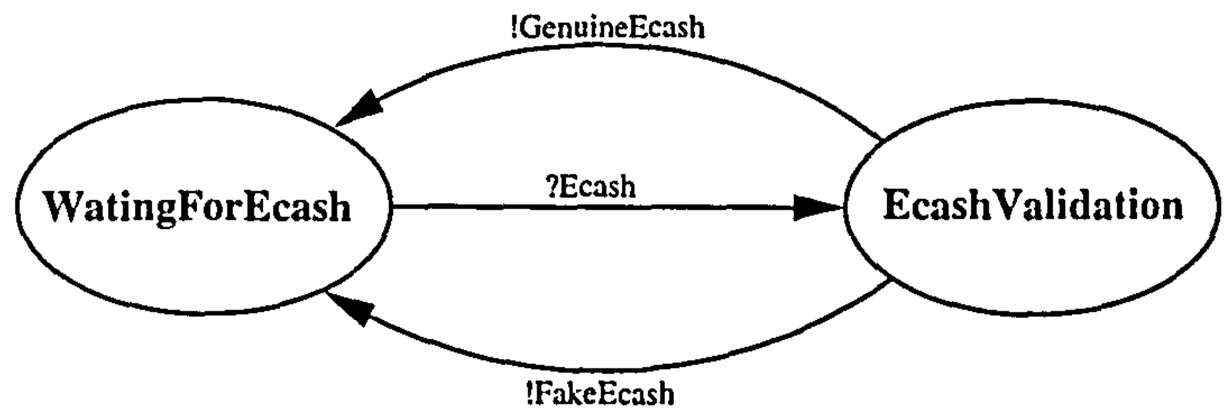
```
/** MSSses LAYER: it manages the PDAs anonymous communications. It receives      **/
/** e-mail messages from the PDA and forwards then to the 'MailSvr', conver-     **/
/** sely it receives e-mail messages from the 'MailSvr' and forwards them to     **/
/** the PDA.                                                                     **/
/** It is responsible for keeping the session key, updating it, timing out the   **/
/** end of the anonymous time. It receives the PDA payment for the anonymous co- **/
/** mmunication session and with the help of the bank assure that the recei--    **/
/** ved e-coin is genuine.                                                       **/
/**                                                                              **/
/**                                    ---- MSSses_to_MailSvr --->               **/
/**                                                                MailSvr       **/
/**              --- tcp_to_MSSses --->     <--- MailSvr_to_MSSses ----          **/
/**                                                                              **/
/**          tcp                        MSSses                                   **/
/**                                          ---- MSSses_to_bank --->            **/
/**                                                                bank          **/
/**             <--- MSSses_to_tcp ----      <--- bank_to_MSSses ----           **/
/**                                                                              **/
/**                                                                              **/
proctype MSSses  /** MSS **/
{

WaitingForEcash:
 do
 :: MSStcp_to_ses ? abort -> got  Aborted
 :: timeout               -> goto Aborted
 :: MSStcp_to_ses ? Ecash -> goto PaySes
 od;

PaySes:
 MSSses_to_bank ! Ecash /* send Ecash to the banker for validation test */
 if
 :: bank_to_MSSses ? GenuineEcashEcash ->
     if
     :: MINPAYMENT <= Ecash <= MAXPAYMENT) ->
```

```
        AnonymousTime= Ecash_to_time(Ecash) /* convert Ecash to time */
        ses_to_tcp ! EcashAccepted;
        goto AnoSes
     :: else -> ses_to_tcp ! EcashRejected; goto WaitingForEcash
     fi
 :: bank_to_ses ? FakeEcash -> MSSses_to_tcp ! EcashRejected; goto WaitingForEcash
 fi;

ExtPayment:
 if
 :: bank_to_MSSses ? GenuineEcashEcash ->
     if
     :: MINPAYMENT <= Ecash <= MAXPAYMENT) ->
        AnonymousTime= AnonymousTime + Ecash_to_time(Ecash) /* convert Ecash to time and */
        ses_to_tcp ! EcashAccepted;                         /* extend anonymous time     */
        goto AnoSes
     :: else -> ses_to_tcp ! EcashRejected; goto AnoSes
     fi
 :: bank_to_ses ? FakeEcash -> MSSses_to_tcp ! EcashRejected; goto AnoSes
 fi;

AnoSes:
 do
 :: MSStcp_to_ses ? EaddrAndTxt -> MSSses_to_MailSvr ! EaddrAndTxt
                   /* receive the msg, deencrypt with Ks and forward it to MailSvr */

 :: MailSvr_to_ses ? YouHaveMail -> MSSses_to_tcp ! YouHaveMail
                   /* receive the msg, encrypt with Ks and forward it to PDA */

 :: MSStcp_to_Sses ? ChangeMyKs -> /* A PDA user requesting a new Ks */
                   MSSses_to_KsTmpIdMan !  ChangeMyKs

 :: KsTmpIdMan_to_ses ? YourNewKs -> /* Your Ks has been hit update it*/
                   Ks=YourNewKs;
                   MSSses_to_tcp ! YourNewKs

 :: MSStcp_to_ses ? Ecash -> goto ExtPayment /* additional payment */

 :: TimeAlert_timeout -> /* alert the user about the end of his anonymous session */
                   MSSses_to_tcp ! TimeAlert

 :: TimeFin_timeout -> /* finish the anonymous session */
                   MSSses_to_tcp ! TimeFin
                   goto abort

  :: KsTmpIdMan_to_ses ? abort -> MSSses_to_tcp ! abort
                       goto Aborted

  :: tcp_to_ses ? abort -> goto Aborted
  od;

Aborted:
 /* clear everything and finish */
 skip
}
```



Ks: session key
PDA: personal digital assitant
KsTmpIdMan: secret key and TmpId manager
MSSses: anonymous session at the MSS
bank: bank work station
Ecash: electronic cash

Figure 6.16: Validation of e–cash at bank work station



Ecash: electronic cash

Figure 6.17: Validation of e–cash

6.6.7 The bank process

Although we do not include all of the details concerning the verification of the e–coin, we assume that the e–coin that Bob sends to Doug is an anonymous e–cash previously blindly signed by Clare (see section 5.8), hence upon receiving the e–coin Clare verifies that it has a valid signature on it and that the coin has not already been spent. As is shown in figure 6.16, Clare responds either with a *GenuineEcash* or *FakeEcash* message depending on whether she is satisfied with the e–coin or not. On this basis Doug decides to accept or reject the payment. It is important to recall from figure 6.12 that the *bank* process is linked to the *MSSses* by a bidirectional channel. In fact the bank workstation can be any computer connected to the Internet as long as it can be reached by the *MSSses* process.

The finite state diagram that shows how the bank process works is depicted in figure 6.17. The *MSSses* and the *bank* processes are linked by a bidirectional channel. As can be appreciated from the figure, the bank process keeps waiting in the *WaitingForEcash* state until the message *Ecash* arrives from the *MSSses* serving the PDA. When such a messages arrives it moves to the *EcashValidation* where it decides whether the Ecash is genuine or not. In the former case it replies with a *GenuineEcash*; in the latter case it sends the *FakeEcash* message to the *MSSses* process.

The Promela code for the *bank* process is shown below. In real life the bank workstation would receive real e–coins and would perform real validation tests on them. At this stage, all we are interested in here is the simulation of a validation test. For this purpose we use the nondeterministic

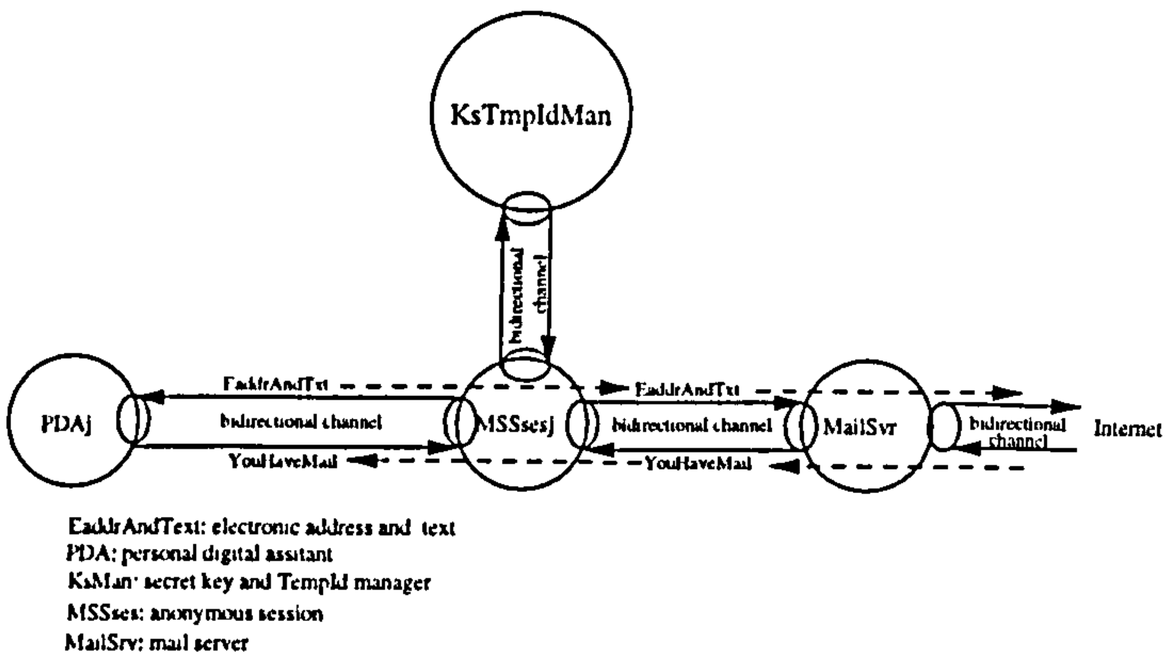


Figure 6.18: The mail server process

nature of the Promela guarded commands (::) by which we arbitrarily decide that an e–coin is genuine or fake. Recall that if more than one guarded command is executable, one of the corresponding sequence is selected at random.

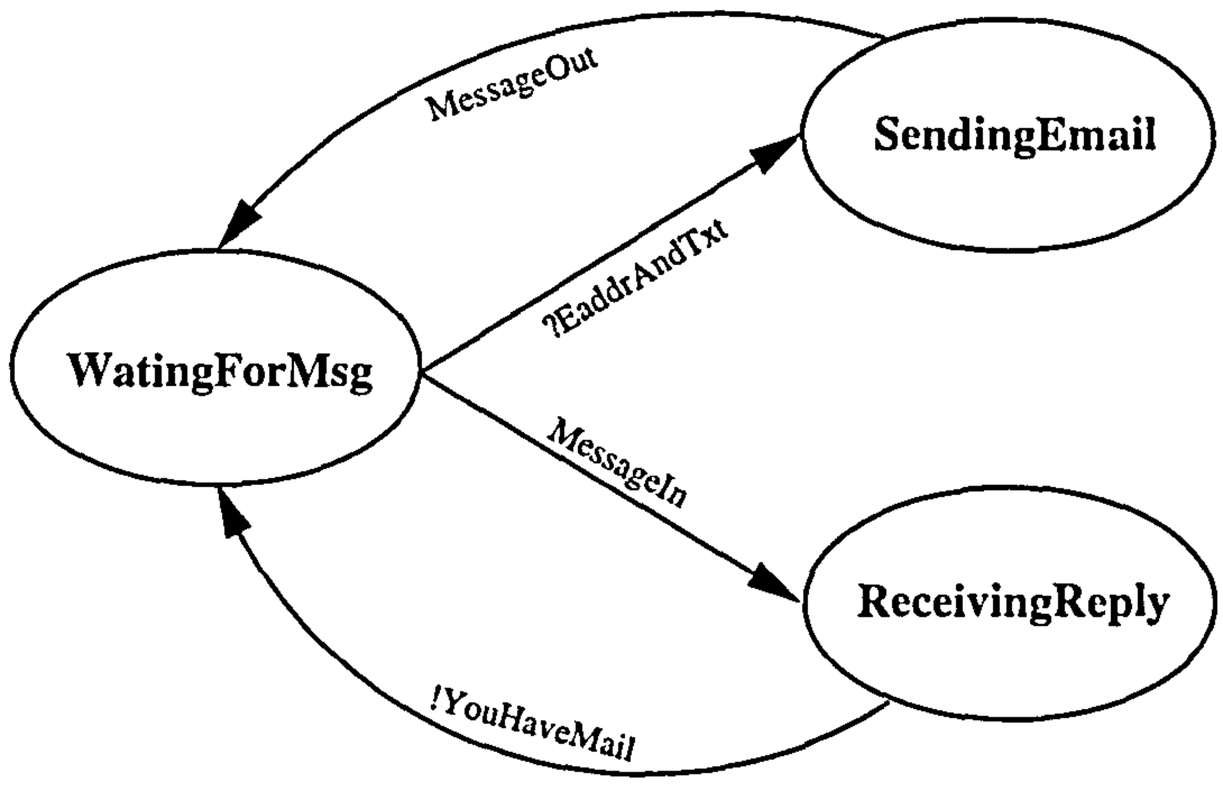
```
/** The bank process: receives Ecash from the anonymous session and tests that   **/
/** the e--coin is acceptable at the bank and has not been spent before.         **/
/** Depending on the result of the test, it responds to the MSSses with either   **/
/** a 'GenuineEcash' or 'FakeEcash' msg.                                         **/
/**                                                                              **/
/**                                                                              **/
/**                --- MSSses_to_bank --->                                       **/
/**        MSSses                              bank                              **/
/**               <--- bank_to_MSSses ---                                        **/
/**                                                                              **/
proctype bank
{
 MSSses_to_bank ? Ecash
 if
 :: (Ecash == GENUINE) -> bank_to_MSSses ! GenuineEcash
 :: (Ecash == FAKE )   -> bank_to_MSSses ! FakeEcash
 if
}
```

6.6.8 The mail server process

The *MailSvr* process serves as a bridge for the *MSSses* process to send and receive e-mail messages to and from computers connected to the Internet. As illustrated in figure 6.18, when Bob sends an anonymous message, the message is originated in Bob's PDA. Secondly, it travels to the MSSses that serves Bob's PDA and from there it goes to the *MailSvr* process which is in charge of sending it to Alice (its final destination) with the help of standard e–mail protocols. If Alice replies, her response travels in the opposite direction until eventually it reaches Bob's PDA.

It is important to notice from figure 6.18 that the mail server is linked to each anonymous session serving a PDA by a bidirectional channel. The mail server is located not necessarily in the MSS, it can be there or in another computer as long as it is reachable by the *MSSsesj* processes through the bidirectional channels.

The finite state diagram that shows how the mail server process works is depicted in figure 6.19. The mail server remains in the *WaitingForMsg* state until either the *EaddrAndTxt* message is received or the *MessageIn* event occurs. The arrival of the *EaddrAndTxt* message from the



Msg: messages
EaddrAndTxt: electronic address and text

Figure 6.19: Finite state diagram of the mail server process

MSSses process indicates that the *MSSses* wants the *MailSvr* process to forward the message to its final destination (to Alice's computer for example). This event moves the finite state diagram to the *SendingEmail* state where it remains until the message is sent —indicated by the *MessageOut* event— and next it returns back to the *WaitingForMsg* state.

The arrival of Alice's reply at the *MailSvr* process is indicated by the occurrence of the *MessageIn* event. The occurrence of this event moves the finite state diagram to the *ReceivingReply* state from where Alice's reply is sent in the *YouHaveMail* message to Bob's *MSSses* process. Upon sending the *YouHaveMail* message, the finite state diagram moves to the *WaitingForMsg* state.

The Promela code for the *MailSvr* process is shown below. All we are interested in at this stage is the simulation of sending and receiving of messages. Thus, the mail server process can be implemented in Promela code as an endless loop which checks whether there is a message to read from the *MSSses* process. If there is any, the message is read and discharged. On the other hand, the arrival a reply from Alice can be modelled by a timeout mechanisms. The firing of the timer is taken as the arrival of one of Alice's replies and then the message *YouHaveMail* is sent to the *MSSses* process.

```
/** The MailServer process: it receives e-mails from the anonymous session process **/
/** and resends them to their final destination using standard e-mail protocols.   **/
/** Also, it receives e-mail messages addressed to Bob and resends them to the     **/
/** anonymous session process.                                                      **/
/**                                                                                 **/
/**                                                                                 **/
/**                --- MSSses_to_MailSvr --->                                    **/
/**        MSSses                                  MailSvr                      **/
/**              <--- MailSvr_to_MSSses ---                                     **/
/**                                                                                 **/
proctype MailSvr
{
 do
 :: MSSses_to_MailSvr ? EaddrAndText ->
                        /* resend the message to its final destination */
                        skip
 :: timeout -> MailSvr_to_MSSses ! YouHaveMail
 od;
```

}

6.6.9 The tcp layer

As with most applications, the protocol we are designing for instance, often needs to send several messages (both control and data messages) from one process to another. Using a protocol that offers only connectionless and unreliable delivery of packets such as the user datagram protocol of the TCP/IP set becomes annoying as the designer has to take care of several details to ensure that messages arrive at their destination safely. What the designer needs in situations like this is a reliable connection-oriented stream protocol that guarantees that messages sent from one process to another arrive safely at their destination. Such communication is often called *end-to-end* and is provided by the transmission control protocol of the TCP/IP architecture.

A TCP connection between two peers guarantees four important features [165, 154]:

1. Explicit initiation and termination of the connection between the two communicating peers.
2. Reliable, in-order, unduplicated delivery of unstructured streams of data.
3. Out-of band indication of urgent data.
4. Flow control to guarantee that the receiving buffer is never overflown.

On top of a protocol with the above characteristics, the difficulties of designing and implementing our protocol are significantly reduced. We can confidently rely on the TCP services to transmit messages between the PDA and the MSS session layers.

The first feature the TCP service guarantees that a connection between the two session layers is established and terminated. The second feature is necessary to ensure that messages received from any of the session layers are delivered to the remote peer without partial or total deletions and reorderings. This must be guaranteed despite the fact that the underlying layer (the UDP protocol) may delete and reorder messages arbitrarily. During the PDA–MSS interaction, it is feasible that urgent messages need to be sent to the remote peer to abort an anonymous session for example, this is exactly what the third feature guarantees. Finally, the flow control guarantees that if any of the communicating parties, the PDA for example, is slower than its peer, information is not lost due to buffer overflow.

With the TCP services in mind we can forget about what is underneath the PDA and MSS TCP processes and connect them directly as illustrated in figure 6.20.

In the figure, it can be seen that the output channel of the PDA tcp process is connected to the input channel of the tcp process of the MSS. Conversely, the output channel of the tcp process of the MSS is connected to the input channel of the tcp process of the PDA. This simplification is possible because of the semantics of Promela channels. A Promela channel implements flow control. Promela channels do not lose or duplicate messages; they deliver them in the right order. To do that, it passes messages in first-in-first-out order. Also, a send operation on a channel is executable only when the addressed channel is not full and a receive operation is only executable when the channel is not empty.

The Promela code for the PDA and MSS tcp processes is quite simple and shown next:

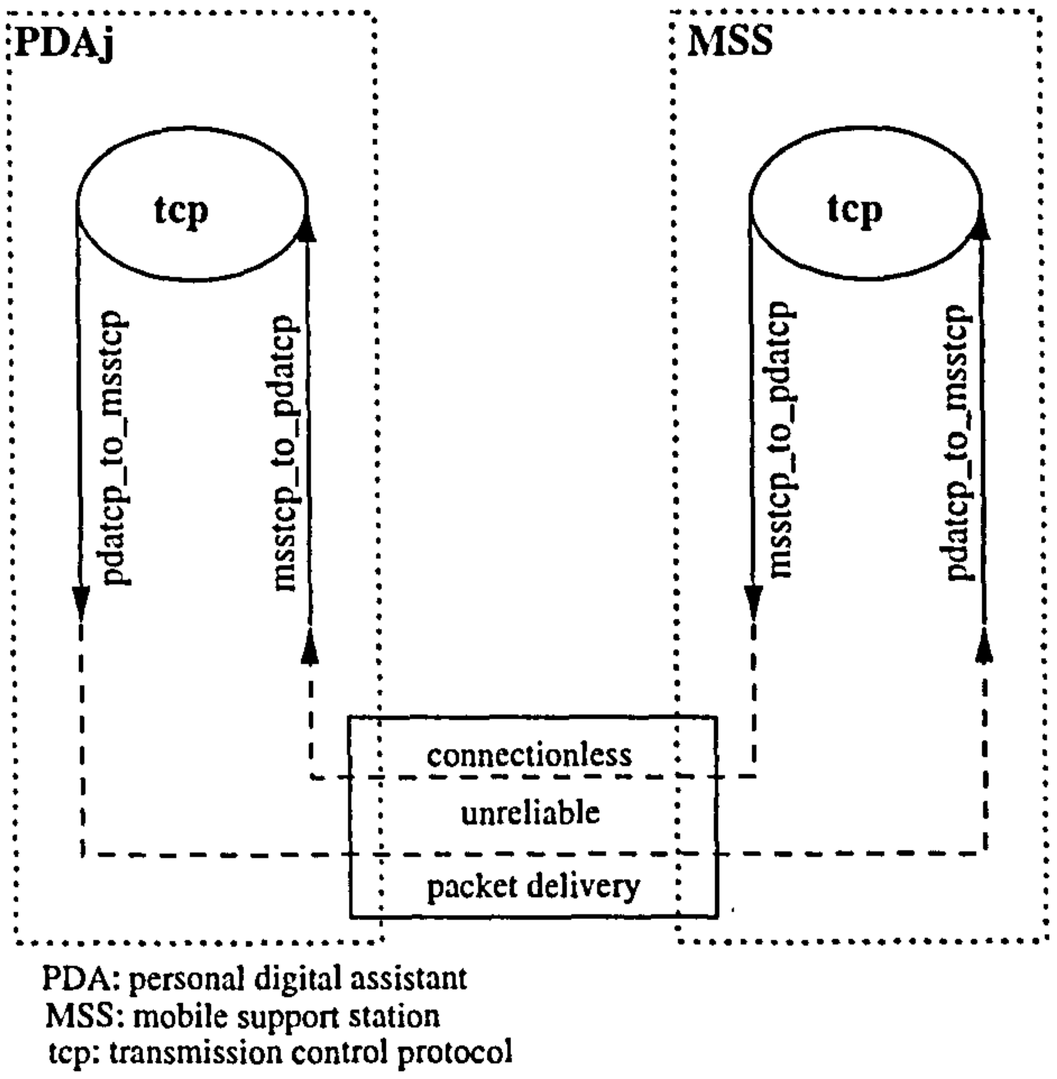


Figure 6.20: Connection of the PDA and MSS tcp processes.

```
/** The PDAtcp process: Implements a Transmission Control Protocol of the      **/
/** TCP/IP set. It is a reliable connection--oriented  stream transport pro-   **/
/** tocol (no deletions, no duplications, no reorderings of messages) on which **/
/** the PDA session layer can rely to send messages to, and receive messages   **/
/** from, the remote MSS session layer.                                        **/
/**                                                                            **/
/**                                                                            **/
/**     --- ses_to_tcp --->      --- pdatcp_to_msstcp --->                     **/
/**  ses                    pdatcp                          MSStcp             **/
/**    <---tcp_to_ses ----       <--- msstcp_to_pdatcp ----                    **/
/**                                                                            **/

proctype pdatcp
{
 do
 :: ses_to_tcp ? anymsg         ->  pdatcp_to_msstcp ! anymsg
 :: msstcp_to_pda_tcp ? anymsg -> tcp_to_ses ! anymsg
 od
}

/** The MSStcp process: Implements a Transmission Control Protocol of the      **/
/** TCP/IP set. It is a reliable connection--oriented  stream transport pro-   **/
/** tocol (no deletions, no duplications, no reorderings of messages) on which **/
/** the MSS session layer can rely to send messages to, and receive messages   **/
/** from, the remote PDA session layer.                                        **/
/**                                                                            **/
/**                                                                            **/
/**     --- ses_to_tcp --->      --- msstcp_to_pdatcp --->                     **/
/**  ses                    msstcp                          pdatcp             **/
/**    <---tcp_to_ses ----       <--- pdatcp_to_msstcp ----                    **/
/**                                                                            **/

proctype msstcp
{
 do
```

```
 :: ses_to_tcp ? anymsg          ->  msstcp_to_pdatcp ! anymsg
 :: pdatcp_to_msstcp ? anymsg    ->  tcp_to_ses ! anymsg
 od
}
```

The flow layer

What is underneath the TCP layer does not influence the behaviour of our protocol since the TCP protocol deals with all the difficulties. However, if for any reason the reader is interested in looking inside the TCP protocol we would recommend him or her to start with the flow control Promela code offered by Holzmann's book [159] which implements the well–known sliding window protocol.

6.7 Summary

Before any attempt to implement a distributed system is made, its communication protocol must be specified. It is well accepted that for a protocol specification to be complete it should explicitly specify five elements, namely: the service provided by the protocol, the assumptions about the environment where the protocol is used, the protocol vocabulary, the format of messages, and the procedure rules. Practice shows that the hardest part of this task is the specification of procedure rules as it concerns guarding the consistency of message exchanges between the interacting processes.

To specify procedure rules one can use finite state machines. Thanks to their graphic nature, finite state machines are easy to read. On the other hand the mathematical theory behind them makes them useful for showing the correct operation of a protocol. However, due to space limitations and the two-dimensional nature of paper, it is not always easy and practicable to show all possible incoming event possibilities including error conditions, local state variables and predicates associated with a protocol. Because of these limitations it might be useful to use them in combination with other methods, with C style pseudo-code, or a protocol verification language such as Promela, for example.

So far the most successful software tool used in the academic environment to trace logical design errors in distributed systems, and in particular in communication protocols, is Spin (Simple Promela INterpreter).

Promela is Spin's input language and provides a vehicle for making abstractions of protocols so that details that are unrelated to the communication processes are suppressed. A Promela program consists of processes, message channels and variables. The state of the whole protocol depends on the state of these three components.

To take advantage of both finite state machines and Promela, the two techniques were used in combination to specify the procedure rules of the anonymizing system introduced in this work: the protocol was divided into layers and then finite state machines were used to specify a simplified version of each layer while a complete specification was presented in Promela. The simplified specification is easy to read and understand while the Promela specification is ready to validate using Spin.

Chapter 7

Validation of the model

7.1 Introduction

Informally one can say that a protocol is correct if it behaves as its user expects. However, to prove that it is correct is a challenging task which involves two strongly connected procedures. Namely *validation* and *conformance test* of the protocol. The aim of the validation is to check that the formal specification of the protocol is logically consistent. Once the protocol designer is satisfied with the validation results the protocol can be implemented and tested to see if the implementation passes a conformance test; i.e. the designer or the user have to check that the external behaviour of the implementation of the protocol in equivalent to its formal specification.

Since we are in the very early stages of the development of our system, we will concentrate only on the validation process. Thus, we are interested only in verifying that the Promela specification of our system, i.e. our Promela validation model, is logically consistent without having in mind any particular implementation.

A validation model is an abstraction of the actual system: it simplifies the system by suppressing irrelevant details but without losing essential features of the system. For a well–modelled system one can assume that if the behaviour of the model is correct, the real system is correct as well. Because of this, system designers talk of validating a system but what they actually mean is validating the model of the system. The real system cannot be validated yet, simply because it has not been implemented yet. Strictly speaking, only the implementor or the user of the final system can talk of validating the system.

As was stated in section 6.6.2, the most widely used software for protocol validation is Spin. It was developed at Bell Labs in 1980, its source code written in ANSI standard C can be easily downloaded from the Internet [166] and compiled for Unix, Linux, Windows95 and WindowsNT platforms.

The Spin package consists of two independent tools: a simulator and a validator that are meant to be used at different stages of the protocol validation process.

7.2 The Spin simulator

As it names implies, the simulator can simulate the execution of a validation program (a *model* in Spin jargon) written in Promela. It simulates Promela code by interpreting its statements on–the–fly. To do its job the simulator performs a single–pass verification procedure making effort to save memory and CPU resources; it tries to store in memory just enough information to complete the verification process and to verify the correctness of the requirements but for the smallest possible

fragment of the whole behaviour of the system. For example, if at a given point during the simulation process the simulator is faced with more than one executable statement (a nondeterministic choice), it selects just one. This means the simulator does not perform any exhaustive reachability analysis but goes only through a single sequence of reachable states in the system which is chosen depending on the value of the seed the random number generator is initialized with; if no seed value is specified, the simulator performs a random simulation [167].

The advantage of using the simulator at an early stage of the system design is that it can immediately tell the system designer about simple inconsistences in his protocol, such as deadlocks, unspecified receptions. It is fast and does not demand a great deal of computer resources since it does not need to construct a global state for the system. Because of this, systems of arbitrary size can be easily simulated. However, since it runs a random simulation only, the absence of errors reported by the simulator does not necessarily mean that the system is error–free. The accurate verification of a system is performed by the Spin validator (see section 7.3).

In a Unix computer, the simulator is simply executed as *% spin -options PromelaCode* where the designer can specify different *options* to tell the simulator to output on the screen what messages are sent or received and by which processes, what line of the code is executed, the value of local and global variables, the value of the seed for the random number generator, and so on.

7.3 The Spin validator

The job of the Spin validator is to validate the *correctness requirements*, (also called *correctness criteria* and *properties*) of Promela code given at its input.

Spin belongs to the category of protocol verification systems that are based on the analysis of the reachability of system states. Before, going further in our discussion let us define what a state is in Spin.

In Spin, a *state* is completely defined by all control flow points of running processes, all values of local and global variables, and the contents of all local and global channels.

A reachability analysis algorithm tries to generate and inspect all the states of the system that are reachable from the initial state; this means that the algorithm will construct all possible execution sequences from the initial state to the final state (possibly more than one). In other words and assuming that the system we are analysing is non–deterministic (i.e. its Promela code contains guarded *::* commands), the algorithm must explore all possible moves. For example, if the validator is faced with the following code:

```
...
InputQueue ? a
if
:: (a > 0) -> statement1
:: (a = 0) -> statement2
:: (a < 0) -> statement3
fi
...
```

Spin has to explore three possible sequences after reading the from the input queue into variable *a*: first it considers that $a > 0$ and executes *statement1*; secondly, it considers that $a = 0$ and executes *statement2*; and finally, the validator assumes $a < 0$ and executes *statement3*.

It is worth noting that for a validation to be possible, the Promela specification of the system must restrict the number of processes, flow control point, variables, channels and slots of channels to a finite number so that the number of states of the system remains finite and the system can be analysed exhaustively by enumerating its reachable states.

Depending on the size of the system, the generation and analysis of all possible states can be computationaly unfeasible. Most of the time the designer of a large system (more than 10^5 reachable system states) is faced with the state space explosion problem. To understand this, we will briefly discuss how Spin works [162].

A system is represented in a Promela model as a set of process. Spin translates each process into a finite state automaton. Next, the asynchronous interleaving product of automata is computed and translated into an automaton. This automaton represents the global system behaviour and is called the *state space of the system* or the *global reachability graph.*

A correctness requirement of a system is expressed in a formal notation called *Linear Temporal Logic* (LTL for short). LTL can be translated into what is known as the *Büchi automaton.*

To perform a verification Spin computes the synchronous product of the Büchi automaton and the automaton that represents the global system behaviour. The result of this computation is another Büchi automaton and is used by Spin to see what language it accepts. If such a language is empty, this means that the correctness requirements expressed in the LTL formula are not satisfied by the system.

The thing to keep in mind during the validation is that to tell whether the language accepted by the Büchi automaton is empty or not Spin has to generate and verify all possible sequences of states of the automaton; this can become prohibitively expensive since in the worse case, the state space of the system has the size of the Cartesian product of all its components: control flow points, processes, local and global variables, and channels.

Once the system is written in Promela code and passed through the simulator, the designer is encouraged to validate it by performing the following steps:

1. By running *% spin -a PromelaCode* we can instruct Spin to generate a standard C program which is known as *the analyser* and by default receives the name of *pan.c*

2. The *pan.c* analyser can be compiled by a standard C compiler to produce an executable analyser as follows: *% cc -o pan pan.c.* Several directives can be specified to indicate whether the analysis of the system is going to be partial or exhaustive; and to optimize memory resources.

3. By typing *% pan –options* the program is executed to perform the analysis of the *PromelaCode.* Several options are at the designer disposition to indicate memory resources for hash tables; selection of hash functions, number of errors before termination, and so on.

4. If the analysis is completed without find any errors in the system and without running out of RAM space the output of *pan* is a few lines stating that no errors were found.

5. The execution of *pan* may have an early termination either because there was not not enough RAM memory in the system or because an error was found.

   - Optimization techniques can be used to ideal with shortage of memory.
   - If an error is detected by the analyser, it outputs the file *pan.t* containing the error trial, and stops.

6. The nature and cause of an error detected by *pan* can be found by instructing Spin to follow the error trial left in *pan.t* by typing *% spin -t PromelaCode*

7.4 Full state space search

As it names implies, the full state space search technique explores all reachable system states of the system. Because of this, this method is also known as *full exhaustive search* and just *full search*.

The main problem with validators based on reachability analysis, and Spin is no exception, is that the number of reachable system states can be prohibitively large in comparison to the amount of RAM memory available to store it and the CPU time available to compute it. As an aside it is important to note that since what matters here is not the total number of system states but only the reachable ones, in the literature the number of reachable system states is inaccurately referred simply as *the number of the states of the system* or simple *the state space*.

In order that the exhaustion of RAM memory and the length of computational time do not take us by surprise and in order to use these resources in the most efficient possible way, it is always advisable to have a rough idea of the size of the system in comparison with the available computational resources.

In the following, let R be the number of reachable states of the system we want to validate with the help of Spin; so if $0 \leq i \leq R-1$ and s_i is one of those reachable states we can define $A = \{s_0, s_1, s_2, ..., s_{R-1}\}$ as the set that contains all the reachable states of the system. Let S be the number of bits or bytes necessary to store a single state from the set of reachable states of the system. Finally, let M be the number of bits or bytes available for Spin use in the RAM memory of the computer.

Grounded on his experience, Holzmann points out that in full exhaustive mode the values for S are normally in the range of 10^1 to 10^2 bytes [159, 168] or what is approximately equivalent in the range of 10^1 to 10^3 bits.

Assuming that it is true, we can estimate that if we run Spin in full exhaustive mode the maximum number of reachable states that can be stored in a computer with a RAM memory of M bytes as follows: $R_{max} = M/S = M/1 \times 10^2$. In other words, the designer will run out of memory if he attempt to run the Spin validator in full exhaustive mode to validate a system with more that $M/1 \times 10^2$ reachable system states.

Another serious boundary that the designer has to keep in mind is the length of the CPU time to perform the analysis. It has been observed [159] that in a computer equipped with current technology the time to analyse a system state is of the order of 10^{-6} seconds. It follows that for a system of $R = 36 \times 10^8$ states, we will need about 1 hour (i.e. 3600 seconds) of computation: $CPU_{time} = 36 \times 10^8 \times 1 \times 10^{-6} = 3600$. Likewise, it can be estimated that a system of the order of 10^{30} states will need about 10^{16} years of CPU time.

When $R \leq R_{max}$, i.e. when the number of reachable system states of the system we are validating does not exceeds the maximum number of reachable system states that the computer can handle, and when the $R < 1 \times 10^8$ (assuming one hour is the maximum amount of CPU time the designer is ready to spend in the validation) the validation is straightforward; Spin performs an exhaustive search to verify the correctness requirements. The only thing the designer has to do is to follow the steps enumerated in section 7.3. When the validation finishes without any errors detected, the designer is certain that his system has been validated with 100% coverage. Although it is not always easy to know in advance, knowing the number of expected states is helpful since this number can be used to calculate the size of the hash table needed and then instruct Spin (*-wN hash table* option) to allocate a hash table to handle that number of states rather than using the default one which is equal to $2^{18} = 262144$ entries i.e. it can store 262 144 states. If for a given system this number happens to be too big it will cause a waste of memory; conversely, if it is to small, it will increase the number of hash collisions, waste CPU time in resolving them and consequently, slow down the verification.

However, it can happen that in the middle of the execution of the *% pan* step Spin runs out of RAM memory. If this happens a message like *pan: out of memory* appears on the output screen and the validation stops; the result of this is that what was meant to be an exhaustive search with a 100% coverage deteriorates into an uncontrolled partial search. Although some results about detected errors are reported on screen they are not reliable since we do not know what part of the system has been validated and what has not; there is no guarantee that the critical parts of the system have been inspected. Hence a *pan: out of memory* message is in fact an indication that further memory optimization is needed to validate the system.

From the above estimations it should be clear that the size of systems that can be validated with the Spin validator is directly determined by memory of the computer or more precisely by the parameter M and by the computational power of the CPU. Both M and CPU are hardware dependent and normally cannot be modified by the designer; fortunately if any of these two resource is exhausted, the designer can still manipulate the parameters S and R which are system dependent to fit his system to the available resources.

7.5 Controlled partial search

From the above discussion one question arises: if an exhaustive search deteriorates into an uncontrolled partial search when the RAM memory is exhausted; is there any technique to drive the Spin validator into a controlled partial search rather than letting it deteriorate into an uncontrolled one? The answer is yes, techniques for controlled partial search have been studied and suggested. In general and in accordance with the approach to fitting the system into the available RAM memory, controlled partial search methods can be divided into two groups: In the first group fall those methods that indicate to the Spin validator what part of the system to inspect. The main difficulty with this technique is that it is not trivial to decide what parts of the system to inspect and what not. Common sense would suggest to inspect those parts of the system where we expect to find errors. However, this is a debatable argument since real life practice shows us that errors appear where the designer does not expect them to be. Another limitation of this approach is that for large systems, the RAM memory available might be not enough to handle the parts of the system we have decided to inspect. The reader interested in a more detailed discussion on these controlled partial search methods is encouraged to refer to [159].

In the second group fall the methods that instead of trying to predict in which parts of the system errors can be found, perform a random simulation, i.e. a *random walk* through the number of reachable system states. The main idea here is to organize the state space of the system so that it can be randomly simulated using precisely M bytes of RAM memory, i.e. the memory available for Spin in the target computer. Thanks to this strategy, systems of sizes that would overflow the available RAM memory in exhaustive search, can be validated in a comparatively small amounts of memory. Because of this and because of the difficulties mentioned above about validating methods that try to guess where errors can be found, Spin focuses on random simulation techniques when it comes to validating a large system. In particular, Spin implements a method called *supertrace* or *bit-state hashing*.

7.6 Supertrace controlled partial search

The supertrace technique was introduced into Spin to validate systems that cannot be validated in full verification mode because $R > R_{max}$, i.e. when the RAM memory requirements are beyond what is available in the target computer.

Let us say that we have a computer with M bytes of RAM memory which is not enough to store R and analyse the system in full exhaustive mode; what Spin does in supertrace mode is to organize the memory and system state space in such a way that it uses precisely M bytes of memory, no more and no less; them from the total system space state, Spin randomly selects the maximum number of states that can be analysed in M bytes of memory so that the coverage of the analysis is as large as possible.

If in full verification mode S bits of memory are required for storing a single state s_i, where S is normally between 10^1 and 10^3 bits, in supertrace mode a single bit can be used to manage the set of reachable states.

A crucial idea behind the supertrace mode is to save RAM memory by not storing each visited state s_i but storing only enough information to tell whether the state has or has not been analysed before. Because of the technique used to store the information about each visited state (explained below), this method is called *bit-state hashing* as well.

Although one bit is enough, in practice Spin uses two bits with the sole purpose of reducing hash conflicts. In fact though it can be set to 1-bit by manipulating the *1-bit or 2-bit hashing option* of Spin, the default is set to 2-bit; this implies two crucial things: First, that a pair of independent hash functions are used (one for each bit); and second, that a hash collision requires a collision on both bits; it is note worthing that this technique is known as *multi-bit hashing* [168].

