Security in signalling and digital signatures

Technische
Universiteit
Eindhoven

Opleiding Wiskunde voor de Industrie Eindhoven

STUDENT REPORT 93-05

SECURITY IN SIGNALLING & DIGITAL SIGNATURES

Sandra Roijakkers

ECMI

Den Dolech 2
Postbus 513
5600 MB Eindhoven

March 1993

WISKUNDE VOOR DE INDUSTRIE
MATHEMATICS FOR INDUSTRY

SECURITY IN SIGNALLING & DIGITAL SIGNATURES

(extended version)

Mrs.Ir S.A.W. Roijakkers

Part I: General information
Part II: Security in signalling
Part III: Digital signatures

University supervisor: Prof.Dr Ir H.C.A. van Tilborg,
Technische Universiteit, Eindhoven
Industrial supervisor: Dr M. De Soete,
N.V. PITS S.A., Brussels

March 1993

Part I
General Information

Contents

1 MBLE/PITS **2**
- 1.1 History . . . 2
- 1.2 Activities of MBLE . . . 3

2 Security in Signalling **4**
- 2.1 Introduction . . . 4
- 2.2 Projects of the CEC . . . 4
- 2.3 The RACE program . . . 5
- 2.4 R1022: Technology for ATD . . . 6
- 2.5 Approach . . . 7

3 Digital Signatures **8**
- 3.1 Introduction . . . 8
- 3.2 Elucidation . . . 8
- 3.3 Approach . . . 9

Chapter 1

MBLE/PITS

1.1 History

On March 18, 1911, in the place Vilvoorde, close to Brussels, 40 people founded the partnership "Lampes Brabant". The word "lampes" is French for bulbs, and Brabant is the province of Belgium where Vilvoorde is situated, and it is not difficult to guess that the company produced and sold bulbs. Two years later the company moved to Anderlecht, the district in the south of Brussels where the company is still located, although on a different address. In 1915 the partnership got the new name "Manufacture Belge de Lampes Electrique" (MBLE), which means, loosely translated, Belgium production of (electronic) bulbs.

The year 1924 was an important year of the firm, because then its first electronics project started, namely the production of vacuum tubes. In 1948 another big step followed: the development and production of professional machines. One year later the head-office moved to the Tweestationstraat, or "Rue de les deux gares" in French. The company has now 430 employees. The original product, the bulbes, were becoming less and less important for the company, and therefore in 1951 the name was changed into "Manufacture Belge de Lampes and de Matriel Electronique". The abbreviated firm name, however, did not change accordingly and remained MBLE.

In 1953 a factory in Evere was opened, and in 1954 the production of semi-conductors started. This grew to one of MBLE's main specialisation fields. In the next six years two factories and a stock were opened in Brussels, and also another office and "MBLE International". The small firm expanded to a big company, and had 5200 employees in 1965.

From then on business went down. Some factories closed, and other ones were taken over by Philips. In fact, Philips is the one shareholder of MBLE's interests. When I started my project, MBLE consisted of only 35 employees, working in the building at the Tweestationstraat, and business was not going too well. But at the end of the year, they "took over" the feeding-producing company N.V. Philips Industrial Activities (PIA) in Wavre form Philips, which was about six times as big and made profit. They continue together as N.V. Philips Industrial and Telecommunication Systems (PITS). The reasons for making this strange manoeuvre are all political. For example, it is interesting for Philips that MBLE remains a Belgium firm, to get orders from the government and subsidies.

1.2 Activities of MBLE

The activities from MBLE cover(ed) various aspects:

- electricity / electronica
  - hybrid integrated switches
  - professional printed circuits
  - non-linear resistances
  - feedings
- automatisation
  - advisory tasks for product development, with as goal the automatisation of production
  - the specification, design and realisation of specific production machines
- Precision mechanica and plating
- Cryptography

Since the activities which are related to cryptography have my main interest, I will spend a few more words on them.

MBLE offers the Belgium market a series of crypto devices for X.25 and fax communication traffic. These applications are part of an efficient security management of the data transferred via teletransmission.

The basic Philips Crypto PPSX2060 guarantees a full duplex end-to-end cryptographic protection of the X.25 packet switch network. All versions use an advanced Philips key management system where each user owns a personal smart card with a PIN code protection.
The Philips Crypto PDFX2035 series protect all data used in facsimile messages.

Chapter 2

Security in Signalling

2.1 Introduction

The first seven weeks of the time I worked at MBLE I spent on research into the placement of security functions in the signalling of a broadband network. This is part of Project 1022 of the RACE Program of the CEC, so it is natural to start with the objectives of the RACE Program in general, followed by the ones of Project 1022. Although it is not necessary to understand my work, I think it is interesting to spent some time on the reasons why the Community sponsors projects like RACE, and what other kinds of projects it sponsors.

I will start in the next section with a short picture of the attitude and politics of the CEC towards the sponsoring of research projects. Afterwards the RACE project will be described. In section 4, I will focus on one of the projects of RACE, namely R1022, and I will narrow the view even further by considering the work that one of the subgroups of R1022 has to do. In the last section of this chapter I will describe how I carried out my task in the work MBLE had to do within this subgroup. The result of this work, that was sent as a chapter in a deliverable to the CEC, can be found in Part II.

2.2 Projects of the CEC

In modern economy, production and innovation cycles grow shorter and shorter. A consequence is that the costs of research and development become very high. In fact, these costs are often decisive in the international competition. Since one of the aims of the CEC is to maintain and consolitate the position of Europe on the world market, it is logical that high tech research will be stimulated.

Of course, the Community does not just want to be another source to get grants from. It aims especially at

- coordination and co-operation across borders, as well as mobility between the worlds of industry and science;
- stimulation of basic research, which is very important nowadays, but for which middle sized and small companies often lack the necessary means;
- integration of research and technology in accordance with the completion of the European internal market, which holds in the first place for normalisation and standardisation.

To achieve these goals, a framework program has been drawn up for a period of four years: from 1990 till 1994. This overlaps the previous program (from 1987 till 1991) by two years, which is done to make the whole continuously.

The program consists of six main areas, which come under three headings:

- Enabling technologies
  - Information and communications technologies
  - Industrial and materials technologies
- Management of natural resources
  - Environment
  - Life sciences and technologies
  - Energy
- Management of intellectual resources
  - Human capital and mobility

A project that will be sponsored by the Community has to be defined within one of these areas.

In total, the CEC will spent 5700 million ecu in those four years, which is equivalent to approximately 13300 million Dutch florins.

To make the European internal market a success, it is important that all aspects of science and technology are covered by the policy of the Community. This is the reason that besides the framework program shown above, also support is given to European schooling and updating courses, and to the circulation and exploitation of research results. The ERASMUS program stimulating the exchange of students, and the set-up of international study programs between universities of the member states is well known to most students. The two other big programs in this area are COMETT, stimulating co-operating between universities and industry by for example international stages, and LINGUA which promotes the education in foreign languages in the Community.

The course "Mathematics for Industry" is the Dutch part of a European course organized by the European Consortium for Mathematics (ECMI), and within ECMI exchange of course members and teachers takes place. The ECMI project is sponsored by ERASMUS and COMETT.

2.3 The RACE program

The term RACE is the abbreviation for *Research and development in Advanced Communications technologies for Europe*. After a pilot phase started in 1985, the first projects got under way in January 1988. In the next period (from 1990 till 1994) the program was continued.

The aim of RACE is to promote a precompetitive R&D to set up an Integrated Broadband Communications Network (IBCN) by 1995-2000. The IBCN is to take over from ISDN (Integrated Services Digital Network), which does not have a broadband width and is therefore unsuitable for the transmission of large flows of data. Broadband communications on the contrary enable large quantities of data to be transmitted at high speed.

The purpose of Integrated Broadband Communications (IBC) is to have a single network of terminals, cables, node processors, computers and satellites with a very high transmission rate. Such a network will provide integrated distribution of

- traditional services like telephone and telex;
- new services, e.g. colour facsimile, high-quality videotex and email;
- conventional or interactive television programs;
- the ultrafast transmission of computer data;
- videoconferencing;
- value-added services, e.g. financial services and electronic data interchange;
- etc.

This integration has the great advantage of avoiding the proliferation of incompatible networks. Integrated broadband communications will therefore not only increase considerabily the ability of private and professional users to exchange data and communicate, but also offer them a wide range of advanced telecommunications services.
Their main effect will be to improve the competiveness of many economic sectors. For example, data exchange between design offices and factories will speed up production cycles. Integrated broadband communications will also directly affect society in a number of ways. In particular, they will permit the decentralisation of economic activities, the opening-up of rural and peripheral regions and the development of teleworking. Thus broadband communications are the essential infrastructure of an information-based economy, and they are at the very heart of the communication revolution.
Apart from its main goal of introducing integrated broadband communications into the European community in 1995, RACE is pursuing several objectives:

- promoting the Community's telecommunications industry;
- developing the competitiveness of European network users;
- creating a single European market in IBC equipment and services;
- developing the poorest regions of the Community, which will thereby be able to benefit fully from advanced telecommunications.

Virtually all the main parties involved in European research and development in the telecommunications sector are participating in the RACE program: national authorities, equipment manufacturers, network operators, information technology industries, universities, research centra, etc. This, for an CEC project rather exceptional, situation makes it possible to take full advantage of the vast intellectual, scientific and technical potentials available in Europe. It also reduces the risks of failure, RACE can act as a catalyst in the key sectors of technological development. Another advantage is that it speeds up the standardisation process, which is a well-known bottleneck in the exploitation of high technology.

2.4 R1022: Technology for ATD

Project 1022 of RACE, or shortly R1022, is titled "Technology for Asynchronous Time Division (ATD)". An asynchronous transmission in data communications is a form of data transmission in which there can be variable time intervals between characters, but the bits within a character are sent with fixed time intervals. Start and stop elements are used to indicate the

beginning and end of characters. As contrasted with synchronous transmission, where each bit is transmitted according to a given time sequence, receiver and sender do not have to maintain exact synchronisation over an extended time period.

The basic assumption of the project is that Asynchronous Transfer Mode (ATM) is the unique transfer mode for all IBCN services. R1022's task is to realise a pre-normative functional integration and deliver a complete demonstrator of the transport principles of the target ATM-based IBCN.

The work that has to be done to achieve this complex goal is divided among several Task Groups (TGs). The first of them, TG1, has the task to investigate all signalling aspects, and produce a deliverable "The state of the art". Signalling messages are messages that are sent separately from ordinary data, and can be used for various call control and connection control functions.

The group consists of seven partners, coming from Belgium, Denmark, Switzerland and the United Kingdom. One of the partners from Belgium is MBLE, and MBLE was responsible for Chapter 5 of this deliverable, titled "Security aspects in signalling".

2.5 Approach

When I started at MBLE, a rough list of subjects that should be treated in the chapter was handed over to me. I also got information on where to find the various standards, and draft standards, on security related issues.

Since the intended readers of the deliverable were not familiar with security principles, in the first sections of the chapter these had to be explained as short and clear as possible. Also an overview was given of security services, and the security primitives that could achieve them. Information on all this could be found in text books on cryptography, articles and course publications. The standards were used to check if certain security services could be achieved using standardised protocols, and if this was the case this standard protocol was explained in some more detail.

In the later sections we had to examine which security services could and should be placed in signalling. For the services that we recommended to offer in signalling, the advantages and disadvantages of the previously sketched primitives were weighed against one another.

The text of the chapter can be found in Part II of this report. Since it is only one chapter out of a coherent deliverable, it is not completely self-contained. In particular, we assume that the reader is familiar with the seven layer OSI model, and the concept of call control and connection control.

Chapter 3

Digital Signatures

3.1 Introduction

The major time of the six months I was at MBLE I worked on various aspects of digital signatures. Before a practical digital signature scheme can be implemented, a lot of problems have to be solved, and a lot of choices have to be made. My task was to study open and Philips-confidential literature in order to gather (theoretical) solutions to the problems and compare their practical use. This resulted in a comprehensive document, which is appended in Part III.

As the document is aimed to be selfcontained, I will here only give a short introduction to the concept of digital signatures, and the reasons why they are useful (or even needed) in electronic communication.

3.2 Elucidation

Electronic data interchange is more and more replacing the traditional paper driven systems. It goes without saying that the security and legal aspects related to the electronic versions of documents should be at least as effective as for the written versions. The task of a paper document is to store information; writing or printing information on a piece of paper has for many centuries been the most convenient way of storing the information. The integrity of the contents could be assured by making it very difficult to make undetected changes to the paper document, and the most common way to do this is by appending a (handwritten) signature. Nowadays, storing information on paper is becoming obsolete. Vast amounts of data are stored in computers and transfered over computer networks, whereby the information is presented as a string of bits. To assure the integrity of these data, a digital analogue of the ordinary signature can be used: a digital signature.

A digital signature has the same task as an ordinary handwritten signature has, i.e. unambiguously identifying the signer. To achieve this, both need to have the following properties:
- everyone is able to verify a signature;
- no one can forge a signature;
- in case a conflict arises, a judge can decide whether or not a specific signature is authentic.

A major difference between an handwritten and a digital signature is that the latter cannot be a constant: all data in electronic communication is just a string of bits, so everyone would be able to forge a constant signature. Therefore a digital signature is a function of the document it signs which only one person can compute, but everyone can verify.

3.3 Approach

Before I started to look into detail to aspects of the signing and verification process, I made a rough sketch of the basis protocol that should be used (Chapter 1). As said before, a digital signature has to be some function on the bitstring representing a document. Functions that can be used are described and compared in the second chapter. Since documents can have any size, they have to be condensed before they can be used as input for the signature algorithm. A description and comparision of various condensing, or hash, functions is given in Chapter 3. The next chapter shows how the hash functions and the signature schemes can be combined. To store the secret information needed to compute a signature, smart cards will be used. More about smart cards ands their usefulness can be found in Chapter 5. Another important aspect of a secure signing and verification process is the management of the keys that will be used. Chapter 6 elaborates on this. Then I will have a look at the legal aspects of digital signatures. Finally, I will treat the relatively new concept of "zero-knowledge". Protocols based on this technique seem to be very promessing for future applications.

Bibliography

[CEC90a]
Commission of the European Communities, Directorate-General XIII-F, *Research and Development in Advanced Communications Technologies in Europe, RACE '90*, March 1990.

[CEC90b]
Commissie van de Europese Gemeenschappen, Directoraat-generaal XII *Wetenschappen, onderzoek en ontwikkeling, Stimulering van onderzoek en technologie door de EG, Vademecum voor genteresseerden*, Economica Verlag, Bonn, april 1990.

[CEC91]
Commission of the European Communities, Directorate-General XIII, *Telecommunications, Information Industries and Innovation, Information and communications technologies in Europe*, catalogue number CD-70-91-095-EN- C, Office for Official Publications of the European Communities, 1991.

Part II
Security in Signalling

Contents

1 Context and general principles **3**
- 1.1 Communication . . . 3
- 1.2 The role of signalling . . . 4
- 1.3 Security . . . 4

2 Security functions in signalling **7**

3 Applications of security functions in signalling **9**

4 Information security **11**
- 4.1 General security services . . . 11
- 4.2 Security primitives . . . 12
  - 4.2.1 Mathematical primitives . . . 12
  - 4.2.2 Physical primitives . . . 16
- 4.3 Authentication protocols . . . 19
- 4.4 Key management, registration and certification authorities . . . 23

5 Security models **26**

6 Security in signalling revisited **27**
- 6.1 Identification . . . 27
- 6.2 Authentication . . . 27
- 6.3 Access control . . . 28
- 6.4 Integrity . . . 28
- 6.5 Confidentiality including stuffing . . . 29
- 6.6 Non-repudiation . . . 29
- 6.7 Network management . . . 29
- 6.8 Conclusion . . . 29

7 Notations **38**

Abstract

In the signalling protocols defined for N-ISDN, no provision was made for security functions. Without this feature, the only way, for customers, to ensure the security of their communications, was to rely on the application layer. This is in strong contrast for instance with services like GSM or DECT, where security is part of the service. The goal of the B-ISDN is to be "the" network supporting all kind of services, and to be able of interacting with all non B-ISDN services. Now that a new generation of signalling protocols for B-ISDN is in preparation within CCITT, there exists an opportunity to improve the situation by making a provision for security functionalities at signalling level. In this document we investigate which security functions could be offered in signalling, if it is preferable to do this, and how the security services to be offered should be implemented.

Chapter 1

Context and general principles

Security is a very broad domain. We will only consider here those security aspects which are related to the communication in B-ISDN, where signalling can be involved. The aim of this first chapter is to specify more clearly the context we will consider.

1.1 Communication

One of the security problems an application has to deal with, is the securing of its information transfer on a telecommunication network. When a communication channel is used, there always exists a risk for a failure, an act of piracy, or for data falsification. Other communication impairments are possible, like connection interruption or data corruption, and have to be analyzed as well from the security point of view.

Different security services can be introduced in communication at specific layers, namely at the OSI-layers as specified, for example, in [T89]. All security services can be provided in the Application layer (layer 7), however, this is out of the scope of the present document.

The application can also make use of the underlying layers to ensure the security of its information transport. An example is given by the electronic mail with X.400. These communication protocols already include a range of possible security functions. Under the hypothesis that security functions are already present in the lower layers, the mechanisms used to fulfil the security requirements at the application level will appear to be quite different.

The underlying layers used by an application do not necessarily include security mechanisms. But when they are present there, they can substantially improve the security for the applications as a whole. To illustrate this, let us consider a network where a radio link is employed. With respect to security, it is obvious that precisely this segment is intrinsically a weak one. If special features (e.g. encryption) are provided to ensure the confidentiality of the transmission on this link, the security will be substantially improved on this segment. Hence, the general security at application level is automatically improved.

The security in a network can be inherent to this network, or it can be ensured by means of supplementary services or by external service providers. Regardless of the chosen possibility, we might expect that specific signalling will be used to guarantee the integrity of the network, this means integrity with respect to the network configuration, the availability and the chosen paths.

An analysis of the security functions which are incorporated in different communication protocols, would be an interesting basis for an investigation on the range of security functions which could be offered by the B-ISDN. Moreover, it could also be used to evaluate to which extent the signalling would be able to support these protocols.

1.2 The role of signalling

Among the security functions to be considered in the B-ISDN, some depend directly on signalling while others do not.

In B-ISDN the data channel is separated from the signalling channel, in a user plane and in a control plane respectively. As a consequence, security functions directly performed on data, such as confidentiality, which have to be implemented in the user plane, cannot be signalling functions. It is not to say that during the call establishment the signalling will not play a role in negotiating the kind of confidentiality to be used. The signalling functions could ensure the exchange of key codes to be used for confidentiality. However, during the transmission itself, the role of the signalling is finished. Moreover, the system must ensure that specific signalling applies to specific data, this means that the link between the signalling and the data needs to be secured as well. We will assume here that a complete separation exists between call control and connection (bearer) control, as it will be the case in the CCITT target signalling protocol for B-ISDN. Therefore, security functions for both of these control protocols have to be foreseen.

In the call control, we find for example end to end authentication procedures; here the signalling can be used to transport the authentication messages.

The connection control is used by intelligent network management. Regarding the security, the decisions are taken at a high level and signalling is used between nodes to control the network configuration. The signalling has to provide the appropriate control messages (layers 1, 2 and 3), including the security messages.

To summarize, the signalling has to provide the means for exchanging control messages related to the security. These messages can be related to the user plane, to the call control and to the intelligent network management. It has to be examined in which cases the signalling protocol can directly provide security functionalities.

1.3 Security

This section will define and describe the basic security elements needed to integrate security in a communication system. We will start with a description of the basic three security services. They can be implemented using security primitives. Primitives are designed to satisfy mathematically precise requirements. They are described in terms of more technical concepts like cryptosystems, hash functions, etc. (see chapter 4).

These security services, which are explained below, are:

- authentication
- integrity
- confidentiality.

Authentication

The authentication of an "entity" is the corroboration that the entity is the one claimed. This means that if an entity has authenticated itself to another entity, the last one is assured that he will communicate with the "genuine" entity he was intended to. An authentication process will detect when another entity is claiming to be the authorised one. It therefore goes further than a simple identification. Notice however, that a secure user identification is necessary to realise the authentication protocols.

In a telecommunication network, there may be a need to authenticate the resources used in the communication process, and possibly also the titular of the organization and/or the responsible for these resources. This means that it is not the authentication of persons which is important here, but the authentication of equipments.

The authentication process, as will be explained in chapter 4, requires the exchange of messages between an entity requesting the authentication, and the entity being authenticated. A task for the signalling can be to act as transport for these messages.

It is obvious that identification and authentication of users or equipment is often not enough. A natural first requirement to a secure communication is that the transmitted information cannot be changed in an unauthorised way without being detected. This means that we need integrity:

Integrity

Data integrity means that the contents of the data cannot be altered without detection. This is of course a crucial issue in data communication, but other kinds of integrity are also important in specific environments.

Network integrity is an important issue, because it directly establishes the level of confidence a user will have in the service offered by the network operator.

Historically, data integrity has always been obtained and certified by means of redundancy (e.g. a signature or checksum) transmitted with the data. Now with the separation of the control functions from the data, other techniques are possible. Clearly, signalling can play a role in this process. The data integrity can be verified during the transmission, or after transmission in order to certify that the "genuine" message has effectively be delivered to the right destination. It is not the role of the signalling to provide the data integrity itself, but it can help the entities involved by transmitting service information related to this integrity.

For a network, integrity means reliability of the connections. This is normally achieved by using re-routing of signals in case of failure, or by using redundant equipments. These measures are security functions, although they are rarely presented as such. Among the signalling functions used by the network management for the control of the routing, only functions added for the improvement of the security will be considered here. The control of the conformity between the network model maintained by the network management, and what is really happening is one of the integrity functions to be examined.

Another important issue is the integrity of the access control, related to the availability of the network.

Confidentiality

Confidentiality means that the information contained in a message can only be accessed by authorised entities. In a communication network, this clearly implies an intervention at the data level. As already indicated, the separation of the data and of the signalling hampers the use of signalling for direct intervention at data level. The signalling can only be involved in an indirect way, during the confidentiality process negotiation.

In a network the confidentiality may be provided on certain "sensitive" segments only, or end to end (from access point to access point). The confidentiality at application level is out of the scope of this study.

In special cases, it is the confidentiality of the signalling messages itself which can be needed. We find already an example of this within GSM. If the B-ISDN will offer the user the possibility to connect to different access points, without pre-registration (for portable terminals), it would be interesting to prevent a possible localisation of this user by non-authorized parties. Therefore the confidentiality of the signalling messages might be required. Another typical example is videoconferencing where layer 6 is used for the coding. This means that the confidentiality must be realised in a lower layer than layer 6 since otherwise the system will not be able to operate.

Other security services

Authentication, integrity and confidentiality are the basic security services. More complex services such as access control, non-repudiation of sender or receiver can be realised with the three services mentioned above. Moreover it is very likely that the additional services can be build with the network functionalities which originally covered the basic three security services. An example for non-repudiation will be given in the next chapter.

Chapter 2

Security functions in signalling

The basic question is: why is security needed in B-ISDN ? Well, simply because such a network is subject to failure and because ill-intentioned persons exist.

Many indications have already been given in the preceding section which show that signalling can be efficiently used to handle security problems both within the call control and within the bearer control.

Obviously, if no specific security functions are introduced in the call control, the possibility will remain to place these security functions at application level, or at communication level. It will only be less effective. On the contrary, improving the security for intelligent network management seems only possible with specific signalling functions.

Some cases where the use of security functions in signalling has already been identified will be reviewed now.

End-to-end authentication

For obvious reasons, it is during the call establishment that the end to end authentication will be the more often used. With signalling, it is possible to achieve this authentication, without the intervention of the application. Access to the application for example, may only be given to authorized entities, after a successful authentication at a lower level. Clearly, this would substantially improve the security.

Access control

The access control normally takes place after an authentication process. It gives an entity the rights to access a particular resource or information. For example in mobile communications, where the users can make their calls from different portable equipments, access control is a main security service. As already explained, the decision to give access is normally not included in the task of the signalling. Access control is under the responsibility of the application, of the service providers, or of the network (e.g. in virtual private networks). Certainly, special signalling functions will nevertheless be required if access control functionalities are desired without the intervention of a third party.

It can be useful, when an access is denied, to pass the information "access denied" to the calling party, at least as a cause in a disconnect message.

Network management

As indicated in part 3 (of the final document), the separation of the call control from the bearer control offers many advantages. Its drawback is that new functions are now needed to guarantee that the connections established by means of bearer control messages correspond well to what they are supposed to be from the call control point of view. This implies the introduction of verification procedures with authentication possibilities.

To achieve a high level of network integrity, the network management shall be able to verify the integrity of a connection at every moment. Instead of a single identification, the authentication of the resources used (user access points, switching nodes, interconnections with other networks, ...) may be necessary.

Having identified (authenticated!) the resources used to interconnect two subscribers, the network management has to declare that a communication path exists between the subscribers. Acting as such, the network management declares in fact that the equipments are genuine and that they are in conformity with his own model of the network. Signalling techniques are needed to verify the concordance between the model and the reality.

To improve the security of a transmission, redundant resources are sometimes used, with some of them in stand-by. In case of malfunction of an equipment, the redundant equipment is put into service by means of signalling. Both resources, the normal and the redundant one, receive the same address, but they are different entities, and from the point of view of the authentication, they are different. It is important to avoid the replacement, by a malicious person, of the redundant unit by an external one with the same address. It follows that the network needs control means.

Non-repudiation of information exchanged between user and network provider

Similar to a registered letter which can be used as a legal proof, certification of information exchanged between the network provider and the user could also be used as a matter of proof. This domain has not yet been explored but could lead to very attractive supplementary services. It certainly can prove to be useful for billing purposes or to detect failures in such bills. Signalling comprising security functions is well suited to support this type of service. Signalling is also needed to ensure that the information given has been correctly collected though the network.

Traffic flow confidentiality

For certain applications (e.g. banking), confidentiality is needed not only with respect to the contents of the information exchanged, but also regarding the simple existence of a transfer of information. Even without having access to the contents of the information, the number of transfers and/or their sizes in a given period of time can have a great significance for pirates. A method to avoid this risk is message stuffing. Dummy messages are transmitted, which are mixed with the genuine ones. By means of appropriate signalling, the user informs the network with confidential signalling whether the data are true or dummy. The price asked by the network operator for transmitting dummy data is very low, and he has the right to modify these dummy data.

Chapter 3

Applications of security functions in signalling

Virtual private networks

A virtual private network requires, in order to be safe, a lot of security functions from the network management such as guarantee of integrity of the virtual network, access control (from access point to access point), and confidentiality. All these functions are invisible to the user. They have to be handled by the intelligent network management by means of specific signalling protocols.

Interacting with other networks

As stated by the CCITT, the B-ISDN must be able of supporting interacting with non B-ISDN services (see I.311). Examples of these services are UMTS, GSM, and DECT. Due to the fact that radiocommunications are used for transmissions to some of the mobile subscribers, it is clear that on these links, mobile telecommunications will intrinsically not provide the same level of protection to its operators and subscribers as B-ISDN will do. Therefore specific security features will be used by these networks for these parts of the transmission. As strong interacting with these networks is the final goal, it will be mandatory for the B-ISDN to provide the signalling functions able to support the signalling from the other networks, and among these functions, there will be specific ones covering the security services.

As a general remark, it is worth noticing that it could also be interesting for the B-ISDN to anticipate on the future needs in this domain, in order to be ready for new applications, and perhaps yet to induce these new applications.

Indirect handling

As figure 3.1 shows, it is possible that the communication between two entities is realised in an indirect way via an intermediate node, a so-called relay system. Since this relay system is part of the network, it consists only of the lower three layers. In this configuration, security between the two end-entities A and B can be introduced up to layer 3.

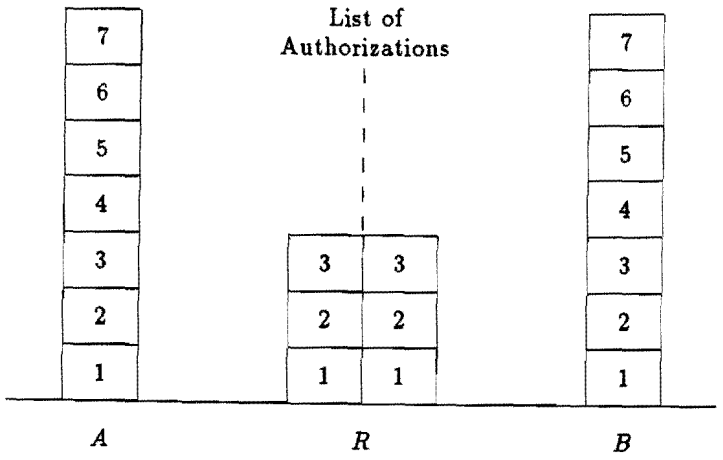


Figure 3.1: Indirect access control

Connectionless

Connectionless protocols are established without any need for addressing of the interacting entities. Only their identification is needed. The entities are acting by themselves, the transfer of data is implicitly done by an underlying service, who also has to handle the security problem.

Chapter 4

Information security

4.1 General security services

Entity identification/authentication

The two principles *identification* and *authentication* are often confused, but they do not have the same meaning. In fact, identification is less strong than authentication: identification is the process that enables recognition of an entity described to a system, whereas authentication is the act of verifying the claimed identity of this entity.

We will only deal with authentication processes, but it is obvious that in order to achieve entity authentication, secure entity identification is necessary. This can be realised through the use of a secure registration of the entities with their credentials such as user privileges, etc., and also with a public key (see 4.4).

ISO/IEC 9798 is a multipart standard dealing with authentication mechanisms based on symmetric and asymmetric techniques. We will deal with these protocols in detail in section 4.3.

Data integrity

In [I7498-2] data integrity is defined as the property that data has not been altered or destroyed in an unauthorized manner. Outside the OSI context, data integrity is sometimes called *data origin authentication*, which according to the OSI definition is the corroboration that the source of data received is as claimed. When two entities communicate using a previously established connection, and after an entity authentication protocol, the origin of the data received is authenticated.

Usually both services are required together. If it can be assured that the data received came from the claimed source, this is not useful in practice if the data may have been changed in an unauthorized way. Conversely, if we know that the data have been transmitted without any unauthorized changes, this is not useful unless we know that the data came from the claimed source and not from an impostor.

Note, that even though a receiver may be convinced of the integrity of a message when he receives it, he may be unable to convince a third party, since he may have been able to produce a convincing message himself. To solve this problem the next security service is needed:

Non-repudiation of origin

This means that the receiver has a proof that the message in fact originated from the genuine sender. Not only the receiver, but any third party can check this proof. Note that non-repudiation of origin is at least as strong as data integrity: if we receive an acceptable proof of origin, the message has not been changed on the way.

So far, nothing solves the replay problem where a malicious third party copies a message and retransmits it. Payment orders are an obvious example that show the need to protect against this threat. By using time stamps or message sequence numbers it can be ensured that exactly the same message is never accepted twice.

The same payment order is an example of a case where the receiver of a message should be unable to later deny having received it. This is a motivation for:

Non-repudiation of receipt

Non-repudiation of receipt (or delivery) means that the receiver of message sends back a receipt to the sender. Anyone, including the sender of course, can verify the validity of this receipt, but only the genuine receiver of the original message is able to produce such a receipt message. Therefore he cannot later deny having received this message.

Finally, the most classical security service is mentioned:

Confidentiality

In [I7498-2] confidentiality is defined as the property that information is not made available or disclosed to unauthorized individuals, entities or processes. In common parlance it just means that the message is transmitted in such a way that no unauthorised entity can learn anything about its content.

4.2 Security primitives

4.2.1 Mathematical primitives

In this section we define the mathematical security primitives required to implement the security services mentioned above.

Data encryption

This primitive ensures confidentiality, i.e. it is concerned with keeping the message secret.

Each data encryption algorithm uses a secret key, and in fact the security of the enciphering relies on this key, while the algorithm used is generally public knowledge. Therefore it is clear that these keys have to be chosen carefully (and have to be managed correctly, which will be elaborated in section 4.4). Usually, this means that a key must be chosen at random from a certain set of "good" keys. Often a candidate key is generated at random, or pseudo-random, and afterwards is checked if this candidate is not a so-called weak key.
In the description of the various encryption methods we will give some attention to the requirements on keys needed.

The data encryption algorithms can be split in two groups: the symmetric, conventional or secret key methods, and the asymmetric or public key methods.

A *conventional* cipher is a cipher that uses a secret key which is known to both the sender and the receiver of the information; the same key is used for encrypting and decrypting the message.

A well-known example of a symmetric cipher is the *Data Encryption Standard (DES)*, designed by IBM in 1976 for the US Government ([NBS77]). The algorithm has also been adopted by the American National Standards Institute (ANSI), where it is known as the Data Encryption Algorithm (DEA) ([ANSI81]). The exact algorithm can be found in the standards and many text books on cryptography, we will only give a sketch of it.

The algorithm uses a 64-bit enciphering key, of which only 56 bits are real key bits, while the remaining 8 bits are parity check bits.
For encryption, each 64-bit block of input data is subject to an initial permutation, followed by 16 equally specified rounds which each use a sub-key derived from the given encryption key, and a final permutation.
Each round is a special function which is composed of transitions, exclusive ors and S-boxes. These S-boxes are eight different substitution tables, each having an input of 6 bits and an output of 4 bits.

For decryption exactly the same key is used while the algorithm is run in reverse order.

In case the enciphering key produces a constant set of sub-keys (which happens exactly for four keys) re-enciphering of the ciphertext with the same key restores the original plaintext. These so-called *weak* keys should be avoided in critical situations. Beside those weak keys *semi-weak* keys exist, whose sub-keys occur in pairs. For each pair of semi-weak keys the sub-keys are the same but in the reverse sequence. It is recommended to avoid them as well when selecting enciphering keys.

Unlike in symmetric cryptosystems, in *public key* cryptosystems the sender and receiver use a different, but related key. In fact, the encryption algorithm with public key P is trapdoor one-way, which means that it is infeasible to compute the corresponding secret key S for the deciphering algorithm from knowledge of the description of the enciphering algorithm and P. Moreover, the algorithms must be such that for each message M holds that

$$dS(eP(M)) = M.$$

The first public key cryptosystem proposed in the open literature is due to Rivest, Shamir and Adleman, and is generally known as the *RSA cryptosystem* ([RSA78]). It is based on modular exponentiation with fixed exponent and modulus, as will be shown in the following description.

Let p and q be two large distinct primes with product n. Denote the least common multiple (lcm) of $p-1$ and $q-1$ by $l(n)$. Choose e coprime to $l(n)$ and compute $d \equiv e - 1 \bmod l(n)$. Note that d can only be computed if the trapdoor p and q is known. The secret key is now $S = \langle d, n \rangle$, while the corresponding public key is given by $P = \langle e, n \rangle$.

The enciphering of message M reads as:

$$C \equiv eP(M) \equiv M^e \bmod n,$$

while decipherment of the cryptogram C is given by

$$dS(C) \equiv dS(M^e \bmod n) \equiv (M^e \bmod n)^d \bmod n \equiv M^{e \cdot d} \bmod n = M,$$

since $e \cdot d \equiv 1 \bmod l(n)$.

Since the system is broken when the factorisation of n is known, special precautions have to be taken into account concerning the choice of the two primes p and q.

In particular:

- p and q should differ in length by only a few digits,
- both $p-1$ and $q-1$ should contain large prime factors,
- the greatest common divisor of $p-1$ and $q-1$ should be small.

For an extensive discussion on these properties we refer to [D83].

Integrity primitives

Integrity primitives are all those that are concerned with the authentication of messages. Just as with data encryption, they can be divided into primitives which are based on conventional cryptography, and primitives which are based on public key cryptography.

Based on *conventional* cryptography are:

MAC: Message Authentication Code
The sender of a message M uses the secret key K he shares with the receiver to compute $MAC(K, M)$ on this message. The message may have any length, but the MAC has a fixed (small) length (e.g. 64 or 128 bits). For any entity that does not know K, it should be computational infeasible to find a message $\tilde{M}$ and a MAC X such that $MAC(K, \tilde{M}) = X$. MACs are being standardized in [I9797].

MDC: Manipulation (or Modification) Detection Code
The sender of a message M computes a manipulation code $MDC(M)$. Again, the message can have any length, but the MDC has a fixed (small) length. It should be impossible to find two different messages M and $\tilde{M}$ such that $MDC(M) = MDC(\tilde{M})$. To a manipulation detection code is often referred as a hash function. ISO/IEC 10118 recommends methods for computing MDCs.

In public key cryptography the authenticity of messages is achieved using digital signatures. Here each user has a secret key that is only known to him, which allows him to create the electronic equivalent of a written signature, while anybody in possession of the corresponding public key can verify this digital signature. If A sends a digitally signed message to a receiver B, then B will not only be convinced that the message was indeed signed by A, but he will also be able to prove to a third party that A actually signed that message.

A public key cryptosystem offers digital signature capacity if:

$$eP(dS(M)) = dS(eP(M)) = M.$$

If a user's secret key is S, and his corresponding public key is P, then his signature on message M is $Sig = dS(M)$. Only this particular user can compute Sig, but everyone can verify its validity by checking that $eP(Sig) = eP(dS(M)) = M$.

Digital signatures can be used to ensure data integrity, but they are also applicable to achieve non-repudiation of origin and receipt.

Various digital signature schemes exist, the one based on RSA probably being the most famous one. However, if RSA is used both for data encryption and digital signatures, it is preferably that each user keeps two distinct pairs of keys. A digital signature scheme giving message recovery which is based on RSA is standardised in [II9796].

In 1985 *El Gamal* ([EG85]) introduced a new digital signature scheme based on the discrete logarithm problem. The scheme uses a large prime p, and a number a, $1 < a < p$. A user chooses a secret key S and calculates the public key $P \equiv a^S \bmod p$. The signing of a message M, $0 \leq M \leq p$, is as follows:

- choose a random k, $0 \leq k < p$, for which the greatest common divisor of k and $p-1$ is 1;
- compute $r \equiv a^k \bmod p$;
- the signature on M is the pair (r, t) with t the solution of

$$a^M \equiv a^{S \cdot r} \cdot a^{k \cdot t} \bmod p,$$

or

$$M \equiv (S \cdot r + k \cdot t) \bmod p - 1,$$

or

$$t \equiv (M - S \cdot r) \cdot k^{-1} \bmod p - 1,$$

which has a solution since the greatest common divisor of k and $p-1$ is 1.

For the verification of the signature, one has to check that for given M, r, and t holds that $a^M \equiv P^r \cdot r^t \bmod p$, since this equals $a^M \equiv a^{S \cdot r} \cdot a^{k \cdot t} \bmod p$.

Notice that the parameter k should not be used more than once, and that the parameter $p-1$ should have at least one large prime factor. If $p-1$ has only small prime factors, then computing the discrete logarithm is easy. Most attacks to the scheme can easily be shown to be equivalent to computing discrete logarithms over GF(p).

A third digital signature scheme has been published by NIST ([NIST91]) as a proposal for a standard. It is called the *Digital Signature Standard (DSS)*. The algorithm uses the following parameters:

- prime p, $2^{511} < p < 2^{512}$;
- prime divisor q of $p-1$, $2^{159} < q < 2^{160}$;
- $g = h^{(p-1)/q} \bmod p$, where h is any integer with $0 < h < p$ such that $h^{(p-1)/q} \bmod p > 1$;
- integer S, $0 < S < q$;
- $P \equiv g^S \bmod p$;
- M, the message to be signed and transmitted;
- k, a random integer with $0 < k < q$;
- H, a (one-way) collision-free hash function.

The integers p, q and g can be public and can be common to a group of users. A user's public key is P, and his corresponding secret key is S. The number k must be different for each signature.

To sign M, the user chooses a random k and computes:

$$r \equiv (g^k \bmod p) \bmod q$$

and

$$t \equiv \left(k^{-1}(H(M) + S \cdot r)\right) \bmod q,$$

where k^{-1} is the multiplicative inverse of k modulo q.

The signature on M is the pair (r, t).

To verify the signature, p, q, g and P must be known. Let $\tilde{M}$, $\tilde{r}$ and $\tilde{t}$ be the received values of M, r and t respectively. Then the following is done:

1. check if $0 < \tilde{r} < g$ and $0 < \tilde{t} < q$,
   if either condition is violated the signature is rejected;
2. compute

$$\begin{aligned} w &\equiv \tilde{t}^{-1} \bmod q, \\ u1 &\equiv \left(H(\tilde{M}) \cdot w\right) \bmod q, \\ u2 &\equiv (\tilde{r} \cdot w) \bmod q, \\ v &\equiv ((g^{u1} \cdot g^{u2}) \bmod p) \bmod q. \end{aligned}$$

If $v \equiv \tilde{r}$ then the signature is accepted.

For a mathematical proof of the validity of this verification process we refer to the Appendix of the standard ([NIST91]).

Finally, Amos Fiat and Adi Shamir introduced in 1986 a signature scheme that is *zero-knowledge* ([FS87]), which means that the verifier learns nothing but that the signature is valid. Formally, a zero-knowledge authentication scheme is an (interactive) protocol which enables a signer to prove that a certain message is sent by him, in which the verifier obtains no information from the prover except this fact and information which he could have produced alone.

Zero-knowledge authentication schemes have to be further investigated and their standardisation can be expected in the coming years.

4.2.2 Physical primitives

Tamper resistancy

In each data system, the data will appear somewhere and sometimes unencrypted. In particular, processing of data usually requires these data to be in clear form.
Therefore no data system can be made secure without some sort of physical protection of the equipment.

Microcomputers processing sensitive information should be consolated in areas that have physical access controls, and connected to a heavy object, possibly supplemented with a movement sensitive alarm pad. To decrease the risk that an intruder will get information from intercepting radiation, tempest equipment is available, like shielded optical fiber.

In general, a tamper resistant device is a technological construction which is a physical reality, and therefore it is unique. It protects the access to the data it contains, and the integrity of those data. Optionally a tamper resistant device can offer data processing facilities and/or data transport facilities.

An example of a tamper resistant device is a chip card. Since chip cards will become very important in the near future, we will elaborate below in some more detail their characteristics.

Chip cards and smart cards

In many applications of information security, the security relies ultimately on the secrecy of a small piece of information: the secret key of the system. A way to store this sensitive piece of information has to be provided.

A chip card is a generic name for any plastic card, usually the size of a credit card, containing a chip. Depending on the properties and features of this chip and its carrier, various types of cards can be distinguished. One of them is the smart card, a chip card where the chip is a microcomputer with programmable memory.

Nowadays, smart cards, or integrated circuit cards, can be considered as a convenient, safe, and inexpensive means for the storage of secret information ([DS91], [V92a]). However, smart cards which are available up to now have only a small capacity with respect to computing power. Since many cryptographic protocols require more, this is a non-neglectable restriction for their applications. Fortunately, smart card technology is growing fast, and more powerful cards will become available soon.

The basic references for such smart cards and the like is the multipart standard ISO 7816. Cards designed in accordance with this standard are made up of a plastic support that contains a small device consisting of a tiny printed circuit board with an integrated microcircuit at its center. On the surface of the board are three reserved areas:

- a location for the printed circuit supporting the microcircuit;
- a location for the magnetic stripe if the support is used as a combined card;
- an embossing area for the user's identity and the card's ISO number.

The microcircuit is attached to the back of the printed circuit. Any data exchange between the microcircuit's memories and the card reader is subjected to security procedures of the microcircuit's CPU. It generally consists of the following parts:

- a *ROM (Read Only Memory)*, containing the card's operating system. The ROM should contain programs and data that are not specific to a card, but that are the same for a large number of cards.
- a *RAM (Random Access Memory)*, a volatile memory that is used by the CPU as a buffer for storing transmission data and as a very fast access memory for storing intermediate results.

- an *(E)EPROM ((Electronically) Erasable Programmable ROM)* contains programmable, non-volatile memory. As the contents of the EPROM can only be erased by UV light, every cell can just once be programmed by the CPU, and the use of EPROM cards is thus limited to applications which do not need a frequent update of the memory. An EEPROM on the other hand can be electronically erased.
- a *CPU (Central Processing Unit)* controlling the internal buses via which it can access all the internal memories. No direct access from outside is possible to them.
- *connection points* or *contacts* to the system.

For the exchange of data between a smart card and an interface device, two protocols are specified so far on an international level, denoted by T=0 and T=1 respectively. Both are asynchronous, half duplex protocols; the main difference between them lies in their handling of the data and the OSI reference model.

T=0 is a byte-oriented protocol, where the only error correction is a parity check immediately after each byte. There is no clear separation of the transport and application layer, so it is impossible to encrypt headers. Furthermore, application data cannot be sent in both header and response of one command. This protocol is standardized since 1989 in [II7816-3].

The standardisation of the block protocol T=1 is from a more recent date. Here the error check is carried out on a block of data, the OSI layers are strictly separated, and application data can be sent in both a request and the response.

To communicate one byte of information, four additional bits are needed: a start bit preceding it, and a parity bit and two stop bits following it.

Various smart cards based on DES exist ([DS91], [V92a]), allowing encryption/decryption and MAC calculations. For a long time public key algorithms could not be implemented in these cards since they require a lot of computational power. This restricted the use of smart cards severely, because precisely public key techniques are generally considered to be very promising for the realisation of security services. However, smart card technology is evaluating fast, and at the moment various manufacturers are working on cards which can perform the large integer arithmetic needed by public key algorithms such as RSA. Recently a new chip 83C825 was developed, which computes a digital signature $X^e \bmod N$ with 512 bit operands in less than half a second. This chip will be used in the first RSA smart card ([DSS92]). Thus, now it is possible to perform digital signature generation and verification within a smart card, which increases the security considerably. Therefore those smart cards will become very important in the near future.

One application of smart cards can be found in GSM, the Global System for Mobile Communications ([V92b], [W92]). In GSM a mobile station can be taken to and used in any of the 18 participating countries, some of them allowing several network operators. The billing is done by the home network operator who normally does on-line authentication. But the user does not need to carry his own mobile station, he only has to take his subscriber card (or Subscriber Identity Module (SIM)) along and insert it into any mobile equipment. This SIM (which is a type of smart card) contains all the necessary information about the subscription, as well as the network specific authentication algorithm and the secret subscriber specific key. The functionality of the SIM is described in GSM 02.17 ([E-G92]), while its interface to the mobile equipment is specified in GSM 11.11 ([E-R92]).

4.3 Authentication protocols

An authentication protocol is a protocol in which the claimed identity of an entity is verified by another entity. The International Standard ISO/IEC 9798 consists of three parts, dealing with authentication mechanisms.

The first part, [I9798-1], is a general model, which explains that usually, for authentication purposes, the entities generate and exchange standardised messages, called tokens. It takes at least the exchange of one token for one entity to be authenticated by the other entity (*unilateral authentication*), and at least the exchange of two tokens to provide both entities with assurance of each other's identity (*mutual authentication*). The entity that is authenticated, and thus claims to have a certain identity, is called the claimant, the entity that verifies the claimant's identity is called the verifier. Precise definitions can be found in the Terminology section.

Part 2 deals with symmetric techniques. This implies that the entities that want to carry out an authentication protocol, either have to share a common secret authentication key in advance, or that a trusted third party is involved ([I9798-2]).

Part 3 of the standard describes entity authentication using asymmetric algorithms. All algorithms are based on the assumptions that the claimant has a secret signature key only known by itself, and that the verifier is in the possession of the valid public key of the claimant ([I9798-3]).

In the protocols *time variant parameters* are used to control uniqueness and/or timeliness, which is required to prevent replay of previously transmitted messages. Some of them also allow for the detection of "forced delays", i.e. delays introduced into the communication medium by an adversary. Three types of time variant parameters are used:

Time stamps make use of a common time reference which logically links a claimant and a verifier. If the difference between the time stamp in a received token, and the time the token is received is within an acceptance window, the message is accepted.
Time stamps can only be used if the time clocks of both parties are synchronised, and if they are not subject to tampering.

Sequence numbers allow a verifier to detect the replay of messages since the claimant and the verifier beforehand agreed on a policy to number messages. If the number sent along with a message does not agree with this policy, the message is rejected.
The use of sequence numbers requires some degree of additional book-keeping. Special procedures may be necessary to reset/restart sequence number counters when normal sequencing is disturbed, for example by a system failure.

Random numbers can be used to prevent replay or interleaving attacks as follows: the verifier sends a random number to the claimant, and the claimant sends this back in the signed part of the response.
Random numbers do not directly control timeliness, but this can be controlled by enforcing a maximal allowable time limit between the challenge and response passes.

Entity authentication using symmetric techniques

The authentication exchanges are based on the assumptions that

- A claimant authenticating itself to a verifier either shares a common secret authentication key with that verifier, or both entities share a secret authentication key with a Trusted Third Party (TTP). This key (or these keys) shall be known only to the two parties that share it.

- If a trusted third party is involved, it shall be trusted by both claimant and verifier.

If any of them is invalid, the authentication process may be compromised or cannot be implemented.

Below we will describe the first four authentication protocols of [I9798-2]. They require at most two passes for unilateral authentication, and at most three passes for mutual authentication, and they do not make use of a trusted third party.

(S1) One pass unilateral authentication.
Claimant A initiates the process by sending

$$TokenAB = eK_{AB}(T_A/N_A \parallel B)$$

to B. B verifies $TokenAB$ by deciphering it and checking the correctness of the distinguishing identifier B, as well as the time stamp or sequence number in order to guarantee timeliness. See figure 4.1.

Figure 4.1: One pass unilateral authentication

(S2) Two pass unilateral authentication.
Here verifier B initiates the process by sending a random number R_B to claimant A. A answers by sending back

$$TokenAB = eK_{AB}(R_B \parallel B)$$

as shown in figure 4.2. B verifies the token by deciphering it and checking the correctness of the distinguishing identifier B, and if R_B equals the number sent to A in the first step.

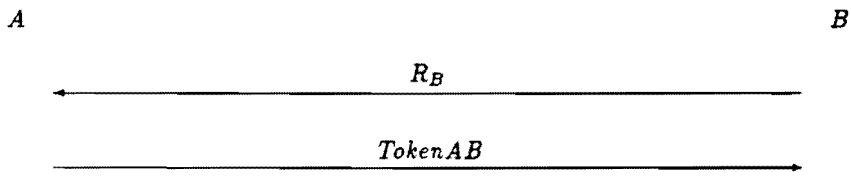


Figure 4.2: Two pass unilateral authentication

(S3) Two pass mutual authentication.
This protocol is just two successive executions of protocol (S1), first an authentication from A to B, then one from B to A.

(S4) Three pass mutual authentication.
A initiates the protocol by sending a random number R_A to B. B answers by sending

$$TokenBA = eK_{AB}(R_B \parallel R_A \parallel A),$$

where R_B is a random number chosen by B. On receipt, A checks, by deciphering the token, that the distinguishing identifier A and the number R_A are correct. Then A sends

$$TokenAB = eK_{AB}(R_A \parallel R_B)$$

to B. Finally, B verifies this token by deciphering it and checking the correctness of R_A and R_B. This protocol is depicted in figure 4.3.

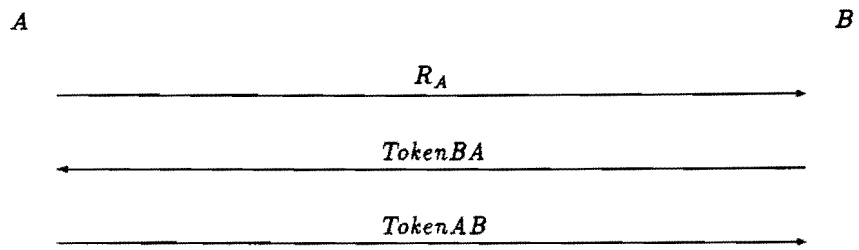


Figure 4.3: Three pass mutual authentication

In [I9798-2] three more protocols are described, which all make use of a trusted third party: a three pass unilateral / four pass mutual authentication protocol, a four pass unilateral / five pass mutual authentication protocol, and a seven pass mutual authentication protocol. Since they require too many passes to be useful for signalling, we will not describe them here.

It is very important to remark that in all the protocols, the tokens can contain extra text fields, in clear or encrypted form. They can be used for purposes as additional redundancy, information concerning data origin authentication or distribution of keys.

Entity authentication using asymmetric techniques

All protocols are based on the assumptions that the claimant has a secret signature key only known by himself, and that the verifier is in the possession of the valid public key of the claimant. A way of obtaining a valid public key is by means of a certificate; how to generate, distribute and revocate them is a separate problem which we will treat in section 4.4. In the protocols below, such a certificate can optionally be send whenever it is appropriate although it will not be mentioned explicitly.

(A1) One pass unilateral authentication.
Claimant A initiates the process by sending

$$TokenAB = T_A/N_A \parallel B \parallel sS_A(T_A/N_A \parallel B)$$

to B, and B checks subsequently the signature using A's public key, the time stamp or sequence number, and B's distinguished identifier. See figure 4.4.

(A2) Two pass unilateral authentication.
As shown in figure 4.5, verifier B initiates the protocol by sending a random number R_B to claimant A. A answers by sending

$$TokenAB = R_A \parallel R_B \parallel B \parallel sS_A(R_A \parallel R_B \parallel B)$$

Figure 4.4: One pass unilateral authentication

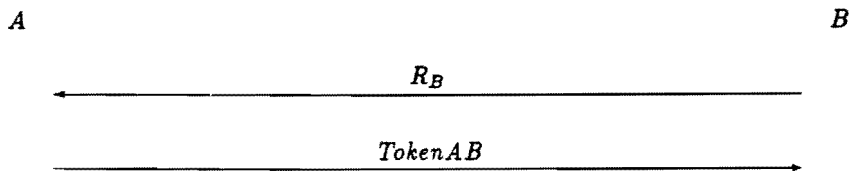


Figure 4.5: Two pass unilateral authentication

to B, which B checks for validity of A's signature, and for correctness of R_B.

(A3) Two pass mutual authentication.
This mutual authentication protocol consist of two subsequent conductions of protocol **(A1)**, first exactly as described there, and thereafter with the roles of A and B interchanged.

(A4) Three pass mutual authentication.
A sends a random number R_A to B. Then B sends

$$TokenBA = R_B \parallel R_A \parallel A \parallel sS_B(R_B \parallel R_A \parallel A)$$

to A. A checks B's signature and the number R_A. Hereafter A sends

$$TokenAB = R_A \parallel R_B \parallel B \parallel sS_A(R_A \parallel R_B \parallel B)$$

to B. Analogously to A's actions, B checks A's signature and the number R_B. In addition B checks the value of R_A. Tis protocol is depicted in figure 4.6.

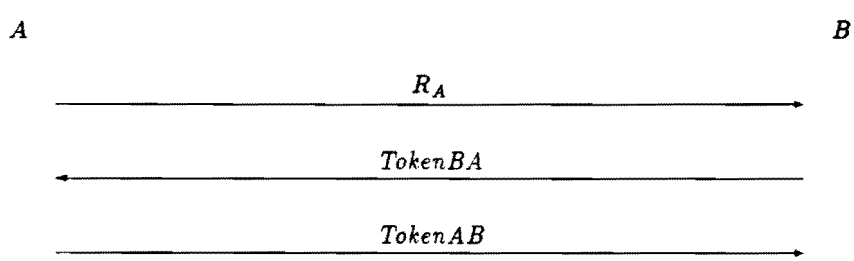


Figure 4.6: Three pass mutual authentication

(A5) Two pass parallel mutual authentication.
This protocol consists the parallel conduction of protocol **(A2)**: A sends R_A to B, and

in parallel B sends R_B to A. A and B reply to each other by sending $TokenAB$ and $TokenBA$ respectively. Simultaneously they check the validity of each others signatures, and that the random number which is previously sent to the other entity agrees with the one contained in the token.

As in the symmetric authentication protocols, each message sent can contain extra text fields whose contents is irrelevant for the authentication itself.

4.4 Key management, registration and certification authorities

Before secure communication based on cryptographic techniques between two or more entities is possible, cryptographic keys have to be distributed among them. The management of those keys is a very important aspect of a security policy, because the whole system relies on them.

[I7498-2] defines key management as "the generation, storage, distribution, deletion, archiving and application of keys in accordance with a security policy". Thus the purpose of key management is to provide all procedures for handling cryptographic keying material to be used in symmetric or asymmetric cryptographic algorithms. The tasks of a key management system can be split in several parts, which will be treated below.

User registration

Any secure system ultimately requires a procedure by which an individual, an organisation or a device is authenticated to the system. A key management scheme only makes sense if it guarantees the link between an entity and its keys. The registration of an entity is to allow automatic identification in the sequel. After its registration, an entity is uniquely identified by a so-called *distinguishing identifier*. Several types of authentication exist. *Absolute identification* is provided if a link between an identifier and some physical representation of the identified entity can be established. Often applications only require *relative identification*, which is a procedure that re-establishes an entity known under some identifier without linking it to another representation.

Entity authentication usually is based on the exchange of certificates. An entity is represented by its credentials that have to be generated upon registration. These credentials serve as a proof of registration.

Trusted third parties

To achieve secure and efficient key distribution, and to manage the keys, usually some kind of Trusted Third Party (TTP) is used. In literature, more or less the same object appears under various names and corresponding abbreviations. If the mechanisms used are based on conventional cryptosystems, the trusted party is generally denoted by the term Key Distribution Center of Key Translation Center. The terms Certification Authority and Key Certification Center are common when public key algorithms are used.

Since the nonexistence of a trusted third party implies that between each pair of users at least once a physical distribution is necessary, cryptosystems usually decide to create one. However, in general, people do not want to trust someone (or some institute) unconditionally. Therefore a trusted third party which is trusted only to the minimum extent which is necessary to avoid

undetected malicious behaviour of the users is the most preferable. Often such a TTP is referred to as a *functionally* trusted third party. It does not know the secret keys of the users, and it cannot impersonate the users, in contrast with an *unconditionally* trusted third party.

Key generation

Another task of a key management system can be the generation of keys, but it is also possible that each entity generates its own keys. Keys shall be generated by using a random or pseudo-random process, such that certain parts of the key space are not more probable than others, and that it is not possible for unauthorized to derive keys. During, or after the generation of a key it has to be checked if this key is not a *weak key*. A weak key is a key which, in relation with the algorithm used, has certain properties that compromise the security of the system.

In this context, *key derivation* should be mentioned, which is a technique that can be used in symmetric cryptosystems to generate a potentially large number of keys from a single seed key (the derivation key) and some variable data like the user identity. This technique allows to separate keys without the need to manage all keys separately. The generation of derived keys utilises a non-reversible process such that the compromise of a derived key does not disclose the seed key or any other derived keys. Key derivation can also be used to update keys and thus obtain new keys without the need for a key distribution process.

Key registration and certification

This aspect of key management is only relevant for public key systems. The main problem to solve in this context is to ensure authenticity of the public keys. This is usually solved by requiring that each user registers with a public key before using the system, and then the system has to provide each user with an authentic copy of this user's public key.

The authenticity of public keys can be ensured by using *certificates* (see Annex B of [I9798-3], Recommendation X.500, and [CCITT88]). Such a certificate contains an entity's distinguishing identifier, the entity's public key, and possible other information like a validity period and/or a serial number. This collection of data is signed by a trusted third party. The verification of a certificate consists of verifying the signature of the trusted third party, and checking, if required, other conditions related to the validity of the certificate such as the validity period.

However, some practical problems remain. An important one among them is that of *blacklisting*: the registration of a user may at some point be invalidated, for example because he violated some rules, or because his secret information is lost or stolen. In that case the user should be blacklisted, i.e. his public key certificate should not be regarded valid anymore. Therefore the system should keep an updated list of users and certificates that are blacklisted, which can be checked by all users. To prevent the list from growing infinitely, certifications should have an expiration date, such that entries in the blacklist can be discarded after some time.

Key establishment: symmetric algorithms

When two parties want to communicate securely, a key has to be established between them. If we exclude the cryptographically non-interesting case where this is done manually (e.g. by courier), essentially two possibilities remain: point-to-point key establishment, and key establishment where a third party is involved.

For *point-to-point key establishment*, it is required that the initiator is able to generate or otherwise acquire a secret key. Furthermore is assumed that the parties already share a key

enciphering key. This key will be used to encipher the negotiation of the parties over the key.

The purpose of a *Key Distribution Center (KDC)* is to generate or acquire, and distribute keys to parties that each share a key enciphering key with the KDC. A *Key Translation Center (KTC)* does only distribute a key generated or acquired by one of the parties to the other party, using the key enciphering key that each party shares with the center. Various protocols for the key establishment using a KDC or KTC based on the authentication protocols standardised in [I9798-2] are under standardisation in ISO/IEC JTC1 SC27.

Key distribution: asymmetric protocols

When public key cryptography is used, a third party is usually denoted by Certification Authority (CA) or Key Certification Center (KCC). From these names we see that its most important task is not to distribute the public keys, which in effect is easy: they can for example be broadcasted, but to guarantee their authenticity.

The danger of a key being compromised is considered to increase with the length of time it has been in use, and the amount of data it has been used to encipher. Therefore, keys have to be changed frequently. A widely accepted, efficient way to do this is to use the (insecure) communication channel itself for key distribution. Of course, the keys must then be enciphered. This is done using *key enciphering keys*. Since those keys are used much less than the actual communication keys, they do not need to be replaced as often. In this way a whole hierarchy can be made, with at the top one or more *master keys*, which need physical protection and distribution.

Key replacement and key deletion

A key shall be replaced when its compromise is known or suspected. A key shall also be replaced within the time deemed to determine it by an exhaustive attack. A replaced key shall not be reused. The replacement key shall not be an easy to determine transformation of the replaced key.

When a key is no longer needed, it has to be destroyed. This means that all records of the key are eliminated, such that no information remaining after the deletion provides any feasibly usable information about the destroyed key. Depending on how the key is stored, it can be destroyed by overwriting, zeroising or destroying the storage medium.

Key storage

A key storage facility provides secure storage of keys, thus, confidentiality and integrity for secret keying material, or integrity for public keys. Secret keying material must be protected by physical security or be enciphered by keys that have physical security.

Since cryptographic keys may get lost due to human error, software bugs or hardware malfunction, it is recommended to store a copy of the current key independently from the original one. Of course this copy needs to be protected as good as the original.

Chapter 5

Security models

A security model is an abstract statement of the important principles of security that a system or product will enforce. For each system and product such a model should exist. However, it is not necessary to make a complete new model for each new system or product, since various formal models are published to which can be referred.
Examples of published formal models of security policy are:

- The *Bell-La Padula model* ([BLP76]), which models access control requirements typical of a national security policy for confidentiality.
- The *Clark and Wilson model* ([CW87]) modelling the integrity requirements of commercial transaction processing systems.
- The *Brewer-Nash model* ([BN89]) modelling access control requirements for client confidentiality, typical of a financial services institution.
- The *Eizenberg model* ([E]), which models access control rights that vary with time.
- The *Landwehr model* ([LHL84]) modelling the data exchange requirements of a message processing network.

Chapter 6

Security in signalling revisited

6.1 Identification

As already said in section 1.3, identification is not a security function, but is necessary to realise authentication. Therefore it is natural to deal with entity identification before considering the more complicated entity authentication. It is important to notice that an entity needs not to be a person, this can just as well be an equipment or an organisation.

In B-ISDN identification protocols according to an international numbering plan will exist (like in N-ISDN : Q931), but these identify the equipment that is used at the end points of a connection, not the entity that is using this equipment. For example, they identify the number of the telephone that is used to make a call, not the person that is making the call using that particular telephone. This contrasts with GSM ([V92b]), where an entity can use every mobile equipment to identify itself.

Since we want another kind of identification than the existing one, some extra information has to be added to the signalling messages. There are several possibilities to do this. It can be done like in GSM, using smart cards. A good architecture would roughly be like in GSM where two identification numbers are needed in every call: the temporary identity of the entity, and the identification of the home network.

6.2 Authentication

Once an entity is identified, it can be authenticated. Only the authentication protocol that is used should be standardised, and the service provider that carries out an authentication should be free to choose his own authentication algorithm. In section 4.3 several authentication protocols are explained, based on symmetric as well as asymmetric techniques. In principle, all of them could be used as standard, and they have all their own advantages and disadvantages.

The use of challenge - response protocols, where the claimant has to answer correctly on a challenge (typically a random number) sent to him by the verifier, would be a good candidate for authentication. To prevent replay, either time stamps or sequence numbers, or random numbers are necessary ([I9798-2], Annex B). A smart card is able to generate (pseudo) random numbers, but cannot generate a time stamp itself.

Furthermore, it would be interesting to choose a protocol that is compatible with the protocol used in GSM (or in the future UMTS), or DECT to facilitate interoperability. In fact, this are challenge - response protocols.

Up till now, all protocols used are based on symmetric techniques. The main reason for this is that there did not exist smart cards which were able to do the large modular arithmetic needed in public key algorithms in a reasonable time. Furthermore, there was not enough place to store the large keys needed. In the very near future this will be possible ([DSS92]), but the half a second needed to compute a digital signature is still rather long for our purposes. Besides, if public key techniques are used, each entity should be able to acquire everyone else public key. This implies that each entity does either need a very large data base, or has to ask a trusted third party for a key it needs.

Finally, the number of messages exchanged should be as small as possible since each extra communication takes extra time.

If we take all these considerations into account, and look for a symmetric protocol, using random numbers, with as few passes as possible, we end up with the two pass unilateral authentication protocol (S2), and the three pass mutual authentication protocol (S4).

6.3 Access control

The definition of access control is, according to [I7498-2], "the prevention of unauthorized use of a resource, including the prevention of use of a resource in an unauthorized manner". In practice, access control consists of an identification/authentication of an entity, followed by the grant (or denial) of access. Optionally some further privileges could be given to the entity.

The decision to give access is not a task of this signalling, this is done by the service provider. Signalling only offers the means to give the service provider all the information it needs in order to make the decision whether access should be given or denied. Signalling can nevertheless contain information (cause information element) on whether the protocol will continue or not and a reason for a premature stop, or information to the calling entity about the extent to which it is given access.

6.4 Integrity

In section 4.2.1 several methods to achieve data integrity are described. Based on conventional cryptography are the MAC and MDC, and based on public key cryptography various digital signature algorithms are shown. One can think of using signalling to send the MAC, MDC, or digital signature, but it is not obvious if this would be a signalling or an application function.

Not only the integrity of data is a security issue, but also the integrity of the network that is used to transmit those data. The network management has to ensure the integrity of the links. With the introduction of security in the network management, signalling will be needed for functions like authentication and access control. Other security considerations, like redundancy, are not related to signalling.

6.5 Confidentiality including stuffing

Confidentiality of a message means that the message is transmitted in such a way that a third party cannot learn anything about its content. This can be done by enciphering the message using the public key of the receiver in asymmetric cryptography, or the key shared between the two communicating parties in symmetric cryptography.

Since the data to be sent consists of the enciphered message, and data and signalling are separated, signalling cannot intervent the transmission at data level. Signalling can only be used to indicate that certain data is encrypted, and which kind of algorithm is used for doing this.

It is too early to elaborate on stuffing in detail now, since it is unknown yet how stuffing can be done, and which kind of signalling will be needed.

6.6 Non-repudiation

The non-repudiation service is based on public key techniques. This will become important in the near future. However, it might be a service that fits more at application level than at signalling level, since in case a sender wants to have a proof of receipt, he will probably ask for this in the message itself and not use separate signalling.

6.7 Network management

The task of the network management is to administer secure user registration and key management. Having these duties, the network management becomes an obvious candidate for the control of the identification of entities.

In case public key techniques are used, the network management will often be the trusted third party, or certification authority that distributes and certifies the public keys.

6.8 Conclusion

The preceding sections have shown the interest of introducing optional security functions in signalling. It is obviously too early to define the representation of these function in the signalling messages. At this stage, the only thing we can say is that the elements to be introduced must be able to transport very long parameters (up to 512 bits).

It is only in the target B-ISDN signalling protocol (see part 1 of the final document) that security functions can be introduced. If the chosen model is in line with Q931, then security functions can be introduced as optional information elements in messages like SETUP. New messages specially dedicated to security have also to be created. On the other hand, if an object oriented approach is chosen (see part 2 of the final document), the security functions may be introduced as a new class of objects in a later stage of the definition work.

Whichever signalling model will be adopted by the CCITT, the introduction of security functions has to be taken into account. It will be an important improvement with regard to the existing situation, and a good step in the direction of a B-ISDN supporting a broad range of new services.

Terminology

Access

A user's ability to communicate with (input to or receive output from) a system to a specified area. Access does not include those persons (customers) who simply receive products created by the system and who do not communicate or interface with the system or its personnel. (NCSC-WA-001-85)

Access control

The prevention of unauthorized use of a resource, including the prevention of use of a resource in an unauthorized manner. ([I7498-2])

Authentication

The act of verifying the claimed identity of an individual, station or originator. (DOE 5636.2A)

Authentication exchange

A mechanism intended to ensure the identity of an entity by means of information exchange. ([I7498-2])

Authentication Token

Information conveyed during a strong authentication exchange, which can be used to authenticate its sender. ([?] and [CCITT88])

Availability

1. The property of being accessible and usable upon demand by an authorized entity. ([I7498-2])
2. The prevention of the unauthorized withholding of information or resources. (ITSEC)

Block chaining

The enciphering of information such that each block of ciphertext is cryptographically dependent upon the preceding ciphertext block. ([?])

Block cipher algorithm (n-bit)

A block cipher algorithm (i.e. a cipher that encrypts plaintext in blocks of a fixed length at a time) with the property that plaintext blocks and ciphertext blocks are n bits in length. ([I10116])

Certificate of a user

The public keys of a user, together with some other information, rendered unforgeable by enciphering with the secret key of the certification authority which issued it. ([?] and [CCITT88])

Certification authority

An authority trusted by one or more uses to create and assign certificates. Optionally the certification authority may create the user's keys. ([?] and [CCITT88])

Claimant

An entity which is or represents a principal for the purposes of authentication, together with the functions involved in an authentication exchange on behalf of that entity. A claimant includes the functions necessary for engaging in authentication exchanges on behalf of a principal. ([I9798-1])

Collision resistant

The property of a function that it is computationally infeasible to construct distinct inputs which give the same output. ([II10118-1])

Confidentiality

1. The property that information is not made available or disclosed to unauthorized individuals, entities or processes. ([I7498-2])
2. The prevention of the unauthorized disclosure of information. (ITSEC)

Credentials

Data that is transferred to establish the claimed identity of an entity.

Cryptographic device

The electronic hardware part, or subassembly, which implements the encryption algorithm. ([?])

Cryptographic equipment

Equipment in which cryptographic functions (e.g. encryption, authentication, key generation) are performed. ([?])

Cryptography

The discipline which embodies principles, means and methods for the transformation of data in order to hide its information content, prevent its undetected modification and/or prevent its unauthorized use. ([I7498-2])

Note:

Cryptography determines the methods used in enciphering and decipherment. An attack on a cryptographic principle, means, or method is cryptanalysis.