For simplicity let us assume that we use Spin in 1-bit hashing mode, then the information for managing the set $A = \{s_0, s_1, s_2, ..., s_{R-1}\}$ of the reachable states of the system is the array $A_{hash} = \{f_0, f_1, f_2, ..., f_{R-1}\}$ of $R-1$ bits where each $f_i = 1|0$ is a binary flag. The address (the position in the array) of each f_i uniquely identifies a state. To access A_{hash} efficiently a hash function is implemented which for a given state s_i locates the corresponding f_i flag in A_{hash} to find out whether the flag is on or off. In the former case the state s_i is considered redundant, (i.e. it has been analysed before) and discarded. In the latter case, the state s_i is considered new; it is analysed and its corresponding bit in A_{hash} is switched on. The array A_{hash} is called the *hash table* and its size is normally equal to the maximum amount of RAM memory available in the target computer, i.e. R is normally equal to M expressed in bits. It is in fact the memory arena where the system is to be analysed.

Note that in supertrace mode Spin does not have a constraint on the number of reachable system states that can be validated since it attempts to select an optimal fraction of the full state space that can be searched using the available RAM resources. However, to use the available RAM and CPU efficiently it is always useful to have an idea about the number of reachable system states we are expecting as this tells the designer how many bytes of memory are needed to store the hash table. By efficiency here we have in mind that the maximum available RAM is used to store the hash table and without asking the target computer to validate a number of reachable system states that is beyond a reasonable computational time.

In practice the value of R is normally unknown, however, we can have a rough idea about it by running the validator just to learn from the output the number of states analysed and the value of the hash factor (explained below). The number of states analysed is reported as *transitions*. If the hash factor is high (over 100) the number of states analysed should be a good approximation of the actual number of reachable system states. Let us assume, that we roughly know the value of R; if we run Spin in 1-bit mode we will need at least $M = R \times 1$ bits of memory to store A_{hash} and at least $M = R \times 2$ bits if we run Spin in 2-bit mode.

Spin provides the option *-wN* to set the size of the hash table as 2^N bits, i.e 2^N entries in the hash table. By default this value is set to 2^{22} i.e. it allocates a hash table of $2^{22} = 4194304$ entries. This means that is we use the algorithm in 1-bit mode that table can handle up to 4 194 304 states; and up to $4194304/2 = 2097152$ if we use it in 2-bit mode. For this to be possible, Spin needs a

computer with at least 2^{22} bits of RAM memory, which is about $2^{19} = 524288$ bytes.

To estimate the coverage of a supertrace run, Spin introduces the notion of hash factor. The *hash factor* is defined as the size of the hash–table divided by the number of states stored [159, 168]. It is calculated by the validator after each run and displayed at the output as an indication of the coverage of the run. For example, is the size of the hash table is $2^{24} = 16777216$ bits and the number of states stored by Spin in the hash table is 111 553, the hash factor is 16777216/111553 = 150.395.

As the number of stored states approaches the size of the hash–table the hash factor approaches 1. A hash factor equal or greater than 100 implies good coverage; conversely, a hash factor near 1 implies poor coverage. The explanation for this is that as the number of stored states in the hash–table increased th number of hash collisions increases as well, this result in parts of the state space not being analysed (this is discussed in section 7.7).

As in exhaustive search (see section 7.4), CPU time is a serious constraint in supertrace mode. Empirically, we have learnt that in the Silicon Graphics computer we use to run Spin (see section 7.10) it takes about 8.0×10^{-5} seconds to analyse a state. For simplicity let us say the CPU time to analyse a single state is of the order of 10^{-5}. If that holds true it will take about one hour (i.e. 3600 seconds) of CPU time to analyse 3.6×10^8 states as $3.6 \times 10^8 \times 1 \times 10^{-5} = 3600$ seconds. Similarly, it will take about one year of CPU time to analyse 3.1×10^{12} states.

7.7 Hash conflicts

To allow for state comparison each system state analysed by Spin must be stored in memory where it can be retrieved efficiently and where new states are stored.

Spin uses (in both exhaustive and controlled partial search) a hash table to maintain the state space of the system being analysed. By applying a hash function to a given state, Spin tries to find out whether the state has or has not been analysed. If the state is found to be a new one (not previously analysed) it is analysed and included in the hash table. In the opposite case, the state is considered an old one (previously analysed) and it is discarded.

One of the most serious flaws encountered in the use of hash methods for constructing search tables is that although in theory it is possible to find one–to–one functions that assign one element and only one to each storage cell, in practice this has been shown to be impossible since normally the elements to be stored are seldom known a priori. Consequently, it is conceivable that the hash function eventually will attempt to assign more than one element to the same cell store; in the literature this issue is called a *hash collision*, *hash clash*, or *hash conflict* and has been well studied. Hash algorithms just accept that collisions are inevitable and rather than attempting to eliminate them entirely, they treat collisions as special case by calling a special procedure which will find an alternative cell store for the current element. This is what is called *resolving hash conflicts* and several techniques has been suggested to deal with the problem [169, 170, 171]; a widely used one is to include the clashing elements in a linked list in the hash table.

The hash algorithm used by Spin is not an exception, therefore, hash conflicts occur during the validation, however, hash conflicts here have a particularity: In a exhaustive full verification where each reached system state is fully stored in the hash table, Spin resolves hash collisions by resorting to a linked list.

Conversely, since the actual contents of reached states are not stored in supertrace mode, hash collisions cannot be corrected. When Spin finds a bit turned on, it cannot tell whether the current state has been analysed or its hash function has clashed. Because of this, Spin can erroneously believe that a state has been analysed, this result in the discard of the state and the failure to visit it and its successors. If this happens pieces of the code might be left unexplored and possible errors

not detected. In other words, in supertrace mode, Spin cannot guarantee 100% coverage due to the possibility of unresolved hash collisions. To reduce the effect of hash conflicts it is recommended to a technique called *sequential multihash* and *multiple hashing* [159, 168].

7.8 Sequential multihash

When due to hash collisions the designer is not satisfied with the coverage of his supertrace run, let us say when the hash factor is below 100, the designer can resort to sequential multihash. This technique consists of repeating the validation with alternate and independent hash functions. The main idea here is to move hash conflicts to a different part of the state space so that Spin can recover the fractions of the state space missed in the previous run.

As explained in detail in [168], by resorting to sequential multihash the level of coverage of supertrace validations can be as high as the designer wants it to be regardless of the size of the system. By continuously repeating the process (each time with a different pair of hash functions) the designer can get as arbitrarily close to 100% coverage as he wishes to [159, 168, 167].

Spin includes the runtime option *-RN* to allow the designer to specify that the supertrace run is to be repeated N times, each time with a different pair of hash functions (see section 7.6); current versions of Spin support up to 32 pairs of such functions. To give one example; the designer can instruct Spin to perform 16 independent supertrace runs by typing *% pan -R16*. This strategy will probably satisfy the most demanding designer regardless of the size of the system; the only problem here being CPU time, for large systems it might take hours to complete the run.

7.9 Correctness requirements

A crucial decision the designer of a protocol has to make is what correctness requirements (absence of deadlocks, mutual exclusion, temporal claims, etc.) to check his system on. This is extremely important not only because this will guarantee that the system is free of a particular kind of error, but also because the inclusion or exclusion of one of these requirements can have significant impact on the number of reachable states of the system and for instance on RAM memory and CPU time demand to validate them.

Although the correctness requirements that are usually validated in protocols are well-known [172], the list of correctness requirements the protocol designer tests her protocol on, depends on two factors: the particular characteristics of the system and the stage of development. The termination correctness requirement for example, can be important for one protocol but not required for other. Similarly, temporal claims are not normally tested in early stages of development, but at the final stages, when the protocol is free of the basic errors.

Promela provides well defined means of expressing different correctness criteria; namely, the designer can include in his Promela specification statements to prompt the Spin validator to check for the following correctness criteria of his system: assertions, system invariants, deadlocks, non-progress cycles, livelocks, and temporal claims.

7.9.1 Assertions and system invariants

An assertion is often expressed as a boolean condition inserted somewhere in the Promela code. It has the form of *assertion(bool_condition)* and is expected to be true whenever a process reached a given state. The assert statement has no effect if the boolean condition holds true; conversely, it generates an error message if the boolean condition becomes false.

If the designer wants a boolean condition to remain true in all reachable system states he can express this as system invariant. A system invariant is just a generalization of an assertion, it has the same form, *assertion(bool_condition)*, and is placed in a separate process that runs concurrently with the one the designer wants to validate; the assert statement is executed precisely once for every state of the system.

7.9.2 Deadlocks

Since Spin expects only systems with a finite number of states; it expects that the one it is validating either terminates after a finite number of state transitions or it goes back to a previously visited state. Both alternatives are considered a valid end to a process. Although the second alternative is not the final state of the system, it is considered and called *a proper end state* in Spin. If the system does not match this correctness criterion it is said to have a *deadlock*. In Promela, a proper end state is identified by a three–character prefix *end–state* label which has the form of **end***something*, where *something* is any sequences of characters accepted by Promela in names used as identifiers. Example of end–state labels are: *end, endcycle, end0, end1* and so on.

7.9.3 Progress cycles and livelocks

In Promela (and other programming languages) infinite cycles are considered correct behaviour for a process as long as the process goes through the states the designer expects.

To express that a process cannot cycle infinitely without visiting certain states Promela provides the statement **progress***something* to mark such states. States marked by such labels are called *progress–states* since the system must go through them to make any progress. An execution sequence that violates this claim is called a *non–progress cycle.*

To express that it is incorrect to cycle infinitely through a given state, Promela provides the statement **accept***something* to mark the state. Such state is called an *acceptance–state.* The name is a bit misleading and comes from the fact that a sequence of statements that contains acceptance–state labels is named an *acceptance cycle.* What we are saying here is that we want a system without any acceptance cycles. The job of Spin is to detect these acceptance cycles if there are any in the system.

As before *something* is any sequences of characters accepted by Promela in names used as identifiers. For example, *progress_svr, progressClt, accept0, accept1*, etc.

Acceptance cycles are also known as *livelock* since a process that goes infinitely often through states marked by acceptance lables is still doing something but trapped in a looop. It cannot escape from there and go through the states the designer wants it to go through.

7.9.4 Temporal claims

In some cases it is necessary to express that a state in which a certain condition is true cannot be followed by a state in which that condition or a different one is false. For example, the designer might want to express that if it is true that a channel with a single slot is full, it cannot remain full after reading a message from it. In Spin these correctness requirements are called *temporal claims* and in Promela are expressed with the help of the statement

```
never{Prom_statement1, Prom_statement2, Prom_statement3, ...}
```

where each *Prom_statement* is a Promela statement that contains the details of the claim; for example assertions, progress–states and acceptance–states labels.

7.9.5 Safety and liveness properties

In protocol validation, properties are grouped into two major classes: *safety properties* and *liveness properties*. Informallly, a safety property states that nothing bad never happens. Let us take a lift as an example. A safety property will state that if the lift is travelling or stopped between two levels its door will never open. On the other hand, a simple liveness property states that something good will eventually happen. Again, let us take a lift as an example. A simple liveness property will state that if a user has arrived at the intended floor, the door will eventually open. In other words, the passenger will eventually terminate his journey.

Another way of explaining safety and liveness properties is by saying that a safety property states what we do not want the system to do. Conversely, a liveness property states what we want the system to do.

These two concepts have been widely used in the literature devoted to correctness of concurrent programs since they were introduced by Lamport [173].

In Spin the concept of safety properties is used to group together assertions and system invariants, deadlocks, and unspecified receptions. On the other hand, non–progress cycles, livelocks and temporal claims fall in the class of liveness properties[159]. As explained in section 7.9.6, the designer can use Spin directives to instruct the validator to validate the properties he is interested in.

It is a well–known fact that it is always simpler specifying what we do not want from a system than specifying what we want, thus, it makes sense to begin the validation of a protocol by validating safety properties first and leave liveness properties for the last stages of the validation.

The reader interested in more details about safety and liveness properties is encouraged to refer to [173, 174, 175, 176] where these concepts are studied in depth.

7.9.6 Cost of correctness requirements

We have just discussed what correctness criteria can be specified in Promela to be validated by Spin; the order in which we introduced them reflects the level of sophistication in the validation and at the same time the cost of performing the validation in terms of RAM memory and CPU time demands.

Holzmann reports ([159, 162]) that it is comparatively cheap to validate assertions and absence of deadlocks. The computational cost for this is linear in the number of reachable states (R) of the system both in RAM memory space and CPU time. To check on progress cycles and livelocks can be twice as hard in terms of CPU time but there is not a noticeable increase in RAM memory requirements. The most expensive correctness criterion to validate is temporal claims. Compared to assertions and absence of deadlocks validation, the cost can be $2N$ times as hard, where N is the number of reachable states in the sequence of statements contained in the *never*{*Prom_statement1,Prom_statement2, Prom_statement3,...*} claim.

It is important to notice that Spin allows us to validate these correctness criteria separately (for example check the system for non–progress cycles only, or for acceptance cycles only) so that the simpler requirements do not contribute to the cost of the more sophisticated ones.

The selection of the correctness criteria to validate is made with the help of Spin directives and options before running the validator: At compilation the designer can specify the *-DSAFETY* directive to indicate that he is interested only in validating safety properties, the definition of safety is explained in section 7.9.5. Similarly, the directive *-DNP* indicates that the designer wants to check on non-progress cycles.

Once the validator is compiled with the appropriate directives, runtime options are used to

check the desired correctness criteria: The execution of *% pan* will explore safety properties of the system. Likewise, the execution of *% pan -a* checks the system on acceptance cycles. Finally, the execution of *% pan -l* will find non-progress cycles.

7.10 Validation platform

To validate our system Spin Version 3.3.5 —28 September 1999 was downloaded from the Internet [177], gcc-compiled, and installed in the following platform:

Computer Silicon Graphic

Model Octane

CPU 2 175MHz MIPS R10000(IP30) Processors with MIPS R10010 FPUs

RAM memory 512 Mbytes

Operating system IRIX64 Release 6.5

gcc compiler gcc version 2.8.1

According to the terminology introduced in section 7.4, we have a computer with $M = 512000000$ bytes. Let us make an estimation of the RAM resources we have at our disposition to validate a system using Spin in full exhaustive and supertrace mode.

In full exhaustive mode and considering that for middle size system the number of bytes to store a single state is of the order of $S = 1 \times 10^2$ (see section 7.4); we can say that in our Silicon Graphic platform we can validate systems with $R_{max} = M/S = 512000000/1 \times 10^2 = 512 \times 10^4$ states. Naturally, if the system is extremely large (with $S = 1 \times 10^3$) we can analyse only $R_{max} = 512 \times 10^3$ states.

By default, in supertrace mode each state occupies only two bits, so $S = 2$; it follows that in a memory of $M = 512000000 bytes = 512000000 \times 8 bits = 4096000000$ bits we can store up to $R_{max} = M/S = 4096000000/2 = 2048000000$ states. To store information about these states we need to allocate a hash table of 4096000000 bits; since it is common practice to express the size of the hash table as 2^N and $2^{31} = 2147843648$ and $2^{32} = 4295687926$ we can say that the largest hash table we can allocate in our computer is 2^{32}.

This is where the CPU time constraint discussed in sections 7.4 and 7.6 have to be taken into account. The available RAM memory allows us to store up to 2048000000 states; however; before any attempt to allocate a hash table for such a number of states it is advisable to have a rough idea about the CPU time needed to analyse them; accordingly with our empirical observations, it take about 8×10^{-5} seconds to analyse a single state (see section 7.6); it follows that the CPU time to analyse 2048000000 states amounts to $CPU_{time} = 2.8 \times 10^9 \times 8 \times 10^{-5} = 2.24 \times 10^5$ seconds; which is about 61 hours.

Similarly, we can estimate that if the designer does not want to spend more than one hour waiting for his results he can analyse only about $3600/8 \times 10^{-5} = 4.5 \times 10^7$ states; to analyse such a number of states in 2-bit mode Spin needs a hash table of $4.5 \times 10^7 \times 2 = 90000000$ bits; since $2^{26} = 67108864$ and $2^{27} = 134217728$; we have to set the size of the hash table equal to 2^{26} to ensure a computation time equal or less that one hour. A supertrace analysis with a hash table larger than 2^{26} becomes prohibitively expensive and has to be avoided even though it provides higher coverage. From this discussion it should be clear that there is a trade-off between run-time and coverage in supertrace mode. Also, it should be clear that we are not able to take full advantage of the 512 000

000 bytes of RAM memory available in our target computer. If only we had a more powerful CPU (say a CPU with as many processors as possible) the story of the coverage would be different.

7.11 An estimation of the size of our system

As can be appreciated from the Promela specification of the system, a special effort was made to keep the size of the system as small as possible in terms of reachable number of states. With the sole purpose of reaching this crucial goal, the number of variables, channels, slots in channels and processes was reduced to the absolute minimum without losing the generality of the validation. In particular, except for two exceptions, all global channels were initialized with a single slot; as for local channels, they were initialized either with a single slot or with three. Similarly, the number of PDAs in the system in the area of coverage of the MSS was set to three. We believe that the interaction of three PDAs with the MSS is enough to explore the whole system and bring to light any possible error in the protocol; moreover, three PDAs is still a reasonable number if we intend to validate the system on a desktop workstation equipped with standard current technology resources.

Besides the effort to keep the system small the resulting system can be easily categorized as a large one. Including comments, the whole of the Promela specification of the system is about 1500 lines long.

Perhaps more relevant than the number of Promela lines is the amount of RAM memory needed to validate such a system and how long it will take to run the Spin validator.

For the moment, let us assume that we want to validate by running the Spin validator in full exhaustive search mode. As we discussed in section 7.4 the amount of RAM needed can be estimated by multiplying R (the number of reachable states of the system) by S (the number of bytes required to store each state). Both S and R are unknown, yet they can be roughly estimated from the Spin output.

With the sole intention of estimating the size of the system and without any attempt at detecting any possible errors in the system, one can run the Spin validator in full search mode and observe its output; even if the validation is not complete due to lack of memory, the designer can still learn the value of S from the output where it is reported as the size in bytes of the *State-vector* or something similar, depending on the Spin version. By this approach we estimated that our system requires about 1600 bytes to store each reachable state.

Similarly, R can be estimated by running the Spin validator in supertrace mode and observe the hash factor and the number of stored states at the output. Following this strategy we learnt that in one supertrace run Spin stored 1.81351×10^{07}; let us assume momentarily that this is the true number of reachable system states. Then M (the amount of memory to analyse the system in full exhaustive mode) will be $M = R \times S = 1.81351 \times 10^{07} \times 1.600 \times 10^{3} = 2.901616 \times 10^{10}$, i.e. we are talking of a RAM memory of the order of 30 Gbytes. Moreover, a low hash factor shown in the output indicates that not all reachable states were stored, some of them were missed due to hash conflicts (see section 7.7); thus, $R = 1.81351 \times 10^{07}$ is just a poor approximation that tell us that to analyse our system in full exhaustive mode we need at least 30 Gbytes of RAM memory.

7.12 Avoiding paging

In [168, 178] Holzmann shows why it is not recommended to use disk memory to store the system state. In theory it is possible, however, in practice it is strongly not recommended as the validation will dramatically slow down due to read/write operations on the disk any time a new state is generated. In fact even if the designer has no intentions of using disk memory it is strongly

recommended to avoid paging so that the speed of the verification is maintained within acceptable limits. By default (see the *pan.h* header file created by Spin) Spin sets the memory bound to run the validator to $2^{25} = 33554432$ or $2^{28} = 268435456$ bytes, depending on whether the target computer is a PC or not; if the default is too small Spin will waste memory, conversely, if the default is too big, the computer will endlessly page. If the designer knows an exact memory bound, he can set it by using either the *-DMEMCNT=N Mbytes* or *-DMEMLIM= N Mbytes* directive during the compilation of the Promela code. The latter directive is supported by Spin 3.3.0 and more recent versions.

In section 7.10 we learnt that $M = 512000000$ for the computer we are using; accordingly, it is advisable to set our bank of memory at 500 Mbytes in order to avoid paging. We can use either the *-DMEMLIM* or *-DMEMCNT* compile directives as follow: $-DMEMLIM = 500$ Mbytes or $-DMEMCNT = 2^{28}$ bytes. Note that $2^{28} = 268435456$ and $2^{29} = 536870912$. As can be seen the former directive gives a more precise control of the memory.

7.13 Reduction of complexity of the systems

Regardless of how much RAM memory and CPU power the designer has available in his computer, there always will be a system that will exhaust his computational resources. In situation like this he can try to validate correctness criteria separately as recommended in section 7.9.6; likewise, he can try to reduce the number of reachable states of his system by decreasing the number of processes, variables and slots in channels; if this does not help he has to resort to a rather different approach; namely, to divide his system into modules (protocol layers) and validate the modules separately.

The flaw of this approach is that it works well when the system can be relatively easily separated into independent modules so that the correctness of each module validated separately holds when all of the modules are put to work together. Except for extremely well defined protocol layers, the separation of a system into modules is not trivial. Also, a separate module is tested with a module tester that plays the rôle of the upper and lower layer of the module being tested. Normally, the module tester is in charge of injecting inputs to the module, receiving its outputs and possible feedbacking some of the outputs. Because of this, there is always a risk that the module tester can introduce spurious errors to the system or hide real ones. Other techniques to reduce the complexity of a system are discussed by Holzmann in [159]; we do not discuss them because they are not relevant for the validation of our system.

7.13.1 Separate and monolithic validation

From the discussion presented in sections 7.11 and the technical information presented in section 7.10 it is clear that the size of our system clearly overflows our computational resources.

For a large system like ours, a full exhaustive search validation is infeasible. Consequently, the designer has to resort to other techniques; namely, to supertrace search (see section 7.6) and modular validation (see section 7.13).

One particular characteristic of our system is that its processes are tightly interrelated; for example, to truly see how PDAi behaves while sending anonymous messages, all of the processes (the public key manager, the secret and TmpId manager, the bank, the mail server, and other PDAs) must be active because the PDAi needs the public key manager, the secret and TmpId manager, the bank and the mail server to perform its work and because other PDAs might have an impact (hit its secret key for example) on PDAi

It can be argued that our system can be separated into independent modules and validate each module separately; we certainly considered this possibility, however, we found that due to the

strong interrelation of the component processes, the validation of one of them independently of the others oversimplified the work of the process and rendered the model and the validation almost meaningless. After all, if there is an error in the protocol, we believe that it is likely to be in the interaction of the processes all together rather than in the interaction of two of then or inside the code of one of them.

Because of this we make a special effort not to indiscriminately divide the system into modules; in fact the only modules we believe that can be safely separated and validated independently of the rest of the system without losing the essential features of the validation are: the public key manager, the bank, and the mail server.

7.14 Selection of correctness requirements

As it can be appreciated from the discussion presented in section 7.10, checking the correctness of a system of middle to large size on all the correctness criteria might be extremely expensive and infeasible due to lack of computational resources. However, in most cases this level of sophistication is not required, particularly at an early stage in the development of a system, the validation of basic safety properties (assertion and system invariants, absence of deadlocks) and proper end–states is more than enough [162, 159].

It is time now to consider what correctness requirements we would like to impose on our anonymizing system. In accordance with our computer resources and stage of development we assume that it is enough to validate our system on the following list of requirements:

- t units of time after arriving at the MSS a PDA must either learn the public key of the MSS or terminate.
- t units of time after learning the public key of the MSS a PDA must obtain a session key and a TmpId or terminate.
- At any time, each PDA in session with the MSS must have a unique session key and unique TmpId.
- A TmpId must last for the duration of the anonymous session.
- A session key can be changed at any time by initiative of the MSS or the PDA.
- Having finished its anonymous session (voluntarily, forced by the MSS or due to an unexpected interruption of the communication channel) a PDA must terminate.
- A server processes (public key manager, mail server, bank server and the KsTmpId manager) must never terminate.
- A payment must be accepted only if the e–cash is genuine and its value is within a certain range (minimum and maximum accepted payment).
- Having provided the appropriate payment a PDA must be credited with an anonymous and finite session time.
- The anonymous session must be terminated by the MSS when the credit for a PDA expires.
- Replies to anonymous messages sent by the PDA are optional. If at the end of an anonymous session some messages are left unanswered the PDA must not block but terminate and the resources allocated to it at the MSS's side must be released.

From the list of correctness requirements one can see that at this stage the emphasis of the validation is on creating the basis for the anonymous communication. Our main aim is to guarantee that each PDA has the right TmpId and the right session key. To put it in abstract words, we can say that we are dealing with a problem of mutual exclusion which can be validated by using safety properties. On the other hand we put emphasis on process termination. More precisely, on avoiding invalid end–states. Process termination belongs to the class of liveness property, however, in Spin it can be validating by using its safety properties directive (see section 7.9.6), since this directive instructs Spin to check that each process terminates in what is considered a proper end–state (see section 7.9.2). In our case, PDAs must always terminate and disappear and servers must always stay and wait for new clients to arrive.

Notice that the validation process has been simplified. At this stage we are not interested in validating non–progress cycles and temporal claims. Our aim is to validate only basic correctness properties. This simplification inevitably reduces the strength of our validation and leaves some windows where potential errors might appear. This is discussed later on in section 7.20.

7.15 Validation of the public key manager module

Recall from the discussion presented in section 6.6.4 that the goal of the public key manager process is to broadcast periodically to the air through a well–known (to the PDAs) port a message containing the public key of the MSS. On the other hand, PDAs interested in learning the MSS public key of the MSS receive the message and verify that the Kpu is correct by checking the signature of the person or institution who vouches for the key. If the Kpu received happens to be incorrect, the PDA waits for the next broadcast till eventually it gets a correct key or a timeout mechanism forces it to terminate. It is worth mentioning that in our Promela code a counter that counts the number of attempt is used instead of a timeout, if the maximum number of attempts is reached the process terminates.

To claim that the Promela code that describes this protocol is correct we have to show two things: First, we have to show that after a certain amount of time each PDA involved is in possession of a certified Kpu or it gives up and terminates. Second, we have to show that the protocol complies with the correctness requirements discussed in section 7.9.

To show that a PDA is in possession of a certified Kpu we use output messages to announce that a PDA is in possession of a Kpu key. To show that the Promela code is correct is more complicated; it is done by analysing the Spin output.

The interaction between the PDA and the MSS is relatively simple; to prove that the protocol is correct it is enough to prove that after a certain amount of time each initialized PDA terminates. By termination we have in mind that each PDA process either finishes its execution or it remains in a valid end state (see section 7.9.2). Also we want to be certain that the value of the Kpu learnt by PDAs belong to the domain of certified Kpu. On the other hand, we have to prove that the KpuMan process cycles back to a valid end state each time after a Kpu is broadcast. To validate a system like this it is more than enough to check its safety properties (see section 7.9.6).

The Promela specification of the KpuMan is relatively simple, however, there is one issue that deserves a few lines of comments here: As it was discussed in section 6.6.2, Promela support only one–to–one communication, neither multicast nor broadcast communication is supported in Promela; for this reason the broadcast of the Kpu key to the air has to be simplified. To get around the problem we do not actually broadcast the Kpu key, it is just placed by the KpuMan in the output channel, where it is eventually read by a PDA, when the Kpu is read, the KpuMan places another one, to be read possible by another PDA, and so on.

The Promela specification of the KpuMan can be categorized as a small system. As can be seen in appendix A.1, the whole code (comment lines included) is about 120 lines.

7.15.1 Simulation results

The results of the KpuMan simulation validation are shown below. Note that the line numbers are not part of the Spin output; they are there to simplify the discussion.

```
% spin -n19 KpuMan  /* feed random generator with n=19 to run Spin simulator */
01:  PDAnum=2 GOT CERTIFIED Kpu= 12 at the NumAttempts==1th
02:  PDAnum=0 GOT CERTIFIED Kpu= 12 at the NumAttempts==3th
03:  PDAnum=1 Failed to get Kpu after NumAttempts= 5 ; going to +++Aborted+++
04:                          PDAnum=1  +++ABORTED+++ BYE-BYE-BYE
```

As can be appreciated from the above lines, at the end of an arbitrary run of the Spin simulator each PDA is either in possession of a certified Kpu key or aborted. In this case, PDA with numbers 2 and 0 managed to get a certified Kpu, while PDA with number 1 fails and terminates in abortion (line 04).

To support the results of the Spin simulator, we will validate the system using the Spin validator.

7.15.2 Validation results

By following the approach presented in section 7.11 we can learn that the number of reachable states in the system to check safety properties is approximately $R = 140415$ and that the number of bytes needed for each state is about $S = 84$. It follows that the memory needed to validate this system is $R \times S = 11797380$ bytes, i.e. only about 12 Mbytes. We can certainly validate the KpuMan in full exhaustive mode using our platform with 512 Mbytes.

By compiling the KpuMan code by *% cc -DMEMLIM=16 -DSAFETY -o pan pan.c* we obtained the executable file called *pan*. Since we expect about $R = 140415$ states, and R is between $2^{17} = 131072$ and $2^{18} = 262144$, it is recommendable to execute *% pan -w18*; this means a hash table with 262144 entries. Recall from section 7.4 that the option *-wN* specifies the size of the hash table as 2^N states to be stored.

Needless to say that there is no risk of exhausting our RAM memory since $262144 \times 84 = 22020096$ bytes is less than our M of 512 000 000 bytes. In regard to CPU time, it takes only about 5 seconds to complete the validation.

```
% spin -a KpuMan /* generate an analyser for the Promela specification */
                 /* of KpuMan and store it in pan.c                     */

% gcc -DMEMLIM=16 -DSAFETY -o pan pan.c /* compile the pan.c verifier into the pan file */
                                        /* analyse SAFETY properties only, use no more  */
                                        /* than 16 Mbytes                               */
% pan -w18 /* allocate a hash table for 2^18= 262144 states and run the Spin validator  */
01: (Spin Version 3.3.5 -- 28 September 1999)
02:         + Partial Order Reduction
03:
04: Full statespace search for:
05:         never-claim             - (none specified)
06:         assertion violations    +
07:         cycle checks            - (disabled by -DSAFETY)
08:         invalid endstates       +
```

```
09:
10: State-vector 84 byte, depth reached 123, errors: 0
11:   140415 states, stored
12:    47624 states, matched
13:   188039 transitions (= stored+matched)
14:       16 atomic steps
15: hash conflicts: 26215 (resolved)
16: (max size 2^18 states)
17:
```

The above lines show that after an exhaustive search no errors concerning safety properties were detected. Spin printed the number of states stored and matched (lines 11 and 12 respectively). Stored states are the states that were stored in the hash table. Finally, matched states are states that were analysed and stored in the hash table and later revisited and discarded. The output also shows (line 15) that during the search 26215 hash conflicts were resolved (see section 7.7).

7.16 Validation of the mail server

From the discussion presented in section 6.6.8 we learnt that the MailSvr process is linked to each MSSses process serving a PDA who has successfully opened an anonymous session. From each MSSses process the MailSvr receives messages Bob want to send to Alice. When a message arrives the MailSvr forwards it to its final destination. Also the MailSvr process receives Alice's replies to Bob. When an Alice's reply arrives, it forwards it to the corresponding MSSses in order that the MSSses forwards it to Bob.

7.16.1 Simulation results

To show that the Promela specification of the MailSvr is correct the output of have to show that the MailSvr receives messages coming from MSSses processes and that the MSSses processes receives replies to some of the messages. Note that replies to Bob messages are at Alice's discretion. She is not expected to reply to all of the messages she receives.

```
% spin -n12 MailSvr /* feed random generator with n=12 to run Spin simulator */
01 MailSvr: has rcvd a msg from PDA tmpId=1

02 MailSvr: has rcvd a msg from PDA tmpId=2
03 MSSses:  rcvd reply for  PDA tmpId= 2

04 MailSvr: has rcvd a msg from PDA tmpId=1
05 MSSses:  rcvd reply for  PDA tmpId= 1

06 MailSvr: has rcvd a msg from PDA tmpId=2

07 MailSvr: has rcvd a msg from PDA tmpId=0
08 MSSses:  rcvd reply for  PDA tmpId= 0
...
21
22 MSSses: tmpId=0 has finished: HAPPY END BYE-BYE-BYE
...
26 MSSses: tmpId=1 has finished: HAPPY END BYE-BYE-BYE
...
30 MSSses: tmpId=2 has finished: HAPPY END BYE-BYE-BYE
31
32
```

From the above output lines we can see that in line 01 the MailSvr receives a messages from the PDA with tmpId=1. Yet there is no reply for it. In line 02 the MailSvr receives a messages from the PDA with tmpId=2 and in line 03 the MSSses receives a reply to that messages. Lines 04 to 08 show similar information. In lines 22, 26, and 30 the output shows the termination of PDA sessions with tmpId=0, tmpId=1, and tmpId=2 respectively.

The output of the Spin simulator is not enough to claim that the Promela specification for the MailSvr complies with the correctness requirements; to prove this we have to validate it using the Spin validator and prove safety properties.

7.16.2 Validation results

By following the approach presented in section 7.11 we empirically learnt that the number of reachable states expected for this system is about $R = 293934$ and that the number of bytes needed to store each state is $S = 240$. It follows that we need about $R \times S = 300000 \times 240 = 70544160$ bytes of RAM memory. Naturally, we can validate this system in full exhaustive mode in our platform omputer with 512 Mbytes. If $2^{18} = 262144$ and $2^{19} = 524288$ We need a hash table with $2^{19} = 524288$ entries only. As to the CPU time, it takes only about 20 seconds to complete the validation.

```
% spin -a MailSvr /* generate an analyser for the Promela specification */
                  /* of MailSvr and store it in pan.c                    */

% gcc -DMEMLIM=500 -DSAFETY -DVECTORSZ=512 -DPC -o pan pan.c /* compile the */
                  /* validator in pan.c into the executable pan file       */

% pan -w19  /* allocate a hash table for 2^19= 524288 states and run the Spin */
            /* validator                                                      */

01  (Spin Version 3.3.5 -- 28 September 1999)
02        + Partial Order Reduction
03
04 Full statespace search for:
05        never-claim             - (none specified)
06        assertion violations    +
07        cycle checks            - (disabled by -DSAFETY)
08        invalid endstates       +
09
09 State-vector 240 byte, depth reached 192, errors: 0
10  293934 states, stored
11  158380 states, matched
12  452314 transitions (= stored+matched)
13       0 atomic steps
14 hash conflicts: 57500 (resolved)
15 (max size 2^19 states)
16
```

The output lines above show that 293934 states were stored, that 57500 hash conflicts were resolved and that no errors were detected by the validator.

7.17 Validation of the bank server

From the discussion presented in section 6.6.7 we already know that the job of the bank process is to verify the genuineness of e–cash sent by PDAs to the MSS as payment for an anonymous session. It is the MSSses process who forwards the e–cash received from a PDA to the bank and then wait for a reply. The reply is either *GenuineEcash* or *FakeEcash*; naturally, in the former case the payment is accepted by the MSS and in the later refused.

To prove that the Promela specification of the bank process is correct we have to show that the bank indeed receives and e–cash and replies to the MSS. Note that for the sole purpose of validating the bank Promela specification the e–cash sent by the MSSses process to the bank is created by the MSSses itself; in the whole system this e–cash comes from a PDA.

7.17.1 Simulation results

The following lines show the output lines of the Promela simulator when its random number generator is initiated with the value of 100. Note that the following assumption were made in the Promela specification: First, e–cash of value –1 are considered a fake; second, cash of value greater than 0 are considered genuine; finally, the number of attempt to send a valid payment is restricted to 3; in no more that three attempts the PDA either succeeds and goes to a state where the anonymous session is supposed to begin and block, or terminates in abortion,

```
% spin -n100 bank /* feed random generator with n=100 to run Spin simulator */
01 BANK: rcvd FakeCash payment=-1 from PDA with tmpId=0
02 BANK: rcvd FakeCash payment=-1 from PDA with tmpId=1
03 BANK: rcvd FakeCash payment=-1 from PDA with tmpId=0

04 BANK: rcvd GenuineEcash payment=15 from PDA with tmpId=2
05
06 MSSses: AnoSes opened for PDA with tmpId=2 (payment=15 )

07 BANK: rcvd FakeCash payment=-1 from PDA with tmpId=0
...   ...   ...
10 MSSses: PDA with tmpId=0 failed to pay: +++ABORTED++++ BYE-BYE-BYE

11 BANK: rcvd FakeCash payment=-1 from PDA with tmpId=1
12 BANK: rcvd GenuineEcash payment=20 from PDA with tmpId=1
13
14 MSSses: AnoSes opened for PDA with tmpId=1 (payment=20 )
15 ...   ...
```

As can be appreciated from the above lines, the PDA with tmpId=0 sends fake e–cash to the bank in its three attempts (lines 01,03,and 07); consequently, it fails to pay for the anonymous session and terminates in abortion (line 10).

The bank receives a fake payment from the PDA with tmpId=1 on lines 02 and 11; however, on line 12 it receives a genuine e–cash from the PDA, thus, an anonymous session is opened for it in line 14.

The PDA with tmpId=2 is the most efficient of the PDA group, it succeeds in sending genuine cash in its first attempt (line 04), thus, an anonymous session is opened for it in line 06.

According with the results, we can argue that the Promela specification of the bank does what we expect, however, to claim that the system is correct we have to verify that the correctness criteria we include in the code, namely, safety properties, hold true by validating it with the Spin validator.

7.17.2 Validation results

As can be appreciated from the Promela code presented in section A.3, the bank process contains (comment lines included) 197 lines. Following the strategies introduced in section 7.11 we learnt that the expected state space for the system is about 764346 states and that we need 232 bytes to store each state. It follows that we need roughly $764346 \times 232 = 177328200$ bytes to analyse the system in full exhaustive mode. We can certainly analyse it in our platform of 512 Mbytes. To use the RAM memory efficiently we can instruct Spin to allocate a hash table for 764346 states (see section 7.4. Since $2^{19} = 524288$ and $2^{20} = 1048576$, we need to set the *-wN* Spin option as *-w20*. From the point of view of CPU time, the system can be considered small, it takes only about 47 seconds for the validation to complete.

```
% spin -a bank /* generate an analyser for the Promela specification */
               /* of bank and store it in pan.c                       */

%  gcc -DMEMLIM=500 -DSAFETY -DPC -DVECTORSZ=256 -o pan pan.c /* compile the     */
             /* verifier stored in pan.c and generate an executable file in pan */

% pan -w20 /* allocate a hash table for 2^20= 104857 entries and run the Spin validator */
01 Fri Dec  3 17:33:53 GMT 1999
02 (Spin Version 3.3.5 -- 28 September 1999)
03        + Partial Order Reduction
04
05 Full statespace search for:
06        never-claim             - (none specified)
07        assertion violations    +
08        cycle checks            - (disabled by -DSAFETY)
09        invalid endstates       +
10
11 State-vector 232 byte, depth reached 148, errors: 0
12  764346 states, stored
13  281699 states, matched
14 1.04604e+06 transitions (= stored+matched)
15       0 atomic steps
16 hash conflicts: 115543 (resolved)
17 (max size 2^20 states)
18
...   ...
```

The above output lines show that the Spin validator stored 764346 states, that the space occupied for each state was 232 bytes. Spin analysed a total of 1046040 states (transition states), 281699 of them are reported as matched, i.e. visited more than one time. Next, in line 16 Spin reports 115543 hash conflicts resolved (see section 7.7). Finally, and the most important, Spin reports in line 11 that no errors were detected during the validation, thus, we can claim that the system is correct.

7.18 Validation of the backbone of system

From the discussions presented in sections 7.9.6 and the technical information about available memory resources presented in 7.10 we learnt that it is infeasible to validate our system by full exhaustive search. In section 7.13.1 we argued why it is not recommended to separate the system indiscriminately into modules and validate each module separately. Because of these reasons we

decided to validate together as many modules as possible. From the whole Promela specification we have separated the KpuMan, the bank and the mail server and nothing else. What is left is what we call the *backbone* of the system since it includes the main components, namely, the three PDAs, the Ks and TmpId manager, and the three MSS session process.

7.18.1 Simulation results

To prove that these processes do what the designer expects, we have run the Spin simulator and analysed its output lines. Since we are dealing with the main components of the system, the output lines here have to show the main aspects of the algorithm we presented in section 5.10.3. Once again, here Bob is a PDA user and Alice is the destination of Bob's messages.

- Each PDA arriving at the MSS either registers with the MSS and is given a Ks key and a TmpId or terminates. In addition, we have to show that a Ks key can be renewed during the anonymous communication session.