Cryptology

The field that encompasses both cryptography and cryptanalysis. (FIPS PUB 39; AR 380-380)

Data encryption key
A cryptographic key used for encrypting (and decrypting) data. (FIPS PUB 112)

Data encryption standard
An unclassified crypto algorithm adopted by the National Bureau of Standards for public use; (NCSC-WA-001-85)

Data integrity
The property that data has not been altered or destroyed in an unauthorized manner. ([I7498-2])

Data origin authentication
The corroboration that the source of data received is as claimed. ([I7498-2])

Data security
The protection of data from unauthorized (accidental or intentional) modification, destruction or disclosure. (OPNAVINST 5239.1A; AR 380, 380; NCSC-WA-001-85)

Decipher
To convert, by use of the appropriate key, enciphered text into its equivalent plain text. (FIPS PUB 39)

Decrypt
To convert, by use of the appropriate key, encrypted (encoded or enciphered) text into its equivalent plain text. (FIPS PUB 39)

Decipherment or decryption
The reversal of a corresponding reversible enciphering. ([I7498-2])

Digital signature
Data appended to, or a cryptographic transformation of, a data unit that allows a recipient of the data unit to verify content and protect against forgery, e.g. by the recipient. ([I7498-2])

Distinguishing Identifier
Information which unambiguously distinguishes an entity in the authentication process. ([I9798-1])

Encipher
To convert plain text into an unintelligible form by means of a cipher system.

Enciphering or Encryption
The cryptographic transformation of data to produce ciphertext. ([I7498-2])

Encryption algorithm
A set of mathematically expressed rules for rendering information unintelligible by effecting a series of transformations through the use of variable elements controlled by the application of a key to the normal representation of the information. (FIPS PUB 39)

Entity
A physical person, an organisation or equipment about which information is stored in a database. ([LS90])

Entity authentication
The corroboration that an entity is the one claimed. ([I7498-2])

Hash code
The result of applying a hash function to data bits. ([II10118-1])

Hash function
1. A (mathematically) function which maps values from a (possibly very) large set of values into a smaller range of values. ([II10118-2])
2. A collision-resistant function which maps a set of arbitrary strings of bits onto a set of fixed-length strings of bits. ([II10118-1])

Identification
The process that enables recognition of a user described to a system. This is generally by the use of machine-readable names. (AR 380-380; NCSC-WA-001-85)

Information
Any communication or reception of knowledge such as facts, data, or opinions, including numerical, graphic or narritive forms, whether oral or maintained in any medium, including computerized data bases, paper, microform, or magnetic tape. ([A-130], DODD 5200.28)

Integrity
1. The assurance, under all conditions, that a system will reflect the logical correctness and reliability of the operating system; the logical completeness of the hardware and software that implement the protection mechanisms; and the consistency of the data structures and accuracy of the stored data. In a formal security model, integrity is interpreted more narrowly to mean protection against unauthorized modification or destruction of information. (MTR-8201)
2. The prevention of the unauthorized modification of information. (ITSEC)

Key management
The generation, storage, distribution, archiving and application of keys in accordance with a security policy. ([I7498-2])

Manipulation detection Code
A mechanism which is used to detect whether a data unit has been modified, either accidentally or intentionally. ([I7498-2])

Message 1. An ordered series of characters intended to convey information. ([C87])
2. An arbitrary amount of information whose beginning and end are defined or implied. ([C87])

Message authentication code (MAC)
A data field used to verify the authenticity of a message. ([?])

Network
A communications medium and all components attached to that medium that are responsible for the transfer of information. Such components may include ADP (Automatic Data Processing) systems, packet switches, telecommunications controllers, key distribution centers, technical control devices, and other networks. (DOE 5636.2A)

Reliability
The probability of a given automated system performing its mission adequately for a period of time intended under the expected operating conditions. (AR 380-380; NCSC-WA-001-85)

Repudiation
Denial by one of the entities involved in a communication of having participated in all or part of the communication. ([I7498-2])

Security
1. The quality or state of being cost-effectively protected from undue losses (e.g., loss of good will, monetary loss, loss of ability to continue operations, etc.). (WB)
2. The combination of confidentiality, integrity and availability. (ITSEC)

Security service
A service, provided by a layer of communicating open systems, which ensures adequate security of the systems or of data transfers. ([I7498-2])

Signature
The data identifying an entity associated with the data linking them to other data and ensuring the integrity of the whole.

Symmetric authentication method
Method for demonstrating knowledge of a secret, in which both entities share common authentic information. ([I10118-2])

Token (exchange AI)
Exchange authentic information conveyed during an authentication exchange. ([I9798-1])

Trusted third party

A security authority, or its agent, trusted by the other entities with respect to security-related activities. ([I9798-1])

Verifier

An entity which is or represents the entity requiring an authenticated identity. A verifier includes the functions necessary for engaging in authentication exchanges. ([I9798-1])

Abbreviations

ANSI
American National Standardisations Institute

B-ISDN
Broadband Integrated Services Digital Network

CA
Certification Authority

CAD
Card Acceptor Device

CCITT
Comit Consultatif International Tlphonique et Tlgraphique
(International Telegraph and Telephone Consultative Committee)

DEA
Data Encryption Standard

DECT
Digital European Cordless Telephone

DES
Data Encryption Standard

DSS
Digital Signature Standard

FIPS PUB
Federal Information Processing Standard Publication

GSM
Global System for Mobile Communications

IEC
International Electrotechnical Commission

ISO
International Organization for Standardization

ISDN
Integrated Services Digital Network

ITSEC
Information Technology Security Evaluation Criteria

KCC
Key Certification Center

KDC
Key Distribution Center

KTC
Key Translation Center

MAC
Message Authentication Code

MDC
Modification/Manipulation Detection Code

N-ISDN
Narrowband Integrated Services Digital Network

NIST
National Institute for Standardization and Technology

OSI
Open Systems Interconnection

PIN
Personal Identification Number

RSA
Rivest Shamir Adleman (cryptosystem)

(T)TP
Trusted Third Party

UMTS
Universal Mobile Telecommunication Services

Chapter 7

Notations

A is the distinguishing identifier of entity A.

B is the distinguishing identifier of entity B.

$dK_X(Z)$ is the result of the decipherment of data Z with a symmetric algorithm using the key K_X.

$dS(Z)$ is the result of the decipherment of data Z with an asymmetric algorithm using the secret key S.

$eK_X(Z)$ is the result of the enciphering of data Z with a symmetric algorithm using the key K_X.

$eK_{XY}(Z)$ is the result of the enciphering of data Z with a symmetric authentication key K_{XY}, shared between entities X and Y.

$eP(Z)$ is the result of the enciphering of data Z with an asymmetric algorithm using the public key P.

KID is a key identifier.

KXY is a secret key associated with entities X and Y, used only in symmetric cryptographic techniques.

N is a sequence number.

N_X is a sequence number issued by entity X.

P_X is a public key associated with entity X, used only in asymmetric cryptographic techniques.

R is a random number.

R_X is a random number issued by entity X.

$sS_X(Z)$ is the signature Sig of data Z using the secret key S_X.

S_X is a secret key associated with entity X, used only in asymmetric cryptographic techniques.

T is a time stamp.

$TokenXY$ is a token sent from entity X to entity Y.

TTP is the distinguishing identifier of the trusted third party, but in subscripts (e.g. for a time stamp) T is used.

T_X is a time stamp issued by entity X.

$Y \parallel Z$ is the result of the concatenation of the data items Y and Z in that order.

Y/Z is the notation for data item Y or data item Z.

Bibliography

[I7498-2]
ISO 7498-2: 1989
OSI Information processing systems - Open Systems Interconnection - Basic Reference Model - Part 2: Security architecture.

[I7816-1]
ISO 7816-1: 1987
Identification cards - Integrated circuit(s) cards with contacts - Part 1: Physical characteristics.

[I7816-2]
ISO 7816-2: 1988
Identification cards - Integrated circuit(s) cards with contacts - Part 2: Dimensions and location of the contacts.

[II7816-3]
ISO/IEC 7816-3: 1989
Identification cards - Integrated circuit(s) cards with contacts - Part 3: Electronic signals and transmission protocols.

[II7816-4]
ISO/IEC 7816-4: CD 1992
Identification cards - Integrated circuit(s) cards with contacts - Part 4: Inter-industry commands for interchange.

[II7816-5]
ISO/IEC 7816-5: DIS 1992
Identification cards - Integrated circuit(s) cards with contacts - Part 5: Numbering system and registration procedure for application identifiers.

[I8372]
ISO 8372: 1987 Information processing - Modes of operation for a 64-bit block cipher algorithm.

[I9594-8]
ISO 9594-8: 1990
Information processing systems - Open Systems Interconnection - The Directory.

[II9796]
ISO/IEC 9796: 1991
Information technology - Security techniques - Digital signature scheme giving message recovery.

[I9797]
ISO 9797: 1989
Data cryptographic techniques - Data integrity mechanism using a cryptographic check function employing block cipher algorithm.

[I9798-1]
ISO 9798-1: 1991
Information Technology - Security techniques - Entity authentication mechanisms - Part 1: General Model.

[I9798-2]
ISO 9798-2: CD 1992
Information Technology - Security techniques - Entity authentication mechanisms - Part 2: Entity authentication using symmetric techniques.

[I9798-3]
ISO 9798-3: DIS 1992
Information Technology - Security techniques - Entity authentication mechanisms - Part 3: Entity authentication using a public key algorithm.

[I10116]
ISO 10116: 1992
Modes of operation for an n-bit block cipher algorithm.

[II10118-1]
ISO/IEC 10118-1: DIS 1992
Information technology - Security techniques - Hash functions - Part 1: General.

[II10118-2]
ISO/IEC 10118-2: DIS 1992
Information technology - Security techniques - Hash functions - Part 2: Hash functions using an n-bit block cipher algorithm.

[A-130]
Office of Management and Budget Circular A-130, "Management of Federal Information Resources" of 12/12/1985.

[DODD5200.28]
US Department of Defense Directive 5200.28 (draft).

[ANSI81]
Data Encryption Algorithm, X3.92, American National Standards Institute, 1981.

[BLP76]
D.E. Bell and L.J. La Padula, Secure Computer Systems: Unified Exposition and Multics Interpretation, Report MTR-2997 Rev.1. Bedford, Mass: The MITRE Corporation, 1976.

[BN89]
D.F.C. Brewer and M.J. Nash, The Chinese Wall Security Policy, Proceedings of the IEEE Symposium on Security and Privacy, May 1989, pp.206-214.

[C87]
John M. Carroll, Computer Security, Second Edition, Butterworth Publishers, 1987.

[CCITT88]
The Directory - Authentication Framework, CCITT Recommendation X.509 (also DIS 9594-8), 1988.

[CW87]
D.D. Clark and D.R. Wilson, A Comparision of Commercial and Military Computer Security Policies, Proceedings of the IEEE Symposium on Security and Privacy, April 1987, pp.184-194.

[D83]
D.E. Denning, Cryptography and Data Security, Addison-Wesley Publ., 1983.

[DH76]
W. Diffie and M.E. Hellman, New Directions in Cryptography, IEEE Trans. on Information Theory, Vol.IT-22 (6), 1976, pp.644-654.

[DP89]
D.W. Davies and W.L. Price, Security for Computer Networks, John Wiley & sons, 1989.

[DS91]
Marijke De Soete, Smart Cards and their Applications, proceedings Compsec International '91 London, Elsevier, 1991, pp.147-156.

[DSS92]
M. De Soete and A.-J. Selezneff, Digital Signatures in Smart Cards: a reality with the new PHILIPS chip 83C852, to be published by Teletrust.

[E]
G. Eizenberg, Mandatory Policy: Secure Systems Model, Toulouse: ONERA/CERT/DERI, undated.

[EG85]
T. El Gamal, A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms, IEEE Trans. on Information Theory, Vol.31-4, 1985, pp.469-472.

[E-G92]
ETSI-GSM, Technical Specification GSM 02.17, Subscriber Identity Modules, Functional Characteristics, Version 3.2.0 (Release 92, Phase 1), 1992.

[E-R92]
ETSI-RES, European Telecommunication Standard, Final Draft, prETS 300 175-7, Digital European Cordless Telecommunications (DECT), Common Interface, Part 7: Security features, May 1992.

[ESAT91]
PC Security, ESAT course "State of the Art and Evolution of Computer Security and Industrial Cryptography", KU Leuven, 1991.

[FS87]
A. Fiat and A. Shamir, How to prove yourself: practical solutions to identification and signature problems, Advances in Cryptology, Proceedings of Crypto '86, Lecture Notes in Computer Science 263, Springer Verlag 1987, pp.186-194.

[LHL84]
C.E. Landwehr, C.L. Heitmeyer and J. McLean, A Security Model for Military Message Systems, ACM Transactions on Computer Systems, Vol.2 No.3, August 1984, pp.198-222.

[LS90]
Dennis Longley and Michael Shain, Data & Computer Security, Dictionary of standard concepts and terms, M stockton press, 1990.

[NBS77]
Data Encryption Standard, Federal Information Processing Standards Publication 46, National Bureau of Standards, US Department of Commerce, January 1977.

[NIST91]
A proposed Federal Information Processing Standard for Digital Signature Standard (DSS), National Institute for Standardisation and Technology, August 1991.

[RSA78]
R. Rivest, A. Shamir and L. Adleman, A method for obtaining digital signatures and public key cryptosystems, Comm. of the ACM, Vol.21, 1978, pp.120-128.

[S90]
Arto Salomaa, Public-Key Cryptography, EATCS monographs on theoretical computer science, Volume 23, Springer Verlag Berlin Heidelberg, 1990.

[T89]
A.S. Tanenbaum, Computer Networks, second edition, Pretence - Hall International Inc., 1989.

[V92a]
Klaus Vedder, Smart Cards, CompEuro 1992 Proceedings, Computer Systems and Software Engineering, 6th Annual European Computer Conference, IEEE, The Netherlands, pp. 630-635, 1992.

[V92b]
Klaus Vedder, Security Aspects of Mobile Communications, Proceedings of the ESAT 91 course KU Leuven, Springer Verlag, to be published.

[W92]
Michael Walker, Security in Mobile and Cordless Telecommunications, CompEuro 1992 Proceedings, Computer Systems and Software Engineering, 6th Annual European Computer Conference, IEEE, The Netherlands, pp. 493-496, 1992.

Part III
Digital Signatures

Contents

1 Introduction **4**

1.1 Basic concepts 4

1.1.1 Digital Signatures 4

1.1.2 Hash functions 5

1.2 General outline 6

1.2.1 The signing process 6

1.2.2 The verification process 7

1.2.3 Remarks 10

2 Digital signature schemes **11**

2.1 RSA 11

2.1.1 Description of the algorithm 11

2.1.2 Key generation 15

2.1.3 Length of the parameters 17

2.1.4 Security 17

2.2 ElGamal (in GF(p), p prime) 18

2.2.1 Description of the algorithm 18

2.2.2 Key generation 19

2.2.3 Length of the parameters 19

2.2.4 Security 19

2.2.5 Fields of characteristic 2 20

2.3 DSA 21

2.3.1 Description of the algorithm 21

2.3.2 Key generation 22

2.3.3 Length of the parameters 22

2.3.4 Security 22

2.4 Comparision of the schemes 23

3 Hash functions **25**

3.1 MD5 25

3.1.1 Description of the algorithm 25

3.1.2 Length of the parameters 28

3.2 DIS 10118-2 29

3.2.1 Description of the single length hash function 29

3.2.2 Description of the double length hash function 30

3.2.3 DES 31

3.2.4 IPES 32

3.2.5 Length of the parameters 33

3.3 SHA 35

3.3.1 Description of the algorithm 35

3.3.2 Length of the parameters . . . 36
3.4 AR/DFP . . . 37
3.4.1 Description of the algorithm . . . 37
3.4.2 Length of the parameters . . . 38

4 Combining hash functions and signature schemes 40
4.1 Parameters . . . 40
4.2 Combinations . . . 41

5 Smart cards 42
5.1 Generals . . . 42
5.2 TB100 . . . 44
5.2.1 Layout . . . 44
5.2.2 Hierarchical Key Design . . . 47
5.2.3 Personal Identification Number *PIN* . . . 48
5.2.4 Authentication protocols . . . 49
5.2.5 The RSA Public Key System . . . 51
5.2.6 Signing and verification . . . 52
5.3 DX Card . . . 55
5.3.1 The chip . . . 55
5.3.2 Personal Identification Number *PIN* . . . 55
5.3.3 Authentication protocols . . . 56
5.3.4 Signing and verification . . . 57

6 Key management 61
6.1 General considerations . . . 61
6.2 Trusted Third Parties . . . 61
6.2.1 Terminology . . . 62
6.2.2 Tasks . . . 62
6.3 Distribution of public keys . . . 63
6.3.1 Public key distribution without a trusted third party . . . 63
6.3.2 Public key certification . . . 63
6.4 Smart card management . . . 64
6.4.1 *PIN* modification . . . 65
6.4.2 Recycling of blocked Smart Cards . . . 65
6.4.3 Invalidation of Smart Cards . . . 65
6.4.4 Smart Card Validity Dates . . . 65
6.4.5 Loss of Smart Cards . . . 66
6.4.6 Smart floppy . . . 66

7 Legal aspects 67
7.1 Tasks of signatures . . . 67
7.2 Privacy aspects . . . 68
7.3 EDI . . . 68

8 Zero-Knowledge protocols 69
8.1 Introduction . . . 69
8.1.1 What is zero-knowledge? . . . 69
8.1.2 Zero-knowledge authentication protocols . . . 69
8.2 Fiat-Shamir . . . 71
8.2.1 Fiat-Shamir authentication protocol . . . 71
8.2.2 Fiat-Shamir signature scheme . . . 74
8.3 Guillou-Quisquater . . . 76

8.3.1 Guillou-Quisquater authentication protocol 76
8.3.2 Guillou-Quisquater signature scheme 78
8.4 Ong-Schnorr improvement of Fiat-Shamir 79
8.4.1 Ong-Schnorr authentication protocol 79
8.4.2 Ong-Schnorr signature scheme 81
8.4.3 Small integer variant 82
8.5 Schnorr 84
8.5.1 Schnorr authentication protocol 84
8.5.2 Schnorr signature scheme 86
8.6 Girault-Pailles 88
8.6.1 Girault-Pailles authentication protocol 88
8.6.2 Girault-Pailles signature scheme 92
8.7 Comparision 93
8.7.1 Overview 93
8.7.2 Choices for the parameters 94
8.7.3 Concluding remarks 95

A **96**
A.1 Rabin primality test 96
A.2 Probprime and probprimeinc 97
A.3 Provprime 99
A.4 Strongprime 101
A.5 Computations of discrete logarithms and factoring of a "hard integer" 102

Chapter 1

Introduction

1.1 Basic concepts

1.1.1 Digital Signatures

A digital signature in electronic mail is a counterpart to a handwritten signature in classic mail. A common feature is that they must provide the following properties ([D83], [N92]):

- The recipient is able to validate the signer's signature on M.
- It is impossible for anyone, including the recipient, to forge a signature.
- In case the signer should disavow signing a message M, it must be possible for a judge or third party to resolve a dispute arising between the (claiming) recipient and the (disavowing) signer.

A major difference between handwritten and digital signatures is that a digital signature cannot be a constant; it must be a function of the document that it signs. If this were not the case, then a signature, due to its electronic nature, could be attached to any document. Furthermore, a signature must be a function of the entire document: changing even one bit of the document should produce a different signature.

According to the definition in [I7498-2], a digital signature is "data appended to, or a cryptographic transformation of, a data unit that allows a recipient of the data unit to verify content and protect against forgery, e.g. by the recipient".

Digital signatures can be used to ensure the following security services:

User authentication is the service which enables a user of the system to convince the party he is communicating with that he really is who he claims to be.

Message integrity means that the contents of a message cannot be changed without detection. A digital signature on a message, transmitted together with this message can be used to ensure the recipient of the message's integrity.

Non-repudiation of origin means that the recipient of a message receives together with the message a proof that the message in fact originated form the claimed sender. Not only the intended recipient, but any third party can check this proof. Note that this security service is at least as strong as message integrity, for if an acceptable proof of origin is received, the message has not been changed on its way.

Non-repudiation of delivery provides the sender of the message with a proof that the intended receiver indeed received the message. To achieve this security service, the recipient has to send upon receipt of the original message a signed receipt back to the sender.

Most digital signature schemes are based upon a particular public key system. Such a public key system includes a procedure producing pairs of keys: a secret key and a public key, and procedures using the two keys. In any public key digital signature system, the secret key is involved in a signature process for signing messages, and the public key is involved in a verification process for verifying signatures ([II9796]).

1.1.2 Hash functions

Once the parameters of a signature scheme are chosen, the length of the input is fixed. This length varies from about 160 to 800 bits nowadays. Messages of any greater length will have to be processed in some way prior to the signing operation.

One possibility to sign a message of arbitrary length would be to divide the message in a sequence of "blocks" of appropriate size for the signature function and then sign each piece individually. That is, if the message M is made up of the sequence

$$M_1, M_2, \ldots, M_t,$$

where each M_i has the required length, then this message could be signed by computing

$$Sign(M_1), Sign(M_2), \ldots, Sign(M_t).$$

Unfortunately, this method has a number of disadvantages ([MPW92]):

1. There is no linkage between different parts of the message, and so the recipient of a signed message will not know if the message components (the M_i) have been reordered, replicated or partially deleted during transmission. A remedy is to include redundancy in the message blocks, but this has the disadvantage that the number of signatures to be computed increases.

2. Everyone can compute signatures of random messages by the public nature of the signature verification process. A forger can choose a value S at random, and compute the corresponding message $M = Ver(S)$. Then S is a valid signature on M. This problem can be removed by adding redundancy to the message to be signed, but then the efficiency of the scheme is reduced even further.

3. The signature algorithm has to be applied t times, which could be very inefficient since many proposed signature schemes are relatively difficult to compute.

These deficiences lead to the introduction of "one-way hash functions" in cryptography. A one-way hash function is a public function h, which should be simple and fast to compute, that satisfies three main properties ([MPW92]):

$h1$: It must be able to convert an arbitrary-length message M into a fixed-length string $h(M)$.

$h2$: It must be *one-way*, which means that given an arbitrary value y in the domain of h, it must be computationally infeasible to find a message M such that $h(M) = y$.

$h3$: It must be *collision-free*, which means that it must be computationally infeasible to construct two messages M and $\tilde{M}$ with the property that $h(M) = h(\tilde{M})$.

The need for $h1$ should be clear. $h2$ is present to prevent a fraudulent interceptor of a message and its signature (say M and $Sign(h(M))$ from replacing M by $\tilde{M}$ with the property that $h(M) = h(\tilde{M})$. The need for $h3$, which in fact implies $h2$, is less obvious. To demonstrate why this property is present, assume that we have a hash function h which does not satisfy $h3$. Then a malicious party Z may be able to construct two message M_1 and M_2 with the same hash result ($h(M_1) = h(M_2)$) such that another user, say X, would happily sign M_1, but would not sign M_2, while Z would like to have X's signature on M_2. Now Z can offer M_1 to X to sign, and claim later on that X signed M_2.

A signature on a message M is now computed by first computing $h(M)$, and using this hash result, possibly after extension, as input to the signature scheme.

[II10118-1] defines a hash function as a "collision-resistant function which maps a set of arbitrary strings of bits onto a set of fixed-length strings of bits". Some alternative terms for a hash function are compressed encoding function, and condensing function. The string of bits which is the output of a hash function appears under various names in literature: hash result, hash value, hash code, Modification - or Manipulation - Detection Code (MDC), residue, check sum, check value, (message) digest, and imprint.

1.2 General outline

In the remainder of this document we will consider various aspects of the signing and the verifying process into detail, but first we briefly describe the idea behind the two processes.

To store the keys needed in the signing and verification process, we will make use of smart cards as tamper resistant devices. We assume that two kinds of smart cards can be used, the TB100 card with a DES chip and the DX card with an RSA chip. Since the DX card provides digital signature generation and verification using the RSA algorithm, the actual signing process can take place in the card itself. This clearly enhances the security.

1.2.1 The signing process

The signing process is depicted in figure 1.1, and consists of the following steps:

1. The user authenticates himself to the smart card by presentation of his *PIN*.
2. Either a unilateral authentication protocol of the smart card of the signer versus the application is executed, or a mutual authentication protocol of as well the smart card versus the application as the application versus the smart card is executed.
3. The document to be signed is brought into the application.
4. The application transforms the document to ASCII format.
5. The signer chooses a hash function from a predescribed set.
6. The application hashes the document in ASCII format to a string with a fixed (short) length, depending on the hash function chosen.
7. The signer chooses an asymmetric algorithm (the signature algorithm) from a predescribed set.

8. If necessary, the application uses a publicly known procedure to expand the hash result to get a string with the required input length for the signature algorithm: the intermediate string.

9. Now there are two possibilities:

   - In case the smart card used is a DX card and the algorithm chosen is RSA:
     (a) The application transfers the intermediate string to the smart card with the request to sign it.
     (b) The smart card signs the intermediate string using the secret RSA key stored on it.
     (c) The smart card transfers the resulting signature to the application.
   - Otherwise:
     (a) The application transfers a request to the smart card for the secret signature key required for the chosen function.
     (b) The smart card transfers this signature key to the application enciphered under a card dependent key. The signature key is stored enciphered on the card, thus it can be read out in "in clear", which is much more efficient than an enciphered read from the card.
     (c) From the identity of the smart card, the application computes from a root key the key needed to decipher the signature key, and obtains by deciphering the signature key.
     (d) The application signs the intermediate string using the obtained signature key.

10. The application creates a signed document containing the original document and the signature, together with a note on who signed the document. Also information on the hash function and on the signature algorithm used are included.

Summarized, the message M (transformed to ASCII format) is hashed to $h(M)$, possibly expanded to $I(h(M))$, and finally signed to obtain the signature $S = Sign(I(h(M)))$:

$$M \rightarrow h(M) \rightarrow I(h(M)) \rightarrow S = Sign(I(h(M))).$$

1.2.2 The verification process

To verify a signed document, the protocol depicted in figure 1.2 and described below has to be executed:

1. The user authenticates himself to the smart card by presentation fo his *PIN*.

2. Either a unilateral authentication protocol of the smart card of the verifier versus the application is executed, or a mutual authentication protocol of as well the smart card versus the application as the application versus the smart card is executed. Note that it is not necessary that a smart card is involved in the verification process, because all parameters needed to verify a signature are public. We assume, however, that only an authenticated smart card has access to the verification program of the application. Optionally further restrictions can be added, for example on the signatures that a particular card is allowed to verify.

3. The signed document is brought into the application.

4. The application reads out the identity of the signer, and the identification of the hash function and signature algorithm used.

5. The application obtains from the signed document the original document and transfers it to ASCII format.

6. The application uses the specified hash function to hash the document in ASCII format to a string with fixed (short) length depending on the hash function.

7. The application obtains the public key of the signer for this signature algorithm and checks its validity. Usually the validity of the public key is ensured by checking its *certificate*: a digital signature of a certification authority on the public key. Using the authority's public key, the application can verify this signature, and hence the validity of the signer's public key.

8. Now there are two possibilities:

   - The first option is always possible: The application uses this public key to verify ("decipher") the obtained signature.
   - In case the smart card used is a DX card and the algorithm chosen is RSA, it is also possible that the verification is done in the card. Thus, the application transfers the public key of the signer and the signature to the smart card, and the smart card "deciphers" the obtained signature with the obtained key. The card transfers the result to the application. Note that there need to be facilities that enable the card to encipher not only with the key stored, but also with a key delivered.

9. If necessary, the application uses a publicly known procedure to shorten the "deciphered signature". In this procedure, the redundancy added in step 8 of the signing process is taken away, and the result is a string with the same length as an output string of the hash function used.

10. The application compares the result with the string obtained in step 6. If they are equal it outputs "OK", and otherwise the output is "not OK".

Summarized, the obtained message M (transformed to ASCII format) is hashed to $h(M)$:

$$M \quad \rightarrow \quad h(M);$$

and on the other hand, the obtained signature $\overline{S} = \overline{Sign(I(h(M)))}$ is "deciphered" to obtain $Ver\left(\overline{Sign(I(h(M)))}\right) = \overline{I(h(M))}$, which is possibly shortened to $\overline{h(M)}$:

$$\overline{S} \quad \rightarrow \quad \overline{I(h(M))} \quad \rightarrow \quad \overline{h(M)}.$$

If the two results $h(M)$ and $\overline{h(M)}$ are equal, the signature is accepted:

$$h(M) \stackrel{?}{=} \overline{h(M)}.$$

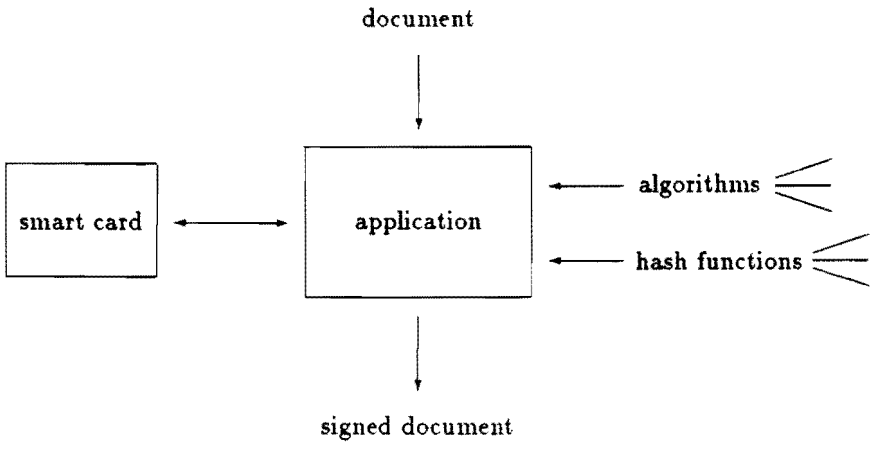


Figure 1.1: The signing process

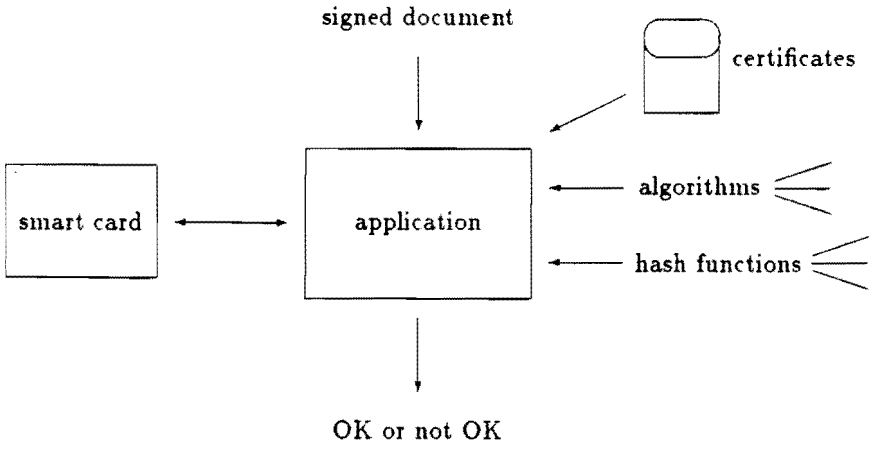


Figure 1.2: The verification process

1.2.3 Remarks

From the sketches of the signing and verification process, we can see that the following functionalities of the equipment used are required:

1. The application (PC) must be able to hash an arbitrary length message to a fixed length imprint.

2. When the DX card is used in combination with the RSA algorithm, the smart card must be able to sign a bitstring of the required length. Otherwise the application has to be able to do this.

3. When the DX card is used in combination with the RSA algorithm, the smart card or the application must be able to "decipher" a signature, given the public key and the digital signature. Otherwise the application has to be able to do this. Note that even when then smart card "deciphers" the signature, the application has to do the remaining part of the signature verification: calculating the hash value and the intermediate string.

Chapter 2

Digital signature schemes

2.1 RSA

The RSA algorithm was published in 1978 by R. Rivest, A. Shamir and L. Adleman ([RSA78]).

2.1.1 Description of the algorithm

First we will give the original algorithm, and afterwards we will describe the signature algorithm of the ISO/IEC standard 9796 ([II9796]) which is basically designed for multiplicative publid key algorithms such as RSA. This standard can withstand certain attacks that the plain RSA algorithm cannot. As mentioned before, it can be used for other algorithms than RSA, but in an annex RSA is used as an example, and this is up till now the only implementation of the standard. We will give the "ISO/IEC 9796-RSA algorithm" after the description of the general standard.

Description of the plain algorithm

preliminaries ([RSA78])

Each signing entity secretly and randomly selects two distinct odd primes p and q. The product of the two primes is a number n, the public modulus, of, say, 512 bits.
Then each signing entity computes $lcm(p-1, q-1)$ and chooses a positive integer v coprime to this number. This number v will be the signer's public verification exponent. In specific applications the public verification exponent may be standardized and then it has to be checked that p and q satisfy the condition above. Small numbers have some practical advantages.
Finally, each signing entity computes its secret signing exponent s as
$s = v^{-1} \bmod lcm(p-1, q-1)$.
The secret key is given by (n, s), and the public key by (n, v).

signing

The signature of a message M is computed as $S = M^s \bmod n$.

verification

Given M and a number S that should be its signature, a verifier checks if $S^v \bmod n$ equals M. If this is the case, the signature is accepted, otherwise S is rejected.

Various attacks on RSA exist, mainly based on the multiplicative properties of this scheme:

Problem 1 ([S92]):
A forger can compute signatures of random messages by choosing at random a signature S and computing the corresponding message M as $M = S^v \bmod n$.

Problem 2 ([S92]):
A forger can create new signatures from old ones: if (M_1, S_1) and (M_2, S_2) are valid message-signature pairs, then (M_1M_2, S_1S_2) satisfies $(S_1S_2)^v = M_1M_2 \bmod n$.

Solution 1&2:
Introduce redundancy in the messages to be signed. This is done in the algorithm described in the ISO/IEC standard 9796 ([II9796]). Here the message is transformed to an intermediate integer I which has a special form, and the probability that S^v has that form is very small when S is chosen without further knowledge (of e.g. s). Also the use of a cryptographic hash function before signing prevents such attacks.

Description of the ISO/IEC 9796 algorithm

This algorithm can be used to transform a hash result of bitlength L_H to a signature with length L_S if it holds that $L_H \leq 8 \cdot \lfloor \frac{L_S+3}{16} \rfloor$.
According to ISO/IEC 9796, the hash result H has to undergo the following processes before the actual signing process:

S1 padding $(H \rightarrow P)$;

S2 extension $(P \rightarrow E)$;

S3 redundancy $(E \rightarrow R)$;

S4 truncation and forcing $(R \rightarrow I)$.

Then the actual signing process can take place:

S5 signature production $(I \rightarrow S)$.

We will describe those processes below.
In all the processes, the values are represented as strings of bits where the most significant bit is the leftmost one.

S1. padding

Hash result H is padded to the left with 0 to 7 zeros as to obtain a string of z bytes.
The number r is defined as the 1 + the number of padded zeros, thus $1 \leq r \leq 8$.
The padded message is called $P = H_z||H_{z-1}||\ldots||H_2||H_1$, where the rightmost $z-1$ bytes equal the corresponding bytes of H, and H_z consists of $r-1$ padding zeros followed by the $9-r$ most significant bits of H.

S2. extension

Define t as the least integer such that $16t \geq L_S - 1$.
Repeat the z bytes of P as many times as necessary, in the original order and concatenated to the left, until a string of t bytes is formed. This string is the extended message $E = \ldots H_1||H_z||\ldots||H_1$. Since t not necessarily divides z, the rightmost bytes of P may occur once more in E than the leftmost ones.

S3. redundancy

The extended message with redundancy R is obtained by interleaving the t bytes of E in odd positions and t bytes of redundancy in even positions. The $2z$-th byte of R codes the message length by its value and position.

For $1 \leq i \leq t$ holds:
- the $(2i-1)$-th byte of R equals the i-th byte of E;
- the $2i$-th byte of R equals the image of the i-th byte of E according to the shadow S specified below, except for the $2z$-th byte where r is exclusive OR-ed to this value.

Thus,

$$R = \ldots S(H_z) \oplus r || H_z || \ldots || S(H_2) || H_2 || S(H_1) || H_1.$$

If byte m consists of the nibbles $\mu_2||\mu_1$, then $S(m)$ consists of the nibbles $\Pi(\mu_2)||\Pi(\mu_1)$.

μ	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
$\Pi(\mu)$	E	3	5	8	9	4	2	F	0	D	B	6	7	A	C	1

S4. truncation and forcing

The intermediate integer I is coded by a string of L_S bits, where the most significant bit is forced to "1". The $L_S - 1$ least significant bits are those of R, except for the least significant byte $\mu_2||\mu_1$ which is replaced by $\mu_1||6$.

S5. signature production

The signature S is obtained as a string of L_S bits by applying to I the signature function under control of the secret signature key: $S = Sign(I)$.

Of course, the reverse actions have to be undertaken when verifying a received signature:

V1 signature opening ($S \rightarrow \tilde{I}$), if OK then

V2 message recovery ($\tilde{I} \rightarrow \tilde{H}$), if OK then

V3 redundancy checking.

The three processes will be described below:

V1. signature opening

The signature S is transformed to the recovered intermediate integer $\tilde{I}$ by applying to S the verification function under control of the public verification key: $\tilde{I} = Ver(S)$.

If $\tilde{I}$ is not a string of L_S bits where the most significant bit is "1" and the least significant nibble is valued to 6, S shall be rejected.

V2. message recovery

First $\tilde{R}$ is recovered as the $2t$-bytes string where the $1 - L_S \bmod 16$ most significant bits are zeros and the $L_S - 1$ least significant bits are those of $\tilde{I}$, except for the least significant byte which is replaced. If $\mu_4||\mu_3||\mu_2||6$ are the four least significant nibbles of $\tilde{I}$, then the two least significant nibbles of $\tilde{R}$ are $\Pi^{-1}(\mu_4)||\mu_2$.

Write

$$\tilde{R} = M_{2t}||\ldots||M_2||M_1.$$

Compute for $1 \leq i \leq t$ the sum $X_i = M_{2i} \oplus S(M_{2i-1})$, and reject S if these all are zero.
Recover z as the first i for which the sum is non-zero.
Recover r as the value of the value of the least significant nibble of this first non-zero sum, and reject S if r is not valued from 1 to 8.

The recovered padded message $\tilde{P}$ is the string of the z least significant bytes in odd positions in $\tilde{R}$:

$$\tilde{P} = M_{2z-1}||\ldots||M_{2i-1}||\ldots||M_3||M_1.$$

Reject S if the $r-1$ most significant bits of $\tilde{P}$ are not all zero.

Finally, recover $\tilde{H}$ as the string of the $8z+1-r$ least significant bits of $\tilde{P}$.

V3. redundancy checking

Calculate from $\tilde{P}$ according to S2 and S3 an extended message with redundancy $\tilde{R}$, and reject S if the $L_S - 1$ least significant bits of $\tilde{R}$ and $\bar{R}$ are not equal.

Description of the ISO/IEC 9796-RSA algorithm

If 512-bit RSA is used, the requirement $L_H \leq 8 \cdot \lfloor \frac{L_S+3}{16} \rfloor$ becomes $L_H \leq 8 \cdot 32 = 256$. This holds for all combinations of RSA with one of the hash functions described in the next chapter.

In the signing process, the first four steps, padding, extension, redundancy and truncation and forcing are exactly executed as in the standard. The signature function in last step is RSA-specific, and requires the following preliminaries:

- Each signing entity selects a positive integer v as its public verification exponent. In specific applications the public verification exponent may be standardized. Small numbers, like 2 or 3, have some practical advantages.
- Each signing entity secretly and randomly selects two distinct odd primes p and q subject to the following conditions:
  - If v is odd, then $p-1$ and $q-1$ shall be coprime to v;
  - If v is even, then $\frac{p-1}{2}$ and $\frac{q-1}{2}$ shall be coprime to v. Moreover, p and q shall not be congruent to each other modulo 8.
  - The product of the two primes is an 512-bit number n.

  The product of those two prime factors is the public modulus n.
- Each signing entity computes its secret signing exponent as the least positive integer s such that $s \cdot v - 1$ is a multiple of
  - $lcm(p-1, q-1)$ if v is odd;
  - $\frac{1}{2} \cdot lcm(p-1, q-1)$ if v is even.

According to the above algorithm, the intermediate integer I is a string of 511 bits. In case v is even and the Jacobi symbol of I with respect to n is -1, then I should be replaced by $I/2$.

The signature is now computed as

$$S = \min(I^s \bmod n, n - (I^s \bmod n)).$$

When a received signature is verified, it is first checked that S is a positive integer less than $n/2$. Then the integer T is computed as $S^v \bmod n$, and thereafter the recovered intermediate integer $\tilde{I}$ is defined as follows
- if $T = 6 \bmod 16$, then $\tilde{I} = T$;
- if $n - T = 6 \bmod 16$, then $\tilde{I} = n - T$.

Moreover, when v is even
- if $T = 3 \bmod 8$, then $\tilde{I} = 2T$;
- if $n - T = 3 \bmod 8$, then $\tilde{I} = 2(n - T)$.

In all other cases, S is rejected.

As in the general standard, S is also rejected if $\tilde{I}$ is not a string of L_S bits where the most significant bit is "1" and the least significant nibble is valued to 6.

The message recovery and redundancy checking is performd exactly as described in the standard.

2.1.2 Key generation

The public key consists of the pair (n, v), and the secret key of (n, s).

In the key generation the choice of the primes p and q is a very important part. They should be chosen such that they do not introduce weaknesses in the RSA system. We mentioned already two attacks on RSA that the algorithm modified accordingly to ISO/IEC 9796 can withstand. Below some more attacks to RSA are described, together with requirements on the primes such that the attack cannot be carried out.

Problem 3 ([D83], [D92]):
For certain keys, re-enciphering the message a small number of times restores the original plaintext ([SN77]). Thus, given $C_0 := M^s \bmod n$ and the public key (n, v), a cryptanalist may be able to determine M by computing $C_i = C_{i-1}^v$ for $i = 1, 2, \ldots$ until C_i equals C_0, say for $i = m$. Then C_{m-1} equals M.

Solution 3:
Clearly, this attack is only worthwhile if m is reasonably small. Rivest ([R78]) showed that if p and q are chosen such that $p - 1$ and $q - 1$ have a large prime factor r_p, r_q respectively, where $r_p - 1$ and $r_q - 1$ have large prime factors t_p, t_q respectively, the probability of succes of this type of attack is extremely small. To be more exact, if v is such that

$$v^{\frac{r_p - 1}{t_p}} \neq 1 \bmod r_p \quad \text{and} \quad v^{\frac{r_q - 1}{t_q}} \neq 1 \bmod r_q,$$

then m is divisible by both t_p and t_q, hence it is at least $t_p \cdot t_q$ ([D92]).

Problem 4 ([D83]):
Blakley et all ([BB78], [BB79]) show that for any choice of keys, at least nine plaintext messages will not be concealed by encipherment; that is, for any s and n, $M^s \bmod n = M$ for at least nine M. A poor choice of keys, however, will conceal less than half of all possible plaintext messages. Although the possibility of picking one out of nine such messages is small if messages are 200 digits long, a poor choice of keys will conceal less than 50% of all possible messages.

Solution 4:
The same authors argue that the system will be more resistant to this type of attack if primes p and q are selected such that $p-1=2\tilde{p}$, and $q-1=2\tilde{q}$, with $\tilde{p}$ and $\tilde{q}$ primes.

Problem 5 ([D92]):
The product n can be easy to factor.

Solution 5:

- Choose p and q such that $p-q$ is larger than 2^{75}.
  Otherwise there is an elementary way to factor n.

- Choose p and q of about the same length.
  Of the algorithms that can be used to factor numbers that have no special properties, the quadratic sieve algorithm (or maybe eventually its variant the number field sieve ([LLMP90])) is currently the best. It can factor 350-400 bit numbers in about 60 days using a large number of computers in parallel. The hardest input for such algorithms are randomly chosen numbers consisting of two prime factors of approximately the same length.

- Choose p and q such that neither $p-1$ nor $q-1$ is smooth (i.e. has only small prime factors), concretely they shall have a prime factor of at least 75 bits.
  If $p-1$ or $q-1$ is not smooth, the factoring method suggested by Pollard can be used. The generalisation of Lenstra works if an elliptic curve over p can be found whose order is a smooth number, but this generalisation is very inefficient. However, if the numbers used are about 350-400 bits, a randomly chosen prime will satisfy the condition with very large probability.

- Choose p and q such that neither $p+1$ nor $q+1$ is smooth, or to be exact: $p+1$ and $q+1$ shall have a prime factor of at least 2^{75} bits.
  If n has a prime factor p such that a particular function of p produces a smooth number, the Cyclotomic Polynomial Method suggested by Bach and Shallit can be used ([BS89]). Of the possible choices for this function the ones involving p^2 or larger powers of p are not practical. The function $p-1$ has already been considered, but $p+1$ should additionally be checked for smoothness.

Summarizing the above security problems for certain choices of p and q, and the ways to overcome them, we find the following precautions that should be taken in selecting p and q:

1. p and q should be of approximately the same length, but $p-1$ and $q-1$ should be more than 75 bits;

2. $p-1$, $q-1$, $p+1$ and $q+1$ should contain large prime factors r_p, r_q, s_p and s_q respectively, all of at least 75 bits;

3. The multiplicative order of e modulo $(p-1)(q-1)$ must be large, which is satisfied if r_p-1 and r_q-1 have prime factors t_p and t_q respectively such that

$$e^{\frac{r_p-1}{t_p}} \neq 1 \bmod r_p, \quad \text{and} \quad e^{\frac{r_q-1}{t_q}} \neq 1 \bmod r_q,$$

   and such that $t_p \cdot t_q$ is at least 75 bits long.

The first part of the condition 1. can easily be achieved by the specification of the interval in which *probprime* or *probprimeinc* (see Annex A.2) has to find a prime, and the probability that

the second part is not satisfied is negligible (but can of course easily be checked).

The algorithm *strongprime* ([D92]) described in Annex A.4 generates a prime p chosen at random from the interval I based on a seed s, such that it can be used in RSA with public exponent v satisfying the last two constraints.

2.1.3 Length of the parameters

The public key of a user is the pair (n, v) of which n has 512 bits, and v has w bits.
$w = 0$ if v is the same fixed exponent for each user and $w = 512$ if v can be chosen freely. A typical value for v, suggested by the banking standard [I11166-1], is $v = 2^{16} + 1 (= 2^{2^4} + 1)$: the fourth Fermat number. This exponent v is a large prime, but computing the v-th power takes only 16 squarings and one multiplication.

The "secret key" (n, s) consists of a 512-bit n and a 512-bit s. The modulus n is in fact a public value, but since it is needed to sign messages it has to be a part of the signing key.

The length of the plaintext M as well as the length of the signature M^* is 512 bits.

2.1.4 Security

Breaking RSA is not harder than factoring, because a factoring algorithm automatically gives a cryptanalytic procedure. The fastest general purpose algorithm known to factor n in p and q is one of the multiple variations of the quadratic sieve algorithm ([P85], [S87]), and it takes about

$$e^{\sqrt{\ln(n)\ln(\ln(n))}}$$

steps.
In [LLMP90] the number field sieve factoring algorithm is presented. This algorithm can only be used to factor n of the form $n = r^e - s$, where r is a small positive integer, and s is a non-zero integer of small absolute value. Numbers n of this form can be factored in expected running time of

$$e^{c\sqrt[3]{\ln(n)(\ln(\ln(n)))^2}},$$

where $c \approx 1.526$. The algorithm can be generalized to factor integers of arbitrary form, but the practical consequences remain to be seen ([LM91b]).

2.2 ElGamal (in GF(p), p prime)

2.2.1 Description of the algorithm

As for the RSA algorithm, we will first describe the original ElGamal algorithm ([EG85]). Afterwards we will give a variant algorithm which slightly differs from the first algorithm. This variant reduces the time needed to compute a signature, while it is as least as secure as the original scheme ([AMV89]).

Description of the original algorithm

preliminaries

A prime p and a generator α of GF(p) are selected ([K87]). Generally p is 512 bits, and in the sequel of the description we will assume that p has this length.

Each signing entity secretly and randomly selects a 512-bit number x which will be its secret key. From the secret key x the entity computes its public key y as $y = \alpha^x \bmod p$.

signing

The signing process consists of a preprocessing step, which can be done off-line even before the message to be signed is known, and the actual signature generation.
In the first step, the signing entity chooses a random number k, $0 < k < p$ subject to the condition that $gcd(k, p-1) = 1$, and it computes r as $\alpha^k \bmod p$.

To sign a 512-bit message M, the signing entity computes the number s:

$$s = k^{-1}(M - x \cdot r) \bmod (p-1).$$

Thus,

$$\alpha^M = \alpha^{x \cdot r} \cdot \alpha^{k \cdot s} \bmod p.$$

The signature S for M is the pair (r, s).

verification

Given $\tilde{M}$, $\tilde{r}$ and $\tilde{s}$, a verifier checks if

$$\alpha^{\tilde{M}} = y^{\tilde{r}} \cdot \tilde{r}^{\tilde{s}} \bmod p.$$

If this does not hold, the signature is rejected.

Description of the variant algorithm

To create signatures, the signing entity must solve for s

$$s = k^{-1}(M - x \cdot r) \bmod (p-1).$$

which requires finding k^{-1} in GF(p). The work to be done is that of the Euclidean algorithm. This is comparable to a full exponentiation, thus, in general, time consuming.
In the variant scheme ([AMV89]), s is computed as

$$s = x^{-1}(M - k \cdot r) \bmod (p-1).$$

The advantage is that the signing entity can compute x^{-1} once, and then repeatedly use it to sign messages.

A verifier checks, given $\tilde{M}$, $\tilde{r}$ and $\tilde{s}$, if

$$\alpha^{\tilde{M}} = y^{\tilde{s}} \cdot \tilde{r}^{\tilde{r}} \bmod p.$$

A forger can choose r fixed and try to find the matching s by solving

$$y^s = \alpha^M \cdot r^{-r} \bmod p,$$

which is the discrete logarithm problem. Another attack is to choose s fixed and try to find the matching r from

$$r^r = a^M \cdot y^{-s} \bmod p.$$

This is the problem of finding a solution to $r^r = \beta \bmod p$ for some β in the field. This is in all likelihood more difficult than the discrete logarithm problem itself.

Thus, the variant scheme is not only more efficient for the signer, but also more secure.

2.2.2 Key generation

The algorithm makes use of a 512-bit prime p and a generator α in the field GF(p). Thus, we need a way to generate a prime p, and a generator of the field.
In Annex A.2 and A.2 algorithms are described which generate probable and provable primes respectively. Before using a (probable) prime generated by one of these algorithms it should be checked that p is not chosen as $p = f(a)$ for a polynomial f which has small coefficients and a low degree, because then the number field sieve can compute discrete logarithms ([S92]).

Since p and α can be common to a large group of users, not every signer has to be able to generate them. However, every signer has to be able to generate a random number which it will use as its secret key.

2.2.3 Length of the parameters

Each user has to know p and α, both consisting of 512 bits. These parameters can be the same for a large group of users, but if there are several user groups with different primes and generators of the field, a user must know or be able to obtain the p's and α's of all groups containing a user whose signature he wants to verify.

The public key y has a length of 512 bits, and also the length of the secret key x is 512 bits.

The signature on a 512-bit message consists of (r, s). Both r and s have a length of 512 bits, so the total signature length is 1024 bits.

2.2.4 Security

To break the original system, a forger can choose r fixed and try to find the matching s by solving

$$r^s = a^M \cdot y^{-r} \bmod p,$$

which is the discrete logarithm problem.
Another attack is to choose s fixed and try to find the matching r from

$$r^s \cdot y^r = a^M \bmod p.$$

It is not proved that this is as hard as the discrete logarithm problem, but up to now it is not possible to solve it in polynomial time ([DS91a]).

In the variant scheme, a forger can choose r fixed and try to find the matching s by solving

$$y^s = \alpha^M \cdot r^{-r} \bmod p,$$

which is the discrete logarithm problem. Another attack is to choose s fixed and try to find the matching r from

$$r^r = a^M \cdot y^{-s} \bmod p.$$

This is the problem of finding a solution to $r^r = \beta \bmod p$ for some β in the field, which is believed to be more difficult than the discrete logarithm problem itself.

We can conclude that the security of the ElGamal signature scheme is strongly dependent on the difficulty of computing discrete logarithms. The computation of the discrete logarithm in GF(p) for p prime takes about

$$e^{\sqrt{\ln(p)\ln(\ln(p))}}$$

steps as $p \to \infty$ ([LMO91]). This estimate is the same as that for factoring large integers of the same size as p which have no special features (see the previous section).

2.2.5 Fields of characteristic 2

The ElGamal signature algorithm is based on the difficulty of computing discrete logarithms in a finite field. In the foregoing we considered only the field GF(p) for p prime. However, the scheme can easily be adapted to work in all finite fields GF(q^n), q prime and n an integer larger than 1. Especially the field GF(2^n) deserves some attention, since arithmetic in GF(2^n) for large n can be performed faster than arithmetic over large primes ([MPW92]).

[vO92] contains an extensive description of the practical methods known to compute discrete logarithms in GF(2^n). His analysis gives a (heuristic expected) asymptotic running time proportional to

$$e^{c \cdot n^{1/3}(\ln n)^{2/3}} \quad \text{where} \quad c \approx 1.35.$$

The same paper also compares the ElGamal system in GF(2^n) and the RSA system using an n-bit modulus N. As a comparision of the running times of the best currently known algorithms of computing discrete logarithms in GF(2^n) and factoring of n-bit integers shows, the ElGamal algorithm is less secure than RSA when the same modulus length is used. This can be compensated for by using a larger bitlength in the ElGamal scheme. As a rough guideline, to match the level of security offered by 512-bit RSA, n should be chosen about 750 in the ElGamal scheme. However, the fields GF(2^n) can be chosen such that they still admit efficient arithmetic. The price to pay for the larger modulus size is somewhat larger key sizes, and greater bandwidth and storage requirements.

2.3 DSA

DSA is an abbreviation for Digital Signature Algorithm. The same algorithm is also known under the name DSS: Digital Signature Standard. The algorithm is under standardization by NIST ([NIST91]) in the US.

2.3.1 Description of the algorithm

preliminaries

The following numbers are selected:

- a prime p, generally 512 bits;
- a prime divisor q of $p-1$, generally 160 bits;
- an integer g defined as $h^{(p-1)/q} \bmod p$, where h is any integer with $1 < h < p$ such that the number g is larger than 1 (a non trivial q-th root of unity).

The prime p can be common for all, or a large group of, users, or it can be individual for each user. Both possibilities have their own advantages and disadvantages. A common p makes computing discrete logarithms modulo this prime p more lucrative. But if each user has its individual p, then this prime has to be a part of this user's public key, which increases the storage or communication of the public key.

Each signing entity secretly and randomly selects an integer x, $1 < x < q$, which will be its secret key. From the secret key x the entity computes its public key y as $y = g^x \bmod p$.

signing

The signing process consists of a preprocessing step, which can be done off-line even before the message to be signed is known, and the actual signature generation.

In the first step, the signing entity chooses a random number k, $0 < k < q$, and it computes r as $r = (g^k \bmod p) \bmod q$.

To sign a 160-bit message M, the signing entity computes the number s:

$$s = k^{-1}(M + x \cdot r) \bmod q.$$

If $s = 0$ it can not be used (the verifier has to compute the inverse of s), thus if s happens to be zero, the signer has to start over again by choosing another k at random ([A92]). The signature S for M is the pair (r, s).

verification

Given $\tilde{M}$, $\tilde{r}$ and $\tilde{s}$, a verifier first checks if $0 < \tilde{r} < q$ and $0 < \tilde{s} < q$; if either condition is violated, the signature is rejected.

Otherwise the following numbers are computed:

- $w = \tilde{s}^{-1} \bmod q$;
- $u_1 = (\tilde{M} \cdot w) \bmod q$;
- $u_2 = (\tilde{r} \cdot w) \bmod q$;
- $v = ((g^{u_1} \cdot y^{u_2}) \bmod p) \bmod q$.

If $v = \tilde{r}$ the signature is accepted.
Indeed, if $\tilde{M} = M$, $\tilde{r} = r$ and $\tilde{s} = s$, then

$$\begin{aligned} v &= ((g^{Mw \bmod q} \cdot (g^x)^{rw \bmod q}) \bmod p) \bmod q \\ &= (g^{(M+xr)w \bmod q} \bmod p) \bmod q \\ &= (g^k \bmod p) \bmod q \\ &= r. \end{aligned}$$

2.3.2 Key generation

The algorithm makes use of a prime p and a prime q which is a divisor of $p-1$. Thus, we need a way to generate those two primes.
In Annex A.2 and A.3 are algorithms described that generate probable and provable primes respectively. Note that also is described how it can be guaranteed that q divides $p-1$.
Before using the (probable) prime p generated by one of these algorithms it should be checked that p is not chosen as $p = f(a)$ where for a polynomial f which has small coefficients and a low degree, because then the number field save can compute discrete logarithms ([S92]).

The standard suggests to find the integer g needed by trial and error.
Given p and q the algorithm is:
1. set h, $1 < h < p-1$, equal to a randomly chosen number;
2. set t equal to $h^{(p-1)/q} \bmod p$;
3. if $t = 1$ go to step 1, otherwise set g equal to t.

Since p, q and g can be common to a large group of users, not every signer has to be able to generate them. However, every signer has to be able to generate a random number that will be its secret key.

2.3.3 Length of the parameters

If the length of p, q and g are chosen acoording to the recommendation in the standard, the triple (p, q, g) consists of (512,160,512) bits. This triple can be common for a group of users. If there are several user groups with different triples, a user must know the p's, q's and g's of all groups containing a user whose signature he wants to verify.

The public key y of a user consist of 512 bits, and the secret key x of a signer is (only) 160 bits long.

The signature pair (r, s) for a 160-bit message is 320 bits long: both r and s are 160-bit numbers.

2.3.4 Security

The security of the DSA signature scheme is based on the difficulty of computing discrete logarithms. The computation of the discrete logarithm in GF(p) for p prime takes

$$e^{\sqrt{\ln(p)\ln(\ln(p))}}$$

steps as $p \to \infty$ ([LMO91]).

2.4 Comparision of the schemes

In the table below the schemes described in the three previous sections will be compared with respect to security and complexity ([vO91], [vO92], [S92]). We have chosen n to be an 512-bit modulus in the RSA scheme, p to be 512 bits in both the ElGamal and DSA algorithm, and q is fixed to 160 bits in DSA.

When the signer can compute part of the signature before the message to be signed is known to him, the number of modular multiplications that can be precomputed is denoted by the addition "(pre)".

	security; fastest algorithm[1]	# modular multiplications	space (in bits)	signature length (in bits)
RSA	factoring; $\exp\left(\sqrt{\ln(n)\ln(\ln(n))}\right)$	signer[2] 30 (pre) 608 verifier 638	secret 1024 public 1024	512
RSA ($v = 3$)	factoring; $\exp\left(\sqrt{\ln(n)\ln(\ln(n))}\right)$	signer[2] 30 (pre) 608 verifier 2	secret 1024 public 512	512
RSA ($v = 2^{16} + 1$)	factoring; $\exp\left(\sqrt{\ln(n)\ln(\ln(n))}\right)$	signer[2] 30 (pre) 608 verifier 17	secret 1024 public 512	512
ElGamal (in GF(p))	discr. log; $\exp\left(\sqrt{\ln(p)\ln(\ln(p))}\right)$	signer[3] 608 (pre) 1 verifier[3] 893	general 1024 secret 512 public 512	1024
DSA	discr. log; $\exp\left(\sqrt{\ln(p)\ln(\ln(p))}\right)$	signer[4] 198 (pre) 1 verifier[4] 277	general 1184 secret 160 public 512	320

Notes:

(1) In practice the computation of discrete logarithms in GF(p) using the best currently known techniques is slightly harder than the factorization on n via the multiple polynomial quadratic sieve ([LMO91]).

(2) This number of multiplications is needed when the signer makes no use of the Chinese Remainder Theorem ([K87]). Using this Theorem, the number of multiplications can be reduced, but the price to pay is that the primes p and q have to be stored instead of n. Thus, 512 bits of secret storage are needed instead of 512 bits that may be public.

(3) If the signer has an auxiliarly storage of 12416 bytes, only 112 off-line multiplications are needed, and if the verifier has an auxiliarly storage of 12416 bytes, the number of multiplications he needs to perform reduces to 693 ([S92]).

(4) An auxiliarly storage of 4864 bytes reduces the number of off-line multiplications for the signer to 43, and the number of multiplications for the verifier to 221 ([S92]).

Chapter 3

Hash functions

3.1 MD5

The MD5 Message-Digest Algorithm ([RD91]) was designed by R. Rivest and S. Dusse in 1991 to be quite fast on 32-bit machines. It is an extension of the MD4 algorithm ([R91]), which was designed to be very fast and is "at the edge" of withstanding a cryptanalytic attack. The MD5 algorithm gives up a little bit in speed for achieving a much greater likelyhood of ultimate security.

3.1.1 Description of the algorithm

MD5 uses four 32-bit registers and four functions which join these registers by means of the boolean functions AND, OR, exclusive OR, and NOT, resulting in a 32-bit word. Starting with a b-bit message $m_0 \ldots m_{b-1}$ the algorithms is as follows:

step 1
Append padding: add "1" and add $0 \leq j < 512$ zeros such that $b + j = 448 \bmod 512$. Padding is always performed, even if the length of the message is already congruent to 448 modulo 512 (in which case 512 padding bits are added).

step 2
Append length: append the binary presentation of b (modulo 2^{64}), resulting in the n-word message $M[0 \ldots n-1]$ (a word is 32 bits, and $16|n$).

step 3
Initialize the registers to the following values, which are given in hexadecimal, low-order bytes first:

word a : 01 23 45 67
word b : 89 AB CD EF
word c : FE DC BA 98
word d : 76 54 32 10

step 4
Process the message in 16-word blocks.

First some notation will be defined (X, Y words):

$X + Y$ means modulo 2^{32} addition of the words X and Y;

$X <<< s$ denotes the 32-bit value obtained by circulary shifting (rotating) X left bij s positions;

$\bar{X}$ denotes the bitwise complement of X;

$X \vee Y$ stands for the bitwise OR of X and Y;

X XOR Y denotes the bitwise XOR of X and Y;

XY is the notation for the bitwise AND of X and Y.

Using this notation four functions that take as input three words, and output one word are defined:

F: $F(X,Y,Z) = XY \vee \bar{X}Z$

G: $G(X,Y,Z) = XZ \vee Y\bar{Z}$

H: $H(X,Y,Z) = X$ XOR Y XOR Z

I: $I(X,Y,Z) = Y$ XOR $(X \vee \bar{Z})$

Define the additative constant t by the integer part of 4294967296 times abs(sin(i)) for i in radians.
Define:
S11 = 7, S12 = 12, S13 = 17, S14 = 22,
S21 = 5, S22 = 9, S23 = 14, S24 = 20,
S31 = 4, S32 = 11, S33 = 16, S34 = 23,
S41 = 6, S42 = 10, S43 = 15, S44 = 21.

Using the functions F, G, H and I, t and the Sij four other functions are defined (a, b, c and d are words):

FF: $FF(W,X,Y,Z,X[k],s,t)$ denotes
$W = X + ((W + F(X,Y,Z) + X[k] + t) <<< s)$;

GG: $GG(W,X,Y,Z,X[k],s,t)$ denotes
$W = X + ((W + G(X,Y,Z) + X[k] + t) <<< s)$;

HH: $HH(W,X,Y,Z,X[k],s,t)$ denotes
$W = X + ((W + H(X,Y,Z) + X[k] + t) <<< s)$;

II: $II(W,X,Y,Z,X[k],s,t)$ denotes
$W = X + ((W + I(X,Y,Z) + X[k] + t) <<< s)$.

Then the following algorithm is executed:

For $i = 0$ to $n/16 - 1$ do:

> For $j = 0$ to 15 do
> Set $X[j]$ to $M[16i + j]$
> end;
>
> $aa = a$;
> $bb = b$;
> $cc = c$;
> $dd = d$;

$FF(a, b, c, d, X[0], S11, 3614090360);$
$FF(d, a, b, c, X[1], S12, 3905402710);$
$FF(c, d, a, b, X[2], S13, 606105819);$
$FF(b, c, d, a, X[3], S14, 3250441966);$
$FF(a, b, c, d, X[4], S11, 4118548399);$
$FF(d, a, b, c, X[5], S12, 1200080426);$
$FF(c, d, a, b, X[6], S13, 2821735955);$
$FF(b, c, d, a, X[7], S14, 4249261313);$
$FF(a, b, c, d, X[8], S11, 1770035416);$
$FF(d, a, b, c, X[9], S12, 2336552879);$
$FF(c, d, a, b, X[10], S13, 4294925233);$
$FF(b, c, d, a, X[11], S14, 2304563134);$
$FF(a, b, c, d, X[12], S11, 1804603682);$
$FF(d, a, b, c, X[13], S12, 4254626195);$
$FF(c, d, a, b, X[14], S13, 2792965006);$
$FF(b, c, d, a, X[15], S14, 1236535329);$

$GG(a, b, c, d, X[1], S21, 4129170786);$
$GG(d, a, b, c, X[6], S22, 3225465664);$
$GG(c, d, a, b, X[11], S23, 643717713);$
$GG(b, c, d, a, X[0], S24, 3921069994);$
$GG(a, b, c, d, X[5], S21, 3593408605);$
$GG(d, a, b, c, X[10], S22, 38016083);$
$GG(c, d, a, b, X[15], S23, 3634488961);$
$GG(b, c, d, a, X[4], S24, 3889429448);$
$GG(a, b, c, d, X[9], S21, 568446438);$
$GG(d, a, b, c, X[14], S22, 3275163606);$
$GG(c, d, a, b, X[3], S23, 4107603335);$
$GG(b, c, d, a, X[8], S24, 1163531501);$
$GG(a, b, c, d, X[13], S21, 2850285829);$
$GG(d, a, b, c, X[2], S22, 4243563512);$
$GG(c, d, a, b, X[7], S23, 1735328473);$
$GG(b, c, d, a, X[12], S24, 2368359562);$

$HH(a, b, c, d, X[5], S31, 4294588738);$
$HH(d, a, b, c, X[8], S32, 2272392833);$
$HH(c, d, a, b, X[11], S33, 1839030562);$
$HH(b, c, d, a, X[14], S34, 4259657740);$
$HH(a, b, c, d, X[1], S31, 2763975236);$
$HH(d, a, b, c, X[4], S32, 1272893353);$
$HH(c, d, a, b, X[7], S33, 4139469664);$
$HH(b, c, d, a, X[10], S34, 3200236656);$
$HH(a, b, c, d, X[13], S31, 681279174);$
$HH(d, a, b, c, X[0], S32, 3936430074);$
$HH(c, d, a, b, X[3], S33, 3572445317);$
$HH(b, c, d, a, X[6], S34, 76029189);$
$HH(a, b, c, d, X[9], S31, 3654602809);$
$HH(d, a, b, c, X[12], S32, 3873151461);$
$HH(c, d, a, b, X[15], S33, 530742520);$
$HH(b, c, d, a, X[2], S34, 3299628645);$

$II(a, b, c, d, X[0], \text{S41}, 4096336452);$
$II(d, a, b, c, X[7], \text{S42}, 1126891415);$
$II(c, d, a, b, X[14], \text{S43}, 2878612391);$
$II(b, c, d, a, X[5], \text{S44}, 4237533241);$
$II(a, b, c, d, X[12], \text{S41}, 1700485571);$
$II(d, a, b, c, X[3], \text{S42}, 2399980690);$
$II(c, d, a, b, X[10], \text{S43}, 4293915773);$
$II(b, c, d, a, X[1], \text{S44}, 22400444497);$
$II(a, b, c, d, X[8], \text{S41}, 1873313359);$
$II(d, a, b, c, X[15], \text{S42}, 4264255552);$
$II(c, d, a, b, X[6], \text{S43}, 2734768916);$
$II(b, c, d, a, X[13], \text{S44}, 1309151649);$
$II(a, b, c, d, X[4], \text{S41}, 4149444226);$
$II(d, a, b, c, X[11], \text{S42}, 3174756917);$
$II(c, d, a, b, X[2], \text{S43}, 718787259);$
$II(b, c, d, a, X[9], \text{S44}, 3951481745);$

$a = a + aa;$
$b = b + bb,$
$c = c + cc;$
$d = d + dd:$

end

step 5
The message digest produced as output is *abcd*. That is, begin with the low order byte of a, and end with the high orde byte of d.

3.1.2 Length of the parameters

The input blocks have a length of 512 bits, and the output is 128 bits long. Four 32-bit registers have to be initialized with given values, which gives an initializing value of 128 bits.

3.2 DIS 10118-2

In the standard ISO/IEC 10118-2, two ways to obtain a hash function using an n-bit block cipher algorithm are described. With an n-bit block cipher is meant a block cipher in which both plaintext block and ciphertext block have a length of n bits.
The first method results in an hash function with an output length equal to the blocklength n of the algorithm, and the function is therefore called a single length hash function. The output length of the hash function obtained by the second method is $2n$, and, as could be expected, the hash function is called a double length hash function.

3.2.1 Description of the single length hash function

Figure 3.1 gives a sketch of the algorithm, and the text describes it in more detail:

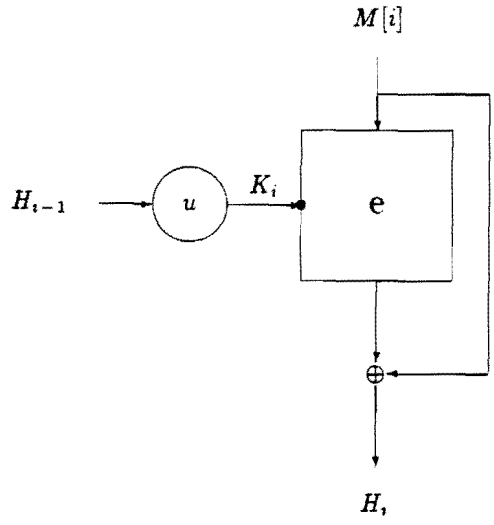


Figure 3.1: Single length hash function

step 1

Append padding to the original message M; two examples of padding methods are:

- method 1
  append $0 \leq j < n$ zeros such that the message length is congruent to 0 modulo n.
- method 2
  add a "1" and add $0 \leq j < n$ zeros such that the message length is congruent to 0 modulo n.

The result is the nq-bit message $M[1 \ldots q]$, where the i-th n-bit message block is denoted by $M[i]$.

step 2

Compute for i from 1 to q the output blocks H_i in an iterative way. For this we need an n-bit block cipher algorithm e, and a function u which transforms an n-bit message block into the keyset of e.
Set H_0 equal to the initializing value IV, and compute for $i = 1, \ldots, n$:
$K_i := u(H_{i-1})$;
$H_i := \mathsf{e}(K_i, M[i]) \oplus M[i]$.

step 3

The hash result is obtained by taking the final output block H_q.