- In an actual implementation of the system the Ks key is used to encrypt/decrypt messages travelling from the PDA to the MSS and in the opposite direction. Since the encryption and decryption of messages are arithmetic operations and irrelevant to the communication they are not shown here.

- We show that Bob can send an e–coin as a payment to the MSS and that the e–coin can be accepted or rejected by the MSS. In addition, we show that after certain amount of time the prepaid time expires and Bob either extends it by sending another e–coin or accepts the abrupt termination of his session. N seconds before the prepaid time expires, Bob receives a warning message.

  In an actual implementation; the time to warn the PDA user and the time to terminate the session are measured by a timeout mechanism; in our validation model, and for the sake of simplicity, we use a counter that decrements one unit of time for every message sent by Bob to Alice. We consider that this does not affect the behaviour of our system.

Note that Alice's replies are not shown in the output of the backbone of the system because the mail server module is out of the validation, this output will be shown is section 7.19 where the whole system is simulated.

Before analysing the output of the simulator it is important to mention that the PDA Promela code is composed of three layers (the PDAuser, PDAses, and PDAtcp), hence, when a PDA finishes it interaction with the MSS, all of these three processes must terminate and the resource (the channels and the TmpIds for example) they use must be released.

This observation, however, immediately leads to an obvious question, can these resources be reused? In theory the answer is yes; they can be reused by new PDAs, however, for this to be safe, we would need a mechanism to recycle the identifiers used to name them. Not to be involved with additional complexity irrelevant to the communication protocol, we decided not to reuse any resource.

Another restriction to keep in mind is that in the Promela specification we restricted the values of valid session keys to 1, 2, 3, and 4; other values are considered invalid. Similarly, valid values for TmpId are: 0, 1, 2, 3, and 4; other values are considered out of domain. Finally, the minimum amount of money accepted for opening or extending an anonymous session is 15 and the maximum accepted is 100. Values less than 0 are considered a fake.

What follows are the output lines of a single run of the simulator whose random number generator was initialized with the arbitrary value of 3456.

```
% spin -n3456 SysBackBone /* feed the random generator with 3456 */
                          /* and run Spin simulator               */

001 PDAses: PDAnum=1; pid=4 RgTED Ks=3 (tmpId= 0) after i=1 attpt
002 MSSses: rcvd FakeEcash | too little or to big money=1 from PDA-tmpId=0

003 PDAses: PDAnum=2; pid=7 RgTED Ks=1 (tmpId= 1) after i= 1 attps
004 MSSses: rcvd GenuineEcash=20 from PDA-tmpId=1

005 MSSses: rcvd FakeEcash | too little or to big money=1 from PDA-tmpId=0

006 PDAses: PDAnum=0; pid=5 RgTED Ks=2 (tmpId= 2) after i= 1 attps
007 MSSses: rcvd GenuineEcash=20 from PDA-tmpId=2

008 MSSses: credit=10 CreditLeft=10 for PDA-tmpId=1

010 PDAses: PDA-tmpId=1 ASKING to change OldKs=1

011 MSSses: rcvd GenuineEcash=20 from PDA-tmpId=0

042 PDAses: tmpId=1 GOT new Ks OldKs=1 NewKs= 4

083 MSSses: credit=0 CreditLeft=7 for PDA-tmpId=1

085 PDAuser: PDA-tmpId=1 has been TimeAlerted
086 PDAuser: PDA-tmpId=1 has sent EXTRA payment=20

093 MSSses: credit=0 CreditLeft=5 for PDA-tmpId=1

099 MSSses: PDA-tmpId=1 SENT extra payment=20 to bank

102 MSSses: credit=0 CreditLeft=8 for PDA-tmpId=2

109 PDAuser: PDA-tmpId=0 has been TimeAlerted

111 PDAuser: PDA-tmpId=0 has sent EXTRA payment=20

115 PDAuser: tmpId=1 AnoTime INCRTED

118 PDAuser: PDA-tmpId=2 has been TimeAlerted
119 PDAuser: PDA-tmpId=2 has sent EXTRA payment=0

124 MSSses: credit=0 CreditLeft=5 for PDA-tmpId=0

126 MSSses: PDA-tmpId=2 SENT extra payment=0 to bank

128 MSSses: PDA-tmpId=0 SENT extra payment=20 to bank

129 MSSses: credit=14 CreditLeft=10 for PDA-tmpId=1

134 PDAuser: AnoTime NOT incremented for PDA-tmpId= 2

135 PDAuser: tmpId=0 AnoTime INCRTED
...
```

As can be seen from the output lines. There are three PDAs interacting with the MSS. To be able to tell them apart before they are given a proper tmpId by the MSS we called them PDAnum=0, PDAnum=1 and PDAnum=2.

- On line 001 the PDAnum=1 manages to register a Ks=3 with the MSS and in return it is given a tmpId=0. Likewise, on line 003 the PDAnum=2 registers a Ks=1 and receives the tmpId=1. Finally, on line 006 the PDAnum=0 is assigned the tmpId=2 after registering Ks=2. Note that both the session keys and the tmpIds are unique. From now on we will refer to PDAs by they tmpIds.

- On line 002 the PDA with tmpId=0 tries to open an anonymous session with the MSS, unfortunately the money it sends is below the minimum payment accepted by the MSS, consequently the payment is refused. On line 005, it fails again. Finally, it manages to pay for the anonymous session on line 011 where it sends 20 units of genuine money.

- On line 004 the PDA with tmpId=1 successfully pays for opening an anonymous session. It sends 20 units of genuine money to the MSS. From the total payment, 10 units of money are reserved as *Creditleft* and the rest as *credit* (see line 008). The *credit* part of the money is immediately used to charge the PDA user for every single message he sends to the MSS. The *CreditLeft* money is used after the *credit* is run out and the PDA user is alerted about his communication time being about to finish. When the *CreditLeft* money runs out, the communication session is abruptly terminated.

- On line 083 we can see that the *credit* money for PDA with tmpId=1 has run out. Thus, on line 085 the PDA is time–alerted; as a result, it sends 20 units of money to the MSS to extend its communication session on line 086. The time extension takes place on line 115. Note that after the PDA is time–alerted and before the money to pay for an extension arrives, the MSS is still serving the PDA and charging it from the *CreditLeft* money (line 093) until it runs out or new money arrives. The result of the extension can be seen on line 129, where the PDA shows *credit=14* and *CreditLeft=10*.

- On line 118 the PDA with tmpId=2 is time–alerted. Thus, it tries to extend its anonymous session by sending a payment=0 to the MSS (line 119), although the payment is genuine for the bank, it is not within the rank of accepted values for the for the MSS, consequently, the payment is refused by the MSS on line 134 and the anonymous time is not incremented..

- The PDA with tmpId=0 is time–alerted on line 109, thus, it sends 20 units of money on line 111; the payment is accepted and the anonymous time incremented on line 135 for the PDA.

- Each PDA can ask the KsTmpId process to change the PDA's Ks key any time during the anonymous communication session. The PDA with tmpId=1 decides to ask for a Ks change on line 010; as a result, it receives a new Ks on line 042.

The output of the Spin simulator is not enough to claim that the backbone of the system is correct; to prove that the system complies with the correctness criteria included in its Promela specification it has to be explored by the Spin validator.

7.18.2 Validation results

The Promela specification of the system backbone is shown in appendix A.4. In terms of number of lines, the size of the system is about 1300 lines (with comment lines included). The precise number of reachable system states and Mbytes of memory needed to validate this system is impossible to know. Yet by following the approach presented in section 7.11 it can be estimated that the system has at least 4.14323×10^7 states and each state occupies about 1780 bytes. The amount of RAM memory to

validate this system in full exhaustive mode would be $4.14323 \times 10^7 \times 1.780 \times 10^3 = 7.374949 \times 10^{10}$, which is approximately 73 Gbytes of memory. It is clear that we cannot validate such a system in our platform using Spin in full exhaustive mode. We have to resort to supertrace mode and take the best of it.

From section 7.10 we learnt that there is a trade–off between run–time and coverage in supertrace mode. Also, we estimated that the CPU time to analyse 4.5×10^7 states is about one hour and that the size of the hash table for such a number of states is 2^{26}.

We decided that CPU times longer than one hour for a single run are beyond the limits, for this reason, we validated the backbone of the system using a hash table of 2^{25} entries. Unfortunately, the hash factor of the validation is only about 2.0, which means a poor coverage. Recall from section 7.6 that the hash factor measures the coverage of the run and that hash factors near 1.0 imply low coverage and hash factor over 100.0 mean high coverage.

In the last resort not to separate the whole system indiscriminately into modules and to improve the coverage of the validation we used sequential multihash techniques (see section 7.8). We ran the Spin validator with 16 different pairs of hash functions (*pan -w25 -R16*).

Before analysing the results of the validation it it worth commenting that to prove that the backbone of the system is correct we focused only on checking safety properties. We believe that at this stage of the development it is enough to claim that the system complains to correctness requirements. Safety property were included in the Promela specification in the form of assertion statements to guarantee several conditions:

- To guarantee that a Ks key is within a certain domain and that each Ks is unique.
- To guarantee that TmpId are within a certain domain and that they are unique.
- To guarantee that the payment for the anonymous session is within the limits accepted by the MSS.
- To guarantee that, after the expiration of Bob's prepaid time, the MSS does no forward any message from Bob to Alice and from Alice to Bob.
- To guarantee that after a certain time all the instantiated processes reach a valid end state. In this case, all the instantiated processes, except for the KsTmpId process which remains blocked in valid end state waiting for the arrival of a new PDA, terminate.

The results of the validation are shown in the following lines:

```
% spin -a SysBackBone /* generate an analyser for the Promela specification */
                      /* of SysBackBon and store it in pan.c                 */

% gcc -DMEMLIM=500 -DVECTORSZ=2048 -DBITSTATE -DSAFETY -DPC -o pan pan.c
                    /* generate an executable spin validator in the pan file */

% pan -w25 -R16    /* run the spin validator using 16 different pair of hash */
                   /* functions                                              */
Run 1:
1.81076e+07 states, stored
2.221e+07 states, matched
4.03176e+07 transitions (= stored+matched)
6.87069e+06 atomic steps
hash factor: 1.85306 (best coverage if >100)
```

```
Run 2:
1.85122e+07 states, stored
2.0463e+07 states, matched
3.89753e+07 transitions (= stored+matched)
8.30765e+06 atomic steps
hash factor: 1.81255 (best coverage if >100)

Run 3:
1.78501e+07 states, stored
2.2038e+07 states, matched
3.98881e+07 transitions (= stored+matched)
6.22926e+06 atomic steps
hash factor: 1.87979 (best coverage if >100)

Run 4:
1.73915e+07 states, stored
2.10173e+07 states, matched
3.84088e+07 transitions (= stored+matched)
6.81976e+06 atomic steps
hash factor: 1.92936 (best coverage if >100)

Run 5:
1.859e+07 states, stored
2.00003e+07 states, matched
3.85903e+07 transitions (= stored+matched)
1.41714e+07 atomic steps
hash factor: 1.80497 (best coverage if >100)

Run 6:
1.83468e+07 states, stored
2.30855e+07 states, matched
4.14323e+07 transitions (= stored+matched)
6.24582e+06 atomic steps
hash factor: 1.8289 (best coverage if >100)

Run 8:
1.86209e+07 states, stored
2.04937e+07 states, matched
3.91146e+07 transitions (= stored+matched)
8.51168e+06 atomic steps
hash factor: 1.80198 (best coverage if >100)

Run 9:
1.83666e+07 states, stored
2.28473e+07 states, matched
4.12139e+07 transitions (= stored+matched)
6.62465e+06 atomic steps
hash factor: 1.82693 (best coverage if >100)

Run 10:
1.85906e+07 states, stored
2.05512e+07 states, matched
3.91417e+07 transitions (= stored+matched)
8.32099e+06 atomic steps
hash factor: 1.80492 (best coverage if >100)

Run 11:
1.81843e+07 states, stored
```

```
1.929e+07 states, matched
3.74743e+07 transitions (= stored+matched)
1.36091e+07 atomic steps
hash factor: 1.84524 (best coverage if >100)

Run 12:
1.77686e+07 states, stored
2.1414e+07 states, matched
3.91826e+07 transitions (= stored+matched)
6.5271e+06 atomic steps
hash factor: 1.88841 (best coverage if >100)

Run 13:
1.79687e+07 states, stored
2.23608e+07 states, matched
4.03296e+07 transitions (= stored+matched)
6.6289e+06 atomic steps
hash factor: 1.86738 (best coverage if >100)

Run 15:
1.80276e+07 states, stored
2.20338e+07 states, matched
4.00614e+07 transitions (= stored+matched)
6.37275e+06 atomic steps
hash factor: 1.86128 (best coverage if >100)

Run 16:
(Spin Version 3.3.5 -- 28 September 1999)
        + Partial Order Reduction

Bit statespace search for:
        never-claim             - (none specified)
        assertion violations    +
        cycle checks            - (disabled by -DSAFETY)
        invalid endstates       +

State-vector 1780 byte, depth reached 2972, errors: 0
1.7194e+07 states, stored
1.76727e+07 states, matched
3.48667e+07 transitions (= stored+matched)
8.49598e+06 atomic steps
hash factor: 1.95152 (best coverage if >100)
(max size 2^25 states)
```

From the output of the Spin validator we have learnt that the system consists of more that 4×10^7 states, that 1780 bytes are needed to store each state, and the best, that after exploring the system with 16 different hash functions no errors were detected. It is perhaps worth mentioning that it takes roughly less than 16 hours of CPU time to complete that validation.

The result also show that the hash factor of each run is only about 2.0. We pointed out in section 7.6 that hash factors greater that 100.0 are recommended because they imply a coverage close to 100% of the total number of reachable system states. However, a hash factor lower than 100.0 should not be underestimated. Holzmann has shown [168] that in some cases a hash factor of 4.0 corresponds to a coverage of 93%.

Grounded on this empirical knowledge we argue that although the hash factor of our validation is only about 2.0 for a run with a single pair of hash functions, by running the validator with 16

different pairs of independent hash functions we reached a coverage closer to 100%. On this account we can safely claim that the backbone of the system is correct.

7.19 Simulation of the whole system

Having validated the public key manager, the mail server, the bank, and the backbone of the system, there should be a high degree of confidence about the correctness of the whole system. Although it is computationaly impossible to validate it, we can still run a single simulator to see how the different components of the system interact. To make the output lines clearer, we decided to simulate it for a single PDA only. The result of the simulation are shown next and are aimed to show that Bob can send anonymous messages to Alice and that if Alice decides to reply, Bob can receive Alice's reply.

```
    % spin -n3456 AnoConComm_1PDA /* feed random generator with= 3456 */
                                  /* and run Spin simulator           */

01  PDAses: PDAnum=0; pid=5 HAS RgTED Ks=5 (tmpId= 0) after i= 1 attps

02  BANK: GenuineEcash payment= 20 rcvd from PDA-tmpId= 0

06  MSSses: tmpId=0 credit=9 CreditLeft=10

12  MSSses: tmpId=0 credit=5 CreditLeft=10

13  ===+++>PDAuser tmpId= 0 chnum= 0  GOT E-MAIL addr= XX   txt= YY

14  MSSses: tmpId=0 credit=4 CreditLeft=10

15  ===+++>PDAuser tmpId= 0 chnum= 0  GOT E-MAIL addr= XX   txt= YY

19  MSSses: tmpId=0 credit=0 CreditLeft=10

20  PDAuser tmpId=0 has been TimeAlerted

21  PDAuser tmpId=0 has sent EXTRA payment=20

...
```

On line 01 Bob's PDA succeeds in registering a session key with the MSS and in return he receives a tmpId=0. Next, on line 02, the PDA pays for its anonymous session. Line 06 shows that the amount of money Bob was credited for is being used. On line 06 he has 19 units of time while on line 12 he has only 15. On line 13 Bob receives an answer to one of the anonymous messages he sent to Alice. The same happens on line 15. On line 19 Bob is running out of session time (he is left only with 10 units of money), therefore, his PDA is time–alerted by the MSS (line 20). On line 21 Bob sends 20 units of money to extend his anonymous session.

7.20 Coverage of the validation

In section 7.14 we specified what correctness requirements our anonymizer systems must meet at this stage to be considered correct. Even though we focused only on safety properties and process termination Spin helped detect several errors:

- Unexpected messages and messages sent to incorrect receivers, which would have cause deadlocks.
- Subtle deadlocks between processes running on the PDA and the MSS.
- Incorrect termination of processes at the MSS after unexpected termination of the communication session at the PDA side. This causes unnecessary hold of resources.
- Subtle errors in the session key and TpmId tables stored by the MSS which were easily detected by assertion violations.

At this stage we significantly reduced the correctness requirements of our system. Except for process termination, we did not validate its liveness properties. Because of this simplification we cannot claim yet that our system complies with correct liveness properties. Consequently, we cannot claim yet that our system, our PDAs for example, does not suffer from service starvation. Moreover, without validating temporal claims we cannot guarantee that contentions in our system are fairly resolved. Likewise, we cannot claim yet that messages sent over the Promela reliable channels (see section 6.6.9) arrive safe, not duplicated and in first–in–first–out order. A brief introduction to service starvation and fairness requirements is presented in [179].

7.21 Spin limitations

Spin is a protocol validator based on exhaustive global state generation. Hence, in theory Spin can validate any system provided that its number of reachable states is finite [172]. Promela does not restrict the number of processes, message queues, length of message queues and variables that can be created. This number can be arbitrarily large as long as it is finite. At first glance the problem seems easy and one can get the impression that the only two things the designer needs to validate large systems are a large memory and plenty of CPU time. However, the point not to be missed is that a large memory can easily mean several Gigabytes. Similarly, plenty of CPU time can easily mean millions of years. Due to the space explosion problem, in practice it is extremely important that the number of processes, message queues, length of message queues and variables be restricted as much as possible, but without losing important features of the system. Failure to keep this number small results in systems with prohibitively large number of reachable system states that can be validated only with poor coverage.

Because of these restrictions Spin does not perform well when it comes to validating protocols with random inputs within a wide range. In our particular case for example, what was supposed to be a key randomly selected from a large domain had to be selected from a random choice between six values. This limitation caused some difficulties in reusing session keys.

Another limitation of Spin validation is that communication in Promela is one–to–one. It does not support either multicast or broadcast. This limitation becomes apparent when the designer wishes to simulate a MSS broadcasting to the air its public key and other messages addressed to the public or to anybody who can understand them.

In Promela, timeouts are intensively used to escape hang states. Interestingly enough, Promela does not have a mean of specifying timeout intervals. Timeout is just a timeout possibility, something that will eventually happen. This abstraction is justified [159, 180] on the basis that if an untimed Promela model is correct, it guarantees to preserve its correctness under all possible real time constraints. However, there are situations where the designer wishes to timeout two or more event in a certain order (event p before event q for example). In cases like this the power of Promela clearly weakens.

A possible way of getting around the lack of time intervals of Promela timeouts is by specifying timeouts in terms of numbers of attempts to do something. For example, to rescue a PDA from hanging at a MSS while trying to learn its public one can restrict the number of incorrect MSS public keys received instead of restricting the trying time to t units of time. Obviously, we are assuming that the MSS never fails to deliver either correct or incorrect keys.

7.22 Summary

To ensure that a distributed system is correct, its communication protocol must be specified and validated. State–of–the–art validation techniques involve the use of software tools which are called protocol validators, or just validators.

In the academic environment, the most successful protocol validator is Spin. It can be downloaded for free from the Internet. Spin is a generic verification system that accepts design specification written in Promela. Spin's validation of correctness is based on the analysis of the reachability of system states.

Small systems can be validated in a full exhaustive search which guarantees 100% coverage as Spin explores all reachable states. However, it can happen that the computer runs out of memory and the full search deteriorates into an uncontrolled partial search. In cases like this, one can resort to a controlled partial search (or supertrace controlled partial search) where Spin makes the best use of the available memory to validate the system regardless of its number of states. Naturally, 100% coverage is not guaranteed but the results are still of practical use. Also, it is feasible that the memory is large enough to store the states of the system but the CPU time to validate it is prohibitively large. CPU time can be reduced by reducing the number of states to validate. The designer can reduce this number simply by limiting the memory allocated to Spin to a certain number of megabytes.

The anonymizing system introduced in this work can be considered large; consequently, to validate it, it was separated into independent modules. Next, each module was validated separately either by full exhaustive search or supertrace controlled partial search.

At early stages of the development of a system it is normally enough to validate only safety properties (deadlocks, unspecified receptions, assertions and system invariants) and process termination. Having passed the safety property tests, the designer can move on to validate liveness properties (progress cycles, livelocks and temporal claims). Since the anonymizing system proposed in this work is at an early stage in of development, is was tested on safety properties only. Regardless of some limitations present in the Spin validator, Spin is certainly helpful to detect design errors. According to the Spin's outputs, the system is error free at this stage.

Chapter 8

Enhancing the basic system

8.1 Introduction

In chapter 5 we introduced a system to make anonymous calls from a PDA based on the public telephone box paradigm. The protocols, in finite state diagrams and Promela language to implement the our proposal were presented in chapter 6. As was stated, our strategy was to keep the system as simple as possible so that its central ideas were easy to explain and understand. This forced us to leave out issues that have to be considered in a real life implementation and usage of the resulting system. In this chapter this issues are discussed and some refinements to the system are suggested. Due to space and time constraints we do not go in to detail in our discussion; our sole intention is to make the reader aware of problems that need further attention. This chapter is therefore tentative.

8.2 Traffic analysis

Message encryption prevents Ebe, an intruder, from understanding the contents of messages exchanged between Bob to Alice. Yet it does not prevent him from overhearing that something is being transmitted through the network.

Though Ebe, cannot extract the contents of the message, he can observe the pattern of the messages and determine the location of the hosts involved in the communication, the frequency and length of messages being transmitted. This type of attack is known as *traffic analysis* and is something we have not taken into account in our system.

In an extreme case, Bob and Alice might be the only two persons connected to the network or to the segment that links them, if this is true, traffic analysis would leave Ebe without any doubt that Bob is e–mailing Alice and the anonymity of Bob would be considerable reduced. To meddle in Bob's affairs Ebe can just e–mail Alice, anonymously if he wishes to, and tell her that the anonymous message she has just received was sent by Bob.

The risk of traffic analysis in our system can be prevented easily —at the expense of efficiency and complexity— by deliberately keeping the communication lines active with bogus message traffic. The MSSs for example may be equipped with a mechanism for sending bogus messages to each other, at random intervals of time and of different length. A snapshot of the network loaded with bogus messages is shown in figure 8.1.

The idea behind this is that if Ebe is meddling with the lines that connect the anonymous caller's current MSS and Alice's WS to the network he can never find out whether the traffic he detects was originated at MSS1, is a bogus message or is a meaningful one. Neither can he tell if

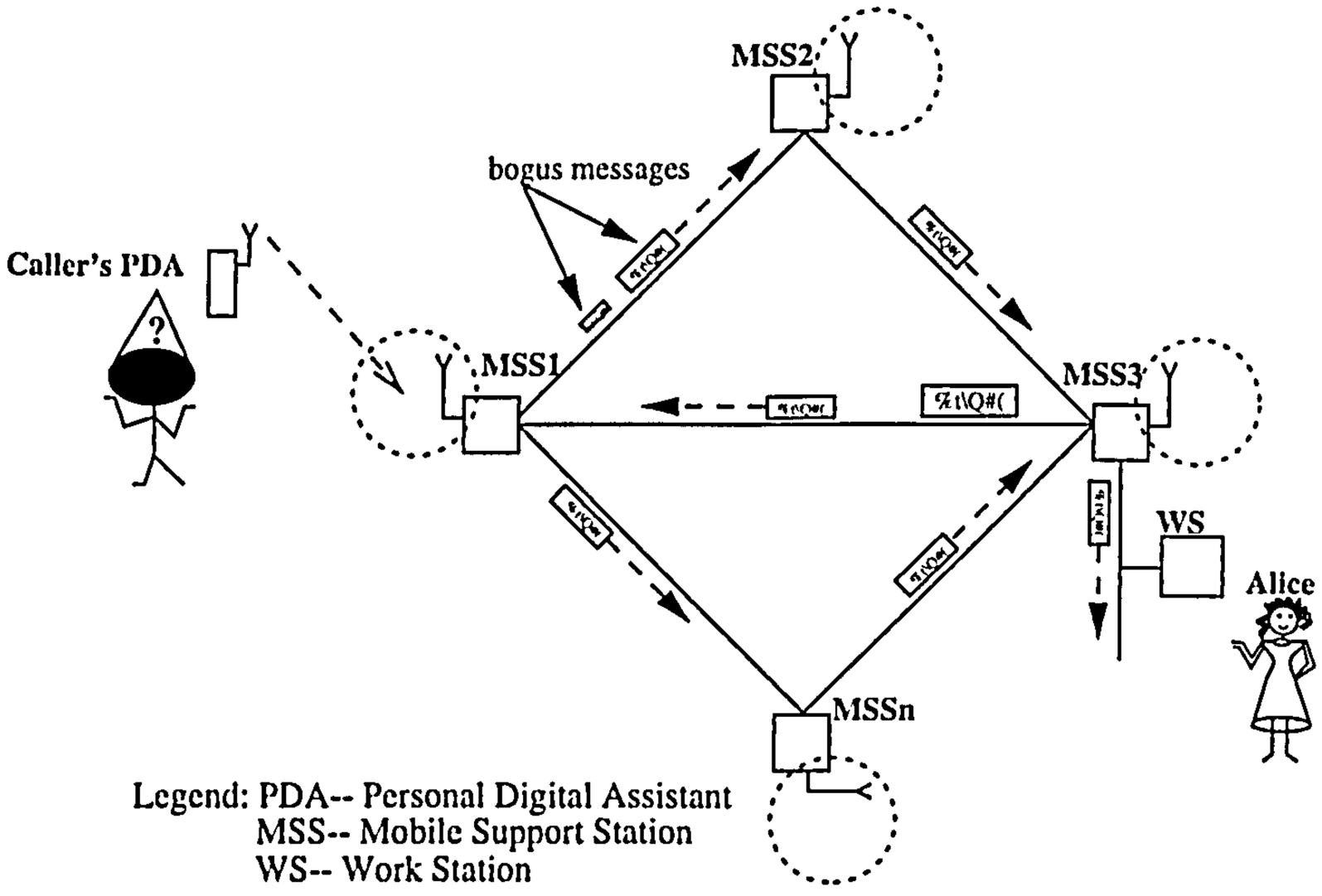


Figure 8.1: Prevention of traffic analysis.

a meaningful message reaching MSS3 from a bogus one.

8.3 Coexistence of physical and electronic cash

The main motivations we had for founding our system on the use of electronic cash were the advantages of e-cash over physical cash discussed in sections 5.5.1 and 5.6. The possibility for the anonymous caller to send his e-coin to the MSS remotely from the park bench where he can be sitting down while sending his anonymous message certainly contributes to hiding his identity.

Another motivation we had in mind when we decided in favour of e-cash against physical cash was the assumption that in the early years of the 21st century e-cash will be widely accepted and available. Note that we are only saying that e-cash will be widely used and not saying that physical cash will disappear as many authors have predicted.

Predicting that physical cash will disappear from the business world is not a safe bet at all but a debatable claim. To please both contentious parties one can say that in the future there will be as much e-cash as physical, i.e. they will coexist and complement each other rather than compete. Though this may sound as an easy way of avoiding confrontations, our last guess makes sense when one goes beyond the boundaries of computer technology to see how new technologies are normally accepted by human beings. With few exceptions, old technologies do not disappear completely from the scene when new ones are deployed, some of them share the space on equal terms, other lose importance and stay behind, however, still in existence. To support this with real life examples we can mention that although cars outperform horses in terms of power and speed, horses are still used, police forces find them appropriate in more than one situation; there are only a few of them but they are still there. Similarly, airplanes are quicker than cars and can fly, notwithstanding, they have not replaced the latter and they never will. Last example, television brings not only sound but pictures as well, yet it peacefully shares the audience with the old-fashioned radio.

The main reason why new technologies coexist with new ones is that it is hard, if not impossible, to create a new technology that outperforms the old one in entirely all aspects. For example, an

airplane outperforms a car when it comes to travelling long distances (thousands of kilometers), yet a car is better for travelling short ones.

Another factor of major importance regarding the resistance to the acceptance of new technologies is inertia. Once a technology becomes entrenched, it is very difficult to change it. Social, cultural, organizational, and emotional factors resists the change.

Humans are sentimental creatures. Thus, after certain time, they fall in love with what they use and find it difficult to depart from it. Consequently, sometimes they would preserve old technologies for the sole pleasure of having it and doing things the old traditional way. Though car lovers will claim the opposite, horse riders will offer convincing arguments to prove that the pleasure of riding a car is nothing compared to riding a horse. Let us mention an example from the electronic world. Only by taking into account the pleasure and the prestige that an old–fashionable mechanical wristwatch gives to his holder one can explain why digital wristwatches have not replaced the mechanical ones when everybody knows that a digital wristwatch is cheaper, more accurate, and practical than a mechanical one.

The introduction of electronic money is not an exception, it complies with the rules we have just discussed. In terms of pleasure, more that one person would claim that one thousand pounds in e–cash is a nice thing to have in your PDA memory. Yet one thousand pounds in five, ten, and twenty pounds notes in your pockets makes them feel better (rich and prestigious). The fact that physical money can be seen, weighed, touched and smelled is bonus to the monetary value of the money, a bonus that counts a great deal for many people. It follows that it is likely that physical cash is going to be around sharing space with electronic cash for a long time.

The reader interested in more details about the stages a new technology goes through before it is widely accepted by the masses is encouraged to refer to [181].

8.4 Text analysis

The most elementary thing the receiver of an anonymous message would do in order guess who is the sender of the anonymous message is to have a look at the text in search for patterns that might lead to the identity of the sender. Patterns like spelling and grammar mistakes, favourite words and sentences, blank spaces, length of lines and others, can provide a great deal of information about the writer of a text. For example, an obvious English mistake would tell Alice that Bob is not a native English speaker.

A simple approach to preventing the analysis of Bob's messages in our system is to make the MSS translate them before sending them to Alice. By translation here we mean converting the text from Bob's writing style into a different one, say, a flat, robot–like style. This translation can be performed by changing words in the original text by their synonyms and by reformatting the text.

A potential problem with text translation is that if words and format are changed arbitrarily, the result of the translation might lose its original meaning. It seems that Bob has to make a choice between the risk of being caught by text analysis and conveying a meaningless message to Alice. To give Bob the chance to decide by himself the MSS can be provided with a selection switch so that Bob can choose between translated and non–translated text.

8.5 Potential risks of e–cash payment

In regards to performance and other practical issues, it is time to discuss the potential danger that e–cash brings to its users. The crucial issue to keep in mind is that e–cash is electronic,

consequently, it is stored, managed, and transmitted by electronic computers and networks, i.e. by systems that are:

- prone to failures due to hardware and software design errors, and
- exposed to security attacks from malicious amateur and professional intruders.

One thousand pounds of e–cash stored in a PDA is worth nothing if the PDA on the network is broken when the PDA user needs to purchase something. So far, we have not been capable of designing unbreakable systems. If this is true it sounds reasonable for a PDA user to have both, e-cash in his PDA memory and physical cash in his pockets.

In the same way, the security of e–cash stored in the PDA memory heavily depends on the security of the PDA, the network, and other computers involved in the e–cash management. Regardless of how strong cryptosystems are, their security will be broken sooner or later.

To talk precisely about the system we propose for sending anonymous and confidential messages from a PDA, we have to admit that the anonymity of the anonymous caller depends on the anonymity of the e–cash used as payment, if the latter is not observed, if the anonymous payment system fails due to a flaw in the protocol or the implementation for example, the anonymity of the caller is seriously compromised. The existence of potential flaws in the protocol or implementation is an inherent weakness of any cryptosystem, something its user should be aware of. Real life has taught us that *unbreakable* cryptosystems do not last for long, sooner or later somebody will hit their point of vulnerability and find a sort of electronic finger prints left on the e–note sent by the PDA; this fact has been admitted by cryptographers long time ago. Thus, if the anonymous caller is really concerned about this issue and does not want to take any risk, he perhaps would prefer using a couple of twenty pence coins to pay for his anonymous call instead of using e–cash. Obviously, he wins in terms of cryptographic security but loses in terms of physical surveillance since he will need to go personally to the MSS to insert his twenty pence into the MSS slot. There he might be observed by somebody and his anonymity compromised.

Another problem with the anonymous payment protocol is that the anonymity of the e–note Bob sends to Clare the banker to be blindly signed is based on the assumption that Clare signs a great deal of e–notes everyday, so that Bob's e–note blends into the wad of notes signed by Clare. The fewer notes Clare signs, the more vulnerable is the system. Again, the use of coins would work better in this situation.

One of the problems of having two equivalent technologies working at the same time is that a double infrastructure is needed to support them, this may be regarded as a luxury since it might double the cost of the system. A public telephone box for example that accepts both coins and credit cards is considerable more expensive that one that supports only one method of payment. The same applies to a MSS that supports coins and e–coins. Doug, the owner of the MSS of our hypothetical network will need to equip them both with an wireless antenna to receive e–cash and a slot to receive coins.

8.6 Cheating with e–cash

In section 4.1.6 the protocol for writing blind signatures on electronic notes was presented. It was mentioned also that this protocol allows Clare the banker not only to detect forgery e–notes but also to find out the identity of the cheater. It detects if Doug, the MSS owner, is presenting an e–note to Clare for a second time. Moreover, it leads to the identity of Bob, the anonymous caller, if he is trying to spend an e–note twice.

In our system no action is taken if Clare detects that Bob is cheating. Bob's forged e–note is refused and nothing else happens. In a real life system legal actions have to be taken again the deceiver, perhaps a sort of Internet police should be informed to investigate and sort out the problem between Bob and Doug.

In our system e–cash is either accepted of refused by Clare, she does not make any distinction between invalid e–cash and forgery. By invalid cash we mean, e–cash that is not recognized by Clare as a legal payment, the use of dollars in the United Kingdom for example. The payment is simply refused, there is no crime to prosecute in this case. On the contrary, by forgery me mean the use of illegal e–cash with the conscious intention of deceiving Clare. For example, the use of hand–made forgery or the use of e–cash that has already been spent.

8.7 Use smart cards to pay anonymously

The need to contact Clare the banker to check for the validity of Bob's e–cash sent to Doug as a payment for an anonymous communication session with the MSS stems from the freedom Bob has to make his own money in his PDA. This might result in Bob double–spending an e–coin either accidentally or maliciously. To prevent Bob from doing this, we had to bring Clare in to the game and consequently make our system slightly more complicated. There is a way of going around this question which consists in decreasing Bob's power to create the e–coins he uses to pay for his anomymous calls.

The essential idea here is for Bob to pay for his anonymous communication sessions with a sort of prepaid telephone magnetic card. The property of the prepaid we are interested in her is that it stores money units (£5.00 for example), that are retrieved from the card by the card reader. Once a money unit is spent, it is permanently deleted from the magnetic band of the card. Consequently, there is no hope of reusing it.

A smart card (see section 4.3) can be used to implement this method to pay for MSS services. Bob can buy a smart card and without writing any personal information in its memory or on its surface, he brings it to Clare the banker to charge it with e–money. To pay, Bob can present Clare with physical cash. The smart card is charged by converting Bob's physical cash into e–notes signed by Clare and loading them into the card memory.

As long as Clare gets paid, she does not care who the card owner is or where the e–coins she signed are going to, thus, she is happy charging a card without asking Bob for any identification. Neither is she interested to know how the e–coins are going to be spent. If this holds true, Bob's identity is protected and the card can be used for anonymous payment to merchants who recognize Clare's signature.

Money is put in and out of the smart card by a program stored by the smart card microcontroller. The program is activated by inserting the card into a card reader and typing the propers commands from the card reader keyboard. Bob does not have any access to the program which is in control of the e–coins stored in memory. Once an e–coin is spent, it is deleted from memory or marked as *spent*. In this manner, there is no chance for double spending.

Smart card readers might be deployed inside PDAs or connected to them, this way Bob can transfer payment for anonymous communications to Doug from his PDA. Upon receiving Bob's payment, Doug sees Clare's signature and knowing that money coming from smart cards is not prone to double spending and forgery, accepts it immediately.

As a side comment, it is worth mentioning that money stored in a smart card might be used to pay for any goods and commodities (at vending machines for example) as long as the merchant has the hardware and software to communicate with the smart card and recognizes Clare's signature.

consequently, it is stored, managed, and transmitted by electronic computers and networks, i.e. by systems that are:

- prone to failures due to hardware and software design errors, and
- exposed to security attacks from malicious amateur and professional intruders.

One thousand pounds of e–cash stored in a PDA is worth nothing if the PDA on the network is broken when the PDA user needs to purchase something. So far, we have not been capable of designing unbreakable systems. If this is true it sounds reasonable for a PDA user to have both, e-cash in his PDA memory and physical cash in his pockets.

In the same way, the security of e–cash stored in the PDA memory heavily depends on the security of the PDA, the network, and other computers involved in the e–cash management. Regardless of how strong cryptosystems are, their security will be broken sooner or later.

To talk precisely about the system we propose for sending anonymous and confidential messages from a PDA, we have to admit that the anonymity of the anonymous caller depends on the anonymity of the e–cash used as payment, if the latter is not observed, if the anonymous payment system fails due to a flaw in the protocol or the implementation for example, the anonymity of the caller is seriously compromised. The existence of potential flaws in the protocol or implementation is an inherent weakness of any cryptosystem, something its user should be aware of. Real life has taught us that *unbreakable* cryptosystems do not last for long, sooner or later somebody will hit their point of vulnerability and find a sort of electronic finger prints left on the e–note sent by the PDA; this fact has been admitted by cryptographers long time ago. Thus, if the anonymous caller is really concerned about this issue and does not want to take any risk, he perhaps would prefer using a couple of twenty pence coins to pay for his anonymous call instead of using e–cash. Obviously, he wins in terms of cryptographic security but loses in terms of physical surveillance since he will need to go personally to the MSS to insert his twenty pence into the MSS slot. There he might be observed by somebody and his anonymity compromised.

Another problem with the anonymous payment protocol is that the anonymity of the e–note Bob sends to Clare the banker to be blindly signed is based on the assumption that Clare signs a great deal of e–notes everyday, so that Bob's e–note blends into the wad of notes signed by Clare. The fewer notes Clare signs, the more vulnerable is the system. Again, the use of coins would work better in this situation.

One of the problems of having two equivalent technologies working at the same time is that a double infrastructure is needed to support them, this may be regarded as a luxury since it might double the cost of the system. A public telephone box for example that accepts both coins and credit cards is considerable more expensive that one that supports only one method of payment. The same applies to a MSS that supports coins and e–coins. Doug, the owner of the MSS of our hypothetical network will need to equip them both with an wireless antenna to receive e–cash and a slot to receive coins.

8.6 Cheating with e–cash

In section 4.1.6 the protocol for writing blind signatures on electronic notes was presented. It was mentioned also that this protocol allows Clare the banker not only to detect forgery e–notes but also to find out the identity of the cheater. It detects if Doug, the MSS owner, is presenting an e–note to Clare for a second time. Moreover, it leads to the identity of Bob, the anonymous caller, if he is trying to spend an e–note twice.

In our system no action is taken if Clare detects that Bob is cheating. Bob's forged e–note is refused and nothing else happens. In a real life system legal actions have to be taken again the deceiver, perhaps a sort of Internet police should be informed to investigate and sort out the problem between Bob and Doug.

In our system e–cash is either accepted of refused by Clare, she does not make any distinction between invalid e–cash and forgery. By invalid cash we mean, e–cash that is not recognized by Clare as a legal payment, the use of dollars in the United Kingdom for example. The payment is simply refused, there is no crime to prosecute in this case. On the contrary, by forgery me mean the use of illegal e–cash with the conscious intention of deceiving Clare. For example, the use of hand–made forgery or the use of e–cash that has already been spent.