3.2.2 Description of the double length hash function

Figure 3.2 below gives a sketch of the algorithm, and the text describes it more detailed:

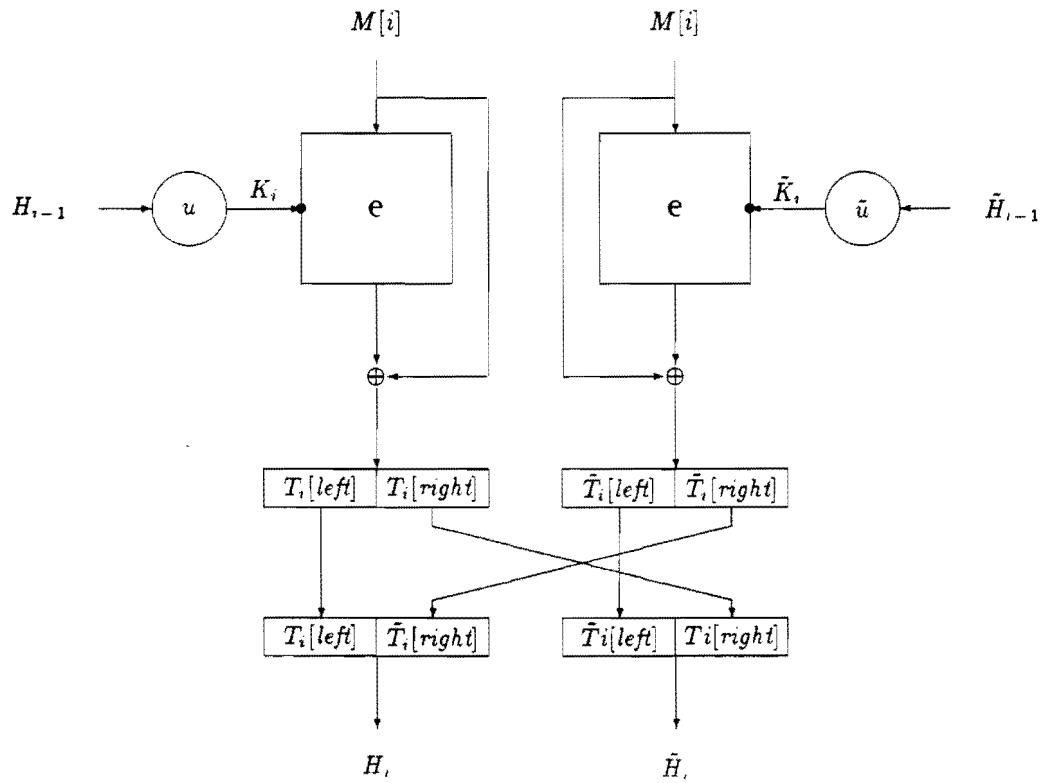


Figure 3.2: Double length hash function

step 1

Append padding to the original message M; two examples of padding methods are given in the description of the single length hash function.
The result is the nq-bit message $M[1 \ldots q]$, where the i-th n-bit message block is denoted by $M[i]$.

step 2

Compute for i from 1 to q the output blocks H_i in an iterative way. For this we need an n-bit block cipher algorithm e, and two functions u and $\tilde{u}$ which transform an n-bit message block into the keyset of e.
Define for an n-bit vector X, $X[left]$ and $X[right]$ as the leftmost $\lceil n/2 \rceil$ and rightmost $\lfloor n/2 \rfloor$ bits of X respectively.
Set H_0 and $\tilde{H}_0$ equal to the initializing values IV and $\tilde{IV}$ respectively, and compute for $i = 1, \ldots, n$:

$K_i := u(H_{i-1})$ and $\tilde{K}_i := \tilde{u}(\tilde{H}_{i-1})$.
$T_i := \mathsf{e}(K_i, M[i]) \oplus M[i]$ and $\tilde{T}_i := \mathsf{e}(\tilde{K}_i, M[i]) \oplus M[i]$.
$H_i := T_i[left] \,||\, \tilde{T}_i[right]$ and $\tilde{H}_i := \tilde{T}_i[left] \,||\, T_i[right]$.

step 3

The hash result is obtained by concatenating H_q and $\tilde{H}_q$.

3.2.3 DES

In an Annex of [?] the algorithm is described for the case where DES is chosen as block cipher algorithm. Here also the initializing vector(s) IV (and $\tilde{IV}$), and the transformation(s) u and $(\tilde{u})$ are fixed. The single and the double lenght hash function employing DES are described below.

Single length hash function

The algorithm is as follows:

step 1

Append padding, resulting in the $64q$-bit message $M[1 \ldots q]$.

step 2

Compute for i from 1 to q the output blocks H_i in an iterative way.

In the description of the computation, we use the following notation:
$B_q := \{0,1\}^q$, i.e. a vector of q bits;
Furthermore, for h_i a hexadecimal digit,
$B(h_1h_2\ h_3h_4\ h_5h_6\ h_7h_8\ h_9h_{10}\ h_{11}h_{12}\ h_{13}h_{14}\ h_{15}h_{16})$ denotes its 64-bit binary representation.

Define the initializing value IV as $B(52\ 52\ 52\ 52\ 52\ 52\ 52\ 52)$.
Define $u : B_{64} \rightarrow B_{56}$ by $u(x_1x_2 \ldots x_{64}) := x_1 10 x_4 x_5 x_6 x_7 x_9 \ldots x_{63})$, i.e. x_2x_3 is forged to 10 and bits $x_8, x_{16}, \ldots x_{64}$ are removed.
Define $\mathrm{DES}(K, X)$ as the DES encryption of the 64-bit vector X under the 64-bit key K.

Compute $K_i := u(H_{i-1})$. Compute $H_i := \mathrm{DES}(K_i, M[i]) \oplus M[i]$.

step 3

The hash result is obtained by taking the final output block H_q.

Double length hash function

The algorithm is as follows:

step 1
Append padding resulting in the $64q$-bit message $M[1 \dots q]$.

step 2
Compute for i from 1 to q the output blocks H_i in an iterative way.
In the description of the computation, we use the following notation:
$B_q := \{0,1\}^q$, i.e. a vector of q bits;
Furthermore, for h_i a hexadecimal digit,
$B(h_1h_2\ h_3h_4\ h_5h_6\ h_7h_8\ h_9h_{10}\ h_{11}h_{12}\ h_{13}h_{14}\ h_{15}h_{16})$ denotes its 64-bit binary representation.

Define the initializing value IV as $B(52\ 52\ 52\ 52\ 52\ 52\ 52\ 52)$.
Define the initializing value $\tilde{I}\tilde{V}$ as $B(25\ 25\ 25\ 25\ 25\ 25\ 25\ 25)$.

Define $u : B_{64} \rightarrow B_{56}$ by $u(x_1x_2 \dots x_{64}) := x_110x_4x_5x_6x_7x_9 \dots x_{63})$, i.e. x_2x_3 is forged to 10 and bits $x_8, x_{16}, \dots x_{64}$ are removed.
Define $\tilde{u} : B_{64} \rightarrow B_{56}$ by $u(x_1x_2 \dots x_{64}) := x_101x_4x_5x_6x_7x_9 \dots x_{63})$, i.e. x_2x_3 is forged to 01 and bits $x_8, x_{16}, \dots x_{64}$ are removed.

Define $\mathrm{DES}(K, X)$ as the DES encryption of the 64-bit vector X under the 64-bit key K.

Compute $K_i := u(H_{i-1})$ and $\tilde{K}_i := \tilde{u}(\tilde{H}_{i-1})$.
Compute $T_i := \mathrm{DES}(K_i, M[i]) \oplus M[i]$ and $\tilde{T}_i := \mathrm{DES}(\tilde{K}_i, M[i]) \oplus M[i]$.
Compute $H_i := T_i[left] \parallel \tilde{T}_i[right]$ and $\tilde{H}_i := \tilde{T}_i[left] \parallel T_i[right]$.

step 3
The hash result is obtained by concatenating H_q and $\tilde{H}_q$.

3.2.4 IPES

IPES is an abbreviation for Improved Proposed Encryption Standard. This is a 64-bit block cipher proposed by X. Lai and J.M. Massey in 1991. It is also possible to choose for e this block cipher. Since IPES is less well known than DES, the algorithm is shown in figure 3.3. See for more details [LM91a] and [L91].

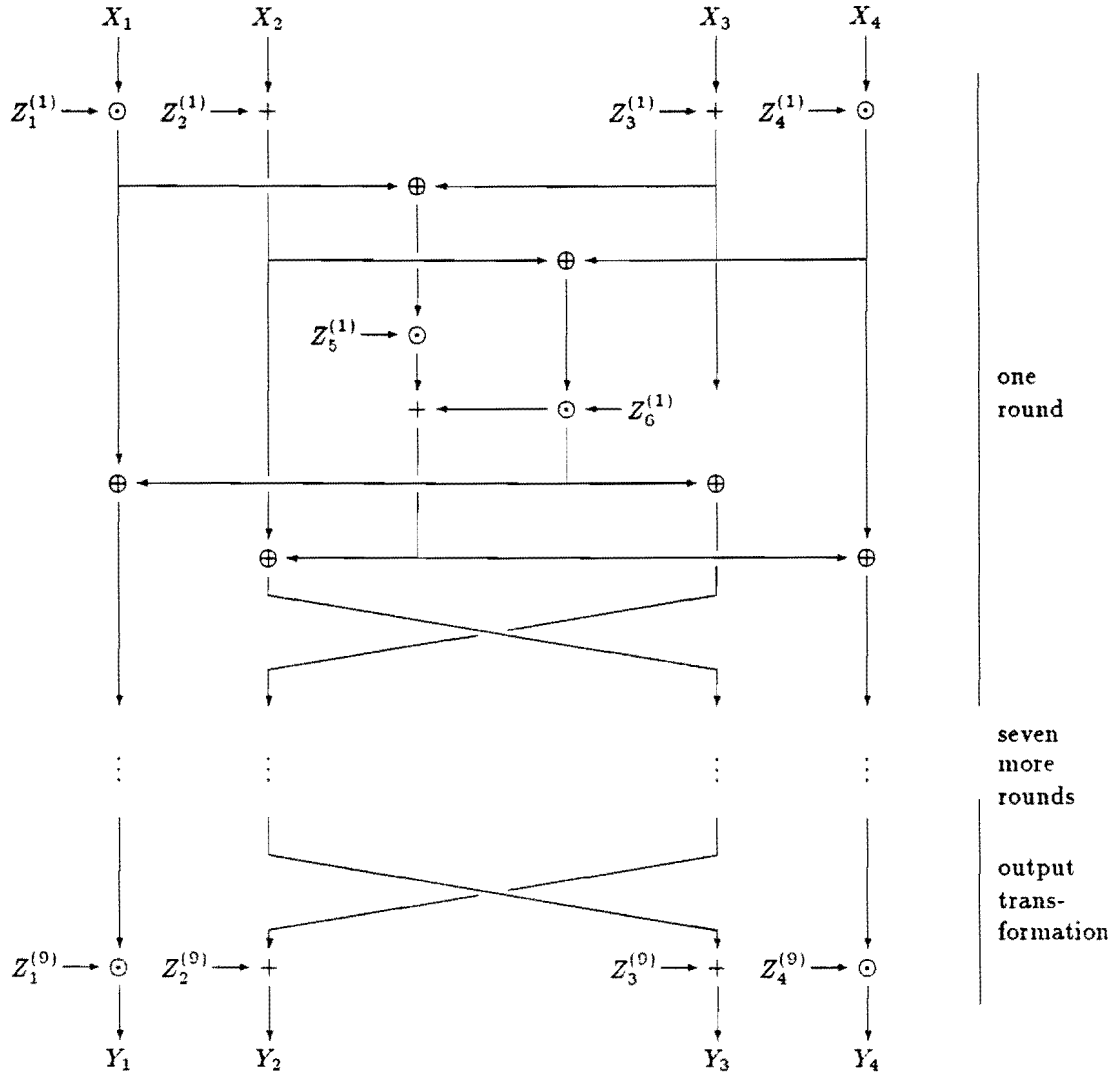
A 64-bit plaintext block is processed as shown in the figure.
The computational graph of the decryption process is exactly the same, the only change occurs in the decryption key subblocks. They are computed from the encryption key subblocks and used in reverse order, as shown in the table below:

	encryption key subblocks	decryption key subblocks
i-th round $(1 \leq i \leq 8)$	$Z_1^{(i)}\ Z_2^{(i)}\ Z_3^{(i)}\ Z_4^{(i)}$ $Z_5^{(i)}\ Z_6^{(i)}$	$Z_1^{(10-i)^{-1}}\ -Z_2^{(10-i)}\ -Z_3^{(10-i)}\ Z_4^{(10-i)^{-1}}$ $Z_5^{(9-i)}\ Z_6^{(9-i)}$
output transform.	$Z_1^{(9)}\ Z_2^{(9)}\ Z_3^{(9)}\ Z_4^{(9)}$	$Z_1^{(1)^{-1}}\ -Z_2^{(1)}\ -Z_3^{(1)}\ Z_4^{(1)^{-1}}$

The 52 key subblocks used in the encryption algorithm are generated from an 128-bit user selected key. To describe how this is done, first an ordering of the key subblocks is defined. From the first to the last, the key subblocks are:
$Z_1^{(1)}, Z_2^{(1)}, \ldots, Z_6^{(1)}, Z_1^{(2)}, \ldots, Z_6^{(2)}, \ldots, Z_6^{(8)}, Z_1^{(9)}, Z_2^{(9)}, Z_3^{(9)}, Z_4^{(9)}$.
The 128-bit user selected key is partitioned into 8 subblocks that are directly used as the first eight key subblocks. Then the user selected key is cyclic shifted to the right by 25 positions, after which the resulting 128-bit block is again partitioned into eight subblocks that are taken as the next eight key subblocks. The obtained 128-bit block is again cyclic shifted to the left by 25 positions to produce the next eight key subblocks, and this procedure is repeated until all 52 key subblocks have been generated.

3.2.5 Length of the parameters

The length of the input blocks is for as well the single length as the double length hash function equal to n.
The output length is n and $2n$ for the single length and double length hash function respectively. The single length hash function requires an initializing value of n bits, and the double length hash function two initializing values of n bits, thus in total $2n$ bits.
Furthermore, the block cipher algorithm needs a key which length is dependent on the algorithm used. For the single length hash function one key is needed, for the double length algorithms two. Thus, for the single length and the double length hash function the algorithm using DES needs 64 and 128 keybits respectively, and when IPES is used 128 and 256 keybits are needed.



The meaning of the variables is:

X_i : 16-bit plaintext subblock

Y_i : 16-bit plaintext subblock

$Z_i^{(r)}$: 16-bit key subblock

$\oplus$: bit by bit exclusive OR of 16-bit subblocks

$+$: addition modulo 2^{16} of 16-bit integers

$\odot$: multiplication modulo $2^{16} + 1$ of 16- bit integers, **where the zero subblock** corresponds to 2^{16}

Figure 3.3: Encryption process of IPES

3.3 SHA

SHA is an abbreviation for Secure Hash Algorithm. The algorithm is also known under the abbrevation SHS: Secure Hash Standard. SHA is under standardization by NIST ([NIST92a]). The algorithm is based on principles similar to those used when designing MD4 ([R91]).

3.3.1 Description of the algorithm

SHA is designed for 32-bit machines, and uses five 32-bit registers, four different functions and four constants. The functions join the contents of some of the registers and a round counter using the boolean functions AND, OR, exclusive OR and NOT. Starting with a b-bit message $m_0 \ldots m_{b-1}$ the algorithm is as follows:

step 1
Append padding: add a "1" and add $0 \leq j < 512$ zeros such that $b + j = 448 \bmod 512$. Note that padding is always performed, even if the message length is already equal to 448 modulo 512. In that case one "1" and 511 zeros are added.

step 2
Append length: append the binary presentation of b (modulo 2^{64}), resulting in the n-word message $M[1 \ldots n]$ (a word is 32 bits, and $16|n$).

step 3
Initialize the registers to the following values, which are given in hexadecimal, high-order bytes first:

h0 : 67 45 23 01
h1 : EF CD AB 89
h2 : 98 BA DC FE
h3 : 10 32 54 76
h4 : C3 D2 E1 F0

step 4
Process the message in 16-word blocks.

First some notation will be defined (X, Y words):

$X + Y$ means modulo 2^{32} addition of the words X and Y;

$\bar{X}$ denotes the bitwise complement of X;

$X \vee Y$ stands for the bitwise OR of X and Y;

X XOR Y denotes the bitwise XOR of X and Y;

XY is the notation for the bitwise AND of X and Y.

Furthermore a function $S(n, X)$ is defined on a word X and an integer n, $0 \leq n < 32$. $S(n, X)$ denotes a circular shift of X by n positions to the left.

Using this notation a sequence of logical functions $f(0, x, y, z), \ldots, f(79, x, y, z)$ is defined. Each f operates on three 32-bit words and produces a 32-bit word as output as follows:

$f(t, x, y, z) = xy \vee \bar{x}z$ $\quad (0 \leq t \leq 19)$,
$f(t, x, y, z) = x$ XOR y XOR z $\quad (20 \leq t \leq 39)$,
$f(t, x, y, z) = xy \vee xz \vee yz$ $\quad (40 \leq t \leq 59)$,
$f(t, x, y, z) = x$ XOR y XOR z $\quad (60 \leq t \leq 79)$.

Also a sequence of constant words $K(0), K(1), \ldots K(79)$ is used in the SHA. In hexadecimal notation these are given by:

$K(t) = 5A827999 \quad (0 \leq t \leq 19)$,
$K(t) = 6ED9EBA1 \quad (20 \leq t \leq 39)$,
$K(t) = 8F1BBCDC \quad (40 \leq t \leq 59)$,
$K(t) = CA62C1D6 \quad (60 \leq t \leq 79)$.

The computation uses two buffers, each consisting of five 32-bit words, and a sequence of eighty 32-bit words. The words of the first buffer are labeled a, b, c, d, e, and the words of the second buffer are labeled $h0$, $h1$, $h2$, $h3$, $h4$. The words of the sequence are labeled $W(0), W(1), \ldots, W(79)$. A single word buffer TEMP is also employed.

To generate the message digest, the 16-word blocks $M(1), M(2), \ldots M(n)$ defined in step 2 are processed in order. The processing of each $M(i)$ involves 80 steps, and goes as follows:

1. Divide $M(i)$ into 16 words $W(0), W(1), \ldots W(15)$, where $W(0)$ is the leftmost word.
2. Define 64 more words by $W_t = W_{t-3} \oplus W_{t-8} \oplus W_{t-14} \oplus W_{t-16}$ for $16 \leq t \leq 79$.
3. Let $a = h0$, $b = h1$, $c = h2$, $d = h3$ and $e = h4$.
4. For $t = 0$ to 79 do
   TEMP $= S(5, a) + f(t, b, c, d) + e + W(t) + K(t)$;
   $e = d$;
   $d = c$
   $c = S(30, b)$;
   $b = a$;
   $a =$ TEMP.
5. Let $h0 = h0 + a$, $h1 = h1 + b$, $h2 = h2 + c$, $h3 = h3 + d$, $h4 = h4 + e$.

step 5

When all blocks are processed, the hash result is the 160-bit string represented by the 5 words

$$h0\ h1\ h2\ h3\ h4.$$

3.3.2 Length of the parameters

The input blocks for this hash function have a length of 512 bits, while the output block is 160 bits long.
Five 32-bits registers have to be initialized with predescribed values, which gives an initializing value of 160 bits. Besides that in each of the rounds a 32-bit round dependent constant is used. Four different ones are defined, which gives 128 bits in total.

3.4 AR/DFP

In 1992, Algorithmic Research has developed the Digital FingerPrint hash function.

3.4.1 Description of the algorithm

The algorithm performs the following computations:

start

b-bit message $m_0 \dots m_{b-1}$.

step 1

padding: add $0 \leq j < 63$ zeros such that $b + j = 0 \bmod 64$, denote the resulting message by $M[1, ...n]$ $(n = \lceil b/64 \rceil)$.

step 2

compute the hash result DFP(M).

In the description of the algorithm, we use the following notation:
$B_n := \{0,1\}^n$, i.e. a vector of n bits;
$B_* :=$ a bit vector of arbitrary length.
Furthermore, for h_i, $1 \leq i \leq 16$, a hexadecimal digit,
$B(h_1h_2\ h_3h_4\ h_5h_6\ h_7h_8\ h_9h_{10}\ h_{11}h_{12}\ h_{13}h_{14}\ h_{15}h_{16})$ denotes its 64-bit binary representation.

Define
$x := B(01\ 23\ 45\ 67\ 89\ AB\ CD\ EF)$;
$k_1 := B(00\ 00\ 00\ 00\ 00\ 00\ 00\ 00)$;
$k_2 := B(2A\ 41\ 52\ 2F\ 44\ 46\ 50\ 2A)$;
$f_1 : B_{64} \times B_* \rightarrow B_{64}$ such that $f_1(k, M) = b(k, M, n-1)$;
$f_2 : B_{64} \times B_* \rightarrow B_{64}$ such that $f_2(k, M) = b(k, M, n)$,
where
$b(k, M, i)$ is recursively defined by
$b(k, M, -1) = b(k, M, 0) = B(00\ 00\ 00\ 00\ 00\ 00\ 00\ 00)$;
$b(k, M, i) = M[i] \oplus \mathrm{DES}(k, M[i] \oplus b(k, M, i-1) \oplus b(k, M, i-2) \oplus x)$ for $1 \leq i \leq n$.
where DES: $B_{64} \times B_{64} \rightarrow B_{64}$ is such that DES(k, b) is the enciphering of b using the key k according to DES.

Define
$c_1 := f_1(k_1, M) = b(k_1, M, n-1) =$
$\quad M[n-1] \oplus \mathrm{DES}(k_1, M[n-1] \oplus b(k_1, M, n-2) \oplus b(k_1, M, n-3) \oplus x)$;
$c_2 := f_2(k_1, M) = b(k_1, M, n) =$
$\quad M[n] \oplus \mathrm{DES}(k_1, M[n] \oplus b(k_1, M, n-1) \oplus b(k_1, M, n-2) \oplus x)$;
$c_3 := f_1(k_2, M) = b(k_2, M, n-1) =$
$\quad M[n-1] \oplus \mathrm{DES}(k_2, M[n-1] \oplus b(k_2, M, n-2) \oplus b(k_2, M, n-3) \oplus x)$;
$c_4 := f_2(k_2, M) = b(k_2, M, n) =$
$\quad M[n] \oplus \mathrm{DES}(k_2, M[n] \oplus b(k-2, M, n-1) \oplus b(k-2, M, n-2) \oplus x)$.

Define also $G : B_{64} \times B_{64} \times B_{64} \rightarrow B_{64}$ by
$G(k, x, y) = \mathrm{DES}(k, x \oplus y) \oplus DES(k, x) \oplus DES(k, y) \oplus y$.

Finally, define
$F_1(M) = G(k_1, G(k_1, c_1, c_2), G(k_1, c_3, c_4))$;
$F_2(M) = G(k_2, G(k_1, c_1, c_2), G(k_2, c_3, c_4))$.

Now $\mathrm{DFP}(M) := (F_1(M), F_2(M))$.

The computation of G, the F_i and DFP is depicted in figure 3.4.

3.4.2 Length of the parameters

The length of the input blocks is 64 bits, and the length of the output blocks is the concatenation of two 64-bit blocks, thus has a length of 128 bits.
The algorithm uses two 64-bit DES keys, and one 64-bit constant (and a 64-bit all-zero constant).

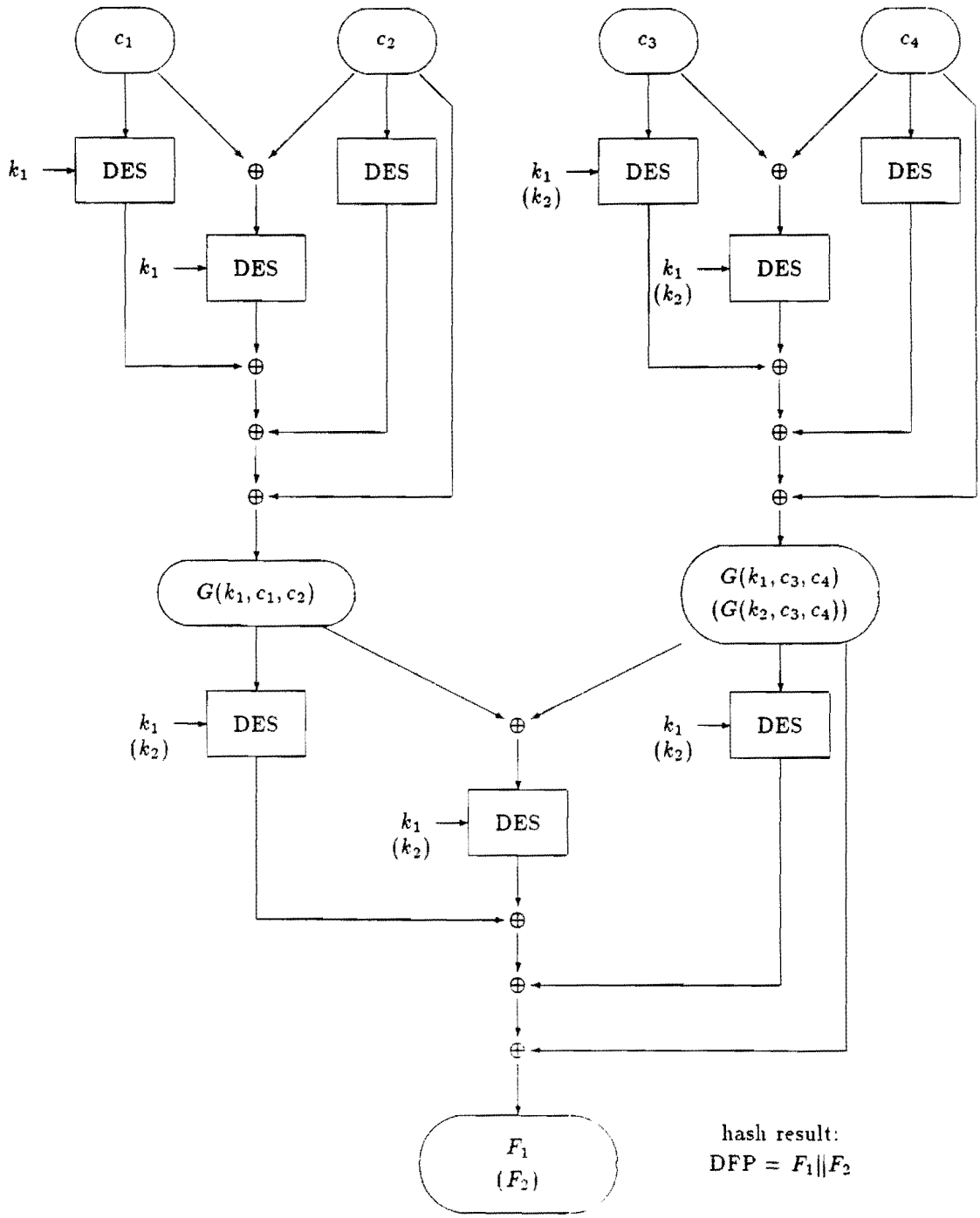


Figure 3.4: Calculation of DFP

Chapter 4

Combining hash functions and signature schemes

In the previous two chapters we have described 3 digital signature schemes and 4 hash functions. To sign a document, generally a hash function has to be chosen to preceed a digital signature algorithm.

4.1 Parameters

About the input length for the signature schemes the following can be stated:

1. The input length of the RSA digital signature scheme is equal to the bitlength of the public modulus that is used. In most applications the modulus length n is chosen to be 512 bits.

2. The input length of messages that are signed applying the ElGamal scheme is equal to the bitlength of the prime p that is used. Generally p is 512 bits long, although lengths of 700 to 800 bits are maybe preferable, since the prime p in the ElGamal scheme has to be larger than then the modulus n in RSA to obtain the same security.

3. In the Proposed Digital Signature Standard, the input length of the messages to be signed equals the length of q, which is a prime divisor of $p - 1$ (p prime). The proposal recommends to choose a 160-bit prime q.

Concerning the output length of the various hash functions described, the following remarks can be made:

1. The output length of the Message Digest Algorithm MD5 is 128 bits.

2. The hash result obtained when the algorithm described in DIS 10118-2 is used, is either one or two times the bitlength n of the block cipher employed. When DES or IPES are used, n is 64 bits. Thus, in these cases the output lenghts are 64 and 128 bits for the single length and double length hash function respectively. The standard describes how to shorten both outputs to an arbitrary length.

3. The output of the Secure Hash Algorithm SHA is 160 bits long.

4. The Digital FingerPrint hash function of Algorithmic Research has an output of 128 bits.

One of the properties the hash function needs to have is collision-freeness, as motivated in the first chapter. However, [MRW89] shows that hash functions giving a 64-bit result cannot be one-way, and [MPW92] suggest 100 bits as the minimum length that should be used. To see how collisions can be found, suppose h is a hash function producing n-bit hash results. If a user constructs two messages, and computes $2^{n/2}$ variants for each message, elementary probability theory says that there is a very strong change ($\approx 1 - e^{-1}$ ([MPW92])) that there will be a pair of variants, one for each message, having the same hash result. An obvious defence against this so-called "birthday attack" is to choose n large enough to make it infeasible to compute $2^{n/2}$ variant pairs. An alternative approach, allowing the use of 64-bit hash functions is suggested by [DP89]. They suggest that the originator of a message should always modify it in some way prior to performing the hash function, typically by appending a random a random value to the message. But as [MRW89] points out, this does not prevent potential frauds by the originator of the message.

This suggest that the algorithm of DIS 10118-2 should not be used as single length hash function with DES or IPES.

4.2 Combinations

When we compare the output lengths of the hash functions, and the recommended (or generally used) input lengths for the signature algorithms, one aspect is manifest:
All hash results are (considerably) shorter than the input length that is required for the signature algorithms, except when SHA is used with DSA. Then the length of the hash result and the input to the signature algorithm are equal.

The fact that the SHA output and the DSA input have an equal length, makes the combination SHA-DSA a very attractive one. This did not happen by accident: NIST designed both algorithms to work together.

The ISO/IEC 9796 digital signature algorithm describes how a hash result of bitlength L_H can be used for an RSA based digital signature scheme which needs an input of length L_S if holds that $L_H \leq 8 \cdot \lfloor \frac{L_S+3}{16} \rfloor$. If 512-bit RSA is used, the requirement for L_H becomes $L_H \leq 8 \cdot 32 = 256$, which holds for all hash functions described in this document. Thus, RSA can be used with any of these hash functions.

If the ElGamal signature scheme is chosen, an algorithm has to be defined to expand an hash result of 64, 128 and/or 160 bits to the required input length, which is 512 or more bits. Analogously, if DSA is used in combination with another hash function then SHA, the hash result of 64 or 128 bits has to be expanded to 160 bits.
This should be done in a way that the input to the signature scheme appears to be a random bitstring in order to prohibit forgery.

Chapter 5

Smart cards

The smart card is chosen to be used in the signature protocol for two reasons. First, a smart card can be used for access control, it allows the identification and the authentication of the user who is going to sign or verify a document. Secondly, a smart card can be used as a tamper resistant device to store the secret keys that will be used in the signature protocol.

The main features of a smart card are its intelligence and its security. It can store large amounts of information, even if it is not connected to a power source. This information can be updated, recalculated, and even coded and decoded by the card itself. Besides its large storage capacity and high intelligence, a key attraction of the smart card is its potential for high security. Smart cards can nowadays be seen as a convenient, safe, and inexpensive means for the storage of secret information.

5.1 Generals

Before we discuss the features of a smart card, it is important to know some more details on the design of a smart card.
The basic reference for smart cards and the like is the multipart standard ISO 7816. Cards designed in accordance with this standard are made up of a plastic support that contains a small device consisting of a chip: a tiny printed circuit board with an integrated micro circuit at its center.

There are three reserved locations on the surface of the support as figure 5.1 shows:

- a location for the printed circuit supporting the micro circuit;
- optionally a location for the magnetic stripe (if the support is used as a combined card);
- a location for embossing the user's identity and the card's ISO number.

The printed circuit conforms to ISO/IEC International Standard 7816-3. It is hermetically fixed in the recess provided on the plastic support. It protects the micro circuit from mechanical stress, radiation, static electricity and acts as an electrical interface between the micro circuit and the smart card reader.
The micro circuit is attached to the back of the printed circuit. Any data exchange between the micro circuit memories and the application is subjected to highly sophisticated security procedures by the micro circuit's CPU.

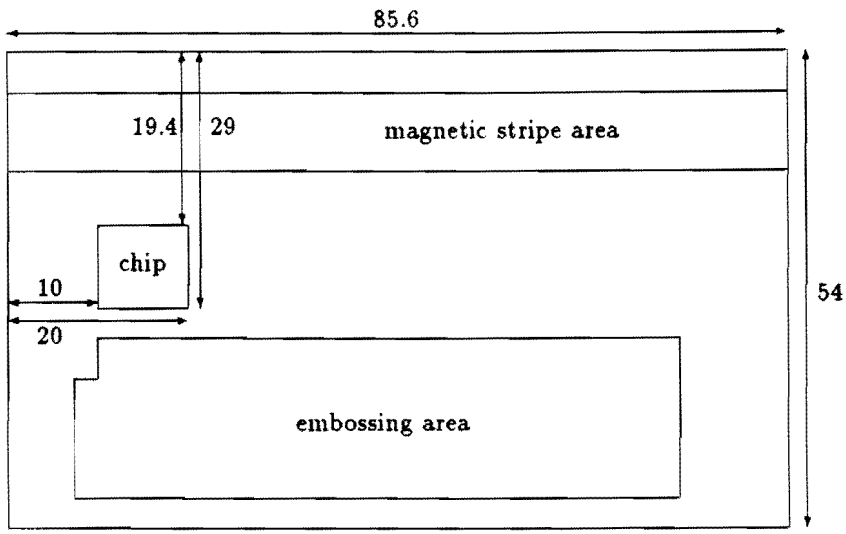


Figure 5.1: Smart card layout

The micro circuit generally contains the following parts:

Data Memory
: The Data Memory can be an EPROM or an EEPROM (=E^2PROM). EPROM is an abbreviation for Erasable Programmable Read Only Memory. Data can only be written once; the possibility of erasing the memory has been disabled. In an EEPROM (Electronically Erasable Programmable Read Only Memory) the erasure of certain data is at the discretion of the card.

ROM
: The Program Memory is a ROM (Read Only Memory). It contains the card's operating system. For example, all the operations to be executed by the CPU, the management of the Data Memory, the security rules, the communication protocol and the algorithm functions such as DES or modular exponentiations.

RAM
: The Working Registers are contained in the RAM (Random Access Memory). They are used by the CPU for the management of internal or temporary data and for storing intermediate results. When the card is disconnected from the power supply, this data is lost.

Central Processing Unit
: The CPU controls the internal buses via which it can access all three memories (Data Memory, Program Memory, RAM). No direct access to these memories is possible from outside. The CPU manages the communication line enabling the micro circuit to communicate with the smart card reader.

Connecting points
: Communications with the system pass via the card's printed circuit. Eight connection points are specified; two of them are reserved for later use and are often not provided. The other ones are used for the supply voltage (5V), the reset of the card, the clock, the ground, the programming voltage for non-volatile memory, and an I/O port.