8.7 Use smart cards to pay anonymously

The need to contact Clare the banker to check for the validity of Bob's e–cash sent to Doug as a payment for an anonymous communication session with the MSS stems from the freedom Bob has to make his own money in his PDA. This might result in Bob double–spending an e–coin either accidentally or maliciously. To prevent Bob from doing this, we had to bring Clare in to the game and consequently make our system slightly more complicated. There is a way of going around this question which consists in decreasing Bob's power to create the e–coins he uses to pay for his anomymous calls.

The essential idea here is for Bob to pay for his anonymous communication sessions with a sort of prepaid telephone magnetic card. The property of the prepaid we are interested in her is that it stores money units (£5.00 for example), that are retrieved from the card by the card reader. Once a money unit is spent, it is permanently deleted from the magnetic band of the card. Consequently, there is no hope of reusing it.

A smart card (see section 4.3) can be used to implement this method to pay for MSS services. Bob can buy a smart card and without writing any personal information in its memory or on its surface, he brings it to Clare the banker to charge it with e–money. To pay, Bob can present Clare with physical cash. The smart card is charged by converting Bob's physical cash into e–notes signed by Clare and loading them into the card memory.

As long as Clare gets paid, she does not care who the card owner is or where the e–coins she signed are going to, thus, she is happy charging a card without asking Bob for any identification. Neither is she interested to know how the e–coins are going to be spent. If this holds true, Bob's identity is protected and the card can be used for anonymous payment to merchants who recognize Clare's signature.

Money is put in and out of the smart card by a program stored by the smart card microcontroller. The program is activated by inserting the card into a card reader and typing the propers commands from the card reader keyboard. Bob does not have any access to the program which is in control of the e–coins stored in memory. Once an e–coin is spent, it is deleted from memory or marked as *spent*. In this manner, there is no chance for double spending.

Smart card readers might be deployed inside PDAs or connected to them, this way Bob can transfer payment for anonymous communications to Doug from his PDA. Upon receiving Bob's payment, Doug sees Clare's signature and knowing that money coming from smart cards is not prone to double spending and forgery, accepts it immediately.

As a side comment, it is worth mentioning that money stored in a smart card might be used to pay for any goods and commodities (at vending machines for example) as long as the merchant has the hardware and software to communicate with the smart card and recognizes Clare's signature.

It is worth closing this section by mentioning that the selection of a smart card as our hardware platform to store and manage Bob's e–cash is just a technological decision based on the fact that a smart card suits our computational and memory needs, and matches nicely current PDA parameters in terms of size, energy, weight, price, and so on. Yet it should be kept in mind that a smart card is just one example of what are known as *secure coprocessors*, i.e tamper–resistant modules that perform computational operations on data stored inside them and destroy it in response to any unauthorise attempt to read or write the data.

We are assuming that data stored in a tamper–resistant module like a smart card is secure. No intruders can access it. To be precise, one should say no intruders without the proper equipment can read it since Bruce Schneier [182] argues that tamper–resistant techniques do not work. Sophisticated attacks based on time spent in cryptographic operations, power consumption, and radiation emissions against a smart card can be devastating.

8.8 Loss of payment in incomplete transactions

In the finite state diagram presented in section 6.6.3 which concerns the interaction between the user and his PDA through the PDA keyboard, there is an entry (`EscPressed`) to allow the user to abort his anonymous session with the PDA any time during the anonymous mailing session. It is conceivable that the abort occurs when a considerable amount of prepaid anonymous communication time is left. No actions is taken to return to the PDA user his unspent e–money, the system just imitates a public telephone box which does not give change. The unspent e–cash goes to Doug. A refinement of the algorithm would be appropriate in order that Bob is given his change.

Likewise, it is possible that the system crashes any time after the payment has been performed, while the e–cash is travelling through the network to Doug or to Clare for example, from the protocol we are proposing, the fate of the unspent or partially spent e–note is not clear, the e–note if left in the limbo. In general the problem we are facing here comes from the fact that our system does not support atomic e–cash transactions [183].

Again, further research in this direction is necessary to trace the e–cash and decide to whom it belongs. In a public telephone box, it always goes to the owner of the telephone Box, regardless of the user's anger. In our anonymizing system a different and fairer approach can be taken; if the anonymous caller does not want his e–note back not to compromise his identity, the e–cash in dispute may go to a charity organization instead of Doug's account.

8.9 Anonymous debit bank accounts

A question that inevitably come to the reader's mind after learning about anonymous cash is whether it is possible to have an anonymous debit bank account. That is to say, an account opened by Bob at Clare's bank without Clare knowing the name of the account owner and with a possibility to withdraw and deposit money anonymously. In terms of technology, the answer is yes. Technically speaking, the only thing we need to open an anonymous account is a pair of public and private keys and an anonymous e–mail system like the one we are presenting in this work. Although the affirmative answer to the question looks like a solution to some problems Bob may have, it is in fact the raising of several issues that have not been researched. There are legal and financial issues that have to be studied before the idea of anonymous debit accounts can be put into practice.

Roughly speaking, the procedure to open an anonymous debit account is as follows.

- Bob brings to Clare a certain amount of money is physical cash —say £1000.00— and a smart card.

- Clare generates a pair of public and private keys for Bob. She stores the private one in the smart card so that nobody sees it and Bob does not forget or lose it.

- Clare opens a record in her computer where she puts Bob's £1000.00 and identifies it with an account number, no names or any personal date, just a unique string of digits. The number is stored in the smart card.

- In public key cryptosystems, the public key is known by everybody, in this case it is kept secret by Clare so that only she can decrypt messages encrypted with Bob's private key.

- To withdraw or deposit money from and to his account Bob sends Clare messages that contains his account number and encrypted with his private key.

- Upon receiving a message from Bob asking for some money, Clare knows that the message is coming from Bob and not from somebody else as only Bob knows the private key.

- To prevent an intruder from linking Bob to his anonymous account, Bob sends his messages to Clare anonymously. He can use the our anonymous system for example.

- Deposits to and withdrawals from Bob's account are made by anonymous e–cash.

It should be clear that the purpose of this section is not to discuss this issue at large but to raise the question. Several problems are hidden behind the algorithm we suggest. For example, once Bob leaves the bank he has no evidence but his private key that the £1000.00 is his money. If the key is lost, damaged or if Clare denies that Bob has an anonymous account with her, Bob loses his money. In practice, bankers are prone to billing mistakes which are normally favourable to their accounts and seldom favourable to their customers'. If Clare makes accidentally or intentionally a billing mistake, Bob will need a mechanism to recover his money, ideally without disclosing his identity.

Another problem is what would happen if an e–note sent to Clare (a deposit) or to Bob (a withdrawal) does not reach its destination; will it be lost?, is there any way for Bob to recover his money?.

It seems that there are more questions than answers in our attempt to create anonymous credit accounts, one may wonder what do we need this for when payments can be performed by e–cash and smart cards as was discussed in sections 5.6.2 and 8.7 respectively. The justification for going through all this hassle is Bob's money safety. An e–coin in a smart card or in PDA memory is lost if the smart card or the PDA is damaged or stolen. Both the smart card and the PDA can be replaced but the e–coin cannot. An e–coin in the bank computer is safer for several reasons. For a start, bank computers are stationary, secondly, they have stronger security measures than mobile devices like smart cards and PDAs have, and after all, if something goes wrong with the bank computer it is not Bob's concern, he is guaranteed that his bank will give his money back when he needs it.

8.10 Anonymous credit bank accounts

It seems that anonymous debit account are feasible, yet it remains to be seen how it works in commercial applications. If this is feasible, one might wonder whether there is a way to open an anonymous credit bank account. That is to say, we want Clare to lend Bob some money without knowing Bob's identity. Though this would be a marvellous thing to have, we argue that there is

no way to open an anonymous credit bank account without putting Clare at the risk of losing some money, something bankers are not keen on, or without putting Bob's anonymity into risk.

Our position is based on the fact that no banker will lend money without being certain that she will be able to catch the credit account owner in case the later fails to pay.

However, one can think of a fairly close approximation to an anonymous credit account. The approach is based on the use of secret splitting techniques [184, 118]. The core idea is to take data D and divide it up into n pieces $D_1, D_2, D_3, ..., D_n$ in such a way that the knowledge of $0 < k \leq n$ D_i pieces makes D easily computable. The knowledge of $k-1$ or fewer D_i pieces is absolutely worthless to recover D.

We can assume that Bob's personal data (name, address, and so on) is D, i.e. a string of bits. Clare is happy to open an anonymous credit account for Bob as long as she is allowed to split D into two pieces: D_1 for herself and D_2 for a government financial authority. If Bob fails to pay his debt before the agreed date she calls the government financial authority, puts D_1 and D_2 together to reconstruct D and rings the police to prosecute Bob. The weak side of this approach is that Bob's identity depends on the willingness of Clare and the government financial authority to keep it; there is a risk that the two of them can get together to conspire against Bob at any time and without a legal reason for it. This risk can be significantly decreased by involving more people into the game, i.e. by splitting D into more than two pieces so that a conspiracy is more difficult to achieve. Nevertheless, regardless of how big the number of pieces is, the risk of conspiracy never disappears. One has to admit that the system is not truly anonymous.

8.11 An improvement to e–voting schemes

One of the most appealing applications of anonymous e–mails is electronic voting. The topic is not new, it has been extensively studied. It is now well understood that a voting protocol has to satisfy the following basic requirements [118]:

- Only authorised voters can vote and only once.
- Votes are anonymous. No one can determine for whom anyone else voted.
- Votes are personal. No one can duplicate anyone else's vote.
- Neither the voters nor the election board can cheat without being discovered.
  - The election board is not allowed to change the choice of a voter, neither is a voter allowed to change anyone else's choice.
  - The election board cannot present a false tally.
- Each valid vote is counted. No one can be excluded from the final tabulation.

Several protocols have been proposed that, to a satisfactory degree, match the previous requirements [118, 185, 186]. Unfortunately, all of the proposed protocols, share a common drawback: they fail when it comes to providing true anonymity. The common failure stems from the fact that they assume the existence of a secure, untraceable electronic mail system to send the votes to the election board, that they do not provide. They are based on anonymizing schemes composed of a set of intermediate computers called *mixes*. The rôle of the mixes is not distinct from the one performed by the e–mail anonymizer we criticized in chapter 3. A mix computer receives an electronic ballot (e–ballot) from a previous one, suppress the voter's name and address and sends

the vote to the next mix computer; the process is repeated till eventually the vote reaches the computer of the election board [187].

As can be seen, the anonymity of the voter can be violated if the mixes collude in disclosing the identity of the voters. A collusion of all mixes might sound unthinkable in a democratic country but not in a totalitarian regime where the whole of the mixes may be under the control of the totalitarian oppressor. Even worse, if a key escrowed cryptosystem is in use, the oppressor can easily gain access to the escrowed keys and reduce the anonymous election to a mere illusion.

We believe that the danger of collusion and key escrow can be sorted out by grounding the proposed electronic voting algorithms on the untraceable anonymous e–mail facilities of our system.

By looking closely at an electronic note and an electronic ballot–paper one can realize that the similarities between them are striking. Both have to be signed by a trusted authority. By a banker in the former case and, in the latter, by a central ballot legitimization agency before the vote is accepted by the election board. In the same manner, both can be used only once. The difference being that the user can make as many e–notes as he wish but it is allowed to make one e–ballot only.

Let us say Bob, is an anonymous voter, Clare a central ballot legitimization agency, and Alice the election board. Likewise, let us assume that Clare has a list of all PDA owners who are entitled to vote. The following is the sketch of an algorithm which with some refinements can lead to a powerful electronic voting scheme that matches the requirements presented at the beginning of this section. It is based on the blind signature protocol presented in section 4.1.6. The reader is encouraged to familiarize with that in order to understand what follows.

- Bob generates n sets of e–ballot each containing a valid vote for each possible outcome, i.e if the vote is a *yes* or *no* question , each set contains two votes, the firsts is marked *yes* and the second—*no*. As with e–notes, each vote contains additional information to ensure that Bob is not cheating.
- Bob sends the set of e–ballot to Clare for a blind signature in a similar was as he sends e–notes to her (see section 5.6.2).
- Clare checks her database to see that Bob is entitled to vote and if she is satisfied, she blindly sings Bob's e–ballot.
- Bob unblinds the signed e–ballot received from Clare and chooses one of the votes (a *yes* or a *no* according to his preferences) and discard the second.
- Now Bob holds a valid e–ballot readily accepted by Alice since it has Clare's signature on it.
- To send his e–ballot to Alice, Bob opens an anonymous session with the MSS. He uses e–notes to pay for it (see section 5.8).
- Upon receiving Bob's e–ballot Alice verifies that it has not been received before; if so she counts it, otherwise the e–ballot is discarded.

It is worth insisting that it is not our intention to present a complete voting algorithm but to show how existing ones can be greatly improved by using the anonymous facilities offered by our system. This make electronic voting schemes more robust against dishonest vote delivery and immune to dishonest key escrowers. A key escrower might learn who is supporting a giving vote but never the identity of the voter.

In theory electronic voting schemes work and work well, yet it remains to be seen how they work in practice. One of the major concerns is that they make it easier to buy and sell votes. Ebe

the intruder for example, can know that Bob is over 18 and entitled to vote, contact him and offer a £1000.00 for the pair of signed votes that Bob has just unblinded after receiving them from Clare. To avoid being detected by the police, Ebe, can negotiate with Bob anonymously, using our own anonymizing system and pay him a £1000.00 in e–cash. A pair of votes in Ebe's hands is just like a coin, he can choice the *yes* or the *no* one and send his choice to Alice, sell the pair or destroy it.

Regardless of the potential danger and not yet studied issues, electronic voting is a useful application, a well implemented scheme makes electronic election more anonymous than traditional ones where due to physical elements involved in the process (physical ballot–boxes) the winner of an election can always know whether a village voted him or his political opponent. This question is answered by electronic voting since nobody can learn which part of the country or city e–ballots are coming from to the electronic ballot–boxes.

8.12 Coexistence of key escrow and non–key escrowed cryptosystems

The core idea behind key escrow mechanisms and their debatable arguments for and against their suitability were discussed in section 4.2.3. We bring it back to this section to make some additional comments about this issue because key escrow significantly plays a fundamental part in our system.

It is not clear yet when this intellectual battle will end and who will win it, if there is a winner, probably not. We argue that since it is difficult to come up with a single system that suits the demands of different applications, in the future it is likely that key escrowed and non–key escrowed cryptosystems will coexist and the user will choose the more suitable for her as long as her choice complies with legal regulation enforced by governments; for example, the use of encryption keys no larger than n–bits, communication restricted to the boundaries of a corporation, etc.

It has been admitted that key escrow is an open door that can easily lead to Internet censorship. Hence, if key escrow is enforced by law, it is likely that there will be a sort of *key escrow immunity* for trusted users by which they may not be key escrowed. For example, diplomats may be given diplomatic key escrow immunity. Likewise, world wide known scientists, academics, writers and other figures of unquestionable intellect may be given the privilege of not having their key escrowed if they do not wish to. If the reason to enforce key escrow if the catch of terrorists, narcotic dealers and other dangerous criminals, governments will find it difficult to present solid arguments in favour of escrowing the key of a well–known peace activist, a Nobel prize winner or a famous writer.

8.13 Key escrow confidentiality and anonymity

Non–key escrowed cryptosystems have the inherent risk that if for any reason the encrypting key is lost, forgotten or damaged the encrypted data is not recoverable. It is lost for ever or in the best case, till the encryption is broken by means of brute force. Depending on the length of the lost key and the power of the computer used, a brute force procedure may take seconds, hours, day, or years to break an encryption. Most of the times this is unacceptable, so it follows that key escrow makes sense. However, it renders the confidentiality of the messages encrypted with the escrowed key seriously compromised. We cannot talk of a truly confidential cryptosystem when the encryption key is available to somebody else, apart from its owner, and can be used without the owner knowing about it. This holds true for our system as well. The confidentiality of the messages exchanged between Bob's PDA and the MSS cannot be guaranteed if the public key of the MSS is escrowed, neither the confidentiality of messages exchanged between the MSS and Alice can be guaranteed. Having the public key of the MSS the government can easily open the message sent

by Bob that contains the session key, having the session key (see 6.6.4) means being able to open every single message in the session, read the contains of the message and find out that the message is addressed to Alice. Again, if Alice responds to Bob by sending her response encrypted with the public key of the MSS or with a session key sent to her by Bob, the government can decrypt Alice's message, read its contents, and find out that it is addressed to somebody, who is currently at the MSS with a given TmpId, fortunately, nothing else is revealed to the government. If the aim of the government was to know the identity of Bob, the former will be certainly frustrated. This is where the power of using TmpId becomes apparent.

A TmpId protects the identity of the PDA user against, the recipient, the MSS and against key escrower as well. This is equivalent to tapping a call from a public telephone box, the meddler can understand the whole of the conversation, can learn the called number and the name of the called person, however, he cannot learn the identity of the caller. This is exactly what our system is imitating, hence, we claim that it is truly anonymous.

8.14 Summary

Simple ideas are easy to understand and extend. It has been proven in practice that simplicity is good and complexity is bad. Simple ideas can work well. Complex ideas can only fail. If the basic idea is simple and good it can serve as the ground for building another system on top of it. If the second system is good and simple, it can serve as the basis for building a third good and simple system; and so on. The Unix operating system is grounded on a simple idea, so are the Internet and the Web. The three of them work and work well. This explain why they have been used successfully to support a great variety of applications. Surprisingly, this lesson is frequently forgotten by the computer science community which is frequently enchanted and diverted by complexity.

The principle of simplicity was not forgotten in this work. The idea of building an anonymizer based on the paradigm of the public phone box is remarkably simple; therefore, it can be enhanced. It can be extended in different directions and useful applications can be built on top of it. Some of these potential applications were briefly discussed in this chapter.

Chapter 9

Conclusions

9.1 Introduction

In this thesis, we have introduced a novel approach to addressing the problem of sending anonymous and confidential messages through the Internet. A system was proposed, described, designed, specified, and validated, and the results of the validation presented. It is time now to describe the experience learnt from this research, to assess our results, to admit their limitations, and to comment on issues related to this topic that would benefit from further investigation.

9.2 Contribution

The necessity of sending confidential and anonymous messages over the Internet was identified about a decade ago when the Internet changed from being an exclusively academic network into a universal network widely and intensively used by the masses, where people from different countries and culture and with different backgrounds, profiles, ideas, and interests, coexist. That was the time when experts in the field raised the issue. Since then, due to the many Internet applications that heavily depend on confidentiality and anonymity, the importance of the issue has now become more apparent. Electronic commerce is probably one of the best examples of an Internet application that will not find a wide acceptance until a satisfactory answer to this problem is found.

A great deal of effort and resources have been devoted to the investigation of confidentiality and anonymity. Several implementations of anonymizers have been proposed and deployed in the Internet. However, none of them has so far given a satisfactory solution to providing true anonymity because the degree of anonymity achieved entirely depends on and is limited by the ability and desire of a third party (a computer or set of computers located between the sender and the receiver of the anonymous message).

This thesis has set out to demonstrate that it is possible to send confidential and anonymous messages without depending on properties of the computer (or computers) located between the sender and the receiver of the anonymous message. To demonstrate our claim, we not only proposed a new approach to providing confidentiality and anonymity, but also designed a protocol based on the approach, specified it in a validation language (Promela) and validated it using a validating software (Spin).

Because we believe that PDAs and other similar, wireless, portable, pocket–sized computers equipped with wireless communication antennae, are going to be widespread in the years to come, and because PDAs fit smoothly into our paradigm, the senders of the anonymous messages are in possession of PDAs. We also assumed that there is a set of MSSs that for a fee provides Internet

access to PDAs.

The main contributions of this research are:

- The proposal of a novel approach to sending confidential and truly anonymous Internet messages from a PDA.
- The specification of a protocol based on the approach and its validation to prove that the protocol satisfies a set of basic correctness criteria.
- Another contribution of this thesis is the introduction IP–addressless computers in the Internet.

9.3 The model

One of the guiding aims of our approach was simplicity. Knowing that complicated systems are difficult to understand, manage and scale, we grounded our approach on a rather simple idea: Our approach to sending anonymous messages is inspired by an old and well–known idea, namely, the paradigm of a public telephone box. We did not try to create something new to solve the problem of sending anonymous messages through the Internet, but instead introduced an existing paradigm from the public telephone network and brought it into the Internet. Similarly, to validate our protocol we tried to keep the validation model as simple as possible but without losing the essential features of the protocol. We believe that after proving that the main modules of ithe system are correct, the system can have additional components added to transform it into a working practical implementation.

To bring the functionality of a public telephone box into the Internet, we needed only to find a functionally equivalent element in the Internet to the elements of the public telephone box. In this way, the telephone communication infrastructure is equivalent to the Internet one: the MSS is a sort of public telephone box used by the general public; and finally, the coins used to operate the public telephone box are replaced by anonymous electronic cash to operate the MSS.

In our paradigm, Bob (the anonymous caller in possession of a PDA) uses the communication services of an MSS to send messages to users reachable through the Internet (Alice for example) and to receive messages while his PDA is switched on and registered with the MSS. Not to give away any information about his identity, Bob connects to the MSS by using a temporary, non–personal, random, identifier (TmpId for short). The TmpId is assigned by the MSS and is valid only for the duration of a session.

To provide confidentiality, we used both public key and secret key cryptographic techniques. In this way, public key encryption is used by the MSS and the PDA to negotiate a TmpId, and a secret key is used later to encrypt messages exchanged between the PDA and the MSS.

Surprisingly, the translation of the paradigm from one network into another was straightforward and the resulting system is remarkably simple. Also, it provides new, valuable features not found in its original environment; namely, the fact that in our paradigm the caller does not need to physically go to the public telephone box to dial the callee's number and make her call, but can do the same remotely using the wireless antenna of her PDA.

The outstanding feature of our approach is that although the MSS is interposed between Bob and Alice, the MSS has no means of finding out about Bob's identity, simply because Bob is not using a personal IP address but rather a TmpId that does not belong to him, just as a public telephone box does not belong to its user. The result of this is that the degree of anonymity achieved by this approach does not depend of the willingness nor strength of the intermediary (the MSS) to keep secrets.

9.4 The validation

We believe that the only reliable way of determining that an idea is of any practical use is to implement it, and put it into practice so that it is exposed to as many users, critics, and unexpected working conditions as possible. To increase the chances that an idea will work correctly, it is advisable to validate its design before implementing it.

We have followed this strategy: we designed the protocol of our system, specified it in the Promela language and validated its basic safety properties (deadlocks, unspecified receptions of messages, and assertion violations) and proper end–states, by using the Spin validator.

From the validation of the protocol we have learnt the following:

- Although we made efforts to keep the Promela validation model simple, its size in terms of CPU and memory used to validate it exhaustively is exceptionally large for current computer technology. The exhaustive validation of the whole model would have required of the order of several Gbytes of RAM memory to store the system states and of the order of several days of CPU time.

  This observation encouraged us to resort to modular validation (separation of the whole Promela validation model into modules to validate them separately) and to supertrace validation (random selection of the maximum number of states that can be validated in the available memory).

  In modular validation the risk of leaving out important features of the protocol is always latent. Similarly, the random nature of the supertrace technique introduces the risk of not exploring the part of the protocol where an error exists. Hence, compared to exhaustive validation, these techniques techniques do not give the most accurate results, yet it is the best we can do for a system of the size of ours. In addition, the results are good enough for practical purposes and reliable enough to claim that the system is correct.

- Our protocol is free from errors related to basic safety properties and improper end–states.

  We are aware that by checking correctness of basic safety properties we checked only for basic errors. We did not check our protocol for subtler safety properties, nor did we check it for liveness properties. Thus, we cannot claim that our validation is complete. It can easily be enhanced. However, we believe that at this stage of development, proving correctness of safety properties gives a significant degree of confidence about its correctness.

  This is probably enough at this stage to appreciate the feasibility or our idea. On the other hand, the main aim of this work was to propose a new paradigm for sending anonymous and confidential messages over the Internet. Exhaustive validation of the protocol fells outside of our interests.

9.5 Limitations of the work and suggestions for future research

In accordance with the results of the validation our protocol is free from basic safety properties and improper end–states. From this stage the designer can step further: it would have been useful to have more time at our disposition to validate liveness properties or convert the Promela specification of the protocol into a working implementation (say C++ code), test its behaviour and then later migrate the C++ code into a real PDA and MSS. This would give the opportunity to see how the performance of the system is affected by the numerous cryptographic operations involved in the

communication which were not included in the Promela specification as cryptographic operation are pure arithmetic operations that are not relevant to the communication protocol.

A minor limitation of our protocol is that it relies on a PDA to send its anonymous and confidential messages. This might not be a serious drawback as we expect that in future PDAs will be widely available. However, if this is not the case, it is an open question as to whether it is possible to send truly anonymous messages from a desktop IP–addressed computer. We suspect that the answer is no. Research in this direction would be useful.

Another minor limitation of our protocol is that it does not support user handoff: once the anonymous caller opens an anonymous session with the MSS, he has to terminate his call at that MSS. We deliberately left this feature out of the protocol, because we consider that even though it would be certainly useful to have support for user handoff, this topic has to be addressed from the point of view of user location and message routing, rather than from the point of view of anonymity and confidentiality. More research is needed to investigate what impact the mobility of the sender of the anonymous messages could have on our protocol.

A more serious limitation of our system is that it does not offer any protection against potential abuse of anonymous messages. The danger lies in Bob's ability to send as many anonymous messages to Alice as he wishes to. One of Bob's messages can contain a virus for example. Alice has no way to prevent Bob from annoying her other than instructing her computer to drop any anonymous message coming to its door. Unfortunately, this approach seems too costly as it prevents Alice from receiving any anonymous message, those coming from Bob as well as other PDA users. This raises the question whether it is possible to provide Alice's computer with protection against Bob's abusive anonymous messages. So far, this problem has not been investigated.

Care should be taken not to think that the system for sending anonymous and confidential messages that we propose is unbreakable. Having in mind that, in practice, the design of an unbreakable system based on cryptographic techniques in arguably an intractable problem, we did not attempt to design an unbreakable system. In theory, unbreakable systems can be designed but they are not practical because they demand significant amounts of computer resources and are too complex to use.

From the cryptographic point of view the security of our system totally depends on the strength of the cryptographic technology we use. If we admit that a cryptographic system sooner or later will be broken, we have to admit that our system sooner or later will be broken as well. Said in a few words, our system is unbreakable for lay users equipped with current technology (i.e. for most people but not for users with strong background in cryptography and sophisticated equipment). For example, we do not have any hope that our system will survive the attack of users equipped with quantum computers of the twenty-first century, which promise to demolish DES and RSA keys in few minutes [188, 189].

We hope that this thesis contributes to the problem of finding a satisfactory solution to the issue of confidentiality and anonymity. We would dare to argue that our system gives a satisfactory answer most applications will be happy with.

Perhaps the field that needs urgently to be researched is the social impact of anonymity.

9.6 Social issues

The main motivation to research the field of confidentiality and anonymity was the potential of its applications. For instance, we realized that the provision of a mechanism to send anonymous messages will encourage PDA users to use their PDAs to perform business transactions, among other attractive applications. Anonymity protects buyers against intrusive merchants by preventing

them from sending the buyer unwanted advertisements. In everyday life, anonymity has other valuable applications which include personal love affairs, medical assistance and freedom to express political opinions. In general, it can be said that anonymity provides protection against intrusion, embarrassment and retaliation. Aside from these advantages, anonymity has several serious and negative side effects that make its deployment in the Internet a controversial issue. The potential danger that might result from its misuse could outweigh its benefits. It would not be difficult to write at length about the pros and cons of anonymity; there are strong arguments for and against it. Because of this, we believe, that before saying that it is a good or bad thing to have in the Internet and before saying that it should be legal or illegal, we have to bring it into practice to test it. Therefore, our position on the issue is that anonymity in the Internet should not be prohibited but regulated so that its potential danger is diminished.

Appendix A

Promela specification of the system

In this appendix we present a complete listing of the set of Promela modules we validated in chapter 7 and we present also the Promela specification of the whole system.

A.1 Promela code for validating the public key manager

```
/*
 * PROGRAMME: Promela validation model for the public key manager.
 * AUTHOR: Carlos Molina Jimenez;
 * ADDRESS: The University of Newcastle upon Tyne
 * DATE OF CREATION: 27 Jul 1999
 * DATE OF LAST UPDATE: 11 Nov 1999
 */

#define CERTIFIED_Kpu       12 /* Kpu certified by a certification authority      */
#define UNCERTIFIED_Kpu     10 /* Kpu not certified by a certification authority  */
#define INVALID_Kpu         00 /* Kpu out of domain                               */
#define MAXNUM_Kpu_ATTEMPTS 5  /* max number of attempts to receive valid Kpu     */

#define YES              1
#define NO               0
#define ANYINT           1  /* any integer                    */
#define NEGINT          -1  /* any negative integer           */
#define MAXNUMPDA        3  /* max number of PDAs in the MSS */

mtype= {Kpu,BogusKpu}

chan KpuPort_to_PDA=[1] of {byte,int}    /* broadcast channel from MSS to PDAs */


/*
 * KpuMan process: it places a msg in KpuPort_to_PDA channel. The msg is read by
 *                 PDAs. It contains certified Kpu key, uncertified Kpu key or bogus
 * message. When the msg is read, the KpuMan places another one, waits until
 * it is read and so on.
 */
proctype KpuMan()
{
 int Kpu_val;

 do
```

```
:: (true) ->
   if
   :: Kpu_val= CERTIFIED_Kpu
   :: Kpu_val= UNCERTIFIED_Kpu
   fi;

   end_cyc:
   if
   :: KpuPort_to_PDA ! Kpu(Kpu_val)
   :: KpuPort_to_PDA ! BogusKpu(INVALID_Kpu)
   fi
 od
}


/*
 * PDAses process: This code is the part the PDAses process and deals with
 *                  the Kpu negotiation between the PDA and the MSS.
 * If the Kpu is learnt successfully by the PDA, it moves to the KsNegotiation
 * state where it tries to obtain a Ks key. The Ks negotiation and the rest of
 * the PDAses code is provided in this appendix in "Promela specification of
 * the backbone of the system". So, this code just blocks when it reaches the
 * negotiation state. The PDAses ends in 'Aborted' if it fails to get the Kpu
 * after MAXNUM_Kpu_ATTEMPTS.
 */
proctype PDAses(int PDAnum)
{
 int PuKey;       /* public key of the MSS */
 int Ks= ANYINT;  /* session key suggested by PDA to MSS after a successful */
                  /* Kpu negotiation                                        */

 int i; /* count num of attempts */

KpuNegotiation:
 PuKey= 0;
 i= 1;
 do
 :: (i <= MAXNUM_Kpu_ATTEMPTS) ->
    if
    :: KpuPort_to_PDA ? Kpu(PuKey) -> /* Kpu rcvd: now check for authenticity */
       if
       :: (PuKey == CERTIFIED_Kpu) -> goto KsNegotiation

       :: else -> i++
       fi

    :: KpuPort_to_PDA ? BogusKpu(PuKey) -> i++ /* bogus msg rcvd: ignore it */

    :: timeout -> /* couldn't hear any msg from this MSS: move to new MSS */
       printf("PDAnum=%d Failed to hear from MSS, going to +++Aborted+++\n", PDAnum);
       goto Aborted
    fi
:: else ->
    printf("PDAnum=%d Failed to get Kpu after N= %d attempts;
            going to +++Aborted+++\n", PDAnum,i-1);
    goto Aborted
od;

KsNegotiation:
```

```
   printf("PDAnum=%d GOT CERTIFIED Kpu= %d after N=%d attempts\n", PDAnum, PuKey,i-1);
   assert(PuKey == CERTIFIED_Kpu);
   end: if
        :: (false) -> skip
        fi;

 Aborted:
  printf("PDAnum= %d  +++ABORTED+++ BYE-BYE-BYE\n",PDAnum)
}


/*
 * Initiate the KpuMan process and MAXNUMPDA processes attempting
 * to get a certified Kpu
 */
init
{
   byte PDAnum;  /* number given to a PDA */

   run KpuMan(); /* create Kpu manager process */

   atomic{       /* create MAXNUMPDA processes, one for each PDA */
   PDAnum= 0;
   do
   :: PDAnum < MAXNUMPDA -> run PDAses(PDAnum); PDAnum++
   :: PDAnum >= MAXNUMPDA  -> break
   od
   }
}
```

A.2 Promela code for validating the mail server process

```
/*
 * PROGRAMME: Promela validation model for validating the mail server process.
 * AUTHOR: Carlos Molina Jimenez;
 * ADDRESS: The University of Newcastle upon Tyne
 * DATE OF CREATION: 27 Jul 1999
 * DATE OF LAST UPDATE: 11 Nov 1999
 */

#define YES              1
#define NO               0
#define ANYINT           1  /* any integer                    */
#define NEGINT          -1  /* any negative integer           */
#define MAXNUMPDA        3  /* max num of PDA in the MSS      */
#define NUMofPDA         3  /* num of PDA visiting the MSS    */
#define QSZ              1  /* num of msg stored in a channel */

#define MAXNUMEMAILS_SENT 4 /* max num of msg sent by a PDA. This's a temporary   */
                            /* restriction to set a finite boundary for the model */

mtype= {aborted,EaddrAndTxt,YouHaveMail};

chan MSSses_to_MailSvr[MAXNUMPDA]=[QSZ] of {byte,int,int};
chan MailSvr_to_MSSses[MAXNUMPDA]=[QSZ] of {byte,int,int};
```

```
/*
 * MSSses process: receives messages from PDAs and replies from MailSvr.
 *                 When a msg is rcvd it is forwarded to the MailSvr process.
 * Replies are forwarded to the corresponding PDA. Messages from PDAs and
 * replies from MailSvr are not actually received here but simulated.
 */
proctype MSSses(int tmpId, chnum)
{
 int  addr; /* store msg address */
 int  txt;  /* store msg body    */
 int  NumM; /* num of msg        */

 NumM=MAXNUMEMAILS_SENT;

  do
  :: (NumM >= 1 && nfull(MSSses_to_MailSvr[chnum])) ->
        MSSses_to_MailSvr[chnum] ! EaddrAndTxt(_pid,_pid);
        NumM--

  :: MailSvr_to_MSSses[chnum] ? YouHaveMail(addr,txt) ->
        printf("MSSses:  rcvd reply for  PDA tmpId= %d \n",tmpId)

  :: (NumM < 1 && empty(MailSvr_to_MSSses[chnum]) && timeout) ->
        if
        :: nfull(MSSses_to_MailSvr[chnum]) ->
                MSSses_to_MailSvr[chnum] !! aborted(tmpId,chnum)

        :: full(MSSses_to_MailSvr[chnum]) && timeout ->
        fi;
        goto End
   od;

End:
 printf("\n\nMSSses: tmpId=%d has finished: HAPPY END BYE-BYE-BYE \n",tmpId)
}




/*
 * MailSvr process: receives messages from the MSSses and pretends to deliver
 *                  them to their final destination. If there is a reply to a
 * delivered message (randomly decided) the MailSvr forwards the reply to the
 * MSSses.
 */
proctype MailSvr()
{
 int  ReplyVec[MAXNUMPDA]; /* reply vector      */
 int  tmpId;               /* temporary Id      */
 int  chnum;               /* channel num       */
 int  addr;                /* store msg address */
 int  txt;                 /* store msg body    */
 byte msgtype; /* type of msg received                          */
 int  f1;      /* field for receiving the first field of a msg  */
 int  f2;      /* field for receiving the second field of a msg */
```

```
int  i;        /* counter                                          */

i=0;
do
:: i <  MAXNUMPDA ->
     ReplyVec[i]= 0;
     i++
:: else -> break
od;

chnum= -1;

do
:: (true) ->
   end_cyc: if
   :: MSSses_to_MailSvr[0] ? msgtype(f1,f2) ->
       assert(msgtype == EaddrAndTxt || msgtype == aborted);
       if
       :: (msgtype == EaddrAndTxt) ->
             addr= f1; txt= f2; ReplyVec[0]= ReplyVec[0] + 1;
             printf("MailSvr: has rcvd a msg from PDA tmpId=0 \n");
             goto  MailRcvd

       :: (msgtype == aborted) -> tmpId= 0; chnum= 0; goto ClearChan
       fi

   :: MSSses_to_MailSvr[1] ? msgtype(f1,f2) ->
       assert(msgtype == EaddrAndTxt || msgtype == aborted);
       if
       :: (msgtype == EaddrAndTxt) ->
             addr= f1; txt= f2; ReplyVec[1]= ReplyVec[1] + 1;
             printf("MailSvr: has rcvd a msg from PDA tmpId=1 \n");
             goto  MailRcvd

       :: (msgtype == aborted) -> tmpId= 1; chnum= 1; goto ClearChan
       fi

:: MSSses_to_MailSvr[2] ? msgtype(f1,f2) ->
       assert(msgtype == EaddrAndTxt || msgtype == aborted);
       if
       :: (msgtype == EaddrAndTxt) ->
             addr= f1; txt= f2; ReplyVec[2]= ReplyVec[2] + 1;
             printf("MailSvr: has rcvd a msg from PDA tmpId=2 \n");
             goto  MailRcvd

       :: (msgtype == aborted) -> tmpId= 2; chnum= 2; goto ClearChan
       fi
   fi;

MailRcvd:
   if
   :: (ReplyVec[0] >= 1) ->
                  if /* Alice doesn't reply to Bob: discharge a msg */
                  :: ReplyVec[0]= ReplyVec[0] -1
                  :: skip
                  fi;
                  if /* make chnum=0 if there's a reply for channel 0 */
                  :: (ReplyVec[0] >= 1) -> chnum= 0
                  :: else -> chnum= -1   /* no replies for channel 0  */
```

```
                fi
:: (ReplyVec[1] >= 1) ->
                if
                :: ReplyVec[1]= ReplyVec[1] -1
                :: skip
                fi;
                if
                :: (ReplyVec[1] >= 1) -> chnum= 1
                :: else -> chnum= -1
                fi
:: (ReplyVec[2] >= 1) ->
                if
                :: ReplyVec[2]= ReplyVec[2] -1
                :: skip
                fi;
                if
                :: (ReplyVec[2] >= 1) -> chnum= 2
                :: else -> chnum= -1
                fi
fi;

if
:: (chnum >= 0 && chnum < NUMofPDA) ->
      if
      :: MailSvr_to_MSSses[chnum] ! YouHaveMail(chnum,_pid) ->
         progress_replySent: skip
      :: timeout-> skip /* PDA is off or has left the MSS */
      fi;
           ReplyVec[chnum]= ReplyVec[chnum] -1;
    :: else-> skip
    fi;

    chnum= -1;
    goto end_cyc;

 ClearChan:
  do
  :: nempty(MailSvr_to_MSSses[chnum]) ->
           MailSvr_to_MSSses[chnum] ? msgtype(f1,f2) /* drop this msg */
  :: empty(MailSvr_to_MSSses[chnum]) -> break
  od;
  goto end_cyc

 od
}
```

```
/*
 * init process: it instantiates MailSvr process and three PDA
 *               processes.
 */
init
{
   int PDAnum; /* PDA number     */
   int tmpId;  /* temporary Id   */
   int chnum;  /* channel number */
```

```
atomic{
run MailSvr();  /* instantiate MailSvr process */
};

PDAnum= 0;
do
:: PDAnum <  NUMofPDA -> /* instantiate MailSvr process */
                 if
                 :: (PDAnum == 0) -> tmpId=0; chnum=0
                 :: (PDAnum == 1) -> tmpId=1; chnum=1
                 :: (PDAnum == 2) -> tmpId=2; chnum=2
                 fi;
                 run MSSses(tmpId, chnum);
                 PDAnum++
   :: PDAnum >= NUMofPDA  -> break
   od
}
```