5.2 TB100

The TB100 Smart Card is a joint development of Philips and BULL. It has the DES algorithm implemented.

5.2.1 Layout

In the TB100 card the micro circuit is an 8-bit microprocessor. It contains a Data Memory consisting of 3 Kbytes of EEPROM; in certain areas of the TB100 card erasure is implemented. These memories can be divided into several zones of confidentiality which can be accessed by specified keys. The Program Memory is a 6 Kbyte ROM which contains, among others, the DES algorithm functions. The working registers are contained in a 128-byte RAM.

Two Data Structures are available in the TB100 Smart Card:

- *Data Files (DFs)* which define storage areas.
  A Data File can be of one of the three levels below:
  - A *Common Data File (CDF)*, of which exactly one exists per Smart Card;
  - An *Application Data File (ADF)*;
  - A *Sub-Application Data File (SDF)*.

  Each Data File is composed of a Header, which defines the Data File, and of a Body that may contain Data Files of a lower level and Zones.
  A Data File can be set in Utilisation Phase, but is really in Utilisation Phase only if the Data File of the upper level is in Utilisation Phase too. Similarly, if a Data File is invalidated, all the structure below is also invalidated.
- *Zones* which serve to store information.
  A Zone can be one of the following:
  - A *Secret Zone* for Cryptographic Keys and Access Codes;
  - An *Access Tracking Zone (ATZ)* for Key submission recording;
  - A *Public Zone* for non-confidential information;
  - A *Working Zone* that can be read, written or erased under parameter dependent access controls.

  A Zone is in Utilisation Phase only when the Data File which holds it is in Utilisation Phase, but it can be invalidated separately from that Data File.

The TB100 Smart Card Words are 32 bits long (full data), and can be used following two modes: In *validated mode* where the word cannot be overwritten and in *token mode* where a not validated word may be overwritten and allows, this way, the bit per bit word consumption. The TB100 Smart Card Keys are 64 bits long (2 Smart Card words), and are identified with a *Key Identifier Field (KIF)* and a *Key Version Field (KVF)*, allowing the key to be indexed.

The management of the Smart Cards must be user-friendly, achieving a high level of security, and the data entry must be kept to a minimum. Chapter 6 deals with various management aspects. In order to avoid that the tool's operator has to enter too much data interactively necessitating a very good knowledge of the smart card, the basic layout for the smart card as shown in figure 5.2 is introduced in the KMT/KMS developed by P.I.T.S. This is a Key Management Service using as basis a Key Management Tool.

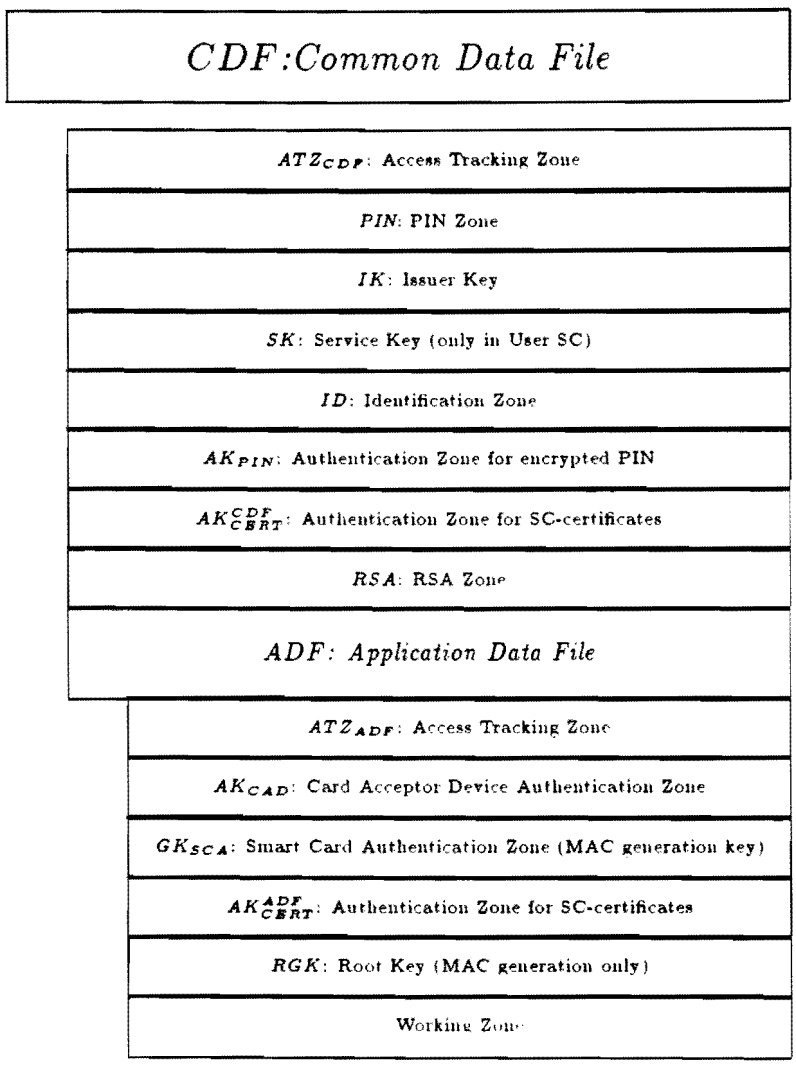


Figure 5.2: General layout of the TB100

Note that several Application Data Files may occur according to the number of applications that are covered by the smart card.

A key which is stored in a Secret Zone can only be accessed by the microprocessor of the smart card, and only for the selected function(s). This means that it is impossible to retrieve the key itself from the smart card. But a key which is stored as data in a Working Zone of the smart card can be retrieved from the card.

The Common Data File CDF can be considered as the upper level of the card. At a minimum it contains the Access Tracking Zone ATZ_{CDF} to record errors which occur during secure operations, a Secret Zone to store the *PIN*, and a Secret Zone to store the Issuer Key IK which allows the recycling of the Smart Cards, *PIN* modifications, and can also be used in read, write and erase rules.
Optionally the CDF can contain a Service Key, stored in a Secret Zone, which allows the creation of new Data Files or new Working Zones. The CDF can furthermore contain a (Public) Identification Zone ID which may be used to record the distinguished name of the owner of the card, the creation date, the person who was responsible for the creation of the card, and the validity period of the card. Two more optional Secret Zones can include authentication keys: the AK_{PIN} which is required to make an encrypted *PIN* presentation, and the AK^{CDF}_{CERT} which is needed for the certified reading of certain smart card data which are stored in the CDF. The CDF can also contain RSA Working Zones where the secret key (n, s) or (p, q, s) and the public key (n, v) are written, possibly together with the validity period of the key pair, a certificate certifying (n, v) and the public key of the certification authority. The RSA Keys are loaded from a previously generated file, and they can be extracted from the smart card.

The Common Data File can contain one or more Application Data Files ADF, which are the second level in the Data Files hierarchy. An ADF has an Access Tracking Zone ATZ_{ADF} to record errors which occur during secure operations, and three secret zones containing a CAD Authentication Key AK_{CAD} to make an authentication of the application versus the smart card, a Smart Card Authentication Key GK_{SCA} which is required to make an authentication of the smart card versus the application, and an authentication key AK^{ADF}_{CERT}, required for the certified reading of certain smart card data which are stored in the ADF. Finally, a Root Key RGK is stored in a Secret Zone as a MAC Generation Key for generating keys for smart cards of a lower level by diversification by means of diversification with external values.

Remark:
The application will generate the AK_{CAD} and GK_{SCA} of a specific smart card by means of diversification with an external value of the appropriate Root Smart Card Authentication Key. This diversification is a MAC generation using the external value and the Root Key, as explained below:

Diversification of keys

Diversified keys or temporary keys in the smart card will be calculated by means of a MAC calculation. This is done in order to avoid problems related to governmental regulations on the encipher/decipher facilities within the smart cards.

The key diversification is depicted in figure 5.3.
First a MAC calculation with the Root Key as key is performed on an internal value of the smart card. Then the MAC result, say X, is XOR-ed with the diversification value, e.g. the card's Serial Number or a random number (in case of a temporary key). Finally, another MAC calculation with the Root Key as key is performed on this XOR. The result of the last MAC

calculation is the diversified key or temporary key SC Key.

In case one really wants to obtain a key which is obtained from the Root Key by diversification with the diversification value, the initial internal value has to be chosen in such a way X equals the zero word. This initial value, once calculated, can be stored as internal parameter together with the MAC Key in a Secret Zone.

The application has to perform a MAC calculation applying the same Root Key with the same diversification value as input to obtain the same SC Key.

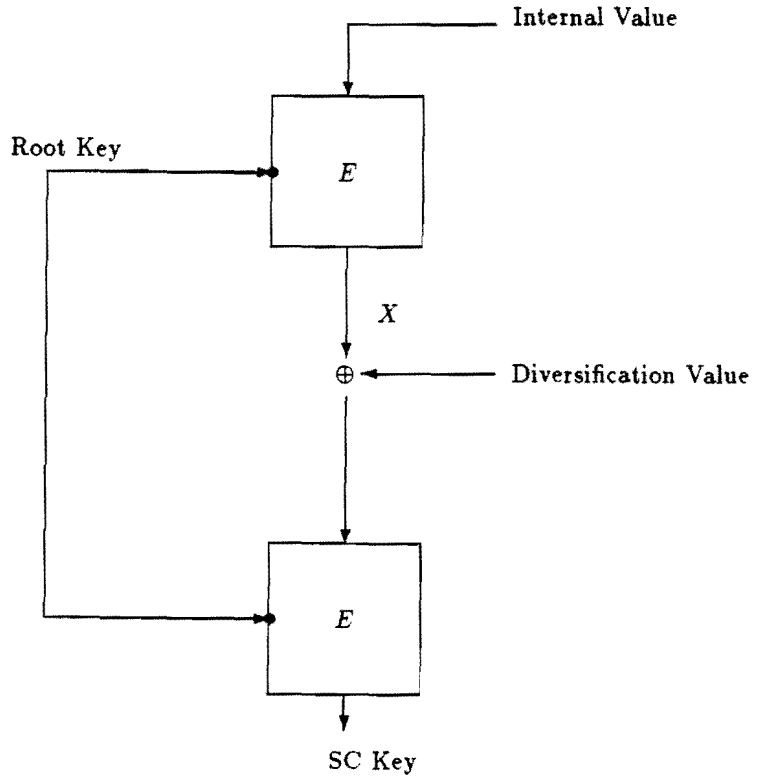


Figure 5.3: Smart Card Key diversification

5.2.2 Hierarchical Key Design

In this section we will describe the hierarchical key design of the KMT/KMS. Therefore we use as basis the three-level hierarchy shown in figure 5.4. The creation of the different hierarchy levels and authorities is closely related to the management of the smart cards, e.g. personalisation, recycling, integration in an application, etc.

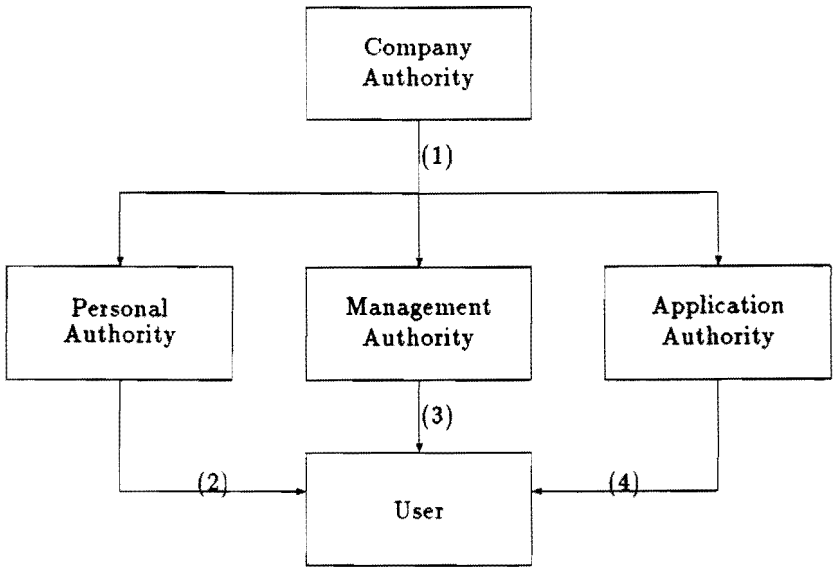


Figure 5.4: The security hierarchy

In figure 5.4 the numbers refer to the following actions:
(1) the creation of the authority cards
(2) the creation of the user cards
(3) the recycling or revalidation of user cards
(4) the loading of a new application and the integration in a new users group.

The structure introduces three different levels in the hierarchy, with at the top the company authority or Security Officer (S.O.). The smart card associated to this level is the basis for all the secret keys which are going to be introduced at the other levels, covering several applications and groups. Each of the intermediate authorities receives from the S.O. a card which is specially designed for the related authority function. The authorities and their cards only have power restricted to their domain. The task of the Personal Authority (P.A.) is the creation of user cards and the storage of the keys which are required for the general card management. The Management Authority (M.A.) takes care of the recycling or revalidation of user cards, and the Application Authority (A.A.) loads (new) applications and integrates users in a new users group. The last level in the security hierarchy corresponds to the users. A user card serves the purpose of user identification (for access control) and the storage of keys and data for specific applications. During its life cycle, it needs the intervention of the specific authorities of the intermediate level at specific moments.

5.2.3 Personal Identification Number *PIN*

A *PIN* is a number that serves as an identification of the user versus the smart card. Only if the owner of the smart card enters the correct *PIN*, he can get access to the information stored on the card.

PIN length

In ISO 9564-1 ([I9564-1]) a *PIN* length of 4 up to 12 characters, not exceeding 6 digits, is recommended. In practice very often *PIN*s of 4 digits (16 bits) are used. Since on the Smart Card the *PIN* must be stored in two Smart Card words (64 bits), a padding of 48 bits is required.

PIN presentation

The Transaction *PIN*, i.e. the *PIN* presented by the smart card owner which will be compared the the reference *PIN* stored on the Smart Card, can be transmitted from the CAD to the Smart Card in clear or in encrypted form.

For the TB100 Smart Card, the presentation of the *PIN* in ciphered form uses a random number which is generated in the Smart Card, but diversifies from the format 1 *PIN* block as described in the ISO standard. However, by means of the random number, the transmitted value is different for every encrypted *PIN* presentation as recommended in the ISO standard.

The key AK_{PIN} is dedicated for the submission of an encrypted *PIN*. The process is depicted in figure 5.5 and described below:

1. The Smart Card generates a random number R and transfers R to the CAD.
2. The CAD obtains AK_{PIN} from the Root Key $RGK_{AK_{PIN}}$ by diversification with the Serial Number of the Smart Card, and it XORs R with the *PIN* code which is entered on it: $PIN \oplus R$. The obtained value is deciphered with the key AK_{PIN}: $d_{AK_{PIN}}(PIN \oplus R)$ and the result is transferred to the Smart Card.
3. The Smart Card enciphers the obtained value with AK_{PIN} and calculates the XOR of the obtained value with *PIN*: $e_{AK_{PIN}}(d_{AK_{PIN}}(PIN \oplus R)) \oplus PIN$. The obtained value is compared with the R sent, and only if the two coincide the *PIN* presentation was correct.

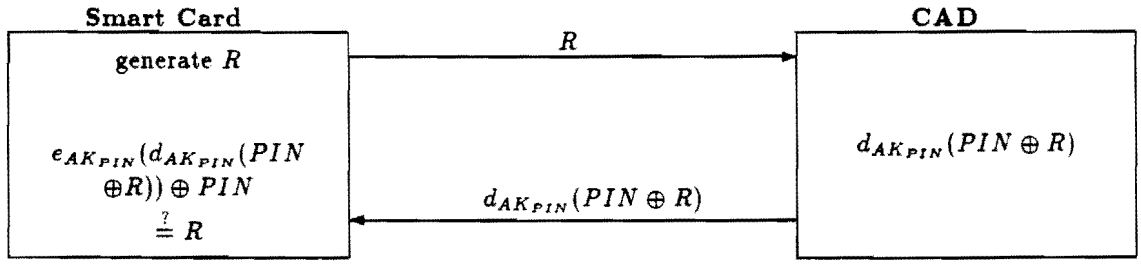


Figure 5.5: Enciphered *PIN* verification

5.2.4 Authentication protocols

We will now describe an authentication protocol that achieves authentication of the application (via the CAD) versus the smart card, and one that achieves authentication of the smart card versus the CAD. Thus, when both protocols are executed after one another, mutual authentication of smart card and CAD is achieved. This mutual authentication is required before a signing or verification protocol will be executed.

CAD authentication versus Smart Card

With the CAD Authentication Key it is possible to perform an authentication process whereby the terminal (application) which is connected to or contains the smart card reader authenticates itself versus the smart card.

The authentication process is, as shown in figure 5.6:

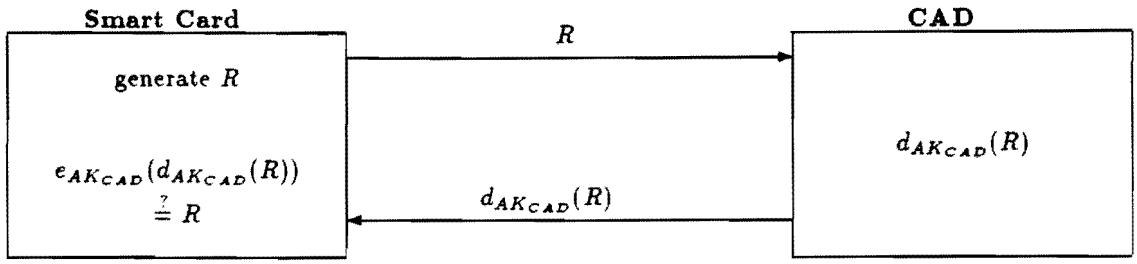


Figure 5.6: CAD authentication

1. The Smart Card generates a random number R and transfers R to the terminal.
2. The terminal obtains AK_{CAD} from the Root Key $RGK_{AK_{CAD}}$ by diversification and decrypts R with this key AK_{CAD}: $d_{AK_{CAD}}(R)$. The result is transferred to the Smart Card.
3. The Smart Card encrypts the received value with the key AK_{CAD}: $e_{AK_{CAD}}(d_{AK_{CAD}}(R))$ and compares the obtained value with the random R sent.

This process is very similar to the two pass unilateral authentication protocol described in section 5.1.2 of the ISO/IEC standard 9798-2 ([II9798-2]). The only difference is that there the CAD decrypts not only R, but R together with the identification of the Smart Card to prevent a so-called reflection attack. This attack, however, is not possible in our situation, because the entities to be authenticated can be divided in two disjoint groups, the smart cards and the terminals. Since the authentication protocol always starts with the transmission of a random number from a CAD to a Smart Card, it will immediately be detected when a CAD tries to impersonate a Smart Card or the other way around.

Smart Card authentication versus CAD

With the Smart Card Authentication Key it is possible to perform an authentication process whereby the Smart Card is authenticating itself versus the CAD.
Since this authentication is that of the CAD versus the Smart Card with the roles of CAD and Smart Card interchanged, it also resembles the one described in ISO/IEC 9798-2 very much; a similar remark as at the end of the section about CAD authentication can be made.

The authentication process is depicted in figure 5.7:

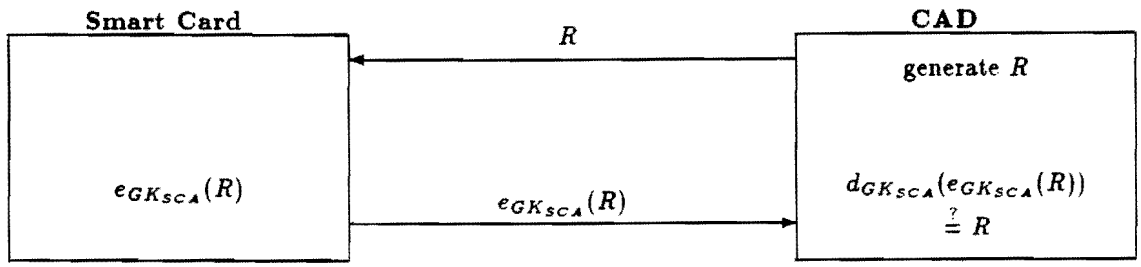


Figure 5.7: Smart Card authentication

1. The CAD generates a random number R and transfers R to the Smart Card.
2. The Smart Card encrypts R with the key GK_{SCA}: $e_{GK_{SCA}}(R)$ and transfers the result to the CAD.
3. The terminal obtains GK_{SCA} from the Root Key $RGK_{GK_{SCA}}$ by diversification and decrypts the received value with the key GK_{SCA}: $d_{GK_{SCA}}(e_{GK_{SCA}}(R))$. The obtained value is compared with the random R sent.

5.2.5 The RSA Public Key System

In the key management system each of the persons involved has a RSA key pair stored on its smart card at *CDF* level. This section aims to describe how this pair is generated and transmitted to every person, and in which way it is stored on a token.

The digital signatures and encryption/decryption which is currently in use, are based on the RSA algorithm which applies a modulus n having a length of 512 bits. The modulus is composed of two primes p and q which each have a length of 256 bits. The secret exponent s is a 512 bit number, while the public or verification exponent v consists of 17 bits.

The RSA key pairs are generated applying a software package. Before writing the keys on the token, they will be encrypted in the kernel of the software packet with a key derived from the token. The generation of a pair RSA keys (n, p, q, s and v) costs, depending on the PC that is used, between 13.3 and 39 seconds at the moment.

On the TB100 Smart Card, three special Working Zones dedicated to the RSA Key pair are created. The reason for dealing with several zones is that this is time efficient, since usually the application only requires the information from one of the zones.
These three Working Zones are:

Working Zone RSA 1 stores the secret key consisting of (p, q, s), or (n, s), and furthermore the validity period (*val*) of the RSA key.

Working Zone RSA 2 stores the public key consisting of n and v with the validity period (*val*) of the RSA key, or the certificate containing that public key, or both.

Working Zone RSA 3 stores the public key (n, v) of the Certification Authority with the validity period (val) of that key.

Remarks

1. In the current key management system, every smart card contains only one RSA key pair. However, it is advisable to use different pairs for signature and for message encryption. Therefore, in the future, the use of different key pairs for those goals will be introduced. For the time being we will use the public key for (short) message encryption if needed by the application. For this reason the public key itself is contained in the Working Zone and not only the certificate.

2. It is recommended that the RSA key pair of the certification authority uses a modulus which is about 200 bits longer than the user keys.

When storing data in Working Zone RSA 1, a check value CV will be computed.
Some of the parameters, namely s, CV, and p and q (if available), are encrypted, applying a key derived from AK_{CERT} by diversification with the card's serial number. The parameters val, and n (if available) are stored in clear. The reason that the secret key information is stored enciphered, and not in clear is to allow the enciphered key to be read out "in clear", which is much less time consuming than enciphered read.

All data in the second RSA Working Zone will be stored in clear. If v and n are available, a check value CV will be calculated.

The data in Working Zone RSA 3 will be stored in clear, but a check value CV will be calculated on v and n.

The application (CAD) will read the keys in clear from the smart card after a *PIN* presentation which is mandatory. The CAD can read from the Working Zone in clear in blocks of 256 bytes, where the reading of each such a block requires about 2 seconds for a TB100 smart card.

5.2.6 Signing and verification

Now we have considered some aspects of the TB100 card, the signing and verification processes as given in the first chapter can be described more detailed.

The signing process

The signing process using the TB100 card is depicted in figure 5.8 and consists of the following steps:

1. The user authenticates himself to the smart card by presentation of his PIN. The transaction from this PIN to the CAD can either be done in clear or in encrypted form. Details can be found in section 5.2.3.

2. The terminal which is connected to or contains the CAD, and the smart card perform a mutual authentication. This is done by performing an unilateral authentication of the CAD versus the smart card, and an unilateral authentication of the smart card versus the CAD. As described in section 5.2.4, both unilateral authentication require two transmissions.

3. The document to be signed is brought into the application.

4. • The application transforms the document to ASCII format.

- The signer chooses a hash function from a predescribed set.
- The application hashes the document in ASCII format to a string with a fixed (short) length, depending on the hash function chosen.
- The application uses a publicly known procedure to expand the hash result to a 512-bit string: the intermediate string.

5. The application obtains the signature key and computes the signature:
   - The application transfers a request to the smart card for the secret RSA signature key.
   - The smart card transfers this signature key, which is stored in Working Zone RSA 1, to the application. The signature key is stored enciphered on the card, thus it can be read out "in clear".
   - From the serial number of the smart card, the application computes from a root key the key needed to decipher the signature key, and obtains by deciphering the signature key in clear.
   - The application signs the intermediate string using the obtained signature key.
6. The application creates a signed document containing the original document and the signature, together with a note on who signed the document. Also information on the hash function used is included.

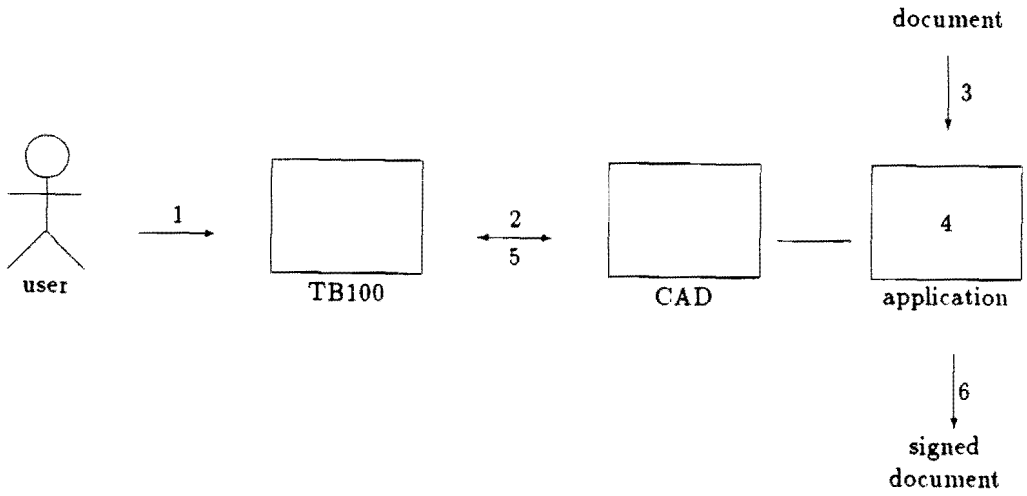


Figure 5.8: The signing process

The verification process

The verification process using the TB100 card is depicted in figure 5.9 and consists of the following steps:

1. The user authenticates himself to the smart card by presentation of his *PIN*. The transaction from this *PIN* to the CAD can either be done in clear or in encrypted form. Details can be found in section 5.2.3.

2. The terminal which is connected to or contains the CAD, and the smart card perform a mutual authentication. This is done by performing an unilateral authentication of the CAD versus the smart card, and an unilateral authentication of the smart card versus the CAD. As described in section 5.2.4, both unilateral authentication require two transmissions.
   Note:
   It is not necessary that a smart card is involved in the verification process, since all parameters needed to verify a signature are public. We assume, however, that only an authenticated smart card has access to the verification protocol of the application, and therefore the above two actions are obliged.

3. The signed document is brought into the application.

4. - The application reads out the identity of the signer, and the identification of the hash function used.
   - The application obtains from the signed document the original document and transfers this original document to ASCII format.
   - The application uses the specified hash function to hash the document in ASCII format to a string with a fixed (short) length depending on the hash function.

5. The application obtains the public key of the signer by means of a certificate and uses it to "decipher" the signature. Detailed information on the construction and use of certificates can be found in the next chapter.

6. The 512-bit result of step 5 is shortened to get a bitstring with a length equal to the output lenght of the hash algorithm used.

7. The outcome is compared with the string stored in step 4, and only if they are equal the application outputs "OK".

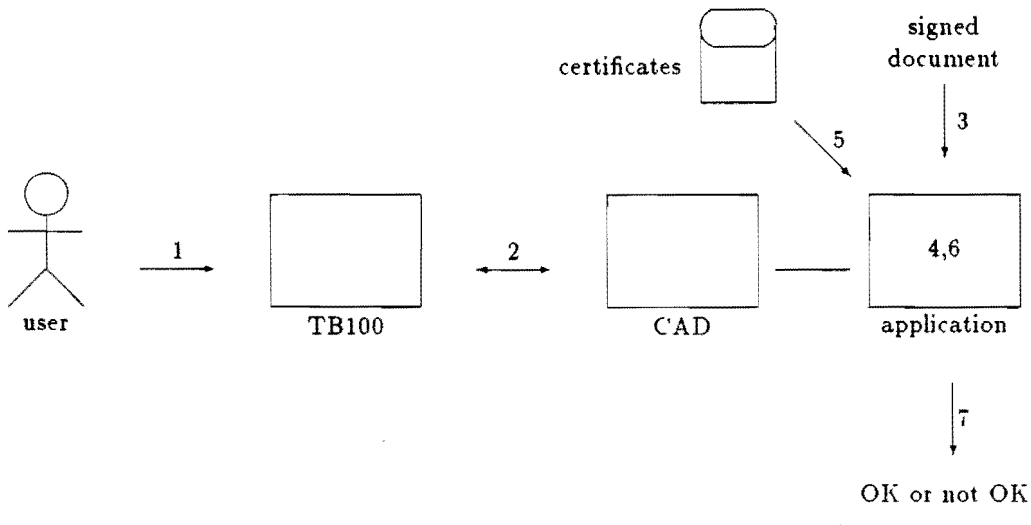


Figure 5.9: The verification process

5.3 DX Card

The DX Smart Card is the first commercially available smart card that has an asymmetric crypto algorithm implemented. While all other smart cards are based on DES, the DX Smart Card contains an RSA chip. The available mask for the DX card has the DES algorithm implemented, but only for internal use. Note that it is possible to use optional coding to implement DES for external use.

5.3.1 The chip

The DX Card contains the chip 83C852, which is embedded in a smart card according to the ISO standard 7816. This chip can perform the equivalent of 40 (8-bit) MIPS for large number arithmetic, and this enables the DX card to compute a digital signature X^s mod n with 512-bit operands very efficiently. The time that is needed is affected by the number of ones in the exponent, and whether or not the card makes use of the Chinese Remainder Theorem (see e.g. [D83]). The Chinese Remainder Theorem requires knowledge of the two primes used to create the modulus, but the time needed to sign a 512-bit message at 6 Mhz drops from 1.5 to 0.5 seconds on average. Note that the use of the Theorem is limited to the signing operation, since the two primes are part of the secret key. However, the basic verification of a signature costs only a few milliseconds, since the public verification exponent is fixed to 3.

About the memories we can say that the Data Memory consists of 2 Kbytes of EEPROM. The Program Memory is a 8 Kbytes ROM, and the working registers are contained in a 256 byte RAM.

The calculation unit, with its associated software, optimizes the calculation time of exponentiations modulo n, used in public key algorithms and zero-knowledge protocols. It needs 196 bytes of RAM for 512-bit operands when the Chinese Remainder Theorem is used, and 83 to 146 bytes when the classical method is applied. The calculation unit does not carry out a complete exponentiation in one step, but provides a set of basic instructions from which the complete exponentiation algorithm can be built by dedicated software. These basic instructions operate on variable length data fields inside RAM and EEPROM. The most important operation is to multiply a 24-bit number or a 32-bit number with a long-word (e.g. 512 bits) and add the result to another long-word. Besides that XORs and shifts might be carried out to give the final result.

5.3.2 Personal Identification Number *PIN*

The Personal Identification Number is used for the authentication of the user versus the smart card.

PIN length

In ISO 9564-1 a *PIN* length of 4 up to 12 characters, not exceeding 6 digits, is recommended. In practice very often *PIN*s of 4 digits (16 bits) are used. On the DX card, a *PIN* has to be stored in 64 bits, so a padding is required.

PIN presentation

The *PIN* presented by the smart card owner can be transmitted from the CAD to the Smart Card in clear or encrypted form.
At P.I.T.S. *PIN* presentation protocols will be written which will basically be the same as the ones for the TB100 card (section 5.2.3).

5.3.3 Authentication protocols

At the moment, P.I.T.S. had not finished authentication protocols for the DX card, for which reason we cannot give a detailed protocol description. However, it is obvious that the protocol descriptions will use that the DX card has an RSA algorithm implemented that can encrypt/decrypt 512-bit messages, using secret and public keys of 512 bits, a 512-bit modulus, and an exponent of up to 512 bits. Below we give a sketch of the basic protocol that will be used.

For the authentication protocols the card contains an RSA key pair. The secret exponent of the Smart Card with identifier X is denoted by s^X_{SC}, and its public exponent by v^X_{SC}. The modulus is denoted by n^X_{SC}. When it does not give confusion the identifier X will be omitted. The secret exponent, the public exponent, and the modulus of a terminal X are denoted by s^X_{CAD}, v^X_{CAD}, and n^X_{CAD} respectively. Again, the identifier X will be omitted in the descriptions whenever possible.
In small systems it is reasonable to assume that each CAD has an up-to-date directory containing the identifiers of all users identified to the system together with the associated public authentication key (v_{SC}, n_{SC}). For large applications this will require too much storage, so then the card has to transfer a certificate containing its public authentication key to the terminal that has to verify a message signed with its secret authentication key. In the protocol descriptions we show the protocols for the case that a certificate has to be send. The certificate containing $v^X_{...}$ will be denoted by $Cert^X_{...}$, and it consists of a message containing $(v^X_{...}, n^X_{...})$ that is signed by some authority. More information about certificates can be found in chapter 6.

Smart Card Authentication versus CAD

The authentication process is depicted in figure 5.10. It equals the asymmetric unilateral authentication protocol of section 5.1.1 in ISO/IEC 9798-3 ([II9798-3]), except for the omission of some parameters that are not needed in our situation.

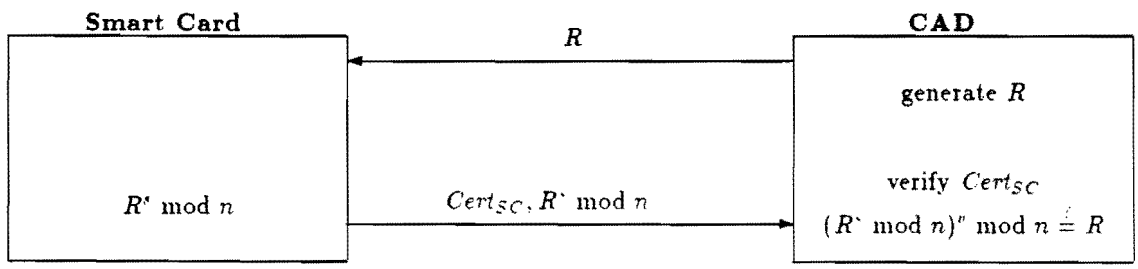


Figure 5.10: Smart Card authentication

1. The CAD generates a random number R and transfers R to the Smart Card.
2. The Smart Card encrypts R with its secret authentication key and transfers the result $R^s \bmod n$ to the CAD together with the certificate $Cert_{SC}$ containing its public key.
3. The CAD verifies the certificate, and retrieves the public authentication key (n, v) from it. Then it computes $(R^s \bmod n)^v \bmod n$, and compares the result with the random R sent.

CAD authentication versus Smart Card

The aim of this protocol is to convince the Smart Card that the CAD which is connected to or contains the smart card reader is a "genuine" one. The same protocol as above can be executed with the roles of CAD and card interchanged, as is depicted in figure 5.11.

1. The Smart Card generates a random number R and transfers R to the CAD.
2. The CAD encrypts R with its secret authentication key and transfers the result $R^s \bmod n$ to the Smart Card together with the certificate $Cert_{CAD}$ containing its public key.
3. The Smart Card verifies the certificate, and retrieves the public authentication key (n, v) from it. Then it computes $(R^s \bmod n)^v \bmod n$ and compares the result with the random R sent.

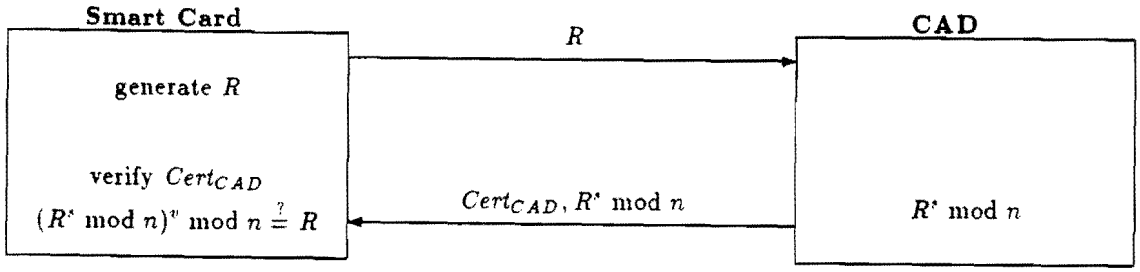


Figure 5.11: CAD authentication

5.3.4 Signing and verification

We will now describe the signing and verification processes as given in the first chapter for the DX card.

The signing process

The signing process using the DX card is depicted in figure 5.12 and consists of the following steps:

1. The user authenticates himself to the smart card by presentation of his PIN. The transaction from this PIN to the CAD can either be done in clear or in encrypted form.
2. The terminal which is connected to or contains the CAD, and the smart card perform a mutual authentication. This is done by performing an unilateral authentication of the CAD versus the smart card, and an unilateral authentication of the smart card versus the CAD.
3. The document to be signed is brought into the application.
4. The document is transformed to a string the DX card can sign:
   - The application transforms the document to ASCII format.
   - The signer chooses a hash function from a predescribed set.
   - The application hashes the document in ASCII format to a string with a fixed (short) length, depending on the hash function chosen.
   - The application uses a publicly known procedure to expand the hash result to a 512-bit string: the intermediate string.
5. The application transfers the intermediate string to the card.
6. The card obtains the signature key and computes the signature.
7. The card transfers the signature to the application.
8. The application creates a signed document containing the original document and the signature, together with a note on who signed the document. Also information on the hash function used is included.

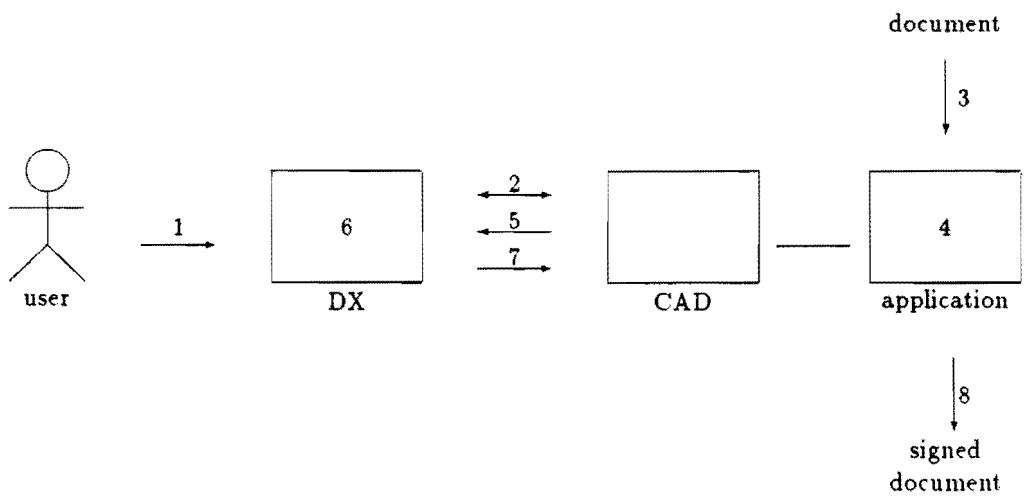


Figure 5.12: The signing process

The verification process

The verification process using the DX card is depicted in figure 5.13 and consists of the following steps:

1. The user authenticates himself to the smart card by presentation of his PIN. The transaction from this PIN to the CAD can either be done in clear or in encrypted form.
2. The terminal which is connected to or contains the CAD, and the smart card perform a mutual authentication. This is done by performing an unilateral authentication of the CAD versus the smart card, and an unilateral authentication of the smart card versus the CAD.

   Note:
   It is not necessary that a smart card is involved in the verification process, since all parameters needed to verify a signature are public. We assume, however, that only an authenticated smart card has access to the verification protocol of the application, and therefore the above two actions are obliged.
3. The signed document is brought into the application.
4. - The application reads out the identity of the signer, and the identification of the hash function used.
   - The application obtains from the signed document the original document and transfers this original document to ASCII format.
   - The application uses the specified hash function to hash the document in ASCII format to a string with a fixed (short) length depending on the hash function.
5. The application obtains the public key of the signer by means of a certificate. Detailed information on the construction and use of certificates can be found in the next chapter.
6. Now there are two possibilities:

   6a The application uses this public key to "decipher" the obtained signature.

   6b The application transfers the public key and the signature to the DX card which "deciphers" the signature using the public key and transfers the result back to the application.
7. The application shortens the 512-bit "deciphered signature", the result of step 6, to get a bitstring with a length equal to the output lenght of the hash algorithm used.
8. The outcome is compared with the string stored in step 4, and only if they are equal the application outputs "OK".

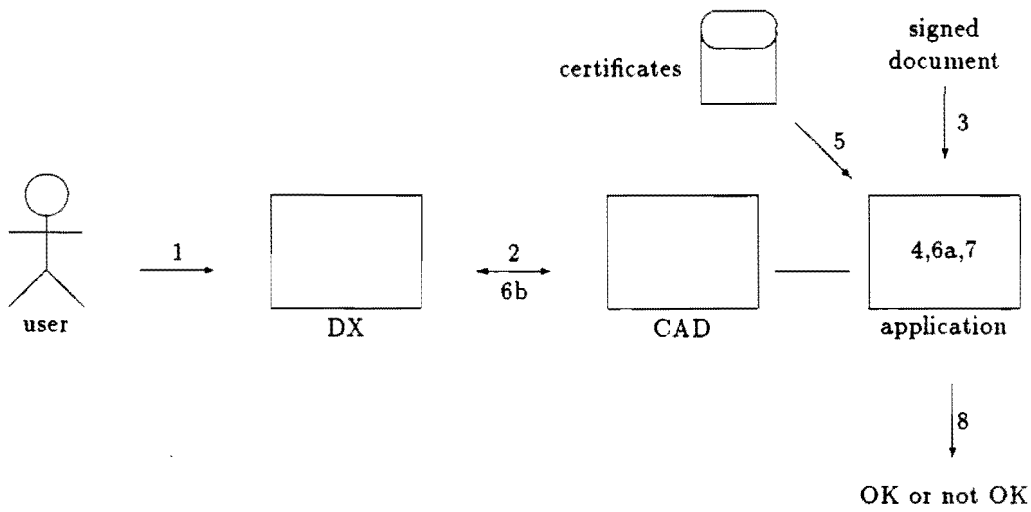


Figure 5.13: The verification process

Chapter 6

Key management

6.1 General considerations

When an application is using cryptographic mechanisms, the security of the application system is directly dependent on the protection afforded to security parameters or keys. The overall goal of key management is to provide procedures for handling cryptographic keying material, or, according to the definition in ISO 7498-2, key management is "the generation, storage, distribution, deletion, archiving and application of keys in accordance with a security policy". A key management system only makes sense if it is able to guarantee the connection between an entity and its uniquely defined representing keys. This connection may be achieved by cryptographic methods, so that registration of entities becomes an essential part of key management.
The following design criteria can be considered to be important:

- Minimise the physical activity.
  This indicates not only that the use of couriers should be kept at a mimimum, but also that entities do not have to travel far to registrate. The last suggest an hierarchical registration approach.
- Ensure that any dishonest entity may be exposed.
  Since it is never possible to prohibit entities from being dishonest, this is the best we can go for. Sometimes it may be necessary to build a system with a sufficiently high security to ensure invulnerability even if several users conspire.
- Minimise the number and size of tamper resistant devices required.
  Perhaps it can be useful to make a distinction between tamper resistancy to protect keys, and tamper detectability to prohibit eavesdropping.
- Minimise the number of secure channels required.
- Achieve maximum flexibility.

6.2 Trusted Third Parties

Except for very small systems, one or more parties that can be trusted (to some extent) are needed for the secure management of cryptographic keys. If no such trusted parties are available, each pair of users who want to communicate must have had physical contact before.

6.2.1 Terminology

The term *Trusted Third Party (TTP)* is used for any "security authority, or its agent, trusted by the other entities with respect to security-related issues" ([II9798-1]). Its task vary from user registration, key management and -distribution to being a judge in case of disputes.

More or less the same object appears under various names, and corresponding abbreviations in literature. If the mechanisms used are based on conventional cryptosystems, the trusted party is generally a Key Distribution Center (KDC) or Key Translation Center (KTC). When public key algorithms are used, a Key Certification Center (KTC) or Certification Authority (CA) will be employed.

6.2.2 Tasks

A TTP has many and varied tasks. They do not need to be carried out, however, by one and the same authority. Especially when the users group is rather large, the tasks should be split among various facilities.

Part of the TTP's duties is related to key management. This starts with *registration* of each user of the system. The purpose of this is to allow automatic *identification* of that entity in the sequel. This identification can be absolute, which means that a link between an ID and some physical representation of the identified entity can be established. The identification can also be relative, than the entity is only re-identified under some ID. An entity is always represented by some public data, its public credentials, and some private data, its secret credentials. When an entity registers, the TTP has to generate or check them. A *certificate* containing the public credentials is issued as a proof of registration. This certificate may involve anything from a protected entry in a certain file to a signature by the TTP on the credentials. This task of the TTP is performed by a so-called a Certification Authority.

Note:
Usually companies or other organisations will register, while employees of the company (or members of the organisation) will sign messages on behalf of that company or organisation. Thus, the signing entities are not the same as the registrated entities. Therefore the certificates on the public keys of the signing entities have to contain not only information on the signing entities, but also of the covering organisation.

Another task of the TTP can be *key generation*, but it is also possible that each entity generates its own keys. Keying material to be used in symmetric algorithms has to be *exchanged* or *distributed* in a way that guarantees its origin, integrity and confidentiality. A key can be exchanged by two entities manually, i.e. by some physical means independently from the communication channel, or automatically, i.e. electronically employing key exchange procedures.

When a key has been established, it has to be *stored* and *protected*. All parts of the *key maintenance* are tasks of the key administration of the TTP. This includes key activation, deactivation, deletion, recovering, black listing, translation, etc.

Besides its tasks related to the management of keys, and partially overlapping them, a TTP will also serve as an "electronic notary" ([BGM91]). As a public notary, an electronic notary should have the duty for confidentiality, the professional code of conduct, and stringent rules for the authentication, verification and secure storage of documents. The notary is trusted because it is indepedent and impartial. Furthermore, documents prepared and signed by a notary are deemed to be authentic and accurate and therefore provide evidence in court.

6.3 Distribution of public keys

In this section we will describe how an entity's public key can be made available to the other entities in an authentic fashion ([II11770-3]). Authenticated distribution of public keys can be achieved without making use of a trusted third party, or involve a certification authority CA. In both cases, the public key of an entity is part of its credentials, which also include at least its distinguished identity. There may other static information regarding the CA, the entity, or the involved algorithms be included in the credentials [II9798-3].

6.3.1 Public key distribution without a trusted third party

In this solution, if an entity (say B) wants to have the public key of another entity (say A), he will get this key directly from A. The protocol is as follows:

1. A sends to B his credentials $Cred_A$ and B's address.
2. B stores A's credentials at a tamperfree place.
3. A computes a check value $h(Cred_A)$ on his credentials and sends it to B using a second, independent and secure channel.
4. B computes the check value on the credentials of A received in the first pass, and compares the result with the check value received. If they are equal, B is convinced of the authenticity of A's public key, and can use it and/or store it in a tamperfree list of active public keys.

6.3.2 Public key certification

Another way to achieve the authentication of the entities' public keys is by exchanging the public keys in the form of certificates. A *certificate* is in [II9798-3] defined as "the credentials of an enity signed with the private key of the certification authority CA and thereby rendered unforgeable".

For this mechanism it is required that each entity A is in possession of a valid certificate $Cert_A$ of his credentials $Cred_A$. An entity (say B) who wants to obtain another entity's (say A's) public key and verify its validity has to be in possession of the public verification key v_{CA} of the certification authority that issued the certificate $Cert_A$. The protocol is as follows:

1. A sends to B his certificate $Cert_A$ and B's distinghuished identifier.
2. B uses the public verification key v_{CA} of the certification authority to verify the authenticity of the credentials and to check the current validity of A's public key.

Before public keys, generated by an entity himself or by the CA on behalf of the entity, can be distributed in the form of certificates, they must be certified. For this, an entity A provides the CA with at least the necessary identifying information and optionally his public key in an authentic way. Upon receipt, the CA verifies the authenticity of the identifying information, optionally generates the asymmetric key pair(s) for A, and adds system specific data to get A's credentials $Cred_A$. This is signed by the CA using its own signature key to produce A's certificate: $Cert_A = s_{CA}(Cred_A)$. The certificate is given back to A who puts it in a directory.

Once a certificate has been generated, no special measures need be taken to ensure its confidentiality or integrity. The certificates may be stored in a public directory such that the users can easily get access to them. Another possibility is that a user sends his certificate to each user when he asks for it; each user may then have a directory to store the certificates obtained

for later use, or he may store the public keys in a tamperfree place.

A certificate has a lifetime which is indicated by a validity period stated in the certificate, or which is otherwise defined by the CA's management.

In a system, usually more than one key certification facility will exist to minimize the physical activity and to spread the working load of the certification. Therefore a construction has to be made that enables entities A and B that are registered at different certification authorities $CA1$ and $CA2$, respectively, to communicate. The notation $Cert_X^Y$ is used for Y's signature on X's credentials.

In [CCITT88] is explained that it is necessary that a chain of authentication authorities between A and B (and thus between $CA1$ and $CA2$) exists. This document assumes the existence of a public directory, to which all users have access, containing all the certificates of certification authorities $Cert_{CA\cdots}^{CA\cdots}$.
The chain can be one of, or a combination of, two essentially different components. These are depicted in figure 6.1.

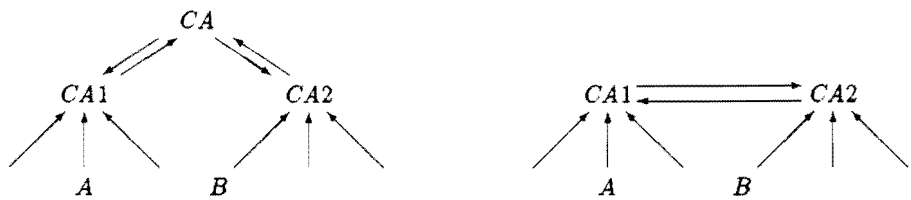


Figure 6.1: Certification Authority chains

In an hierarchical approach, one "top" certification authority certifies the certificates of the certification authorities one level lower and the other way around. In this situation, user A can verify $Cert_B^{CA2}$ using the chain $CA2$ - CA - $CA1$: first A verifies $Cert_{CA}^{CA1}$, then $Cert_{CA2}^{CA}$, and finally $Cert_B^{CA2}$. A can find all certificates needed in the public directory, except $Cert_B^{CA2}$ which he receives from B. Another possibility is that all certification authorities certify each other, so-called cross-certification. Before he can verify $Cert_B^{CA2}$, user A only has to verify $Cert_{CA2}^{CA1}$, which he can find in the public directory. The advantage of cross-certification is clear: the number of verifications A has to perform is smaller. However, the public directory has to contain (much) more certificates than in the hierarchical approach.

6.4 Smart card management

In a key management service not only a secure management of the keys itself has to be defined, but also a management of the tokens used is required. The key management service KMS/KMT developed by P.I.T.S. uses:

- smart cards. In particular, the TB100 card is used, but the design of the KMS/KMT is made easily adaptable to other kinds of smart cards;

- personal calculators, where the KMS/KMT focusses on the Digipass, manufactured by the company DIGILINE. This calculator allows secure data transactions as user authentication, mutual authentication, and data signature (an operation based on a symmetric algorithm to verify the integrity of data). The Digipass is protected by a PIN;
- floppy disks.

The management must be user-friendly and achieve a high level of security.

In the next five parts of this section we will treat some of the aspects related to the management of smart cards in detail. This management is valid for the three level hierarchy as introduced in section 5.2.2. Thereafter, in the last part of this section, we show that the use of smart cards is not obligated; a *smart floppy* can be used instead.

6.4.1 *PIN* modification

For the *PIN* modification of the Smart Cards, two options have been retained:

- no *PIN* modification allowed
- *PIN* modification allowed, with a certain maximum of modifications after the original *PIN* entry.

The option chosen should be indicated in the smart card.

For the TB100 smart card a detailed algorithm is implemented. In the header of the *PIN* zone is indicated whether *PIN* modification is allowed of not, and if it is allowed, how many modifications (at most 6 after the original *PIN* entry).

For the DX smart card, algorithms are not yet implemented. Basically, they can be the same as for the TB100 card.

6.4.2 Recycling of blocked Smart Cards

After a three wrong *PIN* entries, access to the smart card will automatically be blocked. This prohibits that a disrupter who has got hold of the card can by trial and error find the correct *PIN*. If the authorized user of the card has made three false *PIN* entries, it is be possible to unblock the card.

6.4.3 Invalidation of Smart Cards

The invalidation of parts of a smart card prevents further use of those parts.

In the TB100 card Data Files and Zones, the structures of this card (see section 5.2.1), can be invalidated. If a *DF* is invalidated, so are all the structures within it; so, to invalidate the whole card, it is only necessary to invalidate the *CDF*.

6.4.4 Smart Card Validity Dates

In certain zones on a smart card, validity dates may be stored. The modification of these validity dates is under the responsibility of the Management Authority (see section 5.2.2). Which keys have to be presented depends on the write conditions of the specific zone where the Validity Date has to be modified.

6.4.5 Loss of Smart Cards

In case of loss of a smart card the action(s) to be undertaken depend on the level of that specific card. We will give an outline of those actions based on the three-level hierarchical key design as explained in section 5.2.2.

In case the Company Authority looses his Smart Card, a copy of it should be available in a safe. Clearly, a new copy has to be made immediately and stored again in that safe. It is very important to point out that every copy has its own Smart Card Keys (depending on the Smart Card's Serial Number), and since for every action where the Company Authority Card is required, a mutual authentication is mandatory with the CAD applying some of these Smart Card Keys, the system is even protected against an intruder who could put his hands on the valid Company Authority Card and its current *PIN*.

In case one of the Intermediate Authorities looses his smart card, it immediately has to inform the Company Authority (=Security Officer). The Security Officer will personalise a new card. Again, the keys specific to the Smart Card will be different since they depend on the Smart Card's Serial Number. But for the TB100 card the Root Keys, that are dependent on the Root Keys of the Company Authority Smart Card and the secrets stored in a safe, will be the same.

In case a User looses his smart card, he has to inform immediately the Management Authority. Then the Management Authority has to contact the Personalisation Authority and the Application Authority(ies) to personalise a new user card. For the TB100 card the keys of this new card are derived from the same Root Keys stored in the Authority Cards as the keys in the lost card.

6.4.6 Smart floppy

Smart cards are nowadays generally considered as a convenient, safe, and inexpensive means for the storage of the secret keys needed in authentication and signature protocols. However, in order to use smart cards, a smart card reader is needed. Sometimes the requirement for such smart card readers is impractical, for instance when laptops are used outside of the desk. An alternative is to put the secret key on a floppy disk, called a *private smart floppy*, protected by a strong password.

While a smart card is protected by a PIN, a smart floppy is protected by a password. The information stored on the two devices, and the functionality of them is exactly the same.

Chapter 7

Legal aspects

Using the exchange of electronic data to replace traditional, paper based exchange of business documents is still a relatively new concept. Businesses require not only from electronic document handling that it offers positive advantages over present day message delivery systems in terms of speed and accuracy of message handling, but also that it can be used to create business obligations which can be relied upon. Both legal and technical means have a part to play in ensuring the necessary level of business confidence ([BGM91]).

7.1 Tasks of signatures

At the present time, letters of intent are usually manually signed to make them legally valid. Technically, letters of intent could be exchanged in any form, but in most countries specific formats are obliged for specific documents. One of the most important prescriptions is that it has to be in writing ([R92]), because a declarant has "to write personally his name below the letter of intent". The manual signature has five functions, but in principle all of them could as least as good be fullfilled by digital signatures:

1. The **concluding** function of the signature is that it closes and completes the declaration; it makes it from an intentition into a statement. Digital signatures do this better than handwritten ones, since a digitally signed document cannot be changed undetected. It is no longer possible to have blank signed documents: only completely finished documents can be signed.

2. A handwritten signature shows the **identity** of the signer. Also a digital signature achieves the identification of the signer, because the public key needed to verify it is, e.g. by a certificate, linked to the signer's identity (or a pseudonym).

3. An handwritten signature has also an **authenticity** function: it guarantees the origin of the letter of intent. A digital signature subtle improves this function, because it does not guarantee the authenticity of the letter of intent, but of its contents.

4. The signing operation, both manual writing and performing the procedure for computing a digital signature, **warns** the signer that he is making a legally valid statement. This protects him from rushing things.

5. A signature is a **proof** (to a third party) that the signer has committed himself to the text in the letter of intent.

7.2 Privacy aspects

An important advantage of digital signatures above handwritten ones, is that for digital signatures not a name, but a key is responsible for the trusthworthiness of an action. This enables an individual to act under various pseudonyms, which prevents different organisations from combining their data and making up a complete profile of an individual.

By law, individuals have a right for privacy: banks are not allowed to publish their customer's financial situation, physicians should keep medical records secret, and so on. Up till now, this principle right had to be achieved by punishments for contraventions. However, when an individual cannot be linked to his various records, it is impossible to violate his privacy!

7.3 EDI

EDI (Electronic Data Interchange) is the electronic transfer form computer to computer of commercial or administrative dat using an agreed standard to structure the message data. The set of international EDI standards is known as UN/EDIFACT ([I9735]).

In the present EDI community, it is assumed that the users of EDI will already be in an underlying commercial relationship. In the October 1989 version of the United Kingdom EDI Association's Standard Interchange Agreement is stated as a fundamental principle that the Agreement only is related to the interchange of data, and not to the various underlying commercial or contractual obligations of the parties.

It seems wrong to assume that EDI will continue to be used only in closed user communities between existing trading partners. For technological, liberalisation and international trade reasons one can expect that the use of EDI will extend to a larger and less clearly identified user community. When EDI users that have had no commercial contact before want to do business, they need to establish a thrustworthy basis by electronic means. A legally valid digital signature seems to be essential for this.

In the trade sector some initiatives were taken during the last years to deal with the legal aspects of EDI in this area ([DS93]). A legal document, known as an *Interchange agreement* has been introduced. The Uniform rules of conduct for the interchange of data (UNCID, [ICC88]), developed by the International Chamber of Commerce and adopted by the United Nations, provides a basis for establishing interchange agreements. The UNCID rules already include some security requirements from the EDI systems used: entity identification, (message) content integrity, and authentication of the message transfer appear as necessary services. Even an optional request for acknowledgement of receipt from the sender to the receiver is mentioned. However, in most trade applications up to now, the interchange agreements are applied without any of the security services. More recently, the CEC published a model interchange agreement which trading partners may adopt as a legal basis for trading electronically through Europe ([CEC89]).

Chapter 8

Zero-Knowledge protocols

8.1 Introduction

Zero-knowledge techniques are a relatively new concept in cryptography; the first practical protocol was described by A. Fiat and A. Shamir in 1987 ([FS87]). Zero-knowledge authentication and signature schemes require significantly less computational power than public key techniques. This makes them very attractive, especially in an environment where smart cards are employed. The main disadvantage of zero-knowledge techniques is that they usually require many iterations of a basic protocol, thus many interactions between prover and verifier.

8.1.1 What is zero-knowledge?

Suppose a prover P wants to convince a verifier V of something using an interactive protocol in which the two parties are allowed to exchange messages and generate random numbers. The protocol to be used should have the following two properties ([vH92]):

- **completeness:**
  If P is right, then a correct protocol execution by both parties results with overwhelming probability in V accepting the proof.
- **soundness**
  If P is not right, then if V correctly executed the protocol, he will accept P's proof with negligible probability, no matter how P deviates from the correct protocol execution.

A protocol like this is zero-knowledge if V does not learn anything more from the interaction beyond the validity of P's assertion. More formally ([vH92], [SP89]):

- **zero-knowledge**
  No matter how V behaves, the communication between P and V can always be simulated by a (probabilistic polynomial-time) algorithm that does not know P's secrets.

Zero-knowledge interactive proofs are introduced in [GMR85].

8.1.2 Zero-knowledge authentication protocols

In a zero-knowledge authentication protocol, an entity proves his identity without revealing a password or other information ([FFS88]). The computational complexity of zero-knowledge schemes is typically less than that of public key cryptosystems like RSA, and this complexity strongly depends on the security level required. The main principles of an interactive zero-knowledge authentication protocol are ([FP91]:

- The security level of the verifier depends on the number of iterations of an elementary protocol.
- The prover protects himself and his knowledge by giving the verifier in every iteration access only to a randomly selected part of his information.
- Zero-knowledge protocols make use of one-way funtions in order to protect the prover's secret data and to minimize information flow.
- Electronic identification with a zero-knowledge scheme is a non-transitive process, which means that the verifier does not learn anything that would allow him to impersonate the prover.

Zero-knowledge mechanisms typically consist of two phases. The first phase is carried out in a trusted center. After this phase a user may identify himself without intervention of the center in a second phase carried out between himself (the prover) and a verifier. No explicit key management is necessary, the keying material used within the mechanisms is generated by the trusted center and stored at the prover's site only ([FP91]).

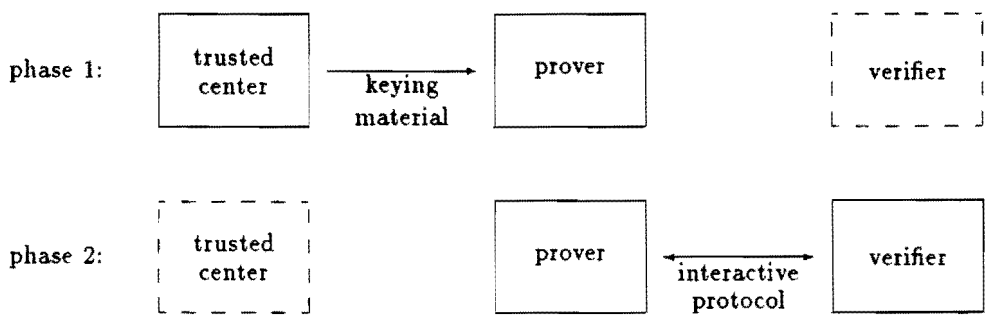


Figure 8.1: Zero-knowledge mechanism

In the next sections we will describe five zero-knowledge schemes for authentication and signing. First the Fiat-Shamir schemes ([FS87]) will be presented. Then the Guillou-Quisquater ([GQ88]) schemes will be described. This is a special version of a generalization of the Fiat-Shamir schemes. Afterwards an efficiency improvement of the Fiat-Shamir schemes from Ong and Schnorr ([OS91]) is presented. Then Schorr's scheme ([S90a]), which uses a somewhat different approach, will be given. Thereafter we will present identity-based schemes of Girault and Pailles ([GP90]) for authentication and signing.

8.2 Fiat-Shamir

At Crypto '86, A. Fiat and A. Shamir ([FS87]) presented a zero-knowledge authentication protocol that can be used with smart cards. This protocol can easily be turned into a signature scheme. We will describe both schemes below.

8.2.1 Fiat-Shamir authentication protocol

Protocol description

Before the center starts issuing cards, it chooses and publishes a modulus n which is the product of two distinct random primes p and q, and a pseudo random function f which maps arbitrary strings to the range $[0, n)$. The function f should be indistinguishable from a truly random function by any polynomially bounded function. [GGM84] describes a family of functions which is provably strong in this sense, but in practice simpler and faster functions (e.g. multiple DES) can be used without endangering the security of the scheme.

The center prepares for each user of the system a string ID, containing all his relevant information. Then the center computes $f(ID, w)$ for small values of w, and picks k distinct w's for $f(ID, w)$ is a quadratic residue modulo n, i.e. there is an $x \in [0, n)$ such that $f(ID, w) = x^2 \bmod n$. These w's are denoted by $w_1, \ldots, w_k$, and the corresponding $f(ID, w_j)$ by v_j. Then the center computes the secret parameters s_j as the smallest square root of v_j^{-1} modulo n. Thus s_j is the smallest number in $[0, n)$ such that $s_j^2 = v_j^{-1} \bmod n$, or, equivalently, $s_j^2 v_j = 1 \bmod n$. The smart card issued to the user contains ID, the k values w_j, and the k values s_j.

Two remarks about this initialisation are:

- For non-perfect functions f it may be advisable to randomize ID by concatenating it to a long random string R chosen by the center, stored in the card, and revealed along with ID.
- It is possible to eliminate the center, and let each user choose its own n and publish it in a public directory. However, this variant makes the scheme much less convenient.

The only general information a verifier needs to store is the universal modulus n and the function f.
When a prover A wants to prove his identity to a verifier B, he proves that he knows $s_1, \ldots, s_k$ without giving away any information about their values. This is done using the following protocol, which is depicted in figure 8.2:

step 1 A sends ID and the k values w_j to B.

step 2 B generates $v_j = f(ID, w_j)$ for $j = 1, \ldots, k$.

Repeat steps 3 to 6 for t times, where the i-th iteration, $1 \le i \le t$, is the following:

step 3 A chooses at random $r_i \in [0, n)$ and sends (part of) $x_i = r_i^2 \bmod n$ to B.

step 4 B sends a random binary vector $b_i = (b_{i1}, \ldots, b_{ik})$ to A.

step 5 A responds with $y_i = r_i \prod_{b_{ij}=1} s_j \bmod n$.

step 6 B checks that $y_i^2 = x_i \prod_{b_{ij}=1} v_j^{-1} \bmod n$, or equivalently that $x_i = y_i^2 \prod_{b_{ij}=1} v_j \bmod n$. If only part of x_i is sent in step 3, B has to check that this part is equal to the corresponding part of $y_i^2 \prod_{b_{ij}=1} v_j \bmod n$.

And finally:

step 7 B accepts A's proof only if all t checks are succesful.

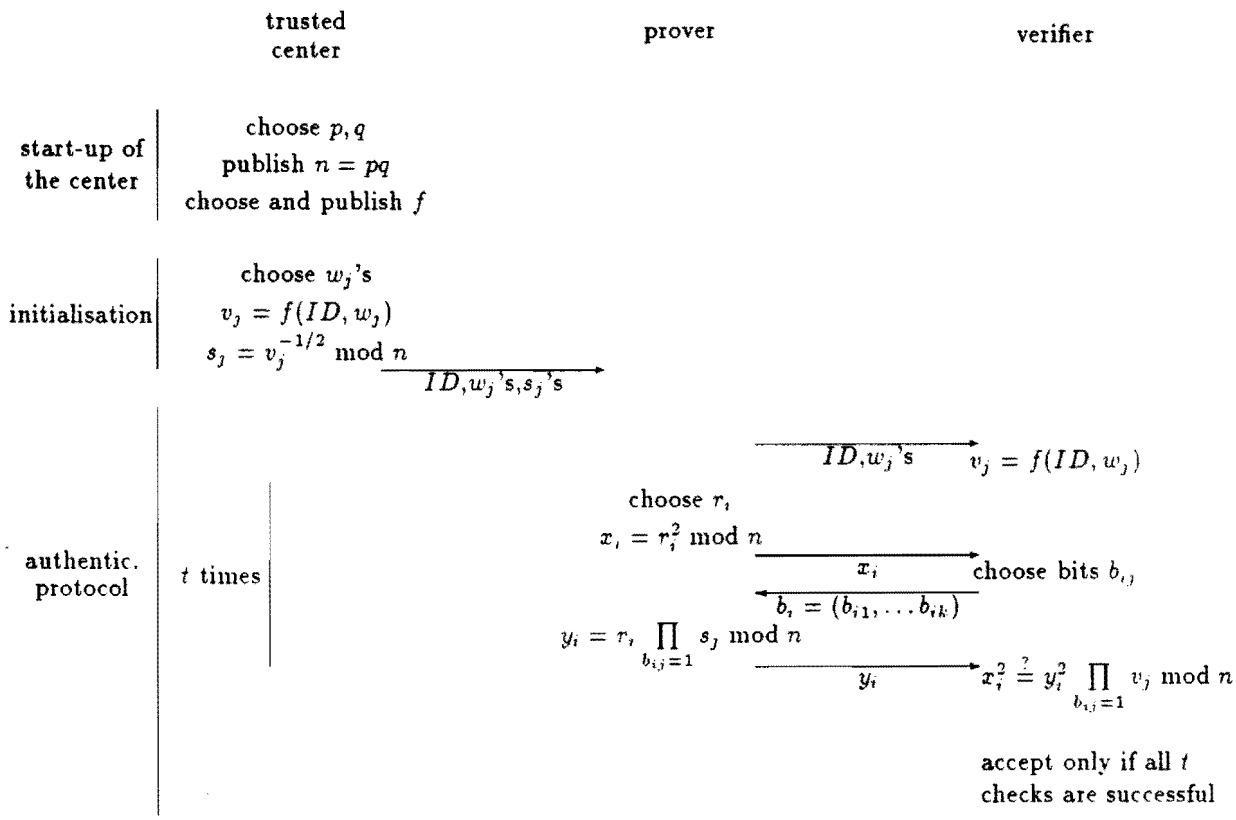


Figure 8.2: Fiat-Shamir authentication protocol

Fiat and Shamir ([FS87]) suggest various generalisations of this scheme: the square roots can be replaced by cubic or higher roots, and the b_i vectors can be made non-binary. However, the security lemmas proven in the next section cannot be generalised as easy. [OO90] calculates some security and parameter conditions for this case. [MS90] reduces for the original scheme the verifier's complexity to less than 2 modular multiplications while the prover's complexity remains unchanged. This is done by letting each user choose his own modulus n, and choosing the v_i values to be the first k primes.

Security

The following lemma, achieving completeness, can easily be proven ([FS87]):

Lemma 1 *If A and B follow the protocol, B always accepts the proof as valid.*

According to Lemma 2 below, the algorithm is also sound ([FS87]):

Lemma 2 *Assume that A does not know the s_j values and cannot compute in polynomial time the square root of any product of the form $\prod_j v_j^{a_j} \bmod n$ where $a_j \in \{-1, 0, 1\}$ for $1 \leq j \leq k$, and not all a_j are zero. If B follows the protocol (and A performs arbitrary polynomial time computations), B will accept the proof as valid with probability bounded by 2^{-kt}.*

In fact, if A does not know the s_j, and cannot compute square roots as stated in the lemma, the best way to cheat is by guessing the correct vectors b_{ij} and sending x_i according to them.

The intuitive reason the proof reveals no information whatsoever about the s_j is that the x_i are random squares, and each y_j contains an independent random variable which masks the values of the s_j. All the messages sent from A to B are thus random numbers with uniform probability distribution, and cheating by B cannot change this fact. This is a sketch of the proof of Lemma 3:

Lemma 3 *For a fixed k and arbitrary t, this is a zero-knowledge proof.*

Complexity

The number of computations that has to be performed by the two parties in the scheme, the amount of information that has to be exchanged, and the storage capacity required are calculated below ([FS87], [FP91]).