A.3 Promela code for validating the bank process

```
/*
 * PROGRAMME: Promela validation model for the bank process.
 * AUTHOR: Carlos Molina Jimenez;
 * ADDRESS: The University of Newcastle upon Tyne
 * DATE OF CREATION: 27 Jul 1999
 * DATE OF LAST UPDATE: 11 Nov 1999
 */

#define MAXNUMPAYATTEMPTS    3 /* max num of attempts to pay for an anonymous session */
#define MAXMONEY           100 /* max money accepted for opening an anonymous call    */
#define GOLDENMONEY         20 /* golden is considered genuine and enough             */
#define MINMONEY            15 /* min money accepted for opening an anonymous call    */
#define TOOLITTLEMONEY       1 /* genuine money but not enough                        */
#define NOMONEY              0 /* no money                                            */
#define FAKEMONEY           -1 /* fake money                                          */
#define MAXEXTPAY           40 /* max amount of money for extra payment: the user can */
                               /* extend his call twice: 20+20 or 15+15 or 20+15      */

#define YES             1
#define NO              0
#define MAXNUMPDA       3  /* max num of PDA in the MSS      */
#define NUMofPDA        3  /* num of PDA visiting the MSS    */
#define QSZ             1  /* num of msg stored in a channel */
#define MAX_Ks          5  /* Valid Ks domain [1...5]        */
#define MIN_Ks          1  /*                                */

mtype= {aborted,abort,Ecash,EcashRejetd,EcashAccepd,GenuineEcash,FakeEcash};

chan MSSses_to_bank[MAXNUMPDA]=[QSZ] of {byte,int,int};
chan bank_to_MSSses[MAXNUMPDA]=[QSZ] of {byte,int,int};



/*
```

```
* MSSses process: simulates to receive a Ecash from a PDA (in fact
*                 the Ecash is created locally); upon receiving it, the
* Ecash is forwarded to the bank for verification of genuineness while
* the MSSses waits for a reply. If the reply is 'FakeCash', it fetches a
* different e-coin and tries again until it receives a 'GenuineEcash'
* answer or MAXNUMPAYATTEMPTS is exhausted. In the former case the
* MSSses enters the anonymous session state. In the latter case, the MSSses
* terminates in abortion.
* This piece of code helps validate the bank process ONLY, so it blocks
* when it reaches the anonymous session state.
 */
proctype MSSses(int tmpId, chnum)
{
 byte msgtype;  /* type of msg received                              */
 int  f1;       /* field for receiving the first field of a msg  */
 int  f2;       /* field for receiving the second field of a msg */
 int  payment;
 int  i;
 bool flag;

 flag= YES;
 i= 1;

FirstPayment: do
:: (i <= MAXNUMPAYATTEMPTS) ->
   if
   :: payment= FAKEMONEY
   :: payment= MINMONEY
   :: payment= GOLDENMONEY
   fi;

   MSSses_to_bank[chnum] ! Ecash(payment,tmpId) ->
   bank_to_MSSses[chnum] ? msgtype(f1,f2);
     if
     :: (msgtype == FakeEcash) ->
           i++  /* go back and fetch another coin from PDA memory */

     :: (msgtype == GenuineEcash) -> payment= f1;
            goto end_AnoSes /* Ecash has been accepted by bank, go and */
     fi;                    /* open an anonymous session for PDA       */

:: else -> MSSses_to_bank[chnum] !! aborted(tmpId,chnum);
           goto Aborted
od;

end_AnoSes:
 do
 :: (flag == YES) ->
    printf("\nMSSses: AnoSes opened for PDA with tmpId=%d (payment=%d )\n",tmpId,payment);
    flag = NO;
 od;

Aborted:
 printf("\n\nMSSses: PDA with tmpId=%d failed to pay: +++ABORTED+++ BYE-BYE-BYE \n",tmpId)
}
```

```
/*
 * bank process: receives Ecash from the MSS and checks for genuineness;
 *               if it is satisfied with the Ecash it replies by sending an
 * 'GenuineEcash' msg, in the opposite case, it replies with 'FakeEcash'
 * msg. Upon sending a reply, it goes back to wait for the next 'Ecash'
 * msg to arrive. For the purpose of the validation, Ecash of value -1
 * is considered to be fake money.
 */
proctype bank()
{
 int tmpId;     /* temporary Id   */
 int chnum;     /* channel number */
 int  payment;
 byte msgtype;  /* type of msg received                          */
 int  f1;       /* field for receiving the first field of a msg  */
 int  f2;       /* field for receiving the second field of a msg */

 chnum= -1;

 do
 :: (true) ->
    end_cyc:if
    :: MSSses_to_bank[0] ? msgtype(f1,f2) ->
         assert(msgtype == Ecash || msgtype == aborted);
         if
         :: (msgtype == aborted) -> chnum= 0; tmpId=0; goto ClearChan
         :: (msgtype == Ecash)   -> payment=f1; chnum= 0; tmpId=0; goto EcashTocheck
         fi

    :: MSSses_to_bank[1] ? msgtype(f1,f2) ->
         assert(msgtype == Ecash || msgtype == aborted);
         if
         :: (msgtype == aborted) -> chnum= 1; tmpId=1; goto ClearChan
         :: (msgtype == Ecash)   -> payment=f1; chnum= 1; tmpId=1; goto EcashTocheck
         fi

    :: MSSses_to_bank[2] ? msgtype(f1,f2) ->
         assert(msgtype == Ecash || msgtype == aborted);
         if
         :: (msgtype == aborted) -> chnum= 2; tmpId=2; goto ClearChan
         :: (msgtype == Ecash)   -> payment=f1; chnum= 2; tmpId=2; goto EcashTocheck
         fi
    fi;

 EcashTocheck: skip;
 /* proc with tmpId= 0 has chnum=0, proc with tmpId=1 has chnum=2, etc */
 assert(0 <= chnum && chnum < NUMofPDA && 0 <= tmpId && tmpId < NUMofPDA);
 if
 :: (payment == FAKEMONEY) -> bank_to_MSSses[chnum] ! FakeEcash(payment,f2);
     printf("BANK: rcvd FakeCash payment=%d from PDA with tmpId=%d \n",payment,tmpId)
 :: else -> bank_to_MSSses[chnum] ! GenuineEcash(payment,f2) ->
     printf("BANK: rcvd GenuineEcash payment=%d from PDA with tmpId=%d \n",payment,tmpId)
 fi;
 goto end_cyc;


ClearChan:
 do
```

```
    :: nempty(bank_to_MSSses[chnum]) ->
        bank_to_MSSses[chnum] ? msgtype(f1,f2) -> skip /* drop this msg */

    :: empty(bank_to_MSSses[chnum]) -> break
    od;
    goto end_cyc

 od
}
```

```
/*
 * init process: instantiates the bank process and
 *               three PDA processes
 */
init
{
 int PDAnum; /* number given to PDA */
 int tmpId;  /* temporary Id        */
 int chnum;  /* channel number      */


 atomic{
   run bank();    /* instantiate the bank process */
 };

 PDAnum= 0;
 do
 :: PDAnum <  NUMofPDA -> /* instantiate 3 PDA processes */
        if
        :: (PDAnum == 0) -> tmpId=0; chnum=0
        :: (PDAnum == 1) -> tmpId=1; chnum=1
        :: (PDAnum == 2) -> tmpId=2; chnum=2
        fi;
        run MSSses(tmpId, chnum);
        PDAnum++
 :: PDAnum >= NUMofPDA  -> break
 od
}
```

A.4 Promela code for validating the backbone of the system

```
/*
 * PROGRAMME: Promela validation model for the backbone of the system
 * AUTHOR: Carlos Molina Jimenez;
 * ADDRESS: The University of Newcastle upon Tyne
 * DATE OF CREATION: 27 Jul 1999
 * DATE OF LAST UPDATE: 11 Nov 1999
 */

#define MAXNUMPAYATTEMPTS    3 /* max num of attempts to pay for an anonymous session */
#define MAXMONEY           100 /* max money accepted for opening an anonymous call    */
#define GOLDENMONEY         20 /* golden, is considered genuine                       */
```

```
#define MINMONEY            15 /* min money accepted for opening an anonymous call    */
#define TOOLITTLEMONEY       1 /* genuine money but not enough                        */
#define NOMONEY              0 /* no money                                            */
#define FAKEMONEY           -1 /* fake money                                          */
#define MAXEXTPAY           40 /* max amount of money for extra payment: the user can */
                               /* extend his call twice: 20+20 or 15+15 or 20+15      */

#define LAST_MSGS           10 /* Num of msg left after TimeExp alarm goes off        */

#define YES              1
#define NO               0
#define ANYINT           1  /* any integer, the value is not important */
#define NEGINT          -1  /* any negative integer, the value is not important */
#define MAXNUMPDA        3  /* max number of PDA in the MSS */
#define NUMofPDA         3  /* num of PDA visiting the MSS  */
#define QSZ              1  /* num of msgs that a channel can store  */
#define LOCALQSZ         3  /* num of msgs that local chan can store */
#define MAX_Ks           5  /* Valid Ks domain [1...5]               */
#define MIN_Ks           1  /*                                       */
#define MAXNUM_Ks_ATTEMPTS  3  /* max num of attempt to register Ks       */
#define MAXNUMMSGREAD       4  /* max num of attempt to receive an        */
                               /* answer from the MSS encrypted with      */
                               /* the suggested Ks and containing a tmpId */

#define MAXNUMEMAILS_SENT  16  /* temporary restriction for validation purposes */

#define KsBlack 1
#define KsBlue  2
#define KsGreen 3
#define KsPink  4
#define KsWhite 6   /* <-- invalid key */

mtype= {aborted,abort,TimeFin,TimeAlert,YourNewKs,ChangeMyKs,
        tmpIdRcvd,Ecash,EcashRejetd, EcashAccepd,GenuineEcash,
        FakeEcash,EaddrAndTxt,YouHaveMail};
        /* This is equiv. to YouHaveMail=1, EaddrAndTxt=2, FakeECash=3,... */

/*
 * The MSS receives Ks proposals from PDAs at its PDA_to_KsPort channel
 * the PDA read the MSS reply from KsPort_to_PDA channel which can
 */
chan PDA_to_KsPort=[1] of {int};
chan KsPort_to_PDA=[NUMofPDA]  of {int,int,int}

chan KsTmpIdMan_to_MSSses[MAXNUMPDA]=[QSZ] of {byte,int,int};
chan MSSses_to_KsTmpIdMan[MAXNUMPDA]=[QSZ] of {byte,int,int};

chan PDAses_to_tcp[MAXNUMPDA]=[QSZ] of {byte,int,int};
chan PDAtcp_to_ses[MAXNUMPDA]=[QSZ] of {byte,int,int};

chan MSSses_to_tcp[MAXNUMPDA]=[QSZ] of {byte,int,int};
chan MSStcp_to_ses[MAXNUMPDA]=[QSZ] of {byte,int,int};

chan MSStcp_to_PDAtcp[MAXNUMPDA]=[QSZ] of {byte,int,int};
chan PDAtcp_to_MSStcp[MAXNUMPDA]=[QSZ] of {byte,int,int};

chan PDAuser_to_ses[MAXNUMPDA]=[QSZ] of {byte,int,int};
chan PDAses_to_user[MAXNUMPDA]=[QSZ] of {byte,int,int};
```

```
/*
 *  KsTmpIdMan process: is in charge of managing session keys and temporary
 *                      Id assigned to PDAs. It guarantees that tmpIds assigned
 * to PDAs are unique. Also, it guarantees that session keys are unique and
 * secret. It asks a PDA to change its Ks when it detects that is has been
 * hit by another PDA. Also, upon request, it provides a PDA with a new Ks.
 */
proctype KsTmpIdMan()
{
 int tmpIdVec[MAXNUMPDA];  /* vector for storing TmpIds */
 int KsVec[MAXNUMPDA];     /* vector for storing Ks     */
 int Ks;                   /* session key               */
 int tmpId;                /* temporary Id              */
 int chnum;                /* channel number            */

 int OldestKs;     /* oldest Ks accepted by MSS and sent to KsPort_to_PDA */

 byte msgtype;  /* type of msg received                                    */
 int  f1;       /* field for receiving the first field of a msg            */
 int  f2;       /* field for receiving the second field of a msg           */
 int fh1;       /* scratch field for receiving the first field of a msg    */
 int fh2;       /* scratch field for receiving the second field of a msg   */
 int NewKey;  /* new Ks key */
 int count;   /* counter    */
 int n;       /* counter    */
 int i;       /* counter    */
 int j;       /* counter    */

 i=0;  /* initialize to 0 and -1 the valid cells in vectors KsVec */
 do    /* and tmpIdVec, respectively                              */
 :: i <  NUMofPDA ->
    assert(0 <= i && i < MAXNUMPDA);
    KsVec[i]= 0;
    tmpIdVec[i]= -1;
    i++
 :: else -> break
 od;

 i= NUMofPDA;  /* initialize to -3 the unused cells in the vectors */
 do            /* KsVec and tmpIdVec respectively                  */
 :: i <  MAXNUMPDA ->
    assert(NUMofPDA <= i && i < MAXNUMPDA);
    KsVec[i]= -3;
    tmpIdVec[i]= -3;
   i++
 :: else -> break
 od;



TestTmpIdKs:
/*
 * test that tmpId assigned to PDA are unique */
 */
atomic{
```

```
i=0;
do
:: (i < MAXNUMPDA) ->
     if
     :: (tmpIdVec[i] >= 0) -> assert(tmpIdVec[i] < NUMofPDA);
                j=0;
                do
                :: (j < MAXNUMPDA) ->
                      if
                      :: (i == j) -> j++
                      :: else ->
                               assert(tmpIdVec[i] != tmpIdVec[j]);
                               j++
                      fi
                :: else -> break /* move to next i */
                od

     :: else -> skip /*  tmpIdVec[i] < 0 are unused elements */
     fi;
     i++
:: else -> break /* comparison finished */
od};

/*
 * test that Ks held by PDA are unique */
 */
atomic{
i=0;
do
:: (i < MAXNUMPDA) ->
     if
     :: (KsVec[i] > 0) -> assert(KsVec[i] <= MAX_Ks);
                j=0;
                do
                :: (j < MAXNUMPDA) ->
                      if
                      :: (i == j) -> j++
                      :: else ->
                               assert(KsVec[i] != KsVec[j]);
                               j++
                      fi
                :: else -> break /* move to next i */
                od

     :: else -> skip /*  KsVec[i] < 0 are unused elements */
     fi;
     i++
:: else -> break /* comparison finished */
od};



tmpIdKsOK: /* tmpId and Ks are all right, continue */
Ks= 0;

end: do  /* valid endstate of this server process */
:: PDA_to_KsPort ? Ks -> goto KsVerification

:: MSSses_to_KsTmpIdMan[0] ? msgtype(f1,f2)->
```

```
                                chnum=0; tmpId= 0; goto MSSsesCare

:: MSSses_to_KsTmpIdMan[1] ? msgtype(f1,f2)->
                                chnum=1; tmpId= 1; goto MSSsesCare

:: MSSses_to_KsTmpIdMan[2] ? msgtype(f1,f2)->
                                chnum=2; tmpId= 2; goto MSSsesCare

od;

KsVerification:
 if
 :: (Ks <= MAX_Ks && Ks >= MIN_Ks) -> /* Is Ks within the valid domain? */

     i=0;
     do /* is Ks already in use by another PDA? */
     :: (i < NUMofPDA) -> /* valid Ks are [1,2,3,4] */
        if
        :: (Ks == KsVec[i]) ->
            printf("KsTmpIdMan: Ks=%d KsVec[%d]=%d is in use by tmpId[%d]=%d \n",
            Ks,i,KsVec[i],i,tmpIdVec[i]);
            goto KsInuse

        :: (Ks != KsVec[i]) -> i++
        fi;
     :: else -> goto KsNotInuse
     od;
:: else ->  /* suggested Ks is incorrect */
     printf("KsTmpIdMan: Ks= %d  is incorrect no reply\n",Ks);
     goto TestTmpIdKs
fi;

KsNotInuse:
  /* Find a free tmpId in tmpIdVec and assign it to PDA                        */
  /* The values of valid tmpId are [0,1,2,3,4]                                 */
  /* tmpIdVec[0]=  0|1|2|3|4 means  tmpId=0 already being used by some PDA     */
  /* tmpIdvec[0]= -1  means  tmpId=0 is free                                   */
  /* tmpIdvec[0]= -2  means  tmpId=0 has been used and can't be recycled yet   */
  /* tmpIdvec[0]= -3  means  tmpId=0 is not in use at all                      */
  /* tmpIdVec[1]=  0|1|2|3|4 means  tmpId=1 already being used by some PDA     */
  /* tmpIdvec[1]= -1  means  tmpId=1 is free                                   */
  /* tmpIdvec[1]= -2  means  tmpId=1 has been used and can't be recycled yet   */
  /* tmpIdvec[1]= -3  means  tmpId=1 is not in use at all                      */
  /* tmpIdVec[2]=  0|1|2|3|4 means  tmpId=2 already being used by some PDA     */
  /* tmpIdvec[2]= -1  means  tmpId=2 is free                                   */
  /* tmpIdvec[2]= -2  means  tmpId=2 has been used and can't be recycled yet   */
  /* tmpIdvec[2]= -3  means  tmpId=2 is not in use at all                      */

  i=0;
  do
  :: (i < NUMofPDA) -> /* tmpId= -3 not in use at all                          */
     if                /* tmpId= -2 not in use: to be recycled later           */
     :: (tmpIdVec[i] != -1) -> i++ /* tmpId is already in use, try next one    */
     :: else -> break              /* tmpId is free, use it                    */
     fi
  :: else -> /* no more tmpId to be assigned: take this as KsINUSE= YES       */
             /* and don't reply no anybody                                    */
```

```
            goto TestTmpIdKs
od;
tmpId= i;  chnum=i; /* tmpId & chnum assigned to new PDA with session key= Ks */
if
:: atomic{nfull(KsPort_to_PDA) -> /* there's space in ch for key(tmpId,chnum) */
    KsPort_to_PDA ! Ks(tmpId,chnum);
    tmpIdVec[i]= i;
    KsVec[i]= Ks;
    run MSSses(tmpId,chnum,Ks)} /* initiate MSS session to take care of new PDA */

:: atomic{full(KsPort_to_PDA) -> /* no space in ch, discharge oldest msg */
     KsPort_to_PDA ? OldestKs(f1,f2);
     tmpIdVec[f1]= -2;              /* empty PDAj from your tables          */
     KsVec[f1]= -2;                 /* and terminate MSSses assigned to PDAj */
     KsTmpIdMan_to_MSSses[f1] !! abort(f1,f2);
     MSSses_to_KsTmpIdMan[f1] ? aborted(fh1,fh2); /* fh1,fh2 scratch var */
     chnum= f1;
     do
     :: nempty(KsTmpIdMan_to_MSSses[chnum]) ->
              KsTmpIdMan_to_MSSses[chnum] ? msgtype(f1,f2) /* drop this msg */
     :: empty(KsTmpIdMan_to_MSSses[chnum]) -> break
     od;
     printf("KsTmpIdMan:removed from ch: Ks=%d tmpId=%d chnum=%d \n",
           OldestKs,f1,f2)}
fi;
goto TestTmpIdKs;

/*
 * Ks suggested by the new PDA is already in use by PDA with tmpId=i and
 * chnum=i. KsMan has to get a new Ks for this PDA and send it to new PDA
 */
KsInuse:
  printf("KsTmpIdMan: suggested Ks=%d is already in use by PDA with tmpId=%d
        chnum=%d\n",Ks,i,i);
  count= 1;
  do
  :: (count <= MAXNUM_Ks_ATTEMPTS) ->
     if
     :: NewKey= KsBlack
     :: NewKey= KsBlue
     :: NewKey= KsGreen
     :: NewKey= KsPink
     /* :: NewKey= KsWhite is out, we assume KsTmpIdMan generates correct Ks only */
     fi;
     n=0;
     do
     :: (n < NUMofPDA) ->
        assert( 0 <= n && n < NUMofPDA);
        if
        :: (KsVec[n] == NewKey) -> count++; break
        :: (KsVec[n] != NewKey) -> n++
        fi;
     :: else -> goto NewKsFound1
     od;
:: else -> /* couldn't find any available Ks */
        atomic{
        KsTmpIdMan_to_MSSses[i] !! abort(i,Ks) -> /* order the MSSses to abort */
        MSSses_to_KsTmpIdMan[i] ? aborted(f1,f2);
        tmpIdVec[i]= -2;
```

```
            KsVec[i]= -2;
            chnum= tmpId;
            do
            :: nempty(KsTmpIdMan_to_MSSses[chnum]) ->
                     KsTmpIdMan_to_MSSses[chnum] ? msgtype(f1,f2) /* drop this msg */
            :: empty(KsTmpIdMan_to_MSSses[chnum]) -> break
            od;
            printf("KsTmpIdMan: has SENT ABORT to tmpId= %d \n",i);
            goto TestTmpIdKs}
    od;

    NewKsFound1:
    assert((MIN_Ks <= NewKey)&&(NewKey <= MAX_Ks)&&(0 <= i)&&(i < NUMofPDA));
    if
    :: atomic{KsTmpIdMan_to_MSSses[i] ! YourNewKs(NewKey,Ks) ->
              KsVec[i]= NewKey}

    :: timeout -> printf("KsTmpIdMan: tmpId=%d chnum=%d Ks=%d NOT THERE,
                  ignore it\n",i,i,Ks)
    fi;
    goto TestTmpIdKs;



MSSsesCare:
 if
 :: (msgtype == ChangeMyKs) ->
 printf("KsTmpIdMan: rcvd ChangeMyKs from PDA-tmpId=%d chnum=%d\n",tmpId,chnum);
 count= 1;
 do
 :: (count <= MAXNUM_Ks_ATTEMPTS) ->
    if
    :: NewKey= KsBlack
    :: NewKey= KsBlue
    :: NewKey= KsGreen
    :: NewKey= KsPink
    /* :: NewKey= KsWhite is out, we assume KsTmpIdMan generates correct Ks only */
    fi;
    n=0;
    do
    :: (n < NUMofPDA) ->
       assert( 0 <= n && n < NUMofPDA);
       if
       :: (KsVec[n] == NewKey) -> count++; break
       :: (KsVec[n] != NewKey) -> n++
       fi;
    :: else -> goto NewKsFound2
    od;
 :: else -> /* couldn't find any available Ks */
       atomic{
       KsTmpIdMan_to_MSSses[chnum] !! abort(tmpId,chnum); /* order MSSses to abort */
       do
       :: MSSses_to_KsTmpIdMan[chnum] ? msgtype(f1,f2)->
           if
           :: (msgtype != aborted) -> skip
           :: (msgtype == aborted) ->
                tmpIdVec[tmpId]= -2;
                KsVec[tmpId]= -2;
                chnum= tmpId;
```

```
                    do
                    :: nempty(KsTmpIdMan_to_MSSses[chnum]) ->
                          KsTmpIdMan_to_MSSses[chnum] ? msgtype(f1,f2) /* drop this msg */
                    :: empty(KsTmpIdMan_to_MSSses[chnum]) -> break
                    od;
                    printf("KsTmpIdMan: has SENT ABORT to tmpId= %d \n",tmpId);
                    goto TestTmpIdKs
                fi
            od}
    od;

    NewKsFound2:
    /* assert((MIN_Ks <= NewKey)&&(NewKey <= MAX_Ks)&&(0 <= tmpId)&&(tmpId < NUMofPDA)); */
     assert(MIN_Ks <= NewKey && NewKey <= MAX_Ks);
     assert(0 <= tmpId && tmpId < NUMofPDA && tmpId == chnum);

    if
    :: atomic{KsTmpIdMan_to_MSSses[tmpId] ! YourNewKs(NewKey,f1) ->
              KsVec[tmpId]= NewKey}

    :: timeout -> printf("KsTmpIdMan: tmpId=%d with chnum=%d Ks=%d NOT THERE,
       ignore it\n",tmpId,chnum,f1)
    fi;



:: (msgtype == aborted) -> /* PDA or MSSses want abort the ano. ses. */
       atomic{
       printf("KsTmpIdMan: going to send ABORT to tmpId= %d \n",tmpId);
       tmpIdVec[tmpId]= -2;
       KsVec[tmpId]= -2;
       chnum= tmpId;
       do
       :: nempty(KsTmpIdMan_to_MSSses[chnum]) ->
                 KsTmpIdMan_to_MSSses[chnum] ? msgtype(f1,f2) /* drop this msg */
       :: empty(KsTmpIdMan_to_MSSses[chnum]) -> break
       od;
       printf("KsTmpIdMan: ABORT FOR tmpId= %d \n",tmpId)}

    :: else -> skip /* UNKNOWN msg: transient failure?*/
    fi;
    goto TestTmpIdKs
}
```

```
/*
 * MSSses process: is in charge of managing the anonymous communication session of
 *                 the PDA. It charges the PDA for the communication, receives the
 * payment, forwards it to the bank for verification, accepts or rejects the payment,
 * warns the PDA user about the prepaid time expiration and abruptly finishes the
 * communication session when the prepaid time expires. Also, it is the link between
 * the PDA and the session and TmpId manager; and the link between the PDA and the
 * mail server.
 */
proctype MSSses(int tmpId, chnum, Ks)
{
```

```
chan q_to_MSStcp=[LOCALQSZ]      of {byte,int,int};
chan q_to_KsTmpIdMan=[LOCALQSZ] of {byte,int,int};
chan q_to_MailSvr=[LOCALQSZ]     of {byte,int,int};
chan q_to_bank=[LOCALQSZ]        of {byte,int,int};

byte msgtype;     /* type of msg received                                  */
int  f1;          /* field for receiving the first field of a msg  */
int  f2;          /* field for receiving the second field of a msg */
int  MyKs;        /* a session key          */
int  addr;        /* e-mail address         */
 int  txt;        /* text in an e-amil msg */
 int  payment;    /* payment for anonymous session: one unit of money is converted  */
                  /* to one msg to be sent                                          */
 int  credit;     /* prepaid payment: num of msg the PDA user has prepaid for       */
 int  CreditLeft; /* prepaid payment before warning msg: number of msg the user can */
                  /* send after the expiration time warning and termination of the  */
                  /* session                                                        */

 run MSStcp(tmpId,chnum,Ks);


FirstPayment:
 do
 :: MSStcp_to_ses[chnum] ? msgtype(f1,f2) ->
      if
      :: (msgtype == Ecash) ->
          payment= f1; tmpId= f2;
if
:: (payment == FAKEMONEY || payment < MINMONEY || payment > MAXMONEY) ->
     printf("MSSses: rcvd FakeEcash | too little or to big money=%d from
             PDA-tmpId=%d\n",payment,tmpI d);
     if
     :: (nfull(MSSses_to_tcp[chnum])) ->
         MSSses_to_tcp[chnum] ! EcashRejetd(payment,tmpId)
     :: timeout -> skip /* MSStcp is sending aborted go and read it */
     fi
:: (payment != FAKEMONEY && MINMONEY <= payment && payment <= MAXMONEY) ->
     printf("MSSses: rcvd GenuineEcash=%d from PDA-tmpId=%d\n",payment,tmpId);
     if
     :: (nfull(MSSses_to_tcp[chnum])) ->
         MSSses_to_tcp[chnum] ! EcashAccepd(payment,tmpId);
         assert(payment > LAST_MSGS);
         credit= payment - LAST_MSGS;
         CreditLeft= LAST_MSGS;
         goto AnoSes
             :: timeout -> skip /* MSStcp is sending aborted, go read it */
             fi
        fi
     :: (msgtype == aborted) -> goto AbortBank
     fi

::  KsTmpIdMan_to_MSSses[chnum] ? abort(f1,f2) -> MSSses_to_tcp[chnum] !! abort(tmpId,chnum) ->
       do
       :: MSStcp_to_ses[chnum] ? msgtype(f1,f2) ->
            if
            :: (msgtype != aborted) -> skip /* discard the msg */
            :: (msgtype == aborted) ->
                printf("MSSses PDA-tmpId=%d aborted by KsTmpIdMan; going +++Aborted+++\n",tmpId);
                goto AbortBank
```

```
            fi
        od
 od;



 assert( 0<= chnum && chnum <= NUMofPDA);

AnoSes:
 do
   /* to MSStcp */
:: KsTmpIdMan_to_MSSses[chnum]  ? [abort(f1,f2)] && nfull(q_to_MSStcp) ->
    KsTmpIdMan_to_MSSses[chnum] ? abort(f1,f2) -> q_to_MSStcp !! abort(f1,f2)

   /* to KsTmpIdMan */
:: MSStcp_to_ses[chnum] ? [aborted(f1,f2)] && nfull(q_to_KsTmpIdMan) ->
    MSStcp_to_ses[chnum] ? aborted(f1,f2) -> q_to_KsTmpIdMan !! aborted(f1,f2)

   /* to MSStcp */
:: KsTmpIdMan_to_MSSses[chnum]  ? [YourNewKs(f1,f2)] && nfull(q_to_MSStcp) ->
    KsTmpIdMan_to_MSSses[chnum] ? YourNewKs(f1,f2) -> q_to_MSStcp ! YourNewKs(f1,f2)

   /* to KsTmpIdMan */
:: MSStcp_to_ses[chnum]  ? [ChangeMyKs(f1,f2)] && nfull(q_to_KsTmpIdMan) ->
    MSStcp_to_ses[chnum] ? ChangeMyKs(f1,f2) -> q_to_KsTmpIdMan ! ChangeMyKs(f1,f2)

   /* to MailSvr */
:: MSStcp_to_ses[chnum]  ? [EaddrAndTxt(f1,f2)] && nfull(q_to_MailSvr) ->
    MSStcp_to_ses[chnum] ? EaddrAndTxt(f1,f2) -> q_to_MailSvr ! EaddrAndTxt(f1,f2)

   /* to bank */
:: MSStcp_to_ses[chnum]  ? [Ecash(f1,f2)] && nfull(q_to_bank) ->
    MSStcp_to_ses[chnum] ? Ecash(f1,f2) -> q_to_bank ! Ecash(f1,f2)

   /* send to MSStcp */
:: nempty(q_to_MSStcp) && nfull(MSSses_to_tcp[chnum]) ->
    q_to_MSStcp ? msgtype(f1,f2);
    if
    :: (msgtype == abort) ->  MSSses_to_tcp[chnum] !! abort(f1,f2)

    :: (msgtype == YourNewKs) -> MSSses_to_tcp[chnum] ! YourNewKs(f1,f2);
                                 MyKs= f1;

    :: (msgtype == GenuineEcash) ->
          payment= f1;
          if
          :: (payment < MINMONEY || payment > MAXMONEY) ->
             MSSses_to_tcp[chnum] ! EcashRejetd(payment,f2)

          :: else ->
             atomic{MSSses_to_tcp[chnum] ! EcashAccepd(payment,f2) ->
               assert(payment > LAST_MSGS);
               credit= payment + credit + CreditLeft;
               credit= credit - LAST_MSGS;
               CreditLeft= LAST_MSGS}
          fi
:: (msgtype == FakeEcash) -> MSSses_to_tcp[chnum] ! EcashRejetd(f1,f2)

:: (msgtype == TimeAlert) ->  MSSses_to_tcp[chnum] !! TimeAlert(f1,f2)
```

```
    :: (msgtype == TimeFin) -> MSSses_to_tcp[chnum] !! TimeFin(f1,f2)

    :: else -> printf("MSSses: tmpId=%d UNKNOWN msg: PANIC-PANIC-PANIC\n\n",tmpId)
    fi

  /* send to KsTmpIdMan */
:: nempty(q_to_KsTmpIdMan) && nfull(MSSses_to_KsTmpIdMan[chnum]) ->
    q_to_KsTmpIdMan ? msgtype(f1,f2);
    if
    :: (msgtype == aborted) -> goto Aborted

    :: (msgtype == ChangeMyKs) -> MSSses_to_KsTmpIdMan[chnum] ! ChangeMyKs(f1,f2)

    fi

:: nempty(q_to_MailSvr) && /* pretend to send msg to MailSvr */
    nfull(q_to_MSStcp) && (credit >=1 || CreditLeft >= 1) ->
    q_to_MailSvr ? msgtype(f1,f2);
    if
    :: (msgtype == EaddrAndTxt) ->
        /* if necessary process the msg */
        assert(credit >= 0 && CreditLeft >= 0);
        printf("MSSses: credit=%d CreditLeft=%d for PDA-tmpId=%d\n\n",credit,
                CreditLeft,tmpId);
        if
        :: (credit >= 1) -> /* charge to 'credit' account */
              /* MSSses_to_MailSvr[chnum] ! EaddrAndTxt(f1,f2); */ skip;
              credit--;
              if
              :: (credit == 0) -> q_to_MSStcp! TimeAlert(f1,f2);
              :: else -> skip
              fi
         :: (credit == 0  && CreditLeft >= 1) -> /* charge to CreditLeft' account */
              /* MSSses_to_MailSvr[chnum] ! EaddrAndTxt(f1,f2); */ skip;
              CreditLeft--;
              if
              :: (CreditLeft == 0) -> q_to_MSStcp ! TimeFin(f1,f2)
              :: else -> skip
              fi
        fi
    fi

:: nempty(q_to_bank) && nfull(q_to_MSStcp) -> q_to_bank ? msgtype(f1,f2);
    if
    ::  (msgtype == Ecash) ->
        payment= f1;
        printf("MSSses: PDA-tmpId=%d has SNT extra payment=%d to bank \n",
               tmpId,payment);
        assert(0 <= chnum && chnum < NUMofPDA && 0 <= tmpId && tmpId < NUMofPDA);
        if
        :: (payment == FAKEMONEY) -> q_to_MSStcp ! FakeEcash(payment,f2);
           printf("BANK: PDA-tmpId=%d chnum= %d SENT FakeCash payment=%d \n",
                  tmpId,chnum,payment)

        :: else -> q_to_MSStcp ! GenuineEcash(payment,f2) ->
            printf("BANK: PDA-tmpId=%d chnum= %d SENT GenuineEcash payment=%d \n",
                   tmpId,chnum,payment)
        fi;
```

```
        :: else ->
             printf("MSSses: tmpId=%d has SENT UNKNOWN msg, ignore it \n",tmpId)
        fi
 od;



Aborted:
 skip;

AbortBank:
 skip;

 /*
  * send abort to KsTmpIdMan
  */
  MSSses_to_KsTmpIdMan[chnum] !! aborted(f1,f2);

  /*
   * clear chan to MSStcp
   */
  do
  :: atomic{nempty(MSSses_to_tcp[chnum]) ->
            MSSses_to_tcp[chnum] ? msgtype(f1,f2)} /* drop this msg */
  :: empty(MSSses_to_tcp[chnum]) ->  break
  od;
 printf("MSSAnoSes tmpId=%d MyKs=%d +++ABORTED++++ BYE-BYE-BYE by KsTmpIdMan \n\n",
        tmpId,MyKs)
}



/*
 * MSStcp process: connection-oriented reliable link between the MSS
 *                 and the PDA. It forwards msg up the protocol stack
 * and to the PDA side. The MSStcp initiates an abort procedure that
 * propagates up the stack protocol whenever it detects that its remote
 * peer (the PDAtcp) is unreachable.
 */
proctype MSStcp(int tmpId, chnum, Ks)
{
 chan q_to_PDAtcp= [LOCALQSZ] of {byte, int, int};
 chan q_to_MSSses= [LOCALQSZ] of {byte, int, int};

 byte msgtype;  /* type of msg received                              */
 int  f1;       /* field for receiving the first field of a msg  */
 int  f2;       /* field for receiving the second field of a msg */

 assert(0 <= chnum && chnum <= NUMofPDA && tmpId == chnum);


 do
 :: nfull(q_to_PDAtcp) && nempty(MSSses_to_tcp[chnum]) ->  /* recv from MSS */
      MSSses_to_tcp[chnum] ? msgtype(f1,f2) -> q_to_PDAtcp ! msgtype(f1,f2)

 :: nfull(q_to_MSSses) && nempty(PDAtcp_to_MSStcp[chnum]) -> /* recv from PDA */
```

```
      PDAtcp_to_MSStcp[chnum] ? msgtype(f1,f2) -> q_to_MSSses ! msgtype(f1,f2)

 :: nempty(q_to_MSSses) && nfull(MSStcp_to_ses[chnum]) -> /* send to MSS */
      q_to_MSSses ? msgtype(f1,f2);
      if
      :: (msgtype == abort) -> /* abort initiated at PDA site */
            MSStcp_to_ses[chnum] !! aborted(f1,f2);
            printf("MSStcp: tmpId=%d initiated at PDA site going +++Aborted+++\n",tmpId);
            goto Aborted

      :: (msgtype == aborted) -> /* abort initiated at MSS */
            MSStcp_to_ses[chnum] !! aborted(f1,f2);
            printf("MSStcp: tmpId=%d initiated at MSS site going +++Aborted+++\n",tmpId);
            goto Aborted

      :: else ->  /* PDA sending a routine msg */
                  MSStcp_to_ses[chnum] ! msgtype(f1,f2)
      fi

    /* send to PDA site */
 :: nempty(q_to_PDAtcp) && nfull(MSStcp_to_PDAtcp[chnum]) && nfull(q_to_MSSses) ->
    q_to_PDAtcp ? msgtype(f1,f2);
      if
      :: (msgtype == abort) -> /* abort initiated by MSS or KsTmpIdMan */
           MSStcp_to_PDAtcp[chnum] !! abort(f1,f2);
           q_to_MSSses !! aborted(f1,f2);
           printf("MSStcp: tmpId=%d aborted by MSS going to +++Aborted+++ soon \n",tmpId);

      :: (msgtype == TimeFin) -> /* End of AnosSes initiated by MSS or KsTmpIdMan */
           MSStcp_to_PDAtcp[chnum] !! TimeFin(f1,f2);
           q_to_MSSses !! aborted(f1,f2);
           printf("MSStcp: tmpId=%d TimeFin by MSS going to +++Aborted+++ soon \n",tmpId);

      :: (msgtype == YourNewKs) -> /* MSSman has changed Ks */
           Ks= f1;
           MSStcp_to_PDAtcp[chnum] ! YourNewKs(f1,f2)

      :: else -> MSStcp_to_PDAtcp[chnum] ! msgtype(f1,f2) /* MSS sent ordinary msg */
      fi


 :: full(MSStcp_to_PDAtcp[chnum]) &&
    timeout -> /* PDA not receiving: assume PDA has aborted its session */
      if
      :: nfull(MSStcp_to_ses[chnum]) -> MSStcp_to_ses[chnum] !! aborted(f1,f2);
                                        goto Aborted

      :: full(MSStcp_to_ses[chnum]) && timeout -> goto ForceAbort
      fi


 :: empty(PDAtcp_to_MSStcp[chnum]) &&
    timeout -> /* PDA not sending: assume PDA has aborted its session */
      if
      :: nfull(MSStcp_to_ses[chnum]) -> MSStcp_to_ses[chnum] !! aborted(f1,f2);
                                        goto Aborted

      :: full(MSStcp_to_ses[chnum]) && timeout -> goto ForceAbort
      fi
```

```
od;


ForceAbort: /* force MSSses to abort (to read 'aborted') from it chan */
do
:: MSStcp_to_ses[chnum] !! aborted(tmpId,chnum) -> break
:: timeout -> MSSses_to_tcp[chnum] ? msgtype(f1,f2) /* drop this msg */
od;
printf("MSStcp tmpId= %d going to +++Aborted +++ by KsTmpIdMan \n",tmpId,chnum);

Aborted:
if
:: atomic{nempty(MSStcp_to_PDAtcp[chnum]) && timeout ->
     do
     :: nempty(MSStcp_to_PDAtcp[chnum]) ->
              MSStcp_to_PDAtcp[chnum] ? msgtype(f1,f2) /* drop this msg */
     :: empty(MSStcp_to_PDAtcp[chnum]) -> break
     od}
:: empty(MSStcp_to_PDAtcp[chnum]) -> skip
fi;

printf("MSStcp: tmpId= %d chanum= %d +++ABORTED+++ BYE-BYE-BYE\n\n",tmpId,chnum)
}
```

```
/*
 * EscKey process:
 * -the fact that the PDA user can press the ESC keyboard at any time to
 *  interrupt his anonymous session is simulated by a 'timeout'
 *  which can go off at any time.
 * -interruption of the PDA user anonymous session can be originated at
 *  the PDAuser, PDAses, PDAtcp or at the MSS site, if this happens
 *  an 'abort' message is received which lead to abort the keyboard pro-
 *  cess.
 */
proctype EscKey(chan user_to_EscKey, EscKey_to_user; int tmpId, chnum)
{
 int  f1;       /* field for receiving the first field of a msg  */
 int  f2;       /* field for receiving the second field of a msg */
 if
 :: user_to_EscKey ? aborted(f1,f2) -> /* abort from PDA or MSS site */
     goto Aborted
 :: timeout -> EscKey_to_user !! abort(tmpId,chnum) ->
     goto Aborted            /* simulates the user pressing ESC   */
                             /* to interrupt his anonymous session */
 fi;

 Aborted:
  printf("EscKey: tmpId=%d aborted at PDA or MSS site BYE-BYE-BYE\n\n",tmpId);
}
```

```
/*
 * PDAuser process: is the interface between the PDA user and the
 *                  anonymous and confidential communication system.
 * It receives PDA user's commands typed on the keyboard and display
 * messages on the PDA screen.
 */
proctype PDAuser(int PDAnum)
{
 byte msgtype;   /* type of msg received                              */
 int  f1;        /* field for receiving the first field of a msg  */
 int  f2;        /* field for receiving the second field of a msg */
 int  tmpId;     /* temporary Id                                      */
 int  chnum;     /* channel number                                    */
bool KsChanged; /* flag to stop the PDA changing its Ks more than once     */
 int  payment;   /* payment for opening or extending an anonymous session   */
 int  MaxExtPay; /* max amount of money a user is allowed to spend in calls */
 int  ExtPay;    /* max amount of money a user hast spent in calls          */
 int  addr;      /* e-mail address                                          */
 int  txt;       /* text in e-mail msg                                      */

bool AbortFlag; /* was the process forced to abort YES/NO ? */

int i;    /* counter of attempts */
int NumM; /* number of e-mails to send */

chan ses_to_user_localch=[1] of {byte,int,int};
chan user_to_esckey=[1]      of {byte,int,int};
chan esckey_to_user=[1]      of {byte,int,int};

 KsChanged= NO;
 AbortFlag= NO;

 /*
  * run PDAses process: ses_to_user_localch is used by PDAuser to
  * receive tmpId and from PDAses
  */
 run PDAses(PDAnum, ses_to_user_localch);



KsNegotiation:
 do
 :: ses_to_user_localch ? tmpIdRcvd(tmpId,chnum) -> break
 :: ses_to_user_localch ? aborted(f1,f2) ->
      printf("PDAuser: PDAnum=%d pid=%d :My PDAses couldn't get a Ks;
              going +++Aborted+++\n",PDAnum,_pid);
     goto Aborted
 od;


 MaxExtPay= MAXEXTPAY;
 ExtPay= 0;

FirstPayment: do
 :: (i <= MAXNUMPAYATTEMPTS) ->
    if
    :: payment= FAKEMONEY
    :: payment= TOOLITTLEMONEY
    :: payment= GOLDENMONEY
```

```
fi;

PDAuser_to_ses[chnum] ! Ecash(payment,tmpId) ->

PDAses_to_user[chnum] ? msgtype(payment,f2) ->
    if
    :: (msgtype == EcashRejetd) ->
           i++  /* go back and fetch another coin from PDA memory */

    :: (msgtype == EcashAccepd) ->
           NumM=MAXNUMEMAILS_SENT;
          /*
           * run the EscKey process */
           */
           run EscKey(user_to_esckey, esckey_to_user, tmpId, chnum);
           goto AnoSes

    :: (msgtype == aborted) ->
          printf("PDAuser: tmpId=%d couldn't PAY; aborted by PDAses|PDAtcp|MSS
                  going +++Aborted+++\n",tmpId);
           goto Aborted
    fi

:: else -> PDAuser_to_ses[chnum] !! abort(tmpId,chnum) ->
     do
     :: PDAses_to_user[chnum] ? msgtype(f1,f2) ->
          if
          :: (msgtype != aborted) -> skip /* discard this msg */


          :: (msgtype == aborted) ->
                printf("PDAuser: tmpId=%d couldn't PAY; going +++Aborted+++\n",tmpId);
                goto Aborted
          fi
     od
od;

AnoSes:
 do
 :: (NumM >= 1 && nfull(PDAuser_to_ses[chnum])) ->
        PDAuser_to_ses[chnum] ! EaddrAndTxt(_pid,_pid); NumM--

 :: PDAses_to_user[chnum] ? YouHaveMail(addr,txt) ->
        printf("PDAuser tmpId= %d chnum= %d  GOT E-MAIL addr= XX txt= YY\n",tmpId,chnum)

:: (KsChanged==NO) -> PDAuser_to_ses[chnum] ! ChangeMyKs(tmpId,chnum); KsChanged= YES
                                              /* Ks can be changed only once */

:: PDAses_to_user[chnum] ? YourNewKs(f1,f2) ->
                             printf("PDAuser tmpId= %d has got a new Ks \n",tmpId)

:: PDAses_to_user[chnum] ? TimeAlert(f1,f2) ->
      printf("PDAuser: PDA-tmpId=%d has been TimeAlerted\n",tmpId);
      if
      :: (ExtPay <= MaxExtPay) ->
            if
            :: payment= FAKEMONEY
            :: payment= TOOLITTLEMONEY
            :: payment= GOLDENMONEY
```

```
            :: payment= NOMONEY
            fi;
            PDAuser_to_ses[chnum] ! Ecash(payment,tmpId);
            printf("PDAuser: PDA-tmpId=%d has sent EXTRA payment=%d\n",tmpId,payment)
       :: else -> skip /* no more extensions allowed */
       fi

 ::  PDAses_to_user[chnum] ? TimeFin(f1,f2) ->
          printf("PDAuser: PDA-tmpId=%d TimeFin going +++Abort+++ \n",tmpId);
          user_to_esckey!! aborted(tmpId,chnum);
          goto Aborted

 :: PDAses_to_user[chnum] ? EcashAccepd(f1,f2) ->
     ExtPay= ExtPay + payment;
     NumM= MAXNUMEMAILS_SENT;
     printf("PDAuser: tmpId=%d AnoTime INCRTED msg to send=%d\n",tmpId,NumM);

 :: PDAses_to_user[chnum] ? EcashRejetd(f1,f2) ->
     printf("PDAuser: AnoTime NOT incremented for PDA-tmpId= %d chnum= %d \n",tmpId,chnum)

 :: PDAses_to_user[chnum] ? aborted(f1,f2) ->
     user_to_esckey!! aborted(tmpId,chnum);
     printf("PDAuser: PDA-tmpId=%d aborted initiated by PDAses | PDAtct |
             MSS going +++Abort+++\n",tmpId);
     goto Aborted

 :: esckey_to_user ? abort(f1,f2) -> /* user pressed ESC key to interrupt session */
       PDAuser_to_ses[chnum] !! abort(chnum,tmpId)
 od;



Aborted:
 AbortFlag= YES;
 printf("PDAuser: +++ABORTED+++ BYE-BYE-BYE\n\n",tmpId,chnum);

End:
 do
 :: nempty(PDAuser_to_ses[chnum]) ->
            PDAuser_to_ses[chnum] ? msgtype(f1,f2) /* drop this msg */
 :: empty(PDAuser_to_ses[chnum])  -> break
 od;

 if
 :: (AbortFlag == NO) ->
       printf("PDAuser: PDA-tmpId=%d chnum= %d HAPPY END BYE-BYE-BYE\n\n",tmpId,chnum)
 :: else -> skip
 fi
}



/*
 * PDAses process: is in charge of learning the Kpu key and negotiating
 *                 a Ks key. It encrypts PDAuser msg before forwarding
 * them down the protocol stack; conversely, it decrypts msg coming from
 * the underneath layer and forward them to the PDAuser layer.
```

```
 */
proctype PDAses(byte MyPDAnum; chan ses_to_user_localch)
{
 chan q_to_tcp=[LOCALQSZ]  of {byte, int, int};
 chan q_to_user=[LOCALQSZ] of {byte, int, int};

 int  Ks;      /* session key of the PDA                              */
 int  tmpId;   /* temporary Id                                        */
 int  chnum;   /* channel number                                      */
 int  payment; /* payment for an anonymous session                    */
 byte msgtype; /* type of msg received                                */
 int  f1;      /* field for receiving the first field of a msg  */
 int  f2;      /* field for receiving the second field of a msg */
 int  addr;    /* e-mail address                                      */
 int  txt;     /* text in e-mail msg                                  */
 int  i;          /* i <= MAXNUM_Ks_ATTEMPTS: counter number of attempt */
                  /* to register a session key                          */
 bool AbortFlag;  /* was the process forced to abort YES/NO ?           */
 bool KsAccepd;   /* The suggested Ks was accepted YES/NO ?             */


KsNegotiation:
 Ks= 0;
 tmpId= 0;

 AbortFlag= NO;
 KsAccepd= NO;

 i= 1;
 do
 :: (i <= MAXNUM_Ks_ATTEMPTS) ->
    if       /* random selection of Ks */

    :: Ks = KsBlack ->
         PDA_to_KsPort ! Ks;
         if
         /*
          * There may be up to 5 msg in the channel buffer. If there is one
          * (not necessarily at the head of the buffer) with mtype= KsBlack
          * retrieve it. Otherwise block until such a msg appears in the
          * channel or timeout goes off
          */
         :: KsPort_to_PDA ?? KsBlack(tmpId,chnum) ->
                                 KsAccepd = YES;
                                 break
         :: timeout ->
              printf("PDAses: PDAnum= %d; sent Ks= %d  i= %d NO answer ->
                      TIMEOUT \n",MyPDAnum,Ks,i);
              i++
         fi

    :: Ks = KsBlue ->
         PDA_to_KsPort ! Ks;
         if
         :: KsPort_to_PDA ?? KsBlue(tmpId,chnum) ->
                                 KsAccepd = YES;
                                 break
         :: timeout ->
              printf("PDAses: PDAnum= %d;  sent Ks= %d  i= %d NO answer ->
```

```
                    TIMEOUT \n",MyPDAnum,Ks,i);
               i++
          fi

     :: Ks = KsGreen ->
          PDA_to_KsPort ! Ks;
          if
          :: KsPort_to_PDA ?? KsGreen(tmpId,chnum) ->
                                  KsAccepd = YES;
                                  break
          :: timeout ->
               printf("PDAses: PDAnum= %d;  sent Ks= %d  i= %d NO answer ->
                       TIMEOUT \n",MyPDAnum,Ks,i);
               i++
          fi

     :: Ks = KsPink ->
          PDA_to_KsPort ! Ks;
          if
          :: KsPort_to_PDA ?? KsPink(tmpId,chnum) ->
                                  KsAccepd = YES;
                                  break
          :: timeout ->
               printf("PDAses: PDAnum= %d;  sent Ks= %d  i= %d NO answer ->
                       TIMEOUT \n",MyPDAnum,Ks,i);
               i++
          fi

     :: Ks = KsWhite ->
          printf("PDAses: PDAnum= %d is going to send Ks= %d \n",MyPDAnum,Ks);
          PDA_to_KsPort ! Ks;
          if
          :: KsPort_to_PDA ?? KsWhite(tmpId,chnum) ->
                                  KsAccepd = YES;   /* this should never happen */
                                  break             /* this should never happen */
          :: timeout ->
               printf("PDAses: PDAnum= %d; sent Ks= %d i= %d NO answer ->
                       TIMEOUT \n",MyPDAnum,Ks,i);
               i++
          fi

     fi

:: (i > MAXNUM_Ks_ATTEMPTS) ->
     break /* give up registering a Ks */
od;


/*
 *  communication between the PDAuser and the PDAses
 */
if
:: (KsAccepd == NO) ->
     printf("PDAses: PDAnum=%d failed to register Ks after i=%d attempts:
             going +++Aborted+++\n",MyPDAnum,i-1);
     ses_to_user_localch !! aborted(NEGINT,NEGINT);
     goto Aborted
:: else ->
     ses_to_user_localch ! tmpIdRcvd(tmpId,chnum);
```

```
     printf("PDAses: PDAnum=%d; pid=%d RgTED Ks=%d (tmpId= %d) after the %dth
             attempts \n",MyPDAnum,_pid,Ks,tmpId,
i);
     /*
      * initiating the underneath layer
      */
     run PDAtcp(tmpId,chnum,Ks)
 fi;
```

```
FirstPayment:
 do
 :: PDAuser_to_ses[chnum] ? msgtype(f1,f2) ->
     if
     :: (msgtype == Ecash) ->
                    /* if necessary do anything to the msg */
                   PDAses_to_tcp[chnum] ! msgtype(f1,f2)

     :: (msgtype == abort) -> PDAses_to_tcp[chnum] !! abort(f1,f2) ->
           do
           :: PDAtcp_to_ses[chnum] ? msgtype(f1,f2);
                if
                :: (msgtype != aborted) -> skip /* discard the msg */
                :: (msgtype == aborted) -> PDAses_to_user[chnum] !! aborted(f1,f2);
                                                goto Aborted
                fi
           od
     fi;

:: PDAtcp_to_ses[chnum] ? msgtype(f1,f2) ->
     if
     :: (msgtype == EcashRejetd) ->
                    /* if necessary do anything to the msg */
                    PDAses_to_user[chnum] ! msgtype(f1,f2)

     :: (msgtype == EcashAccepd) ->
                    skip; /* if necessary do anything to the msg */
                    payment= f1;
                    PDAses_to_user[chnum] ! EcashAccepd(payment,tmpId);
                    goto AnoSes

     :: (msgtype == aborted) ->
                    PDAses_to_user[chnum] !! aborted(f1,f2);
                    goto Aborted
     fi
 od;
```

```
AnoSes:
 do
    /* from user to ses to tcp */
 :: nfull(q_to_tcp) && nempty(PDAuser_to_ses[chnum]) ->
     PDAuser_to_ses[chnum] ? msgtype(f1,f2) -> q_to_tcp ! msgtype(f1,f2)

    /* from tcp to ses to user */
 :: nfull(q_to_user) && nempty(PDAtcp_to_ses[chnum]) ->
     PDAtcp_to_ses[chnum] ? msgtype(f1,f2) -> q_to_user ! msgtype(f1,f2)
```

```
   /* send to user */
:: nempty(q_to_user) && nfull(PDAses_to_user[chnum]) ->
    q_to_user ? msgtype(f1,f2);
    if
    :: (msgtype == YourNewKs) ->
                  printf("PDAses: tmpId=%d GOT new Ks OldKs=%d
                          NewKs= %d\n",tmpId,Ks,f1);
                  Ks= f1;
                  /* if necessary process the msg */
                  PDAses_to_user[chnum] ! YourNewKs(f1,f2)

    :: (msgtype == TimeAlert) ->
                  /* if necessary process the msg */
                  PDAses_to_user[chnum] ! TimeAlert(f1,f2)

    :: (msgtype == EcashAccepd) ->
                  /* if necessary process the msg */
                  PDAses_to_user[chnum] ! EcashAccepd(f1,f2)

    :: (msgtype == EcashRejetd) ->
                  /* if necessary process the msg */
                  PDAses_to_user[chnum] ! EcashRejetd(f1,f2)

    :: (msgtype == YouHaveMail) ->
                  /* if necessary process the msg */
                  PDAses_to_user[chnum] ! YouHaveMail(f1,f2)

    :: (msgtype == aborted) ->
                  /* if necessary process the msg */
                  PDAses_to_user[chnum] !! aborted(f1,f2);
                  printf("PDAses: tmpId=%d abort initiated by PDAtcp
                          or MSS going +++Aborted+++ \n",tmpId);
                  goto Aborted

    :: (msgtype == TimeFin) ->
                  /* if necessary process the msg */
                  PDAses_to_user[chnum] ! TimeFin(f1,f2);
                  printf("PDAses: tmpId=%d TimeFin MSS going +++Aborted+++ \n",tmpId);
                  goto Aborted
    fi;


:: nempty(q_to_tcp) && nfull(PDAses_to_tcp[chnum]) -> /* send to tcp */
    q_to_tcp ? msgtype(f1,f2);
    if
    :: (msgtype == ChangeMyKs) ->
                  printf("PDAses: PDA-tmpId=%d ASKING to change OldKs=%d \n",tmpId,Ks);
                  /* if necessary process the msg */
                  PDAses_to_tcp[chnum] ! ChangeMyKs(f1,f2)

    :: (msgtype == EaddrAndTxt) ->
                  /* if necessary process the msg */
                  PDAses_to_tcp[chnum] ! EaddrAndTxt(f1,f2)

    :: (msgtype == Ecash) ->
                  /* if necessary process the msg */
                  PDAses_to_tcp[chnum] ! Ecash(f1,f2)
```

```
    :: (msgtype == abort) ->
                  /* if necessary process the msg */
                  PDAses_to_tcp[chnum] !! abort(f1,f2)
    fi

 od;


Aborted:
 AbortFlag= YES;
 printf("PDAses: +++ABORTED+++ BYE-BYE-BYE\n\n");

End:
 do
 :: atomic{nempty(PDAses_to_tcp[chnum]) ->
           PDAses_to_tcp[chnum] ? msgtype(f1,f2)} /* drop this msg */
 :: empty(PDAses_to_tcp[chnum])  ->  break
 od;

 if
 :: (AbortFlag== NO) -> printf("PDAses: HAPPY END BYE-BYE-BYE\n\n")
 :: else-> skip
 fi
}
```

```
/*
 * PDAtcp process: connection-oriented reliable link between the PDA
 *                 and the MSS. It forwards msg up the protocol stack
 * and to the MSS side. The PDAtcp initiates an abort procedure that
 * propagates up the stack protocol whenever it detects that its remote
 * peer (the MSStcp) is unreachable.
 */
proctype PDAtcp(int tmpId, chnum, Ks)
{
 chan q_to_PDAses= [LOCALQSZ] of {byte, int, int};
 chan q_to_MSStcp= [LOCALQSZ] of {byte, int, int};

 byte msgtype;  /* type of msg received                          */
 int  f1;       /* field for receiving the first field of a msg  */
 int  f2;       /* field for receiving the second field of a msg */

assert(0 <= chnum && chnum <= NUMofPDA && tmpId == chnum);

do
   /* from PDAtcp to PDAses */
:: nfull(q_to_PDAses) && nempty(MSStcp_to_PDAtcp[chnum]) ->
     MSStcp_to_PDAtcp[chnum] ? msgtype(f1,f2) -> q_to_PDAses ! msgtype(f1,f2)

   /* PDAses to PDAtcp */
:: nfull(q_to_MSStcp) && nempty(PDAses_to_tcp[chnum]) ->
     PDAses_to_tcp[chnum] ? msgtype(f1,f2) -> q_to_MSStcp ! msgtype(f1,f2)

   /* send to PDAses */
:: nempty(q_to_PDAses) && nfull(PDAtcp_to_ses[chnum]) ->
     q_to_PDAses ? msgtype(f1,f2);
```

```
if
:: (msgtype == abort) -> /* abort initiated by MSS */
              PDAtcp_to_ses[chnum] !! aborted(f1,f2);
              printf("PDAtcp: tmpId=%d initiated by MSS going
                      +++Aborted+++\n", tmpId,chnum);
              goto Aborted

:: (msgtype == TimeFin) -> /* TimeFin initiated by MSS */
              PDAtcp_to_ses[chnum] !! TimeFin(f1,f2);
              printf("PDAtcp: tmpId=%d TimeFin initiated by MSS going
                      +++Aborted+++\n",tmpId);
              goto Aborted

:: (msgtype == aborted) -> /* abort initiated by PDAuser or PDAses or PDAtcp */
              PDAtcp_to_ses[chnum] !! aborted(f1,f2);
              printf("PDAtcp: tmpId=%d initiated at PDA side going
                      +++Aborted+++\n",tmpId);
              goto Aborted

:: (msgtype == YourNewKs) -> /* MSSman has changed Ks */
               Ks= f1;
               PDAtcp_to_ses[chnum] ! YourNewKs(f1,f2)

:: else ->  /* MSS sending a routine msg */
              PDAtcp_to_ses[chnum] ! msgtype(f1,f2)
fi

   /* send to MSStcp */
:: nempty(q_to_MSStcp) && nfull(PDAtcp_to_MSStcp[chnum])
                       && nfull(q_to_PDAses) ->
     q_to_MSStcp ? msgtype(f1,f2);
     if /* PDA sent ordinary msg */
     :: (msgtype != abort) -> PDAtcp_to_MSStcp[chnum] ! msgtype(f1,f2)

     :: (msgtype == abort) -> /* abort initiated by PDA */
          PDAtcp_to_MSStcp[chnum] !! abort(f1,f2);
          q_to_PDAses !! aborted(f1,f2);
          printf("PDAtcp: tmpId=%d aborted by PDA going to
                  +++Aborted+++ soon \n",tmpId)
     fi

:: full(PDAtcp_to_MSStcp[chnum]) &&
   timeout -> /* MSS not reading: assume MSS aborted session */
           if
           :: nfull(q_to_PDAses) -> q_to_PDAses !! aborted(f1,f2)
           :: full(q_to_PDAses) && timeout -> goto ForceAbort
           fi

od;

ForceAbort:
 do
 :: PDAtcp_to_ses[chnum] !! aborted(tmpId,chnum) -> break
 :: timeout -> PDAses_to_tcp[chnum] ? msgtype(f1,f2) /* drop this msg */
 od;

Aborted:
if
:: atomic{nempty(PDAtcp_to_MSStcp[chnum]) && timeout ->
```

```
      do
      :: nempty(PDAtcp_to_MSStcp[chnum]) ->
                PDAtcp_to_MSStcp[chnum] ? msgtype(f1,f2) /* drop this msg */
      :: empty(PDAtcp_to_MSStcp[chnum]) -> break
      od}
 :: empty(PDAtcp_to_MSStcp[chnum]) -> skip
 fi;

  printf("PDAtcp: tmpId= %d chanum= %d +++Aborted+++ BYE-BYE-BYE\n\n",tmpId,chnum)
}
```

```
/*
 * instantiates the participants processes
 */
init
{
   int PDAnum;  /* number of PDA */

   atomic{
   run KsTmpIdMan(); /* instantiate the tmpId and Ks manager process */
   };

   PDAnum= 0;

   do /* instantiate NUMofPDA processes, one for each PDA */
   :: PDAnum <  NUMofPDA ->
                          run PDAuser(PDAnum);
                          PDAnum++
   :: PDAnum >= NUMofPDA  -> break
   od
}
```

A.5 Promela specification of the whole system

```
/*
 * PROGRAMME: Promela validation model for the anonymous and confidential communicator
 * AUTHOR: Carlos Molina Jimenez;
 * ADDRESS: The University of Newcastle upon Tyne
 * DATE OF CREATION: 27 Jul 1999
 * DATE OF LAST UPDATE: 11 Nov 1999
 */


/*
 * This code assumes that the Ks negotiation has been verified, so lines related to
 * KpuMan process have been commented
 */
/* #define CERTIFIED_Kpu       12 *//* Kpu certified by a certification authority      */
/* #define UNCERTIFIED_Kpu     10 *//* Kpu not certified by a certification authority  */
/* #define INVALID_Kpu         00 *//* Kpu out of domain                               */
/* #define MAXNUM_Kpu_ATTEMPTS 10 *//* max number of attempts to receive the right Kpu */
```

```
#define MAXNUMPAYATTEMPTS    3 /* max num of attempts to pay for an anonymous session */
#define MAXMONEY           100 /* max money accepted for opening an anonymous call    */
#define GOLDENMONEY         20 /* golden, silver,                  are genuine and    */
#define SILVERMONEY         15 /* enough money for opening an anonymous call          */
#define MINMONEY            15 /* min money accepted for opening an anonymous call    */
#define TOOLITTLEMONEY       1 /* genuine money but not enough                        */
#define NOMONEY              0 /* no money                                            */
#define FAKEMONEY           -1 /* fake money                                          */
#define MAXEXTPAY           40 /* max amount of money for extra payment: the user can */
                               /* extend his call twice: 20+20 or 15+15 or 20+15      */

#define LAST_MSGS           10 /* Num of msg left after TimeExp alarm goes off        */

#define YES                  1
#define NO                   0
#define ANYINT               1  /* any integer                    */
#define NEGINT              -1  /* any negative integer           */
#define MAXNUMPDA            3  /* max number of PDA in the MSS   */
#define NUMofPDA             3  /* num of PDA visiting the MSS    */
#define QSZ                  1  /* num of msgs stored by a channel */
#define LOCALQSZ             3  /* num of msgs store by a channel  */
#define MAX_Ks               5  /* Valid Ks domain [1...5]         */
#define MIN_Ks               1  /*                                 */
#define MAXNUM_Ks_ATTEMPTS   3  /* max num of attempt to register Ks       */
#define MAXNUMMSGREAD        4  /* max num of attempt to receive an        */
                                /* answer from the MSS encrypted with      */
                                /* the suggested Ks and containing a tmpId */

#define MAXNUMEMAILS_SENT  16   /* temporary restriction for validation purposes */

#define KsBlack 1
#define KsBlue  2
#define KsGreen 3
#define KsPink  4
#define KsRed   5
#define KsWhite 6   /* <-- invalid key */

mtype= {aborted,abort,TimeFin,TimeAlert,YourNewKs,ChangeMyKs,
        tmpIdRcvd,Ecash,EcashRejetd, EcashAccepd,GenuineEcash,
        FakeEcash,EaddrAndTxt,YouHaveMail};
        /* This is equiv. to YouHaveMail=1, EaddrAndTxt=2, FakeECash=3,... */

/*
 * This code assumes that the Ks negotiation has been verified, so lines related to
 * KpuMan process have been commented
 */
/* mtype={Kpu,BogusKpu,KpuRcvd} *//* msg type used by KpuMan */
/* chan KpuPort_to_PDA=[1] of {byte,int}; *//* Kpu broadcast channel from MSS to PDAs */

/*
 * The MSS receives Ks proposals from PDAs at its PDA_to_KsPort channel
 * the PDA read the MSS reply from KsPort_to_PDA channel which can
 */
chan PDA_to_KsPort=[1] of {int};
chan KsPort_to_PDA=[NUMofPDA]  of {int,int,int}

chan KsTmpIdMan_to_MSSses[MAXNUMPDA]=[QSZ] of {byte,int,int};
chan MSSses_to_KsTmpIdMan[MAXNUMPDA]=[QSZ] of {byte,int,int};
```

```
chan PDAses_to_tcp[MAXNUMPDA]=[QSZ] of {byte,int,int};
chan PDAtcp_to_ses[MAXNUMPDA]=[QSZ] of {byte,int,int};

chan MSSses_to_tcp[MAXNUMPDA]=[QSZ] of {byte,int,int};
chan MSStcp_to_ses[MAXNUMPDA]=[QSZ] of {byte,int,int};

chan MSStcp_to_PDAtcp[MAXNUMPDA]=[QSZ] of {byte,int,int};
chan PDAtcp_to_MSStcp[MAXNUMPDA]=[QSZ] of {byte,int,int};

chan PDAuser_to_ses[MAXNUMPDA]=[QSZ] of {byte,int,int};
chan PDAses_to_user[MAXNUMPDA]=[QSZ] of {byte,int,int};

chan MSSses_to_bank[MAXNUMPDA]=[QSZ] of {byte,int,int};

chan bank_to_MSSses[MAXNUMPDA]=[QSZ] of {byte,int,int};

chan MSSses_to_MailSvr[MAXNUMPDA]=[QSZ] of {byte,int,int};
chan MailSvr_to_MSSses[MAXNUMPDA]=[QSZ] of {byte,int,int};


/*
 * This code assumes that the Ks negotiation has been verified, so lines related to
 * KpuMan process have been commented
 */
/*
 * KpuMan process: it places a msg in KpuPort_to_PDA channel. The msg is read by
*                  PDAs. It contains certified Kpu key, uncertified Kpu key or bogus
* message. When the msg is read, the KpuMan places another one, waits until
* it is read and so on.
*/
/* proctype KpuMan()
 * {
 * int Kpu_val;
 *
 * do
 * :: (true) ->
 *  if
 *  :: Kpu_val= CERTIFIED_Kpu
 *  :: Kpu_val= UNCERTIFIED_Kpu
 *  fi;
 *
 * end_cyc:
 * if
 *  :: KpuPort_to_PDA ! Kpu(Kpu_val)
 *  :: KpuPort_to_PDA ! BogusKpu(INVALID_Kpu)
 *  fi
 * od
 * }
 */



/*
 *  KsTmpIdMan process: it is in charge of managing session keys and temporary
 *                      Id assigned to PDA. It guarantees that tmpIds assigned
 * to PDA are unique. Also, it guarantees that session keys are unique and
```

```
 * secret. It asks the PDA to change its Ks when it detects that is has been
 * hit by another PDA. Also, upon request, it provides the PDA with a new Ks.
 */
proctype KsTmpIdMan()
{
 int tmpIdVec[MAXNUMPDA];  /* vector for storing TmpIds */
 int KsVec[MAXNUMPDA];     /* vector for storing Ks     */
 int Ks;                   /* session key               */
 int tmpId;                /* temporary Id              */
 int chnum;                /* channel number            */

 int OldestKs;     /* oldest Ks accepted by MSS and sent to KsPort_to_PDA */

 byte msgtype;  /* type of msg received                                    */
 int  f1;       /* field for receiving the first field of a msg            */
 int  f2;       /* field for receiving the second field of a msg           */
 int fh1;       /* scratch field for receiving the first field of a msg    */
 int fh2;       /* scratch field for receiving the second field of a msg   */
 int NewKey;  /* new Ks key */
 int count;   /* counter    */
 int n;       /* counter    */
 int i;       /* counter    */
 int j;       /* counter    */

 i=0;  /* initialize to 0 and -1 the valid cells in vectors KsVec */
 do    /* and tmpIdVec, respectively                               */
 :: i <  NUMofPDA ->
    assert(0 <= i && i < MAXNUMPDA);
    KsVec[i]= 0;
    tmpIdVec[i]= -1;
    i++
 :: else -> break
 od;

 i= NUMofPDA; /* initialize to -3 the unused cells in the vectors */
 do           /* KsVec and tmpIdVec respectively                  */
 :: i <  MAXNUMPDA ->
    assert(NUMofPDA <= i && i < MAXNUMPDA);
    KsVec[i]= -3;
    tmpIdVec[i]= -3;
   i++
 :: else -> break
 od;

 TestTmpIdKs:
 /*
  * test that tmpId assigned to PDA are unique */
  */
 atomic{
 i=0;
 do
 :: (i < MAXNUMPDA) ->
      if
      :: (tmpIdVec[i] >= 0) -> assert(tmpIdVec[i] < NUMofPDA);
                 j=0;
                 do
                 :: (j < MAXNUMPDA) ->
```

```
                    if
                    :: (i == j) -> j++
                    :: else ->
                            assert(tmpIdVec[i] != tmpIdVec[j]);
                            j++
                    fi
                :: else -> break /* move to next i */
                od

    :: else -> skip /*  tmpIdVec[i] < 0 are unused elements */
    fi;
    i++
:: else -> break /* comparison finished */
od};

/*
 * test that Ks held by PDA are unique */
 */
atomic{
i=0;
do
:: (i < MAXNUMPDA) ->
    if
    :: (KsVec[i] > 0) -> assert(KsVec[i] <= MAX_Ks);
                j=0;
                do
                :: (j < MAXNUMPDA) ->
                        if
                        :: (i == j) -> j++
                        :: else ->
                                assert(KsVec[i] != KsVec[j]);
                                j++
                        fi
                :: else -> break /* move to next i */
                od

    :: else -> skip /*  KsVec[i] < 0 are unused elements */
    fi;
    i++
:: else -> break /* comparison finished */
od};


progress_tmpIdKsOK: skip; /* tmpId and Ks are all right, continue */
Ks= 0;

end: do  /* valid endstate of this server process */
:: PDA_to_KsPort ? Ks -> goto KsVerification

:: MSSses_to_KsTmpIdMan[0] ? msgtype(f1,f2)->
                            chnum=0; tmpId= 0; goto MSSsesCare

:: MSSses_to_KsTmpIdMan[1] ? msgtype(f1,f2)->
                            chnum=1; tmpId= 1; goto MSSsesCare

:: MSSses_to_KsTmpIdMan[2] ? msgtype(f1,f2)->
                            chnum=2; tmpId= 2; goto MSSsesCare
```

```
od;


KsVerification:
 if
 :: (Ks <= MAX_Ks && Ks >= MIN_Ks) -> /* Is Ks within the valid domain? */

     i=0;
     do /* is Ks already in use by another PDA? */
     :: (i < NUMofPDA) -> /* valid Ks are [1,2,3,4,5] */
        if
        :: (Ks == KsVec[i]) ->
            printf("KsMan: Ks=%d KsVec[%d]=%d is in use by tmpId[%d]=%d \n",
                    Ks,i,KsVec[i],i,tmpIdVec[i]);
            goto KsInuse

        :: (Ks != KsVec[i]) -> i++
        fi;
     :: else -> goto KsNotInuse
     od;
:: else ->  /* suggested Ks is incorrect */
    printf("KsMan: Ks= %d  is incorrect I didn't reply\n",Ks);
    goto TestTmpIdKs
fi;

KsNotInuse:
  /* Find a free tmpId in the tmpIdVec and assign it to the PDA              */
  /* The values of valid tmpId are [0,1,2,3,4]                              */
  /* tmpIdVec[0]=  0|1|2|3|4 means  tmpId=0 already being used by some PDA  */
  /* tmpIdvec[0]= -1  means  tmpId=0 is free                                */
  /* tmpIdvec[0]= -2  means  tmpId=0 has been used and can't be recycled yet */
  /* tmpIdvec[0]= -3  means  tmpId=0 is not in use at all                   */
  /* tmpIdVec[1]=  0|1|2|3|4 means  tmpId=1 already being used by some PDA  */
  /* tmpIdvec[1]= -1  means  tmpId=1 is free                                */
  /* tmpIdvec[1]= -2  means  tmpId=1 has been used and can't be recycled yet */
  /* tmpIdvec[1]= -3  means  tmpId=1 is not in use at all                   */
  /* tmpIdVec[2]=  0|1|2|3|4 means  tmpId=2 already being used by some PDA  */
  /* tmpIdvec[2]= -1  means  tmpId=2 is free                                */
  /* tmpIdvec[2]= -2  means  tmpId=2 has been used and can't be recycled yet */
  /* tmpIdvec[2]= -3  means  tmpId=2 is not in use at all                   */

  i=0;
  do
  :: (i < NUMofPDA) -> /* tmpId= -3 not in use at all                       */
     if                /* tmpId= -2 not in use: to be recycled later        */
     :: (tmpIdVec[i] != -1) -> i++ /* this tmpId already in use, try next one */
     :: else -> break              /* this tmpId free, use it               */
     fi
  :: else -> /* no more tmpId to be assigned: take this as KsINUSE= YES */
             /* and don't reply no anybody                               */
             goto TestTmpIdKs
  od;
  tmpId= i;  chnum=i; /* tmpId & chnum assigned to new PDA with session key= Ks */
  if
  :: atomic{nfull(KsPort_to_PDA) -> /* there's space in cha for key(tmpId,chnum) */
      KsPort_to_PDA ! Ks(tmpId,chnum);
      tmpIdVec[i]= i;
      KsVec[i]= Ks;
```

```
      run MSSses(tmpId,chnum,Ks)} /* initiate MSS session to take care of new PDA */

   :: atomic{full(KsPort_to_PDA) -> /* no space in cha, discharge oldest msg */
        KsPort_to_PDA ? OldestKs(f1,f2);
        tmpIdVec[f1]= -2;              /* empty PDAj from your tables          */
        KsVec[f1]= -2;                 /* and terminate MSSses assigned to PDAj */
        KsTmpIdMan_to_MSSses[f1] !! abort(f1,f2);
        MSSses_to_KsTmpIdMan[f1] ? aborted(fh1,fh2); /* fh1,fh2 scratch var */
        chnum= f1;
        do
        :: nempty(KsTmpIdMan_to_MSSses[chnum]) ->
                  KsTmpIdMan_to_MSSses[chnum] ? msgtype(f1,f2) /* drop this msg */
        :: empty(KsTmpIdMan_to_MSSses[chnum]) -> break
        od;
        printf("KsMan:removed from ch: Ks=%d tmpId=%d chnum=%d \n",OldestKs,f1,f2)}
      fi;
      goto TestTmpIdKs;

   /*
    * The Ks suggested by the new PDA is already in use by PDAi with tmpId= i and
    * chnum=i. KsMan has to get a new Ks for PDAi.
    */
   KsInuse:
     printf("KsMan: suggested Ks=%d is already in use by PDA with
             tmpId=%d chnum=%d\n",Ks,i,i);
     count= 1;
     do
     :: (count <= MAXNUM_Ks_ATTEMPTS) ->
        if
        :: NewKey= KsBlack
        :: NewKey= KsBlue
        :: NewKey= KsGreen
        :: NewKey= KsPink
        :: NewKey= KsRed
        /* :: NewKey= KsWhite is out, we assume KsMan generates correct Ks only */
        fi;
        n=0;
        do
        :: (n < NUMofPDA) ->
           assert( 0 <= n && n < NUMofPDA);
           if
           :: (KsVec[n] == NewKey) -> count++; break
           :: (KsVec[n] != NewKey) -> n++
           fi;
        :: else -> goto NewKsFound1
        od;
     :: else -> /* couldn't find any available Ks */
           atomic{
           KsTmpIdMan_to_MSSses[i] !! abort(i,Ks) -> /* order MSSses to abort */
           MSSses_to_KsTmpIdMan[i] ? aborted(f1,f2);
           tmpIdVec[i]= -2;
           KsVec[i]= -2;
           chnum= tmpId;
           do
           :: nempty(KsTmpIdMan_to_MSSses[chnum]) ->
                KsTmpIdMan_to_MSSses[chnum] ? msgtype(f1,f2) /* drop this msg */
           :: empty(KsTmpIdMan_to_MSSses[chnum]) -> break
           od;
           printf("KsMan: has SENT ABORT to tmpId= %d \n",i);
```

```
        goto TestTmpIdKs}
od;

NewKsFound1:
assert((MIN_Ks <= NewKey)&&(NewKey <= MAX_Ks)&&(0 <= i)&&(i < NUMofPDA));
if
:: atomic{KsTmpIdMan_to_MSSses[i] ! YourNewKs(NewKey,Ks) ->
          KsVec[i]= NewKey}

:: timeout -> printf("KsMan: tmpId=%d chnum=%d Ks=%d NOT THERE, ignore it\n",
                  i,i,Ks)
   fi;
   goto TestTmpIdKs;


MSSsesCare:
 if
 :: (msgtype == ChangeMyKs) ->
   printf("KsMan: tmpId=%d chnum=%d has rcvd ChangeMyKs\n",tmpId,chnum);
   count= 1;
   do
   :: (count <= MAXNUM_Ks_ATTEMPTS) ->
      if
      :: NewKey= KsBlack
      :: NewKey= KsBlue
      :: NewKey= KsGreen
      :: NewKey= KsPink
      :: NewKey= KsRed
      /* :: NewKey= KsWhite is out, we assume KsMan generates correct Ks only */
      fi;
      n=0;
      do
      :: (n < NUMofPDA) ->
         assert( 0 <= n && n < NUMofPDA);
         if
         :: (KsVec[n] == NewKey) -> count++; break
         :: (KsVec[n] != NewKey) -> n++
         fi;
      :: else -> goto NewKsFound2
      od;
 :: else -> /* couldn't find any available Ks */
      atomic{
      KsTmpIdMan_to_MSSses[chnum] !! abort(tmpId,chnum); /* order MSSses to abort */
      do
      :: MSSses_to_KsTmpIdMan[chnum] ? msgtype(f1,f2)->
         if
         :: (msgtype != aborted) -> skip
         :: (msgtype == aborted) ->
               tmpIdVec[tmpId]= -2;
               KsVec[tmpId]= -2;
               chnum= tmpId;
               do
               :: nempty(KsTmpIdMan_to_MSSses[chnum]) ->
                     KsTmpIdMan_to_MSSses[chnum] ? msgtype(f1,f2) /* drop this msg */
               :: empty(KsTmpIdMan_to_MSSses[chnum]) -> break
               od;
               printf("KsMan: has SENT ABORT to tmpId= %d \n",tmpId);
               goto TestTmpIdKs
```

```
            fi
        od}
    od;

    NewKsFound2:
     assert(MIN_Ks <= NewKey && NewKey <= MAX_Ks);
     assert(0 <= tmpId && tmpId < NUMofPDA && tmpId == chnum);

    if
    :: atomic{KsTmpIdMan_to_MSSses[tmpId] ! YourNewKs(NewKey,f1) ->
              KsVec[tmpId]= NewKey}

      :: timeout -> printf("KsMan: tmpId=%d with chnum=%d Ks=%d NOT THERE,
                            ignore it\n",tmpId,chnum,f1)
      fi;



  :: (msgtype == aborted) -> /* the PDA side or MSSses wants abort the ano. ses. */
     atomic{
     printf("KsMan: going to send ABORT to tmpId= %d \n",tmpId);
     tmpIdVec[tmpId]= -2;
     KsVec[tmpId]= -2;
     chnum= tmpId;
     do
     :: nempty(KsTmpIdMan_to_MSSses[chnum]) ->
               KsTmpIdMan_to_MSSses[chnum] ? msgtype(f1,f2) /* drop this msg */
        :: empty(KsTmpIdMan_to_MSSses[chnum]) -> break
        od;
        printf("KsMan: ABORT FOR tmpId= %d \n",tmpId)}

     :: else -> skip /* UNKNOWN msg: transient failure? */
     fi;
     goto TestTmpIdKs


}




/*
 * MSSses process: is in charge of managing the anonymous communication session of
 *                 the PDA. It charges the PDA for the communication, receives the
 * payment, forwards it to the bank for verification, accepts or rejects the payment,
 * warns the PDA user about the prepaid time expiration and abruptly finishes the
 * communication session when the prepaid time expires. Also, it is the link between
 * the PDA and the KsTmpIdMan; and the link between the PDA and the mail server.
 */
proctype MSSses(int tmpId, chnum, Ks)
{
 chan q_to_MSStcp=[LOCALQSZ]     of {byte,int,int};
 chan q_to_KsTmpIdMan=[LOCALQSZ] of {byte,int,int};
 chan q_to_MailSvr=[LOCALQSZ]    of {byte,int,int};
 chan q_to_bank=[LOCALQSZ]       of {byte,int,int};

 byte msgtype;  /* type of msg received                              */
 int  f1;       /* field for receiving the first field of a msg  */
```

```
int  f2;        /* field for receiving the second field of a msg */
int  MyKs;      /* a session key */
int  addr;      /* e-mail address */
int  txt;       /* text in an e-amil msg */
int  payment;     /* payment for anonymous session: one unit of money is converted to  */
                  /* one msg to be sent                                                */
int  credit;      /* prepaid payment: num of msg the PDA user has prepaid for       */
int  CreditLeft; /* prepaid payment before warning msg: num of msg the user can    */
                  /* send after the expiration time warning and termination of the  */
                  /* session                                                        */

 run MSStcp(tmpId,chnum,Ks);


progress_FirstPayment: do
 :: MSStcp_to_ses[chnum] ? msgtype(f1,f2) ->
      if
      :: (msgtype == Ecash) ->
          payment= f1; tmpId= f2;
          MSSses_to_bank[chnum] ! Ecash(payment,tmpId);
          if
          :: bank_to_MSSses[chnum] ? GenuineEcash(payment,f2) ->
             if
             :: (payment < MINMONEY || payment > MAXMONEY) ->
                   MSSses_to_tcp[chnum] ! EcashRejetd(payment,tmpId)

             :: else -> MSSses_to_tcp[chnum] ! EcashAccepd(payment,tmpId) ->
                   assert(payment > LAST_MSGS);
                   credit= payment - LAST_MSGS;
                   CreditLeft= LAST_MSGS;
                   goto progress_AnoSes
             fi
          :: bank_to_MSSses[chnum] ? FakeEcash(payment,f2) ->
                   MSSses_to_tcp[chnum] ! EcashRejetd(payment,tmpId)
          fi
      :: (msgtype == aborted) -> goto AbortBank
      fi

::  KsTmpIdMan_to_MSSses[chnum] ? abort(f1,f2) ->
      MSSses_to_tcp[chnum] !!  abort(tmpId,chnum) ->
      do
      :: MSStcp_to_ses[chnum] ? msgtype(f1,f2) ->
           if
           :: (msgtype != aborted) -> skip /* discard the msg */
           :: (msgtype == aborted) ->
                printf("MSSses tmpId= %d aborted by KsTmpIdMan; going
                       +++Aborted+++\n",tmpId);
                goto AbortBank
           fi
      od
 od;


 assert( 0<= chnum && chnum <= NUMofPDA);

progress_AnoSes: do
    /* to MSStcp */
 :: KsTmpIdMan_to_MSSses[chnum]  ? [abort(f1,f2)] && nfull(q_to_MSStcp) ->
```

```
    KsTmpIdMan_to_MSSses[chnum] ? abort(f1,f2) -> q_to_MSStcp !! abort(f1,f2)

  /* to KsTmpIdMan */
:: MSStcp_to_ses[chnum]  ? [aborted(f1,f2)] && nfull(q_to_KsTmpIdMan) ->
    MSStcp_to_ses[chnum] ? aborted(f1,f2) -> q_to_KsTmpIdMan !! aborted(f1,f2)

  /* to MSStcp */
:: KsTmpIdMan_to_MSSses[chnum]  ? [YourNewKs(f1,f2)] && nfull(q_to_MSStcp) ->
    KsTmpIdMan_to_MSSses[chnum] ? YourNewKs(f1,f2) -> q_to_MSStcp ! YourNewKs(f1,f2)

  /* to MSStcp */
:: MailSvr_to_MSSses[chnum]  ? [YouHaveMail(f1,f2)] && nfull(q_to_MSStcp) ->
    MailSvr_to_MSSses[chnum] ? YouHaveMail(f1,f2) -> q_to_MSStcp ! YouHaveMail(f1,f2)

  /* to MSStcp */
:: bank_to_MSSses[chnum]  ? [GenuineEcash(f1,f2)] &&  nfull(q_to_MSStcp) ->
    bank_to_MSSses[chnum] ? GenuineEcash(f1,f2) -> q_to_MSStcp ! GenuineEcash(f1,f2)

  /* to MSStcp */
:: bank_to_MSSses[chnum]  ? [FakeEcash(f1,f2)] &&  nfull(q_to_MSStcp) ->
    bank_to_MSSses[chnum] ? FakeEcash(f1,f2) -> q_to_MSStcp ! FakeEcash(f1,f2)

  /* to KsTmpIdMan */
:: MSStcp_to_ses[chnum]  ? [ChangeMyKs(f1,f2)] && nfull(q_to_KsTmpIdMan) ->
    MSStcp_to_ses[chnum] ? ChangeMyKs(f1,f2) -> q_to_KsTmpIdMan ! ChangeMyKs(f1,f2)

  /* to MailSvr */
:: MSStcp_to_ses[chnum]  ? [EaddrAndTxt(f1,f2)] && nfull(q_to_MailSvr) ->
    MSStcp_to_ses[chnum] ? EaddrAndTxt(f1,f2) -> q_to_MailSvr ! EaddrAndTxt(f1,f2)

  /* to bank */
:: MSStcp_to_ses[chnum]  ? [Ecash(f1,f2)] && nfull(q_to_bank) ->
    MSStcp_to_ses[chnum] ? Ecash(f1,f2) -> q_to_bank ! Ecash(f1,f2)

  /* send to MSStcp */
:: nempty(q_to_MSStcp) && nfull(MSSses_to_tcp[chnum]) ->
    q_to_MSStcp ? msgtype(f1,f2);
    if
    :: (msgtype == abort) ->  MSSses_to_tcp[chnum] !! abort(f1,f2)

    :: (msgtype == YourNewKs) -> MSSses_to_tcp[chnum] ! YourNewKs(f1,f2);
                                 MyKs= f1;

    :: (msgtype == YouHaveMail) -> MSSses_to_tcp[chnum] ! YouHaveMail(f1,f2)

:: (msgtype == GenuineEcash) ->
        payment= f1;
        if
        :: (payment < MINMONEY || payment > MAXMONEY) ->
             MSSses_to_tcp[chnum] ! EcashRejetd(payment,f2)

        :: else ->
             atomic{MSSses_to_tcp[chnum] ! EcashAccepd(payment,f2) ->
               assert(payment > LAST_MSGS);
               credit= payment + credit + CreditLeft;
               credit= credit - LAST_MSGS;
               CreditLeft= LAST_MSGS}
        fi
:: (msgtype == FakeEcash) -> MSSses_to_tcp[chnum] ! EcashRejetd(f1,f2)
```

```
    :: (msgtype == TimeAlert) ->  MSSses_to_tcp[chnum] !! TimeAlert(f1,f2)

    :: (msgtype == TimeFin) -> MSSses_to_tcp[chnum] !! TimeFin(f1,f2)

    :: else -> printf("MSSses: tmpId=%d UNKNOWN msg: PANIC-PANIC-PANIC\n\n",tmpId)
    fi

  /* send to KsTmpIdMan */
:: nempty(q_to_KsTmpIdMan) && nfull(MSSses_to_KsTmpIdMan[chnum]) ->
    q_to_KsTmpIdMan ? msgtype(f1,f2);
    if
    :: (msgtype == aborted) -> goto Aborted

    :: (msgtype == ChangeMyKs) -> MSSses_to_KsTmpIdMan[chnum] ! ChangeMyKs(f1,f2)

    fi

  /* send to MailSvr */
:: nempty(q_to_MailSvr) && nfull(MSSses_to_MailSvr[chnum]) &&
    nfull(q_to_MSStcp) && (credit >=1 || CreditLeft >= 1) ->
    q_to_MailSvr ? msgtype(f1,f2);
    if
    :: (msgtype == EaddrAndTxt) ->
        /* if necessary process the msg */
        assert(credit >= 0 && CreditLeft >= 0);
        printf("MSSses: tmpId=%d credit=%d CreditLeft=%d \n\n",
              tmpId,credit,CreditLeft);
    if
    :: (credit >= 1) -> /* charge to 'credit' account */
          MSSses_to_MailSvr[chnum] ! EaddrAndTxt(f1,f2);
          credit--;
          if
          :: (credit == 0) -> q_to_MSStcp! TimeAlert(f1,f2);
          :: else -> skip
          fi
    :: (credit == 0  && CreditLeft >= 1) -> /* charge to CreditLeft' account */
          MSSses_to_MailSvr[chnum] ! EaddrAndTxt(f1,f2);
          CreditLeft--;
          if
          :: (CreditLeft == 0) -> q_to_MSStcp ! TimeFin(f1,f2)
          :: else -> skip
          fi
    fi
fi

  /* send to bank */
:: nempty(q_to_bank) && nfull(MSSses_to_bank[chnum]) ->
    q_to_bank ? msgtype(f1,f2);
    if
    :: (msgtype == Ecash) ->
        MSSses_to_bank[chnum] ! Ecash(f1,f2);
        payment= f1;
        printf("MSSses: tmpId=%d has SNT extra payment=%d to bank \n",
               tmpId,payment)
    fi
od;
```

```
Aborted:
 /*
  * send abort to MailSvr
  */
  do
  :: nfull(MSSses_to_MailSvr[chnum]) ->
          MSSses_to_MailSvr[chnum] !! aborted(f1,f2); break
  :: full(MSSses_to_MailSvr[chnum]) &&  full(MailSvr_to_MSSses[chnum]) ->
          MailSvr_to_MSSses[chnum] ? msgtype(f1,f2) /* drop this msg */
  od;

AbortBank:
 /*
  * send abort to bank
  */
  do
  :: atomic{nfull(MSSses_to_bank[chnum]) ->
          MSSses_to_bank[chnum] !! aborted(f1,f2); break}
  :: full(MSSses_to_bank[chnum]) &&  full(bank_to_MSSses[chnum]) ->
          MSSses_to_bank[chnum] ? msgtype(f1,f2) /* drop this msg */
  od;

 /*
  * send abort to KsTmpIdMan
  */
  MSSses_to_KsTmpIdMan[chnum] !! aborted(f1,f2);

  /*
   * clear chan to MSStcp
   */
  do
  :: atomic{nempty(MSSses_to_tcp[chnum]) ->
       MSSses_to_tcp[chnum] ? msgtype(f1,f2)} /* drop this msg */
  :: empty(MSSses_to_tcp[chnum]) ->  break
  od;

 printf("MSSAnoSes tmpId=%d MyKs=%d +++ABORTED++++ BYE-BYE-BYE
         by KsTmpIdMan \n\n",tmpId,MyKs)
}

/*
 * MSStcp process: connection-oriented reliable link between the MSS
 *                 and the PDA. It forwards msg up the protocol stack
 * and to the PDA side. The MSStcp initiates an abort procedure that
 * propagates up the stack protocol whenever it detects that its remote
 * peer (the PDAtcp) is unreachable.
 */
proctype MSStcp(int tmpId, chnum, Ks)
{
 chan q_to_PDAtcp= [LOCALQSZ] of {byte, int, int};
 chan q_to_MSSses= [LOCALQSZ] of {byte, int, int};

 byte msgtype;  /* type of msg received                              */
 int  f1;       /* field for receiving the first field of a msg  */
 int  f2;       /* field for receiving the second field of a msg */
```

```
assert(0 <= chnum && chnum <= NUMofPDA && tmpId == chnum);


do
   /* recv from MSS site */
:: nfull(q_to_PDAtcp) && nempty(MSSses_to_tcp[chnum]) ->
     MSSses_to_tcp[chnum] ? msgtype(f1,f2) -> q_to_PDAtcp ! msgtype(f1,f2)

   /* recv from PDA site */
:: nfull(q_to_MSSses) && nempty(PDAtcp_to_MSStcp[chnum]) ->
     PDAtcp_to_MSStcp[chnum] ? msgtype(f1,f2) -> q_to_MSSses ! msgtype(f1,f2)

   /* send to MSS site */
:: nempty(q_to_MSSses) && nfull(MSStcp_to_ses[chnum]) ->
     q_to_MSSses ? msgtype(f1,f2);
     if
     :: (msgtype == abort) -> /* abort initiated at PDA site */
          MSStcp_to_ses[chnum] !! aborted(f1,f2);
          printf("MSStcp: tmpId=%d initiated at PDA site going
                  +++Aborted+++\n",tmpId);
          goto Aborted

     :: (msgtype == aborted) -> /* abort initiated at MSS */
          MSStcp_to_ses[chnum] !! aborted(f1,f2);
          printf("MSStcp: tmpId=%d initiated at MSS site going
                  +++Aborted+++\n",tmpId);
          goto Aborted

     :: else ->  /* PDA sending an ordinary msg */
          MSStcp_to_ses[chnum] ! msgtype(f1,f2)
     fi

   /* send to PDA site */
:: nempty(q_to_PDAtcp) && nfull(MSStcp_to_PDAtcp[chnum]) && nfull(q_to_MSSses) ->
   q_to_PDAtcp ? msgtype(f1,f2);
     if
     :: (msgtype == abort) -> /* abort initiated by MSS or MSSksMan */
          MSStcp_to_PDAtcp[chnum] !! abort(f1,f2);
          q_to_MSSses !! aborted(f1,f2);
          printf("MSStcp: tmpId=%d aborted by MSS going to
                  +++Aborted+++ soon \n",tmpId);

     :: (msgtype == TimeFin) -> /* end of AnosSes: initiated by MSS or KsTmpIdMan */
          MSStcp_to_PDAtcp[chnum] !! TimeFin(f1,f2);
          q_to_MSSses !! aborted(f1,f2);
          printf("MSStcp: tmpId=%d TimeFin by MSS going to
                  +++Aborted+++ soon \n",tmpId);

     :: (msgtype == YourNewKs) -> /* MSSman has changed Ks */
          Ks= f1;
          MSStcp_to_PDAtcp[chnum] ! YourNewKs(f1,f2)

     :: else -> MSStcp_to_PDAtcp[chnum] ! msgtype(f1,f2) /* MSS sent routine msg */
     fi


:: full(MSStcp_to_PDAtcp[chnum]) &&
   timeout -> /* PDA not receiving: assume PDA has aborted its session */
```

```
      if
      :: nfull(MSStcp_to_ses[chnum]) -> MSStcp_to_ses[chnum] !! aborted(f1,f2);
                                        goto Aborted

      :: full(MSStcp_to_ses[chnum]) && timeout -> goto ForceAbort
      fi


 :: empty(PDAtcp_to_MSStcp[chnum]) &&
    timeout -> /* PDA not sending: assume PDA has aborted its session */
      if
      :: nfull(MSStcp_to_ses[chnum]) -> MSStcp_to_ses[chnum] !! aborted(f1,f2);
                                        goto Aborted

      :: full(MSStcp_to_ses[chnum]) && timeout -> goto ForceAbort
      fi
od;


ForceAbort: /* force MSSses to abort (to read 'aborted') from it chan */
do
:: MSStcp_to_ses[chnum] !! aborted(tmpId,chnum) -> break
:: timeout -> MSSses_to_tcp[chnum] ? msgtype(f1,f2) /* drop this msg */
od;
printf("MSStcp tmpId= %d going to +++Aborted +++ by KsTmpIdMan \n",tmpId,chnum);


 Aborted:
 if
 :: atomic{nempty(MSStcp_to_PDAtcp[chnum]) && timeout ->
       do
       :: nempty(MSStcp_to_PDAtcp[chnum]) ->
            MSStcp_to_PDAtcp[chnum] ? msgtype(f1,f2) /* drop this msg */
       :: empty(MSStcp_to_PDAtcp[chnum]) -> break
       od}
 :: empty(MSStcp_to_PDAtcp[chnum]) -> skip
 fi;

 printf("MSStcp: tmpId= %d chanum= %d +++ABORTED+++
         BYE-BYE-BYE\n\n",tmpId,chnum)
}
```

```
/*
 * bank process: receives an e-coin and verify that it is
 *               genuine and that it has not been spent.
 */
proctype bank()
{
 int  tmpId;    /* temporary Id */
 int  chnum;    /* channel number */
 int  payment;  /* e-coin         */
 byte msgtype;  /* type of msg received                              */
 int  f1;       /* field for receiving the first field of a msg  */
 int  f2;       /* field for receiving the second field of a msg */

 chnum= -1;
```

```
do
:: (true) ->
   end_cyc:if
   :: MSSses_to_bank[0] ? msgtype(f1,f2) ->
        assert(msgtype == Ecash || msgtype == aborted);
        if
        :: (msgtype == aborted) ->
              chnum= 0; tmpId=0; goto ClearChan
        :: (msgtype == Ecash)   ->
              payment=f1; chnum= 0; tmpId=0; goto progress_ecashTocheck
        fi

   :: MSSses_to_bank[1] ? msgtype(f1,f2) ->
        assert(msgtype == Ecash || msgtype == aborted);
        if
        :: (msgtype == aborted) -> chnum= 1; tmpId=1; goto ClearChan
        :: (msgtype == Ecash)   ->
              payment=f1; chnum= 1; tmpId=1; goto progress_ecashTocheck
        fi

   :: MSSses_to_bank[2] ? msgtype(f1,f2) ->
        assert(msgtype == Ecash || msgtype == aborted);
        if
        :: (msgtype == aborted) ->
              chnum= 2; tmpId=2; goto ClearChan
        :: (msgtype == Ecash)   ->
              payment=f1; chnum= 2; tmpId=2; goto progress_ecashTocheck
        fi
   fi;

   progress_ecashTocheck: skip;
   /* proc with tmpId= 0 has chnum=0, proc with tmpId=1 has chnum=2, etc */
   assert(0 <= chnum && chnum < NUMofPDA && 0 <= tmpId && tmpId < NUMofPDA);
   if
   :: (payment == FAKEMONEY) -> bank_to_MSSses[chnum] ! FakeEcash(payment,f2);
        printf("BANK: tmpId= %d chnum= %d SENT FakeCash payment= %d to PDA \n",
               tmpId,chnum, payment)

   :: else -> bank_to_MSSses[chnum] ! GenuineEcash(payment,f2) ->
        printf("BANK: tmpId= %d chnum= %d SENT GenuineEcash payment= %d to PDA \n",
               tmpId,chnum, payment)
   fi;
   goto end_cyc;

   ClearChan:
    do
    :: nempty(bank_to_MSSses[chnum]) ->
         bank_to_MSSses[chnum] ? msgtype(f1,f2) /* drop this msg */
    :: empty(bank_to_MSSses[chnum]) ->
         break
    od;
    goto end_cyc

 od
}
```

```
/*
 * MailSvr process: receives msg coming from Bob's PDA and simulates that
 *                          they are sent to Alice's (their final destination)
 * e-mail address. Alice's computer is connected somewhere to the Internet.
 * Similarly, it simulates that it receives replies to Bob's msgs and for-
 * wards them to Bob. The decision about replying or not to an e-mail is ta-
 * ken randomly.
 */
proctype MailSvr()
{
 int  ReplyVec[MAXNUMPDA]; /* msg awaiting for replies */
 int  tmpId;               /* temporary Id             */
 int  chnum;               /* channel number           */
 int  addr;                /* e-mail address           */
 int  txt;                 /* text in e-mail msg       */
 byte msgtype;  /* type of msg received                              */
 int  f1;       /* field for receiving the first field of a msg  */
 int  f2;       /* field for receiving the second field of a msg */
 int  i;        /* counter                                       */

 i=0;
 do
 :: i <  MAXNUMPDA ->
      ReplyVec[i]= 0;
      i++
:: else -> break
od;

chnum= -1;

do
:: (true) ->
   end_cyc: if
   :: MSSses_to_MailSvr[0] ? msgtype(f1,f2) ->
        assert(msgtype == EaddrAndTxt || msgtype == aborted);
        if
        :: (msgtype == EaddrAndTxt) ->
              addr= f1; txt= f2; ReplyVec[0]= ReplyVec[0] + 1; goto  progress_mailRcvd

        :: (msgtype == aborted) -> tmpId= 0; chnum= 0; goto ClearChan
        fi

:: MSSses_to_MailSvr[1] ? msgtype(f1,f2) ->
    assert(msgtype == EaddrAndTxt || msgtype == aborted);
    if
    :: (msgtype == EaddrAndTxt) ->
          addr= f1; txt= f2; ReplyVec[1]= ReplyVec[1] + 1; goto  progress_mailRcvd

    :: (msgtype == aborted) -> tmpId= 1; chnum= 1; goto ClearChan
    fi

:: MSSses_to_MailSvr[2] ? msgtype(f1,f2) ->
    assert(msgtype == EaddrAndTxt || msgtype == aborted);
    if
    :: (msgtype == EaddrAndTxt) ->
          addr= f1; txt= f2; ReplyVec[2]= ReplyVec[2] + 1; goto  progress_mailRcvd
```

```
        :: (msgtype == aborted) -> tmpId= 2; chnum= 2; goto ClearChan
        fi
    fi;

progress_mailRcvd: skip;
    if
    :: (ReplyVec[0] >= 1) ->
                    if /* Alice doesn't reply to Bob: discharge a msg */
                    :: ReplyVec[0]= ReplyVec[0] -1
                    :: skip
                    fi;
                    if /* make chnum=0 if there're any reply for channel 0 */
                    :: (ReplyVec[0] >= 1) -> chnum= 0
                    :: else -> chnum= -1   /* no replies for channel 0 */
                    fi
 :: (ReplyVec[1] >= 1) ->
                    if
                    :: ReplyVec[1]= ReplyVec[1] -1
                    :: skip
                    fi;
                    if
                    :: (ReplyVec[1] >= 1) -> chnum= 1
                    :: else -> chnum= -1
                    fi
 :: (ReplyVec[2] >= 1) ->
                    if
                    :: ReplyVec[2]= ReplyVec[2] -1
                    :: skip
                    fi;
                    if
                    :: (ReplyVec[2] >= 1) -> chnum= 2
                    :: else -> chnum= -1
                    fi
 fi;

 if
 :: (chnum >= 0 && chnum < NUMofPDA) ->
       if
       :: MailSvr_to_MSSses[chnum] ! YouHaveMail(chnum,_pid) ->
          progress_replySent: skip
       :: timeout-> skip /* PDA is off or has gone away from MSS */
       fi;
       ReplyVec[chnum]= ReplyVec[chnum] -1;
 :: else-> skip
    fi;

    chnum= -1;
    goto end_cyc;

 ClearChan:
  do
  :: nempty(MailSvr_to_MSSses[chnum]) ->
       MailSvr_to_MSSses[chnum] ? msgtype(f1,f2) /* drop this msg */
  :: empty(MailSvr_to_MSSses[chnum]) ->
       break
  od;
  goto end_cyc
```

```
 od
}




/*
 * EscKey process:
 * -the fact that the PDA user can press the ESC keyboard at any time to
 *  interrupt his anonymous session is simulated by a 'timeout'
 *  which can go off at any time.
 * -interruption of the PDA user anonymous session can be originated at
 *  the PDAuser, PDAses, PDAtcp or at the MSS site, if this happens
 *  an 'abort' message is received which lead to abort the keyboard pro-
 *  cess.
 */
proctype EscKey(chan user_to_EscKey, EscKey_to_user; int tmpId, chnum)
{
 int  f1;       /* field for receiving the first field of a msg  */
 int  f2;       /* field for receiving the second field of a msg */
 if
 :: user_to_EscKey ? aborted(f1,f2) -> /* abort from PDA or MSS site */
      goto Aborted
 :: timeout -> EscKey_to_user !! abort(tmpId,chnum) ->
      goto Aborted             /* simulates the user pressing ESC   */
                               /* to interrupt his anonymous session */
 fi;

 Aborted:
  printf("EscKey: tmpId=%d aborted at PDA or MSS site BYE-BYE-BYE\n\n",tmpId);
}




/*
 * PDAuser process: is the interface between the PDA user and the
 *                  anonymous and confidential communications system.
 * It receives PDA user's commands typed on the keyboard and display
 * messages on the PDA screen.
 */
proctype PDAuser(int PDAnum)
{
 byte msgtype;  /* type of msg received                          */
 int  f1;       /* field for receiving the first field of a msg  */
 int  f2;       /* field for receiving the second field of a msg */
 int  tmpId;    /* temporary Id                 */
 int  chnum;    /* channel number               */
 /* int  PuKey; *//* public key of the MSS */
 bool KsChanged; /* flag to stop the PDA changing its Ks more than once     */
 int  payment;   /* payment for opening or extending an anonymous session   */
 int  MaxExtPay; /* max amount of money a user is allowed to spend in calls */
 int  ExtPay;    /* max amount of money a user hast spent in calls          */
 int  addr;      /* e-mail address                                          */
 int  txt;       /* text in e-mail msg                                      */

 bool AbortFlag; /* was the process forced to abort YES/NO ? */
```

```
int i;    /* counter of attempts */
int NumM; /* number of e-mails to send */

chan ses_to_user_localch=[1] of {byte,int,int};
chan user_to_esckey=[1]      of {byte,int,int};
chan esckey_to_user=[1]      of {byte,int,int};

KsChanged= NO;
AbortFlag= NO;
```

```
 /*
  * run PDAses process: ses_to_user_localch is used by PDAuser to
  * receive tmpId and from PDAses
  */
 run PDAses(PDAnum, ses_to_user_localch);


/*
 * This code assumes that the Ks-negotiation has been verified, so lines related to
 * Ks negotiation have been commented
 */
/*
 * progress_KpuNegotiation:
 */
 /* do
  * :: ses_to_user_localch ? KpuRcvd(f1,f2) ->
  *         printf("PDAuser: PDAnum=%d pid=%d : Kpu learnt \n",PDAnum,_pid);
  *         break
  * :: ses_to_user_localch ? aborted(f1,f2) ->
  *         printf("PDAuser: PDAnum=%d pid=%d :My PDAses couldn't get a Ks; going
  *                 +++Aborted+++\n",PDAnum,_pid);
  *         goto Aborted
  * od;
  */

progress_KsNegotiation:
 do
 :: ses_to_user_localch ? tmpIdRcvd(tmpId,chnum) -> break
 :: ses_to_user_localch ? aborted(f1,f2) ->
          printf("PDAuser: PDAnum=%d pid=%d :My PDAses couldn't get a Ks; going
                  +++Aborted+++\n",PDAnum,_pid);
          goto Aborted
 od;
```

```
 MaxExtPay= MAXEXTPAY;
 ExtPay= 0;

progress_FirstPayment: do
 :: (i <= MAXNUMPAYATTEMPTS) ->
    if
    :: payment= FAKEMONEY
    :: payment= TOOLITTLEMONEY
    :: payment= GOLDENMONEY
    :: payment= SILVERMONEY
    fi;

    PDAuser_to_ses[chnum] ! Ecash(payment,tmpId) ->
```

```
PDAses_to_user[chnum] ? msgtype(payment,f2) ->
    if
    :: (msgtype == EcashRejetd) ->
           i++  /* go back and fetch another coin from PDA memory */

    :: (msgtype == EcashAccepd) ->
           NumM=MAXNUMEMAILS_SENT;
          /*
           * run the EscKey process */
           */
           run EscKey(user_to_esckey, esckey_to_user, tmpId, chnum);
           goto progress_AnoSes

       :: (msgtype == aborted) ->
           printf("PDAuser: tmpId=%d couldn't PAY; aborted by PDAses|PDAtcp|MSS going
                  +++Aborted+++\n",tmpId);
           goto Aborted
       fi

:: else -> PDAuser_to_ses[chnum] !! abort(tmpId,chnum) ->
       do
       :: PDAses_to_user[chnum] ? msgtype(f1,f2) ->
            if
            :: (msgtype != aborted) -> skip /* discard this msg */
            :: (msgtype == aborted) ->
                 printf("PDAuser: tmpId=%d couldn't PAY; going +++Aborted+++\n",tmpId);
                 goto Aborted
            fi
       od
od;

progress_AnoSes: do
 :: (NumM >= 1 && nfull(PDAuser_to_ses[chnum])) ->
       PDAuser_to_ses[chnum] ! EaddrAndTxt(_pid,_pid); NumM--

 :: PDAses_to_user[chnum] ? YouHaveMail(addr,txt) ->
       printf("===+++>PDAuser tmpId= %d chnum= %d  GOT E-MAIL
              addr= %d txt= %d\n",tmpId,chnum,addr,txt)

 :: (KsChanged==NO) -> PDAuser_to_ses[chnum] ! ChangeMyKs(tmpId,chnum); KsChanged= YES
                                               /* Ks can be changed only once */

:: PDAses_to_user[chnum] ? YourNewKs(f1,f2) ->
     printf("PDAuser tmpId= %d has got a new Ks \n",tmpId)

:: PDAses_to_user[chnum] ? TimeAlert(f1,f2) ->
       printf("$$$>PDAuser tmpId=%d has been TimeAlerted\n",tmpId);
       if
       :: (ExtPay <= MaxExtPay) ->
            if
            :: payment= FAKEMONEY
            :: payment= TOOLITTLEMONEY
            :: payment= GOLDENMONEY
            :: payment= SILVERMONEY
            :: payment= NOMONEY
            fi;
            PDAuser_to_ses[chnum] ! Ecash(payment,tmpId);
```

```
            printf("$$$>PDAuser tmpId=%d has sent EXTRA
                    payment=%d\n",tmpId,payment)
      :: else -> skip /* no more extensions allowed */
      fi

::  PDAses_to_user[chnum] ? TimeFin(f1,f2) ->
        printf("PDAuser: tmpId=%d TimeFin going +++Abort+++ \n",tmpId);
        user_to_esckey!! aborted(tmpId,chnum);
        goto Aborted

 :: PDAses_to_user[chnum] ? EcashAccepd(f1,f2) ->
      ExtPay= ExtPay + payment;
      NumM= MAXNUMEMAILS_SENT;
      printf("PDAuser: tmpId=%d AnoTime INCRTED msg to send=%d\n",tmpId,NumM);

 :: PDAses_to_user[chnum] ? EcashRejetd(f1,f2) ->
      printf("PDAuser: tmpId= %d chnum= %d AnoTime NOT incremented\n",tmpId,chnum)

 :: PDAses_to_user[chnum] ? aborted(f1,f2) ->
      user_to_esckey!! aborted(tmpId,chnum);
      printf("PDAuser: tmpId=%d aborted initiated by PDAses | PDAtct | MSS going
              +++Abort+++\n",tmpId);
      goto Aborted

 :: esckey_to_user ? abort(f1,f2) -> /* user pressed ESC key to interrupt session */
      PDAuser_to_ses[chnum] !! abort(chnum,tmpId)
od;

Aborted:
 AbortFlag= YES;
 printf("PDAuser: +++ABORTED+++ BYE-BYE-BYE\n\n",tmpId,chnum);

End:
 do
 :: nempty(PDAuser_to_ses[chnum]) ->
      PDAuser_to_ses[chnum] ? msgtype(f1,f2) /* drop this msg */
 :: empty(PDAuser_to_ses[chnum])  ->
      break
 od;

 if
 :: (AbortFlag == NO) ->
       printf("PDAuser: tmpId= %d chnum= %d HAPPY END BYE-BYE-BYE\n\n",tmpId,chnum)
 :: else -> skip
 fi
}



/*
 * PDAses process: is in charge of learning the Kpu key and negotiating
 *                 a Ks key. It encrypts PDAuser msg before forwarding
 * them down the protocol stack; conversely, it decrypts msg coming from
 * the underneath layer and forward them to the PDAuser layer.
 */
```

```
proctype PDAses(byte MyPDAnum; chan ses_to_user_localch)
{
 chan q_to_tcp=[LOCALQSZ]  of {byte, int, int};
 chan q_to_user=[LOCALQSZ] of {byte, int, int};

 /* int  PuKey;*/
 int  Ks;      /* session key of the PDA */
 int  tmpId;   /* temporary Id           */
 int  chnum;   /* channel number         */
 int  payment; /* payment for an anonymous session */
 byte msgtype; /* type of msg received             */
 int  f1;       /* field for receiving the first field of a msg  */
 int  f2;       /* field for receiving the second field of a msg */
 int  addr;    /* e-mail address                   */
 int  txt;       /* text in e-mail msg             */
 int  i;          /* i <= MAXNUM_Ks_ATTEMPTS: counter number of attempt */
                  /* to register a session key                          */
 bool AbortFlag;  /* was the process forced to abort YES/NO ?           */
 bool KsAccepd;   /* The suggested Ks was accepted YES/NO ?             */


/*
 * This code assumes that the Ks negotiation has been verified, so lines related to
 * Ks negotiation have been commented
 */
/* KpuNegotiation: */
/*
* Part of the PDAses process that deals with the Kpu negotiation between the
* PDA and the MSS.
* Both the PDAuser and the PDAses end in 'Aborted' if the PDAses fails to
* get the Kpu after MAXNUM_Kpu_ATTEMPTS.
*/
/* PuKey= 0;
 * i= 1;
 * do
 * :: (i <= MAXNUM_Kpu_ATTEMPTS) ->
 *  if
 *  :: KpuPort_to_PDA ? Kpu(PuKey) -> // Kpu rcvd: now check for authenticity
 *     if
 *     :: (PuKey == CERTIFIED_Kpu) ->
 *          ses_to_user_localch ! KpuRcvd(ANYINT,ANYINT);
 *          printf("MyPDAnum=%d has learnt Kpu after N= %d; going to
 *                  KsNegotiation\n", MyPDAnum,i-1);
 *          goto KsNegotiation
 *     :: else -> i++
 *     fi
 *
 *  :: KpuPort_to_PDA ? BogusKpu(PuKey) -> i++ // bogus msg rcvd: ignore it
 *
 *  :: timeout ->  // couldn't  hear any msg from this MSS: move to a new one
 *       ses_to_user_localch ! aborted(ANYINT,ANYINT);
 *       goto Aborted
 *  fi
 * :: else ->
 *  printf("MyPDAnum=%d Failed to get Kpu after N= %d; going to
 *          +++Aborted+++\n", MyPDAnum,i-1);
 *  ses_to_user_localch ! aborted(ANYINT,ANYINT);
 *   goto Aborted
 * od;
```

```
  *
  */

KsNegotiation:
 Ks= 0;
 tmpId= 0;

 AbortFlag= NO;
KsAccepd= NO;

i= 1;
do
:: (i <= MAXNUM_Ks_ATTEMPTS) ->
   if       /* random selection of Ks */

   :: Ks = KsBlack ->
        PDA_to_KsPort ! Ks;
        if
        /*
         * There may be up to 5 msg in the channel buffer. If there is one
         * (not necessarily at the head of the buffer) with mtype= KsBlack
         * retrieve it. Otherwise block until such a msg appears in the
         * channel or timeout goes off
         */
        :: KsPort_to_PDA ?? KsBlack(tmpId,chnum) ->
                                KsAccepd = YES;
                                break
        :: timeout ->
                printf("PDAses: PDAnum= %d;  sent Ks= %d  i= %d NO answer ->
                        TIMEOUT \n",MyPDAnum,Ks,i);
                i++
        fi

:: Ks = KsBlue ->
     PDA_to_KsPort ! Ks;
     if
     :: KsPort_to_PDA ?? KsBlue(tmpId,chnum) ->
                                KsAccepd = YES;
                                break
     :: timeout ->
                printf("PDAses: PDAnum= %d;  sent Ks= %d  i= %d NO answer ->
                        TIMEOUT \n",MyPDAnum,Ks,i);
                i++
     fi

:: Ks = KsGreen ->
     PDA_to_KsPort ! Ks;
     if
     :: KsPort_to_PDA ?? KsGreen(tmpId,chnum) ->
                                KsAccepd = YES;
                                break
     :: timeout ->
                printf("PDAses: PDAnum= %d;  sent Ks= %d  i= %d NO answer ->
                        TIMEOUT \n",MyPDAnum,Ks,i);
                i++
     fi

:: Ks = KsPink ->
```

```
PDA_to_KsPort ! Ks;
if
:: KsPort_to_PDA ?? KsPink(tmpId,chnum) ->
                        KsAccepd = YES;
                        break
:: timeout ->
          printf("PDAses: PDAnum= %d;  sent Ks= %d  i= %d NO answer ->
                  TIMEOUT \n",MyPDAnum,Ks,i);
          i++
fi
:: Ks = KsRed ->
     PDA_to_KsPort ! Ks;
     if
     :: KsPort_to_PDA ?? KsRed(tmpId,chnum) ->
                             KsAccepd = YES;
                             break
     :: timeout ->
               printf("PDAses: PDAnum= %d;  sent Ks= %d  i= %d NO answer ->
                       TIMEOUT \n",MyPDAnum,Ks,i);
               i++
     fi

:: Ks = KsWhite ->
     printf("PDAses: PDAnum= %d is going to send Ks= %d \n",MyPDAnum,Ks);
     PDA_to_KsPort ! Ks;
     if
     :: KsPort_to_PDA ?? KsWhite(tmpId,chnum) ->
                             KsAccepd = YES;   /* this should never happens */
                             break             /* this should never happens */
        :: timeout ->
                  printf("PDAses: PDAnum= %d;  sent Ks= %d  i= %d NO answer ->
                          TIMEOUT \n",MyPDAnum,Ks,i);
                  i++
        fi

   fi

:: (i > MAXNUM_Ks_ATTEMPTS) ->
    break /* give up registering a Ks */
od;


/*
 *  communication between the PDAuser and the PDAses
 */
if
:: (KsAccepd == NO) ->
    printf("PDAses: PDAnum=%d failed to register Ks after i=%d attempts: going
            +++Aborted+++\n",MyPDAnum,i-1);
    ses_to_user_localch !! aborted(NEGINT,NEGINT);
    goto Aborted
:: else ->
    ses_to_user_localch ! tmpIdRcvd(tmpId,chnum);
    printf("PDAses: PDAnum=%d; pid=%d HAS RgTED Ks=%d (tmpId= %d) after i= %d
            attempts \n",MyPDAnum,_pid,Ks,tmpId,i);
    /*
     * initiating the underneath layer
     */
    run PDAtcp(tmpId,chnum,Ks)
```

```
 fi;


progress_FirstPayment: do /* progress of PDAses proc to FirstPayment state */
 :: PDAuser_to_ses[chnum] ? msgtype(f1,f2) ->
     if
     :: (msgtype == Ecash) ->
                    skip /* if necessary do anything to the msg */

     :: (msgtype == abort) -> PDAses_to_tcp[chnum] !! abort(f1,f2) ->
           do
           :: PDAtcp_to_ses[chnum] ? msgtype(f1,f2);
                if
                :: (msgtype != aborted) -> skip /* discard the msg */
                :: (msgtype == aborted) -> PDAses_to_user[chnum] !! aborted(f1,f2);
                                                goto Aborted
                fi
           od
     fi;

     if
     :: PDAses_to_tcp[chnum] ! msgtype(f1,f2)
     fi

:: PDAtcp_to_ses[chnum] ? msgtype(f1,f2) ->
     if
     :: (msgtype == EcashRejetd) ->
                    skip /* if necessary do anything to the msg */

     :: (msgtype == EcashAccepd) ->
                    skip; /* if necessary do anything to the msg */
                    payment= f1;
                    PDAses_to_user[chnum] ! EcashAccepd(payment,tmpId);
                    goto progress_AnoSes

     :: (msgtype == aborted) ->
                    PDAses_to_user[chnum] !! aborted(f1,f2);
                    goto Aborted
     fi;

     if
     :: PDAses_to_user[chnum] ! msgtype(f1,f2)
     fi
od;
```

```
progress_AnoSes: do
     /* from user to ses to tcp */
 :: nfull(q_to_tcp) && nempty(PDAuser_to_ses[chnum]) ->
      PDAuser_to_ses[chnum] ? msgtype(f1,f2) -> q_to_tcp ! msgtype(f1,f2)

      /* from tcp to ses to user */
 :: nfull(q_to_user) && nempty(PDAtcp_to_ses[chnum]) ->
      PDAtcp_to_ses[chnum] ? msgtype(f1,f2) -> q_to_user ! msgtype(f1,f2)

    /* send to user */
 :: nempty(q_to_user) && nfull(PDAses_to_user[chnum]) ->
```

```
q_to_user ? msgtype(f1,f2);
if
:: (msgtype == YourNewKs) ->
            printf("PDAses: tmpId=%d GOT new Ks OldKs=%d NewKs= %d\n",
                    tmpId,Ks,f1);
            Ks= f1;
            /* if necessary process the msg */
            PDAses_to_user[chnum] ! YourNewKs(f1,f2)

:: (msgtype == TimeAlert) ->
            /* if necessary process the msg */
            PDAses_to_user[chnum] ! TimeAlert(f1,f2)

:: (msgtype == EcashAccepd) ->
            /* if necessary process the msg */
            PDAses_to_user[chnum] ! EcashAccepd(f1,f2)

:: (msgtype == EcashRejetd) ->
            /* if necessary process the msg */
            PDAses_to_user[chnum] ! EcashRejetd(f1,f2)

:: (msgtype == YouHaveMail) ->
            /* if necessary process the msg */
            PDAses_to_user[chnum] ! YouHaveMail(f1,f2)

:: (msgtype == aborted) ->
            /* if necessary process the msg */
            PDAses_to_user[chnum] !! aborted(f1,f2);
            printf("PDAses: tmpId=%d abort initiated by PDAtcp or MSS going
                   +++Aborted+++ \n",tmpId);
            goto Aborted

   :: (msgtype == TimeFin) ->
            /* if necessary process the msg */
            PDAses_to_user[chnum] ! TimeFin(f1,f2);
            printf("PDAses: tmpId=%d TimeFin MSS going
                    +++Aborted+++ \n",tmpId);
            goto Aborted
   fi;


  /* send to tcp */
:: nempty(q_to_tcp) && nfull(PDAses_to_tcp[chnum]) ->
   q_to_tcp ? msgtype(f1,f2);
   if
   :: (msgtype == ChangeMyKs) ->
            printf("PDAses: tmpId=%d user ASKING to change OldKs=%d \n",tmpId,Ks);
            /* if necessary process the msg */
            PDAses_to_tcp[chnum] ! ChangeMyKs(f1,f2)

   :: (msgtype == EaddrAndTxt) ->
            /* if necessary process the msg */
            PDAses_to_tcp[chnum] ! EaddrAndTxt(f1,f2)

   :: (msgtype == Ecash) ->
            /* if necessary process the msg */
            PDAses_to_tcp[chnum] ! Ecash(f1,f2)

   :: (msgtype == abort) ->
```

```
                /* if necessary process the msg */
                PDAses_to_tcp[chnum] !! abort(f1,f2)
    fi

 od;


Aborted:
 AbortFlag= YES;
 printf("PDAses: +++ABORTED+++ BYE-BYE-BYE\n\n");

End:
 do
 :: atomic{nempty(PDAses_to_tcp[chnum]) ->
      PDAses_to_tcp[chnum] ? msgtype(f1,f2)} /* drop this msg */
 :: empty(PDAses_to_tcp[chnum])  ->
      break
 od;

 if
 :: (AbortFlag== NO) -> printf("PDAses: HAPPY END BYE-BYE-BYE\n\n")
 :: else-> skip
 fi
}

/*
 * PDAtcp process: connection-oriented reliable link between the PDA
 *                 and the MSS. It forwards msg up the protocol stack
 * and to the MSS side. The PDAtcp initiates an abort procedure that
 * propagates up the stack protocol whenever it detects that its remote
 * peer (the MSStcp) is unreachable.
 */
proctype PDAtcp(int tmpId, chnum, Ks)
{
 chan q_to_PDAses= [LOCALQSZ] of {byte, int, int};
 chan q_to_MSStcp= [LOCALQSZ] of {byte, int, int};

 byte msgtype;  /* type of msg received                              */
 int  f1;       /* field for receiving the first field of a msg  */
 int  f2;       /* field for receiving the second field of a msg */

 assert(0 <= chnum && chnum <= NUMofPDA && tmpId == chnum);

do
   /* from PDAtcp to PDAses */
:: nfull(q_to_PDAses) && nempty(MSStcp_to_PDAtcp[chnum]) ->
     MSStcp_to_PDAtcp[chnum] ? msgtype(f1,f2) -> q_to_PDAses ! msgtype(f1,f2)

   /* PDAses to PDAtcp */
:: nfull(q_to_MSStcp) && nempty(PDAses_to_tcp[chnum]) ->
     PDAses_to_tcp[chnum] ? msgtype(f1,f2) -> q_to_MSStcp ! msgtype(f1,f2)

   /* send to PDAses */
:: nempty(q_to_PDAses) && nfull(PDAtcp_to_ses[chnum]) ->
     q_to_PDAses ? msgtype(f1,f2);
     if
```

```
:: (msgtype == abort) -> /* abort initiated by MSS */
            PDAtcp_to_ses[chnum] !! aborted(f1,f2);
            printf("PDAtcp: tmpId=%d initiated by MSS going
                    +++Aborted+++\n",tmpId,chnum);
            goto Aborted

:: (msgtype == TimeFin) -> /* TimeFin initiated by MSS */
            PDAtcp_to_ses[chnum] !! TimeFin(f1,f2);
            printf("PDAtcp: tmpId=%d TimeFin initiated by MSS going
                    +++Aborted+++\n",tmpId);
            goto Aborted

:: (msgtype == aborted) -> /* abort initiated by PDAuser or PDAses or PDAtcp */
            PDAtcp_to_ses[chnum] !! aborted(f1,f2);
            printf("PDAtcp: tmpId=%d initiated at PDA side going
                    +++Aborted+++\n",tmpId);
            goto Aborted

:: (msgtype == YourNewKs) -> /* MSSman has changed Ks */
             Ks= f1;
             PDAtcp_to_ses[chnum] ! YourNewKs(f1,f2)

:: else ->  /* MSS sending a routine msg */
            PDAtcp_to_ses[chnum] ! msgtype(f1,f2)
fi

  /* send to MSStcp */
:: nempty(q_to_MSStcp) && nfull(PDAtcp_to_MSStcp[chnum])
                       && nfull(q_to_PDAses) ->
    q_to_MSStcp ? msgtype(f1,f2);
    if  /* PDA sent ordinary msg */
    :: (msgtype != abort) -> PDAtcp_to_MSStcp[chnum] ! msgtype(f1,f2)

    :: (msgtype == abort) -> /* abort initiated by PDA */
       PDAtcp_to_MSStcp[chnum] !! abort(f1,f2);
       q_to_PDAses !! aborted(f1,f2);
       printf("PDAtcp: tmpId=%d aborted by PDA going to +++Aborted+++
               soon \n",tmpId)
    fi

:: full(PDAtcp_to_MSStcp[chnum]) &&
   timeout -> /* MSS not reading: assume MSS aborted session */
          if
          :: nfull(q_to_PDAses) -> q_to_PDAses !! aborted(f1,f2)
          :: full(q_to_PDAses) && timeout -> goto ForceAbort
          fi

od;

ForceAbort:
 do
 :: PDAtcp_to_ses[chnum] !! aborted(tmpId,chnum) -> break
 :: timeout -> PDAses_to_tcp[chnum] ? msgtype(f1,f2) /* drop this msg */
 od;

Aborted:
if
:: atomic{nempty(PDAtcp_to_MSStcp[chnum]) && timeout ->
```

```
      do
      :: nempty(PDAtcp_to_MSStcp[chnum]) ->
           PDAtcp_to_MSStcp[chnum] ? msgtype(f1,f2) /* drop this msg */
      :: empty(PDAtcp_to_MSStcp[chnum]) ->
           break
      od}
:: empty(PDAtcp_to_MSStcp[chnum]) -> skip
fi;

 printf("PDAtcp: tmpId= %d chanum= %d +++Aborted+++ BYE-BYE-BYE\n\n",tmpId,chnum)
}



/*
 * instantiates the participants processes
 */
init
{
   int PDAnum;  /* number of PDA */

  /*
   * This code assumes that the Ks negotiation has been verified, so lines related to
   * Ks negotiation have been commented
   */
   /* run KpuMan(); *//* create the Kpu manager process */

   atomic{
   run KsTmpIdMan(); /* instantiate the tmpId and Ks manager process */
   run bank();       /* instantiate the bank process                 */
   run MailSvr()     /* instantiate the MailSvr process              */
   };

   PDAnum= 0;

   do /* instantiate NUMofPDA processes, one for each PDA */
   :: PDAnum <  NUMofPDA ->
                         run PDAuser(PDAnum);
                         PDAnum++
   :: PDAnum >= NUMofPDA  -> break
   od
}
```

Bibliography

[1] George Lawton. The Internet's challenge to privacy. *Computer*, 31(6), June 1998.

[2] Lorrie Faith Cranor, Roger Clarke, Josef Dietl, Daniel Jaye, and Yves Le Roux. Laws, self–regulation, and p3p: Will w3c's privacy platform help make the web safe for privacy. *Computer Networks and ISDN Systems*, 30(1–7):751–753, April 1998. Proceedings of the Seventh International World Wide Web Conference 14–18 April 1998, Brisbane Australia.

[3] Lincoln D. Stein. The world wide web security faq. http://www.w3.org/Security/Faq/www-security-faq.htm, April 1998.

[4] Chris Pounder. Security and the new data protection law. *Computers & Security*, 17(2), 1998.

[5] UK Parliament. Data protection act 1998. http://www.hmso.gov.uk/acts/acts1998/19980029.htm, July 1998.

[6] Charles P. Pfleeger and Deborath M. Cooper. Security and privacy: Promising advances. *IEEE Software*, sep–oct 1997.

[7] *World Wide Web Journal*, 2(3), Summer 1997.

[8] Simson Garfinkel and Gene Spafford. *Practical Unix and Internet Security.* O'Reilly & Associates, Inc., second edition, 1996.

[9] B.C̃lifford Neuman. Security, payment, and privacy for network commerce. *IEEE Journal on Selected Areas in Communications*, 13(8), October 1995.

[10] Philippe A. Janson. Security mangement and management of security. In Morris Sloman, editor, *Network and Distributed System Management*, chapter 15, pages 403–429. Addison–Wesley Publishing Company, Great Britain, 1994.

[11] Anish Bhimani. Securing the commercial Internet. In Dorothy E. Denning and Peter J. Denning, editors, *Internet besieged. Countering Cyberspace Scofflaws*, chapter 24, pages 407–419. Addison–Wesley, 1998. Published also in Communications of the ACM V.39 N.6 1996.

[12] S.M. Bellovin. Security problems in the TCP/IP protocol suite. *Computer Communication Review*, 19(2), April 1998.

[13] Vijay K. Garg and Joseph E. Wilkes. *Wireless and Personal Communications Systems.* Prentice Hall PTR, Upper Saddle River, NJ, 1996.

[14] Joseph E. Wilkes. Privacy and authentication needs of PCS. *IEEE Personal Communications*, 2(4), August 1995.

[15] Daniel Guinier. From eavesdropping to security on the cellular telephone system GSM. *SIG Security Audit & Control Review, acm Press*, 15(2), April 1997.

[16] Donal O'Mahony. UMTS: The fusion of fixed and mobile networks. *IEEE Internet Computing*, 2(1), January–February 1998.

[17] Ioannis N. Kriakas, André W. Jarvis, Vincent E. Phillips, and Derek J. Richards. Third–generation mobile network architectures for the universal mobile telecommunications systems (UMTS). *Bell Labs Technical Journal*, 2(3), Summer 1998.

[18] Ken Buchanan, Rodger Fudge, David McFarlane, Tim Phillips, Akio Sasaki, and Howard Xia. Imt–2000: Service provider's perspective. *IEEE Personal Communications*, 4(4), August 1997.

[19] Joao Schwartz Dasilva, Demosthenes Ikonomou, and Heiko Erben. European R&D program on third–generation mobile communication systems. *IEEE Personal Communications*, 4(1), February 1997.

[20] Barry M. Leiner, Robert J. Ruth, and Ambatipudi R. Sastry. Golas and challenges of the DARPA GloMo program. *IEEE Personal Communications*, 3(6), December 1996.

[21] Otto Spaniol, Andreas Fasbender, Simon Hoff, Josef Kaltwasser, and Jurgen Kassubek. Impacts on mobility on telecommunication and data communication networks. *IEEE Personal Communications*, 5(2), October 1995.

[22] David J. Frank. Application and technology forecast. In Wolfgang Nebel and Jean Mermet, editors, *Low Power Design in Deep Submicron Electronics*, chapter 2, pages 9–44. Kluwer Academic Publishers, The Netherlands, 1997.

[23] Noah Davis. Personal digital assistants: Part 1. *Computer*, 29(9), September 1996.

[24] Noah Davis. Personal digital assistants: Part 2. *Computer*, 29(11), November 1996.

[25] Erik P., Steven W. Depp, William E. Pence, Scott Kirkpatrick, M. Sri-Jayantha, and Ronald R. Troutman. Technology directions for portablr computers. *Proceedings of the IEEE*, 83(4), 1995.

[26] R.M. Needham. Computers and communications. In Robin Milner, editor, *Computing Tomorrow. Future Research Direction in Computer Science*, chapter 14, pages 284–294. Cambridge University Press, Great Britain, 1996.

[27] Marc Rotenberg. Communications privacy: Implications for network design. *Communications of the ACM*, 36(8), August 1993.

[28] Frank M. Tuerkheimer. The underpinnings of privacy protection. *Communications of the ACM*, 36(8), August 1993.

[29] Jeff Smith. Privacy policies and practices: Inside the organizational maze H. *Communications of the ACM*, 36(12), December 1993.

[30] Gerald Kovacich. Electronic–internet business and security. *Computers & Security*, 17(2), 1998.

[31] The European Commission of the European Union. Electronic commerce —an introduction. http://www.ispo.cec.be/ecommerce/introduc.htm, July 1998.

[32] Charles Cresson Wood. A management view of Internet electronic commerce security. *Computers & Security*, 16(4), 1997.

[33] Charles Petrie. The edge of e–cash. *IEEE Internet Computing*, 1(6), 1997.

[34] David Chaum. How much do you trust big brothers. *IEEE Internet Computing*, 1(6), 1997.

[35] Anish Bhimani. Securing the commercial Internet. *Communications of the ACM*, 39(6), June 1996.

[36] Philip R. Zimmermann. *The Official PGP User's Guide.* The MIT Press, USA, 1995.

[37] Philip Zimmermann. Pgp user's guide, Volume I: Essential topics. http://www.cl.cam.ac.uk/PGP/#pks, October 1994.

[38] Andre Bacard. Anonymous remailers. http://www.well.com/user/abacard/remail.html, November 1996.

[39] Inc. Anonymizer. Anonymizer. http://www.anonymizer.com/main.html, April 1998.

[40] Lucent Technologies. The lucent personalized web assistant. http://www.bell-labs.com/project/lpwa/index.html, April 1998.

[41] Mike Reiter and Avi Rubin. Crowds: Anonymity loves company. http://www.research.att.com/projects/crowds/, April 1998.

[42] TRUSTe. TRUST home page. http://www.truste.org/, July 1998.

[43] W3C. About the world wide web consortium. http://www.w3.org/Consortium/, July 1998.

[44] John F. Shoch. Inter–network naming, addressing, and routing. In *Proceedings. Computer Communications Networks. Compcon78. September 5–8, 1978.* IEEE Computer Society, 1978.

[45] Zaw-Sing Su. Identification in computer networks. In *Proceedings of the 8th Data Communications Symposium*, N. Falmouth, Massachusetts, October 1983. ACM SIGCOMM Computer Communication Review, Vol. 13, No. 4.

[46] John Ioannidis, Dan Duchamp, and Jr. Gerald Q. Maguire. IP-based protocols for mobile internetworking. In *SIGCOM'91 Conference. Communications Architecture and Protocols*, Zurich, Swizerland, September 3-6 1991. ACM.

[47] R.G. Toulson and C.M. Phipps. *Confidentiality.* Sweet & Maxwell, Great Britain, 1996.

[48] Colin Munro. Confidence in government. In Linda Clarke, editor, *Confidentiality and the law*, chapter 1, pages 1–21. Lloyd's of London press LTD., Great Britain, 1990.

[49] System Security Study Committee, Computer Science and Telecommunication Board, Commission on Physical Sciences, Mathematical, and Applications, and National Research Council. *Computer at Risk.* National Academic Press, Washington D.C., U.S.A., 1996.

[50] William Stallings. Simple network management protocol. In Morris Sloman, editor, *Network and Distributed System Management*, chapter 7, pages 165–196. Addison–Wesley Publishing Company, Great Britain, 1994.

[51] Arnoud Galactus Engelfriet. Anonymity and privacy on the Internet. URL:http://www.stack.nl/ galactus/remailers/index.html, December 1996.

[52] Sara Baase. *A Gift of Fire: Social, Legal and Ethical Issues in Computing.* Prentice Hall, 1987.

[53] Simson Garfinkel with Gene Spafford. Cryptography and the web. *World Wide Web Journal*, 2(3), Summer 1997.

[54] Sesshadri Mohan and Ravi Jain. Two user location strategies for personal communication services. *IEEE Personal Communications*, 1(1), First Quarter 1994.

[55] Jay E. Padgett, Christoph G. Gunther, and Takeshi Hattori. Overview of wireless personal communications. *IEEE Communications Magazine*, 33(1), January 1995.

[56] Donald C. Cox. Wireless personal communications: What is it? *IEEE Personal Communications*, 2(2), April 1995.

[57] Raymond Steele. The evolution of personal communications. *IEEE Personal Communications*, 1(2), Second Quarter 1994.

[58] A. Schill and S. Kummel. Design and implementation of a support platform for distributed mobile computing. *Distributed Systems Engineering*, 2(3), September 1995.

[59] Stanley Chia. The universal mobile telecommunication system. *IEEE Communications Magazine*, 30(12), December 1992.

[60] Moe Rahnema. Overview of the GSM system and protocol architecture. *IEEE Communications Magazine*, 31(4), April 1993.

[61] Andrew S. Tanenbaum. *Computer Networks.* Prentice–Hall, Inc., third edition, 1996.

[62] Fred Halsall. *Data Communications, Computer Networks and Open Systems.* Addison–Wesley Publishing Company, USA, fourth edition, 1996.

[63] A. Robin Potter. Implementation of PCNs using DCS1800. *IEEE Communications Magazine*, 30(12), December 1992.

[64] Barry Varley. User administration and accounting. In Morris Sloman, editor, *Network and Distributed System Management*, chapter 14, pages 381–402. Addison–Wesley Publishing Company, Great Britain, 1994.

[65] Michael H. Callendar. Future public land mobile telecommunication systems. *IEEE Personal Communications*, Fourth Quarter 1994.

[66] Kenneth C. Budka, Hong Jiang, and Steven E. Sommars. Cellular digital packet data networks. *Bell Labs Technical Journal*, 2(3), Summer 1997.

[67] Yair Frankel, Amir Herzberg, paul A.K̃arger, Hugo Krawcyk, Charles A. Kunzinger, and Moti Yung. Security issues in a CDPD wireless network. *IEEE Personal Communications*, 2(4), August 1995.

[68] Apostolis K. Salkintzis and Christodoulos Chamzas. Mobile packet data technology: An insight into MOBITEX architecture. *IEEE Personal Communications*, 4(1), February 1997.

[69] Michael Miller. Introduction to satellite communications. In Michael J. Miller, Branka Vucetic, and Less Berry, editors, *Satellite Communications*, chapter 1, pages 1–56. Kluwer Academic Publisher, Norwell Massachusetts USA, 1993.

[70] Michael Miller. Mobile satellite system design. In Michael J. Miller, Branka Vucetic, and Less Berry, editors, *Satellite Communications*, chapter 3, pages 103–143. Kluwer Academic Publisher, Norwell Massachusetts USA, 1993.

[71] Ana Luz Ruelas. *México y Estados Unidos en la Revolución Mundial de las Telecomunicaciones.* UNAM, México, 1995. Disponible en: http://www.lanic.utexas.edu/la/Mexico/telecom/.

[72] G. Maral. *VSAT Networks.* John Willey & Sons, Great Britain, 1995.

[73] Gary Comparetto and Rafols Ramirez. Trends in mobile satellite technology. *Computer*, 30(2), February 1997.

[74] Lee Goldberg. The Internet in space: Problems and solutions. *Computer*, 30(2), February 1997.

[75] Willian Stallings. *Data and Computer Communications.* Macmillan Publishing Company, USA, fourth edition, 1994.

[76] Zoran M. Markovic. Satellites in non-geostationary orbits. In Michael J. Miller, Branka Vucetic, and Less Berry, editors, *Satellite Communications*, chapter 9, pages 345–391. Kluwer Academic Publisher, Norwell Massachusetts USA, 1993.

[77] Peter Dondl. Standardization of the satellite component of the UMTS. *IEEE Personal Communications*, 2(5), October 1995.

[78] Udo Flohr. Electric money. *BYTE*, 21(6), June 1996.

[79] ComLinks.com. Iridium. http://www.comlinks.com/sys/iridium.htm, 1999.

[80] Motorola. Iridium global satellite communications. http://www.satphone.net/iridmain.htm, 2000.

[81] Hans De Boer. RACE mobile communications. *Electronic & Communication Engineering Journal*, 5(3), June 1993.

[82] Fumio Teraoka, Yasuhiko Yokote, and Mario Tokoro. A network architecture providing host migration transparency. In *SIGCOM'91 Conference. Communications Architecture and Protocols*, Zurich, Swizerland, September 3-6 1991. ACM.

[83] Charles Perkins. Providing continuous network access to mobile hosts using TCP/IP. *Computer Networks and ISDN Systems*, 26(3), 1993.

[84] Hiromi Wada, Takashi Yozawa, Tatsuya Ohnishi, and Yasunori Tanaka. Mobile computing environment based on internet packet forwarding. In *Proceeding of Winter Usenix*, San Diego CA, January 25-29 1993. USENIX Association.

[85] Gihwan Cho and Lindsay Marshall. An efficient location and routing scheme for mobile computing environments. *IEEE Journal on Selected Areas in Communications*, 13(5), June 1995.

[86] Baruch Awerbuch and David Peleg. Concurrent online tracking of mobile users. *Computer Communications Review*, 21(4), September 1991.

[87] Andreas Fasbender, Frank Reichert, Eckhard Geulen, Johan Hjelm, and Thomas Wierlemann. Any network, any terminal, anywhere. *IEEE Personal Communications*, 6(2), April 1999.

[88] WAP Forum. Wap forum specifications. http://www.wapforum.org/what/technical.htm, 2000.

[89] WAP Forum. Wap forum specifications: Wireless application protocol architecture specification. version 30–apr—1998. http://www.wapforum.org/what/technical.htm, 2000.

[90] WAP Forum. Wap forum specifications: Wireless application protocol wireless application environment overview. version 04–nov—1999. http://www.wapforum.org/what/technical.htm, 2000.

[91] Anne C. Lear. Palmos gains upper hand in handheld market. *Computer*, 33(1), January 2000.

[92] George Lawton. Vendors battle over Mobile–OS market. *Computer*, 32(2), February 1999.

[93] John Bates. The state of the art in distributed and dependable computing. A cabernet–sponsored report, University of Cambridge, UK, October 1998. Also available at http://www.newcastle.research.ec.org/cabernet/sota/index.html.

[94] Michael Wu and Willy Zwaenepoel. eNVy: A non–volatile, main memory storage system. *ACM SIG PLAN NOTICES*, 29(11), November 1994. ASPLOS–VI Proceedings Sixth International Conference on Architectural Support for programming languages and Operating Systems.

[95] Robert A. Powers. Batteries for low power electronics. *Proceedings of the IEEE*, 83(4), 1995.

[96] Imielinski Tomasz and B. R. Badrinath. Wireless computing challenges in data management. *Communications of the ACM*, 37(10), October 1994.

[97] Wolfgang Nebel. Introduction. In Wolfgang Nebel and Jean Mermet, editors, *Low Power Design in Deep Submicron Electronics*, chapter Introduction, pages 1–8. Kluwer Academic Publishers, The Netherlands, 1997.

[98] Kaushik Roy and Mark C. Johnson. Software design for lower power. In Wolfgang Nebel and Jean Mermet, editors, *Low Power Design in Deep Submicron Electronics*, chapter 6.3, pages 533–460. Kluwer Academic Publishers, The Netherlands, 1997.

[99] Tim Lich and Jeff Slaton. Strongarming portable communications. *IEEE Micro*, 18(2), March–April 1998.

[100] Luc Claesen, Hans de Kuyper, and Ronny Tits. Lower power applications at system level. In Wolfgang Nebel and Jean Mermet, editors, *Low Power Design in Deep Submicron Electronics*, chapter 2, pages 9–43. Kluwer Academic Publishers, The Netherlands, 1997.

[101] Norman Adams, Rich Gold, Bill N. Schilit, and Michael M. Tso. An infrared network for mobile computers. In *Mobile and Location–Independent Computing Symposium*, Cambridge Massachusetts, August 2–3 1993. Usenix Association.

[102] Francisco J. López-Hernández and A. Santamaría. Wireless LANs: An overview. In A. Santamaría and F.J. López-Hernádez, editors, *Wireless LAN Systems*, chapter 1, pages 1–21. Artech House, 1994.

[103] Joseph M. Kahn, John R. Barry, Malik D. Audeh, Jeffrey B. Carruthers, William J. Krause, and Gene W. Mars. Non–directed infrared links for high–capacity wireless LANs. *IEEE Personal Communications*, 1(2), Second Quarter 1994.

[104] Departmental. Radio LANs. *LAN Magazine*, March 1996.

[105] Kwang-Cheng Chen. Medium access control of wireless LANs for mobile computing. *IEEE Network*, 8(5), September-October 1994.

[106] Harshal S. Chhaya and Sanajy Gupta. Performance of asynchronous data transfer methods of IEEE 802.11 MAC protocol. *IEEE Personal Communications*, 3(5), October 1996.

[107] Bernard Bourin. High performance radio mobility in LANs. In Mark Harrington, editor, *European Telecommunications Standardization and the Information Society.* Atalink Ltd on behalf of the ETSI, 1995. Available at
http://www.etsi.fr:80/ecs/reports/stateart/bourin.htm.

[108] Roy Rada, Carl Cargill, and John Klensin. Consensus and the web. *Communications of the ACM*, 41(7), July 1998.

[109] TRUSTe. Who is TRUSTe.
http://www.truste.org/webpublishers/aboutus.html, July 1998.

[110] Lorrie Cranor. Platform for privacy preferences (p3p) project.
http://www.w3.org/P3P/, 2000.

[111] Christine Varney. Talks about advancing commerce and protecting consumers. *World Wide Web Journal*, 2(3), Summer 1997.

[112] A. Luotonen and K. Altis. World wide web proxies. *Computer Networks and ISDN Systems*, 27:147–154, November 1994.

[113] Inc. Anonymizer. Use anonymizer for free.
http://www.anonymizer.com/surf_free.html, April 1998.

[114] Inc. Anonymizer. Access your paid account.
http://www.anonymizer.com/surf_access.html, April 1998.

[115] Inc. Anonymizer. Send anonymous e-mail.
http://www.anonymizer.com/email/remailer-simple.cgi, April 1998.

[116] Inc. Anonymizer. Anonymous web publishing.
http://www.anonymizer.com/publishing.html, April 1998.

[117] Michael K. Reiter and Aviel D. Rubin. Crowds: Anonymity for web transactions. Technical Report DIMACS Technical Report 97–15, AT&T Labs—Research, Murray Hill, New Jersey, USA, August 1997. Available at http://www.research.att.com/projects/crowds/.

[118] Bruce Schneier. *Applied Cryptography.* John Wiley & Sons, Inc., second edition, 1996.

[119] Roger M. Needham. Cryptography and secure channels. In Sape Mullender, editor, *Distributed Systems*, chapter 20, pages 531–541. Addison–Wesley, acm PRESS, second edition, 1993.

[120] Carl H. Mayer and Stephen M. Matyas. *Cryptography: A new Dimension in Computer Data Security.* John Wiley & Sons, 1982.

[121] Electronic Frontier Foundation. *Cracking DES. Secrets of Encryption Research, Wiretap Politics & Chip Design.* O'Really, 1998.

[122] Whitfield Diffie and Martin E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, IT–22(6), November 1976.

[123] R.L̃. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public–key cryptosystems. *Communications of the ACM*, 21(2), February 1978.

[124] Dorothy Elizabeth Robling Denning. *Cryptography and Data Security.* Addison–Wesley Publishing Company, 1982.

[125] Solovay R. and Strassen V. A fast monte–carlo test for primality. *SIAM Journal on Computing*, 6, March 1977.

[126] David Chaum. Security without identification: Transaction systems to make big brother obsolete. *Communications of the ACM*, 28(10), October 1985.

[127] Daniel C. Lynch and Leslie Lundquist. *Digital Money.* John Willey & Sons, Inc., 1996.

[128] Thierry Moreau. A probabilistic flaw in PGP design? *Computing & Security*, 15(1), 1996.

[129] *Communications of the ACM*, 36(3), March 1993.

[130] *Communications of the ACM*, 37(11), November 1994.

[131] Clint N. Smith. Government promotion of key recovery encryption. *World Wide Web Journal*, 2(3), Summer 1997.

[132] Hal Abelson, Ross Anderson, Steven M. Bellovin, Josh Benaloh, Matt Blaze, Whitfield Diffie, John Gilmore, Peter G. Neumann, Ronald L. Rivest, Jeffrey I. Schiller, and Bruce Schneier. The risks of key recovery, key escrow, and trusted third-party encryption. *World Wide Web Journal*, 2(3), Summer 1997.

[133] John Browning. The netizen: I encrypt, therefore I am. *WIRED*, 5(11), November 1997.

[134] Janet Reno (The United Stated Attorney General). Law enforcement in cyberspace address. In Dorothy E. Denning and Peter J. Denning, editors, *Internet besieged. Countering Cyberspace Scofflaws*, chapter 27, pages 439–447. Addison–Wesley, 1998. Speech presented to the Commonwealth Club of california, Jun 1996.

[135] Bruce Sterling. Remarks at computers, freedom, and privacy conference IV, Chicago. In Dorothy E. Denning and Peter J. Denning, editors, *Internet besieged. Countering Cyberspace Scofflaws*, chapter 29, pages 475–480. Addison–Wesley, 1998.

[136] Brian A. LaMacchia. Bal's PGP public key server. http://pgp5.ai.mit.edu/, July 1999.

[137] Simson Garfinkel and Gene Spafford. *Web Security & Commerce.* O'Reilly & Associates, Inc., 1997.

[138] Stephen T. Kent. Internet privacy enhanced mail. In Dorothy E. Denning and Peter J. Denning, editors, *Internet besieged. Countering Cyberspace Scofflaws*, chapter 19, pages 295–318. Addison–Wesley, 1998.

[139] Stephen T. Kent. Internet privacy enhanced mail. *Communications of the ACM*, 36(8), August 1993.

[140] Aviel D. Rubin and Daniel E.G̃eer Jr. A survey of web security. *Computer*, 31(9), September 1998.

[141] Frederick J. Hirsch. Introducing SSL and certificates using SSLeay. *World Wide Web Journal*, 2(3), Summer 1997.

[142] William Stallings. *Network and Internetwork Security Principles and Practice.* Prentice Hall, 1995.

[143] Dorothy E. Denning. International encryption policy. In Ravi Kalakota and Andrew B. Whinston, editors, *Reading in Electronic Commerce*, chapter 9, pages 105–118. Addison–Wesley, 1997.

[144] Dorothy E. Denning and Dennis K. Branstad. A taxonomy for key escrow encryption systems. *Communications of the ACM*, 39(3), March 1996.

[145] Stephen T. Walkers, Steven B. Lipner, Carl M. Ellison, and David M. Balenson. Commercial key recovery. *Communications of the ACM*, 39(3), March 1996.

[146] David Paul Maher. Cryptobackup and key escrow. *Communications of the ACM*, 39(3), March 1996.

[147] Ravi Ganesan. The yaksha security system. *Communications of the ACM*, 39(3), March 1996.

[148] Per Christoffersson, editor. *Cryto User's Handbook. A guide for Implementors of Cryptographic Protection In Computer Systems.* North–Holland, The Netherland, 1988.

[149] L. Jean Camp, Marvin Sirbu, and J.D. Tygar. Token and national money in electronic commerce. In *Conference Proceedings: The First USENIX Workshop on Electronic Commerce*, New York, New York, July 11-12, 1995. USENIX Association.

[150] N. Asokan, Phillipe A. Janson, Michael Steiner, and Michael Waidner. The state of the art in electronic payment systems. *Computer*, 30(9), September 1997.

[151] Digicash Company. Digicash web page. http://www.digicash.com/, 1997.

[152] B.C̃lifford Neuman. A flexible framework for network payment. In Ravi Kalakota and Andrew B. Whinston, editors, *Reading in Electronic Commerce*, chapter 9, pages 229–243. Addison–Wesley, 1997.

[153] Digicash Publications (Company information). Digicash–numbers that are money. http://www.digicash.com/publish/digibro.html, 1996.

[154] Douglas E. Comer. *Internetworking with TCP/IP. Principles, Protocols, and Architecture*, volume 1. PRENTICE HALL, third edition, 1995.

[155] Carlos Molina-Jiménez and Lindsay Marshall. Anonymous and confidential communications from an IP addressless computer. In *Handheld and Ubiquitous Computing, Proceedings of the First International Symposium, HUC'99*, Karlsruhe, Germany, Sep 1999. Springer. Lecture Notes in Computer Science, 1707.

[156] David Naccache and David M'Raihi. Cryptographic smart cards. *IEEE Micro*, 16(3), June 1996.

[157] José Luis Zoreda and José Manuel Otón. *Smart Cards.* Artech House, 1994.

[158] Mahdi Abdelguerfi, Burton S. Kaliski Jr, and Wayne Patterson. Public–key security systems. *IEEE Micro*, 16(3), June 1996.

[159] Gerard J. Holzmann. *Design and Validation of Computer Protocols.* Prentice Hall, 1991.

[160] Raouf Boutaba, Karim El Guemhioui, and Petre Dini. An outlook on intranet management. *IEEE Communications magazine*, 35(10), October 1996.

[161] Jost Weinmiller, Morten Schläger, Andreas Festag, and Adam Wolisz. Performance study of access control in wireless LAN–IEEE DFWMAC and ETSI 10 hiperland. *Mobile Networks and Applications*, 2(1), June 1997.

[162] Gerard J. Holzmann. The model checker Spin. *IEEE Transactions on Software Engineering*, 23(5), May 1997. also available at: http://netlib.bell-labs.com/netlib/spin/whatispin.html.

[163] Edsger W. Dijkstra. Guarded commands, nodeterminacy and formal derivation of programs. *Communications of the ACM*, 18(8), August 1975.

[164] C.A.R. Hoare. Communicating sequential processes. *Communications of the ACM*, 21(8), August 1978.

[165] Samuel J. Leffler, Marshall Kirk McKusick, Michael J. Karels, and John S. Quaterman. *The Design and Implementation of the 4.3BSD Unix Operating System.* Addison–Wesley Publishing Company, United States of America, 1989.

[166] Bell labs: Spin Web Home Page. On-the-fly, LTL model checking with spin. http://netlib.bell-labs.com/netlib/spin/whatispin.html, June 1999.

[167] Bell labs: Spin Web Home Page. What's new in spin versions 2.0 and 3.0 (updated for version 3.0, july 1997). http://cm.bell-labs.com/cm/cs/what/spin/Man/WhatsNew.html, August 1997.

[168] Gerard J. Holzmann. An analysis of bitstate hashing. In *Proc. IFIP/PSTV95, Conf. on Protocol Specification, Testing, and Verification*, Warsaw, Poland, June 13–66 1995.

[169] M. J.R̃. Shave. *Data Structures*. McGraw–Hill, Inc., 1975.

[170] Herman A. Maurer. *Data Structures and Programming Techniques*. Prentice Hall, Inc., 1997.

[171] Lawrence H. Miller. *Advanced Programming. Design and Structure Using Pascal*. Addison Wesley Publishing Company, 1986.

[172] P.M. Merlin. Specification and validation of protocols. *IEEE Transactions on Communications*, 27(11), November 1979.

[173] Leslie Lamport. Proving the correctness of multiprocess programs. *IEEE Transactions on Software Engineering*, SE–3(2), March 1977.

[174] Leslie Lamport and Fred B. Schneider. Formal foundation for specification and verification. In *Distributed Systems. Methods and Tools for Specification. An Advanced Course*, Lecture Notes in Computer Science, Volume 190. Springer–Verlag, 1985.

[175] William E. Weihl. Specifications of concurrent and distributed systems. In Sape Mullender, editor, *Distributed Systems*, chapter 3, pages 27–53. Addison–Wesley, acm PRESS, second edition, 1993.

[176] Felix C. Gartner. Fundamentals of fault–tolerant distributed computing in asynchronous environments. *ACM Computing Surveys*, 31(1), March 99.

[177] Bell labs: Spin Web Home Page. Spin readme: Downloading and installation unix systems. http://ftp.cit.gu.edu.au/ shiv/#S2, October 1999.

[178] Gerard J. Holzmann. An improved protocol reachability analysis technique. *IEEE Transactions on Software–Practice and Experience*, 18(2), May 1998.

[179] M. Ben–Ari. *Principles of Concurrent and Distributed Programming*. Prentice–Hall, Inc., 1990.

[180] PROM97. Spin version 3.0: Language reference. http://cs.uwp.edu/staff/fossum/Classes/CS477/spin/HTML/promela.html, December 1997.

[181] Donald A. Norman. *The Invisible Computer*. The MIT Press, 1998.

[182] Bruce Schneier. Cryptographic design vulnerabilities. *Computer*, 31(9), September 1998.

[183] J.D. Tygar. Atomicity in electronic commerce. In Dorothy E. Denning and Peter J. Denning, editors, *Internet besieged. Countering Cyberspace Scofflaws*, chapter 23, pages 389–405. Addison–Wesley, 1998.

[184] Adi Shamir. How to share a secret. *Communications of the ACM*, 22(11), November 1979.

[185] Wen-Shenq Juang and Chin-Laung Lei. A collision–free secret ballot protocol for computerized general elections. *Computer & Security*, 15(4), 1996.

[186] Joan Borrel and Josep Rifá. An implementable secure voting scheme. *Computer & Security*, 15(4), 1996.

[187] David L. Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Communications of the ACM*, 24(2), 1981.

[188] Simon Singh. *The Code Book: The Science of Secrecy from Ancient Egypt to Quantum Cryptography.* Fourth Estate, 1999.

[189] Andrew M. Steane and Eleanor G. Rieffel. Beyond bits: The future of quantum information processing. *Computer*, 33(1), January 2000.