Time

Both the prover and the verifier need to do t modular squarings, and on average $t \cdot k/2$ modular multiplications, which gives $t(k+2)/2$ multiplications. In addition, the verifier has to compute the k values v_j.

Communication

The prover has to send the string ID to the verifier, and during the proof t times an x_i, k bits b_{ij}, and an y_i have to be exchanged. Neglecting ID, this sums up to $t(2\log_2 n + k)$ bits.

Space

To store the secret values s_j, the prover needs a $k \log_2 n$-bit ROM. Besides that he needs a ROM to store his identity ID and the k values w_j. If the w_j's are chosen small, the place needed to store them is very small, $\mathcal{O}(k \log_2 k)$, and we will neglect this. Finally, the prover needs a $\log_2 n$-bit RAM to store r_i in each step of the proof.
The verifier needs a $k \log_2 n + \log_2 n + k$-bit RAM to store the values v_j, and x_i and b_i during the proof.

The time, space, communication and security of the scheme can be traded of in many possible ways, depending on the choices of k and t. For an identification scheme, a typical security level is 2^{-20}. For a 512-bit modulus, this can be achieved by choosing $k = 5$ and $t = 4$, which gives on average 14 multiplications, 323 bytes exchanged and the need for a 320-byte secure memory. If, for example, k is increased to 18, and if the vectors b_i have at most three ones in them, in each iteration 988 ($\approx 2^{10}$) vectors are possible. Thus, with two iterations we get already the 2^{-20} security level. The number of transmitted bytes drops to $2(640 + 18)/8 = 165$, and the average number of multiplications for both parties drops to $2(1 + 2.8) = 7.6$. The drawback is that a 1152-byte secure ROM is needed. Note that the verifier needs to compute at most 6 out of the 18 values v_j to verify the proof. The optimal choices of k and t and the matrix b_{ij} depend on the relative costs of the various resources.

Further improvements in speed can be obtained by parallelizing the operations. A can prepare x_{i+1} and y_{i+1} while B is still checking x_i and y_i, and parallel multipliers can be used to compute products in log k dept.

8.2.2 Fiat-Shamir signature scheme

The role of the verifier B in the interactive authentication protocol is passive, but important. Since the vectors b_i are chosen at random, they contain no information, but their unpredictability prevents cheating by the prover A. To turn this scheme into a signature scheme, B's role is replaced by the function f.

Protocol description

To sign a message M, A carries out the following protocol:

step 1 A picks random $r_1, \ldots, r_t \in [0, n)$ and computes $x_i = r_i^2 \bmod n$ for all r_i.

step 2 A computes $f(M, x_1, \ldots, x_t)$ and uses its first kt bits as values b_{ij} $(1 \leq i \leq t, 1 \leq j \leq k)$.

step 3 A computes for $i = 1, \ldots, t$

$$y_i = r_i \prod_{b_{ij}=1} s_j \bmod n$$

and sends ID, his k values w_j, M, the matrix b_{ij} and all the y_i to B. A's signature on M consists of the matrix b_{ij} and the y_i.

To verify A's signature on M, B acts as follows:

step 1 B computes $v_j = f(ID, w_j)$ for $j = 1, \ldots, k$.

step 2 B computes for $i = 1, \ldots, t$

$$z_i = y_i^2 \prod_{b_{ij}=1} v_j \bmod n.$$

step 3 B computes $f(M, z_1, \ldots, z_t)$, and verifies that the first kt bits are the b_{ij}.

Security

To prove the security of the above signature scheme, we assume that f is a truly random function. Then the following lemma follows easily:

Lemma 4 *If A and B follow their protocols, B always accepts the signature as valid.*

One obvious way to forge signature for arbitrary messages is to guess the matrix b_{ij}. If this is done T times, the probability of success is $T \cdot 2^{-kt}$. In [FS87] is shown that this attack is essentially optimal if factoring of the modulus n is infeasible. This holds even if the forger has been provided with an arbitrary amount of signatures of his choice.

We want to stress that the above signature scheme coming from a zero-knowledge scheme is not zero-knowledge. In fact, signature schemes cannot be zero-knowledge by definition, since if everyone can recognize valid signatures but no one can forge them, B cannot generate by

himself A's messages with the same probability distribution. However, the information about the s_j values that B gets from signatures generated by A is so implicit that it cannot be used to forge signatures.

Complexity

In the signature scheme, an adversary knows in advance whether his signature will be accepted as valid, and thus by experimenting with 2^{kt} random values r_i, he is likely to find a signature he can send to B. Consequently, the product kt must be larger for a signature scheme than for an identification scheme; $kt = 72$ is a general accepted value.

The length of the signature, consisting of b_{ij} and y_i for $1 \leq i \leq t, 1 \leq j \leq k$, is $k(t + \log_2 n)$.

8.3 Guillou-Quisquater

In the Fiat-Shamir protocol, the security level is 2^{-kt}. To obtain an acceptable level, the number of iterations (the interaction between prover and verifier), and/or the memory needed by the prover need to be large. L.C. Guillou and J.J. Quisquater introduced in [GQ88] an optimization of this protocol. They generalize the squaring by introducing a parameter c, which can have any value ≥ 2, and compute c-th powers and roots. The parameters k and t are both fixed to 1. In this way they achieve a protocol with only one iteration, thus three steps, and low memory. However, the price to pay for this is longer computations.

8.3.1 Guillou-Quisquater authentication protocol

Protocol description

Before describing the protocol, we will define what shadows and imprints are:

- **shadow**
  For a short message (half the length of n), the secret operation S consists of
  - completing the message with a similar-sized redundancy (the shadow), followed by
  - extracting the c-th root of the obtained element.

  This method with shadow produces credentials.
  Due to multiplicative properties of RSA, the shadow must not be expressed multiplicatively in terms of the message.
- **imprint**
  A long message is not signed as chained blocks, but the following secret operation S is carried out: an imprint h (shorter than n) of the message is computed using a one-way collisionfree hash function.

In the description we will use a notation and terminology which differs significantly from the original paper to show the resemblance with the Fiat-Shamir protocol of the previous section.

The center computes for a user with identity ID the shadowed identity as described above. The message completed with the shadow is denoted by v, and the c-th root of v by s. The value s is issued to the user.
A prover A can now prove his identity to a verifier B using the protocol depicted in figure 8.3:

step 1 A sends ID to B.

step 2 B uses ID to compute v.

step 3 A chooses at random $r \in [0, n)$ and sends (part of) $x = r^c \bmod n$ to B.

step 4 B sends a random b, $(0 \leq b < c)$ to A.

step 5 A responds with $y = r \cdot s^b$.

step 6 B compares the given bits of x with the corresponding bits of $y^c v^{-b} \bmod n$, and accepts A's proof if they are equal.

Security

Lemmas 1 and 2 of the Fiat-Shamir protocol, achieving completeness and soundness, can easily be adapted to hold for the Guillou-Quisquater protocol:

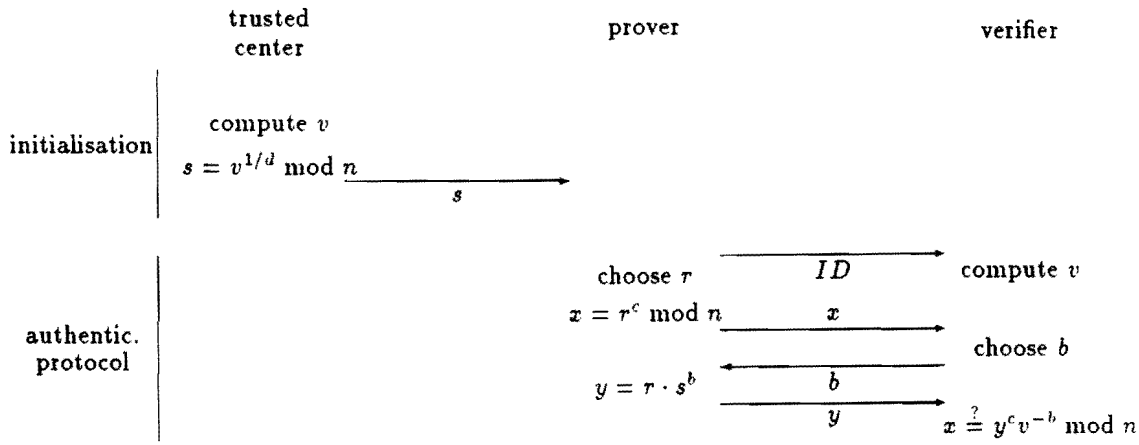


Figure 8.3: Guillou-Quisquater authentication protocol

Lemma 5 *If A and B follow the protocol, B always accepts the proof as valid.*

Lemma 6 *Assume that A does not know s and cannot compute in polynomial time the c-th root of $v^a \bmod n$, where $0 < |a| < c$. If B follows the protocol (and A performs arbitrary polynomial time computations), B will accept the proof as valid with probability bounded by $1/c$.*

A cannot do essentially better than guessing the correct b, and the probability of succes is $1/c$.

Intuitively, the proof reveals no information whatsoever about s because x is a random c-th power, and y contains an independent random variable which masks the value of s. This makes the following lemma plausible:

Lemma 7 *This is a zero-knowledge proof.*

As long as the size of the exponent c is sufficient to reach directly the level of security requested, no repetitions are needed.

Complexity

The number of computations that has to be performed by the two parties in the scheme, the amount of information that has to be exchanged, and the storage capacity required are calculated below.

Time

Both the prover and the verifier need to do 2 exponentiations modulo n, and one modular multiplication. Since exponentiation modulo n can be implemented with about $\frac{3}{2} \cdot \log_2(exponent)$ modular multiplications ([FP91]), the two parties have to perform $3\log_2 c + 1$ modular multiplications each. In addition, the verifier has to compute v.

Communication

The prover has to send the string ID to the verifier, and during the proof x, an integer smaller than c, and an y have to be exchanged. Neglecting ID, this sums up to $2\log_2 n + \log_2 c$ bits.

Space
To store the secret s value, the prover needs a $\log_2 n$-bit ROM. He also needs a ROM to store ID. To store r during the proof, a $\log_2 n$-bit RAM is needed.
The verifier needs a RAM of $\log_2 n + \log_2 c$ bits to store v and b during the proof.

8.3.2 Guillou-Quisquater signature scheme

To sign a message M, A carries out the following protocol:

step 1 A picks at random an $r \in [0, n)$ and computes $x = r^c \bmod n$.

step 2 A computes $b = f(M, x)$.

step 3 A computes $y = r \cdot s^b \bmod n$ and sends his ID and w, M, b and y to B. Thus, A's signature on M consists of b and y.

To verify A's signature on M, B acts as follows:

step 1 B computes $v = f(ID, w)$.

step 2 B computes $z = y^c v^{-b} \bmod n$.

step 3 B verifies that $b = f(M, z)$.

The following lemma can easily be proven:

Lemma 8 *If A and B follow their protocols, B always accepts the signature as valid.*

The signature is the pair (b, y), and has a length of $\log_2 n + \log_2 c$ bits.

8.4 Ong-Schnorr improvement of Fiat-Shamir

H. Ong and C.P. Schorr presented on Eurocrypt '90 two improvements to the Fiat-Shamir authentication and signature scheme ([OS91]). The communication of the authentication protocol is reduced to one round, while the efficiency of the scheme is preserved. This reduces also the length of the signatures. For the secret keys small integers can be used, which reduces the time for signature generation by a factor 3 or 4.

8.4.1 Ong-Schnorr authentication protocol

Protocol description

The center chooses two primes p and q, and computes and publishes their product n. The center also chooses a key pair consisting of a private and public key, and publishes the public key.

Each user chooses at random k numbers $s_j \in [1, n)$ such that $gcd(s_j, n) = 1$. These compose his private key $s = (s_1, \ldots, s_k)$. The corresponding public key is $v = (v_1, \ldots, v_k)$ such that $v_j = s_j^{-2^t} \bmod n$.

When a user registers at the center, the center prepares an identification string ID, and generates a certificate $C(ID, v)$ for the identification string and the user's public key.

The authentication protocol that is executed by a prover A and a verifier B is as follows:

step 1 *(preprocessing)* A chooses at random $r \in [1, n]$ and computes $x = r^{2^t} \bmod n$.

step 2 *(initiation)* A sends ID, v, $C(ID, v)$, and x to B.

step 3 B verifies $C(ID, v)$ and sends a random string $b = \{b_{ij}\}$ $(1 \leq i \leq t, 1 \leq j \leq k)$ to A.

step 4 A responds with

$$y = r \prod_j s_j^{\sum b_{ij} 2^{i-1}} \bmod n.$$

step 5 B computes z,

$$z = y^{2^t} \prod_j v_j^{\sum b_{ij} 2^{i-1}} \bmod n$$

and accepts A's proof of identity if $z = x$.

This protocol is depicted in figure 8.4.

Security

Using the definitions, a straightforward calculation proves the following:

Lemma 9 *If A and B follow the protocol, B always accepts the proof as valid*

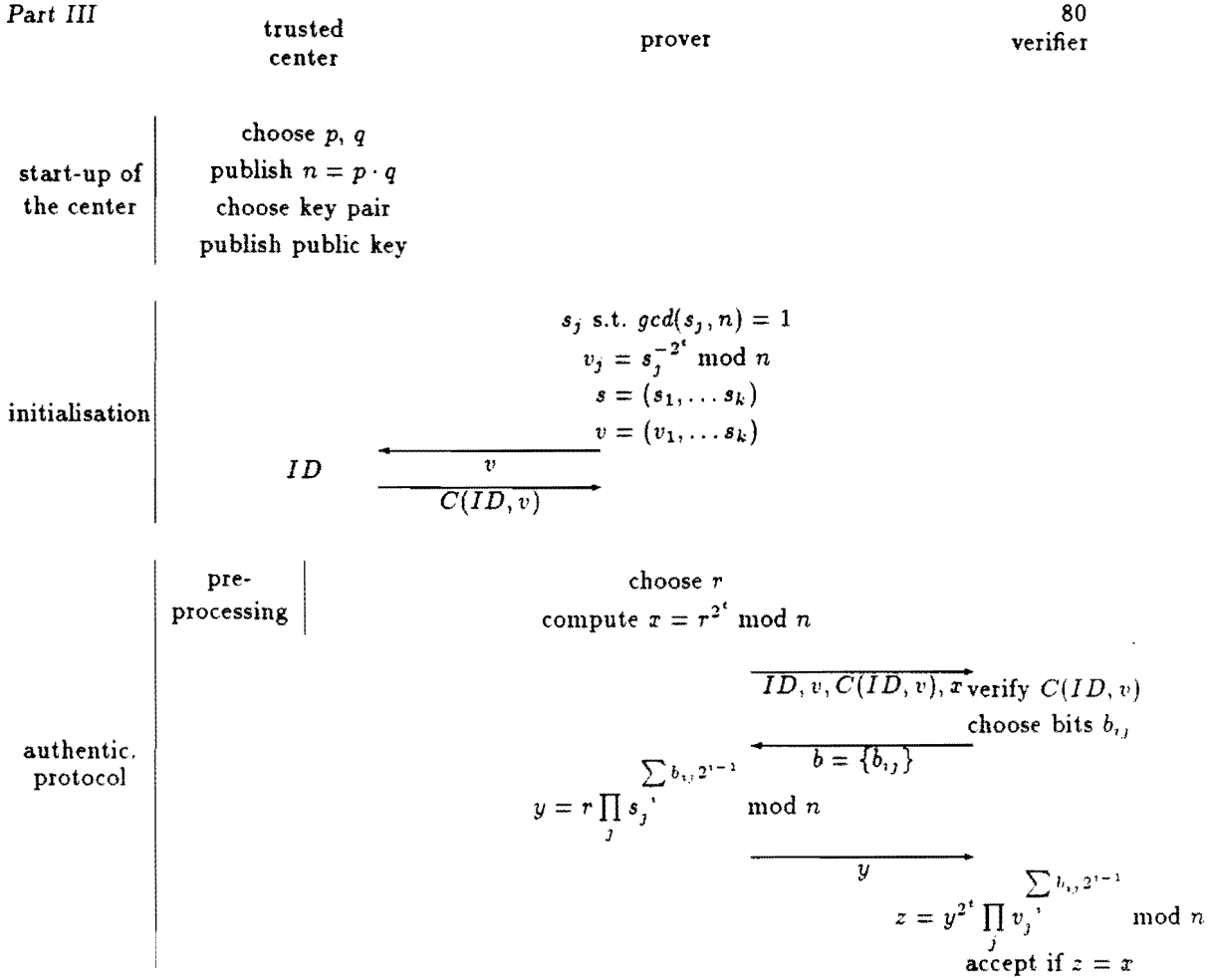


Figure 8.4: Ong-Schnorr authentication protocol

Next we consider the possibilities of cheating for A and B.

A fraudulent A can cheat by guessing the challenge b and sending for an arbitrary r

$$x = r^{2^t} \prod_j v_j^{\sum b_{ij} 2^{t-i}} \bmod n$$

in step 2, and in step 4 $y = r$.
The probability of success for this attack is 2^{-kt}, and in [OS91] is proven that this success rate cannot be increased unless some non-trivial 2^t-th root modulo n can be computed easily.

Complexity

Time

Prover A has to do a preprocessing step of computing r^{2^t}, which costs t squarings. During the protocol A has to compute

$$r \prod_j s_j^{\sum b_{ij} 2^{t-i}} \bmod n.$$

Using the algorithm below, this can be done in on average $t(k+2)/2-1$ modular computations for a randomly chosen b ([OS91]):

$$\begin{aligned} y &:= \prod_{b_{i,j}=1} s_j \bmod n; \\ y &:= y^2 \prod_{b_{i,j}=1} s_j \bmod n \text{ for } i = t-1, \ldots, 1; \\ y &:= y \cdot r \bmod n. \end{aligned}$$

Verifier B has to do a similar computation; he has to calculate

$$y^{2^t} \prod_j v_j^{\sum b_{i,j} 2^{i-1}} \bmod n.$$

The following algorithm uses for a random chosen b on average $t(k+2)/2+1$ computations ([OS91]):

$$\begin{aligned} z &:= y^2 \prod_{b_{i,j}=1} v_j \bmod n; \\ z &:= z^2 \prod_{b_{i,j}=1} v_j \bmod n \text{ for } i = t-1, \ldots, 1. \end{aligned}$$

Communication
The prover has to send ID, v, $C(ID, v)$ and x to the verifier in step 2, which, neglecting ID and $C(ID, v)$, are $(k+1)\log_2 n$ bits. In step 3 the verifier has to send a kt-bit string, and the prover's response consists of a $\log_2 n$-bit y. Thus, together $(k+2)\log_2 n + kt$ bits have to be communicated.

Space
The prover needs a $k \log_2 n$-bit ROM to store the secret s, and ROM which does not have to be secret to store ID and the $k \log_2 n$-bit v. He also needs a RAM of $\log_2 n$ bits to store r during the proof.
The verifier needs a RAM of $k \log_2 n + \log_2 n + kt$ bits to store v, x and b during the proof.

8.4.2 Ong-Schnorr signature scheme

Protocol description

For the signature scheme the same preliminaries as for the authentication scheme are needed, and one extra: the center has to choose and publish a one-way hash function $h : \mathcal{Z}_n \times \mathcal{Z}_n \rightarrow \{0,1\}^{kt}$.

To sign a message M, A carries out the following protocol:

step 1 *(preprocessing)* A chooses at random $r \in [1, n)$ and computes $x = r^{2^t} \bmod n$.

step 2 A computes $b = \{b_{ij}\} := h(M, x)$.

step 3 A computes

$$y = r \prod_j s_j^{\sum b_{i,j} 2^{i-1}} \bmod n.$$

The output is A's signature on M: (b, y).

To verify A's signature, B needs (b, y), v and M, and he acts as described below:

step 1 B checks the certificate $C(ID, v)$.

step 2 B computes

$$z = y^{2^t} \prod_j v_j^{\sum_{i=1}^{t} b_{i,j} 2^{i-1}} \mod n.$$

step 3 B checks that $h(M, z) = b$.

Security

A similar calculation as in the authentication scheme shows that $z = x$, and therefore $h(M, z) = h(M, x) = b$. This proves the following lemma:

Lemma 10 *If A and B follow the protocol, B always accepts the signature as valid*

In order to falsify a signature for message M, the cryptanalyst has to solve the equation

$$b = h\left(M, y^{2^t} \prod_j v_j^{\sum_{i=1}^{t} b_{i,j} 2^{i-1}} \mod n\right)$$

for b and y.
No efficient way is known to solve this equation.

Complexity

Signer and verifier have to do the same computations as for the signature scheme, except for the computation of an extra computation of $h(M, \cdot)$. Also the same number of bits have to be transferred, and the same size and kind of memory is needed.

The signature (b, y) has a length of $\log_2 n + kt$ bits.

8.4.3 Small integer variant

In the same article the authors propose a variant on the above schemes that reduces the size of the secure memory needed by the prover, and accelerates the generation of signatures considerably ([OS91]). This is done by choosing the s_j to be small integers. The security of the variation is based on the assumption that computing 2^t-th roots modulo n is difficult. No particular algorithm is known to compute 2^t-th roots modulo n given that these 2^t-th roots are of order $n^{2^{-t+1}}$.

For $j = 1, \ldots, k$, let s_j be a random prime in $[1, 2^{64}]$. This interval is large enough to ensure that the s_j cannot be found by exhaustive enumeration. The parameter t must be at least 4, so that $s_j^{2^t}$ is at least of order n^2. The s_j must be primes, for if $s_j = \alpha \cdot \beta$ with $\alpha, \beta \in [1, 2^{32}]$, s_j can be found by solving

$$\beta^{-2^t} = v_j \alpha^{2^t} \mod n \text{ for } \alpha, \beta \in [1, 2^{32}],$$

and this can be done in about 2^{32} steps.

Let us now consider the efficiency of the schemes. Suppose $\sum_j b_{ij} \leq 8$ for $i = 1, \ldots, t$. Then

$$\prod_{b_{ij}=1} s_j < 2^{512},$$

and computing this product does not require any modular reduction. Thus, step 5 of the authentication protocol, and step 3 of the signature generation protocol require at most $2t - 1$ modular multiplications for the prover and signer respectively. The other multiplications are with small numbers, and the effort computing the product (which is less than 2^{512} of at most eight of those small numbers can be taken to cost about half a modular multiplication ([OS91]). Thus, computing the y in both schemes costs an equivalent of $\frac{5}{2}t + 1$ full modular exponentiations. The computation of $x = r^{2^t}$ takes t additional modular squarings, but these can be done in preprocessing mode before the on-line authentication starts, or the message to be signed is known.

8.5 Schnorr

At Crypto '89 C.P. Schnorr presented somewhat different identification and signature schemes ([S90a]) which are based on the discrete logarithm problem instead of the difficulty of factoring.

8.5.1 Schnorr authentication protocol

Protocol description

For $n \in \mathcal{N}$, let $\mathcal{Z}_n$ be the ring of integers modulo n. We identify $\mathcal{Z}_n$ with $\{1, \ldots, n\}$.

The protocol is depicted in figure 8.5, and described below.

Before the (key authentication) center starts issuing cards, it initiates the scheme by choosing:
- primes p and q such that $p-1$ divides q;
- $\alpha \in \mathcal{Z}_p$ with order q;
- its own private and public key.

Every user has a private key s which is a random number in $\mathcal{Z}_q$. The corresponding public key v is $v = \alpha^{-s} \bmod p$. When a user wants to register, the center generates an identification string ID and signs the pair (ID, v) consisting of ID and the user's public key v to get $C(ID, v)$.

To prove his identity to a verifier B, user A carries out the protocol below:

step 1 A sends ID, v and $C(ID, v)$ to B.

step 2 B checks v using $C(ID, v)$.

step 3 A picks at random $r \in \mathcal{Z}_q$. Then A computes $x = \alpha^r \bmod p$, and sends x to B.

step 4 B sends a random number $e \in [0, 2^t - 1)$ to A.

step 5 A responds with $y = r + se \bmod q$.

step 6 B computes $\bar{x} = \alpha^y v^e \bmod p$ and accepts A's proof of identity if $x = \bar{x}$.

To reduce the number of bits transmitted, A can hash x to $h(x)$ and send this value to B in step 3. In step 6, B has now to compare $h(\bar{x})$ and $h(x)$.

Security

It is not difficult to check the validity of the following lemma, stating that the algorithm is sound:

Lemma 11 *If A and B follow the protocol, B always accepts the proof as valid.*

Next we consider the possibilities of cheating for A and B.

A fraudulent A can cheat by guessing the correct e and sending $x = \alpha^r v^e$ in step 3, and $y = r$ in step 5. The probability of success for this attack is 2^{-t}. A lemma in [S90a] proves that this success rate cannot be increased unless computing $\log_\alpha v$ is easy.
The verifier B is free to choose e in step 4, so he can try to choose e in order to obtain useful information from A. But since x is random, x reveals no information. The parameter y equals $y = \log_\alpha x + es \bmod q$, and since B cannot compute $r = \log_\alpha x$ from x, it is unlikely that B can

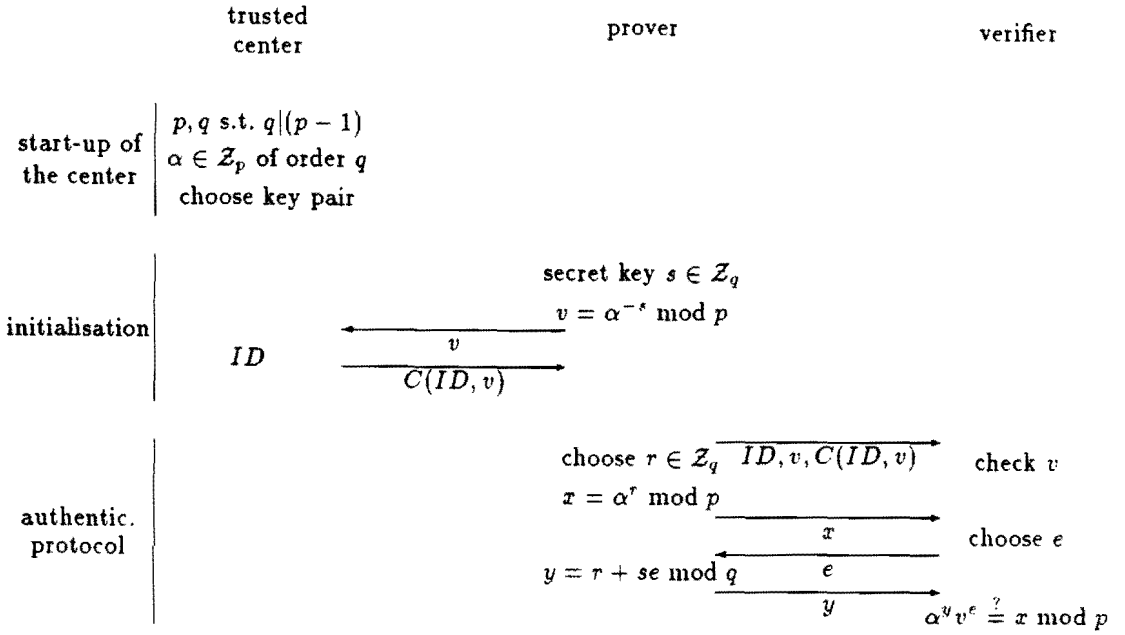


Figure 8.5: Schnorr authentication protocol

choose e as to obtain any useful information about s from y.

Strictly spoken, the scheme is not zero-knowledge because the tripel (x, y, e) may be a particular solution of the equation $x = \alpha^y v^e \bmod p$ due to the fact that the choice of e may depend on x.

Complexity

The complexity of the identification scheme is again divided in the computational complexity, the amount of communication required, and the space that is needed.

Time
The prover A has to do one exponentiation (to the power r) modulo p and one multiplication modulo q. Since $r < q$, this gives about $\frac{3}{2}\log_2 q$ modular multiplications for A. Besides the multiplications, A can have to compute a hash result.
B has to compute one y-th power and one e-th power modulo p, and multiply the results modulo p. Neglecting the last multiplication, this sums up to $\frac{3}{2}(\log_2 q + t)$ modular multiplications, but when the algorithm below is used, $\frac{3}{2}\log_2 q + \frac{1}{4}t$ multiplications suffice:

Write

$$y = \sum_{i=0}^{\lceil \log_2 q \rceil - 1} y_i 2^i \quad \text{with} \quad y_i \in \{0, 1\}.$$

and

$$e = \sum_{i=0}^{\lceil \log_2 q \rceil - 1} e_i 2^i \quad \text{with} \quad e_i \in \{0, 1\}, e_i = 0 \text{ for } i \geq t.$$

Compute αv in advance, and obtain $\bar{x}$ as follows:

$i := \lceil \log_2 q \rceil$; $z := 1$;
while $i \geq 0$ do $i := i - 1$; $z := z^2 \alpha^{y_i} v^{e_i} \bmod p$;
$\bar{x} := z$.

On average, half of the bits y_i with $i \geq t$ are zero, and $e_i = y_i = 0$ holds for $\frac{1}{4}$-th of the $i < t$, thus $\frac{1}{2}(\log_2 q - t) + \frac{1}{4}t$ modular multiplications with z^2 have to take place. Neglecting the precomputation of αv and taking into account the squarings of z, we get on average $\frac{3}{2}\log_2 q + \frac{1}{4}t$ modular multiplications.
In addition to the computation of $\alpha^y v^e$, B has to check the certificate and possibly has to compute $h(\bar{x})$.
Since most of the computational complexity is put on the verifier's side, this protocol is well-suited for applications where only unilateral authentication of a smart card versus an application is required.

Communication
During the proof, ID, v, $C(ID, v)$, $h(x)$, e and y have to be transfered. Neglecting ID and $C(ID, v)$, this are at most (when $h(x) = x$) $2\log_2 p + \log_2 q + t$ bits.

Space
The prover needs a $\log_2 q$-bit ROM to store his secret key s, and a ROM to store ID, his certificate, and the $\log_2 p$-bit v. To store r during the proof, a RAM of $\log_2 q$ bits is needed.
The verifier needs a RAM of $\log_2 h(x) + t$ bits to store $h(x)$ and e during the proof, which is at most $\log_2 p + t$.

8.5.2 Schnorr signature scheme

The Schnorr identification scheme can be turned into the signature scheme described below. For that a (hash) function h is needed. This can be the same function as the function that is used to reduce the number of bits transmitted by hashing x to $h(x)$ in the identification scheme.

Protocol description

To sign a message M, A performs the following steps:

step 1 A picks at random $r \in \mathcal{Z}_q$ and computes $x = \alpha^r \bmod p$.

step 2 A computes $e = h(x, M) \in [0, 2^t - 1)$.

step 3 A computes $y = r + se \bmod q$.

To verify A's signature (e, y) on M, B computes $\bar{x} = \alpha^y v^e \bmod p$. Then B computes $h(\bar{x}, M)$, and accepts the signature if the result equals e.

If A generated the signature according to the protocol, B will always accept his signature since

$$\alpha^y v^e = \alpha^{r+se}(\alpha^{-s})^e = \alpha^r = x.$$

Complexity

The work for signature generation consist of one exponentiation to the power r and one multiplication (of s and e).
The message M is not used in the computation of α^r, thus this preprocessing step can be before the real protocol starts, and can be stored by A. The exponentiation costs $\frac{3}{2} \cdot \log_2 q$ modular multiplications.

The multiplication of the q-bit s, and the t-bit e is negligible in typical applications where $q \approx 140$ bits and $t \approx 72$.

Signature verification consists mainly of the computation of $\bar{x} = \alpha^y v^e \bmod p$, which can be done in $\frac{3}{2}\log_2 q + \frac{1}{4}t$ modular multiplications.

The length of the signature (e, y) is $t + \log_2 q$.

8.6 Girault-Pailles

M. Girault and J.-C. Pailles constructed an identity-based scheme providing zero-knowledge authentication and authenticated key exchange ([GP90]). We will describe of these two only the authentication scheme. As with the authentication schemes discussed in the previous sections, this scheme can easily be turned into a signature scheme, which will be described too.

The attributes of a user consist in all public-key schemes of his identity ID and a pair (s, P) of a secret key s and a public key P. The public keys need not be protected for confidentiality; on the contrary: they have to be as public as possible. But this publicity makes them vulnerable to integrity-attacks. Therefore the attributes of a user must also contain a guarantee that P is really the public key of the user with identity ID. Depending on the form of this guarantee, several types of schemes can be distinghuised. But all require the existence of a trusted center.

- In *certificate-based* schemes, the guarantee G consists of a digital signature on the pair (ID, P) computed and delivered by the center, the certificate $G = C(ID, P)$. In this case the four attributes ID, s, P and G are different. When someone needs to authenticate the user with identity ID, he gets the public triplet (ID, P, G) and checks G with the help of the center's public key. This is the approach of [I9594-8].
- *Identity-based* schemes are introduced by Shamir on Crypto '84 ([S85]). Here the public key is nothing but the identity (i.e. $P = ID$), and the guarantee is nothing but the secret key (i.e. $G = s$), so that only two attributes exist instead of four. The advantages of this approach are obvious: no certificate needs to be stored and checked. However, it also has its drawbacks. In particular, the center can impersonate any user at any moment, since it has calculated the secret keys.
- In the *intermediate* scheme of Girault and Pailles the guarantee is equal to the public key (i.e. $G = P$), so that there are three attributes: ID, s and P. The scheme is neither a certificate-based, nor an identity-based one, but its characteristics are closer to those of an identity-based one, which explains why the authors named it identy-based. It has the advantage of having no certificates of an identity-based scheme, and the advantage of an certificate-based scheme that each user can chose his own secret key, and that the center cannot interfer it from the public key.

8.6.1 Girault-Pailles authentication protocol

Protocol description

First a center chooses a modulus n, which is a product of two secret primes p and q. We must have $p = 2f\tilde{p} + 1$, and $q = 2f\tilde{q} + 1$, where f, $\tilde{p}$ and $\tilde{q}$ are distinct primes, $\tilde{p}, \tilde{q} >>> f$. The center also chooses two exponents e and d such that $ed = 1 \bmod (p-1)(q-1)$. Moreover, the center chooses an integer g of order f modulo both p and q, such that the multiplicative group generated by g in $\mathcal{Z}_n$ is very large. Note that the requirement that g has order f modulo p and q implies that g has also order f modulo n. Discussions on these choices follow in the security section, after the protocol description.
The integers n, f, e and g are public, whilst p, q and d are kept secret by the center.

Each user of the system chooses a large random number $s < f$ as his secret key, and computes $g^s \bmod n$ and gives it to the authority. After having verified the user's identity, the center generates an identification string ID and computes

$$V = ID^{-d} \bmod n.$$

Subsequently the center computes

$$P = Vg^{-s} \bmod n,$$

and he transfers both V and P to the user.

In summary, a user's memory contains:
- the universal integers n, f, and h;
- the user's credentials ID and P;
- the user's secret key s.

Now A can prove his identity to B by proving that he knows s. The protocol is as follows:

step 1 A chooses at random $x \in [1, f)$ and sends (part of) $t = h^x \bmod n$ to B, together with ID and P.

step 2 B chooses at random $c \in [0, e)$ and sends c to A.

step 3 A sends $y = x + sc \bmod f$ to B.

step 4 B compares the given bits of t with the corresponding bits of $h^y(P^e ID)^c \bmod n$, and accepts A's proof if they are equal.

Note that A can compute t in advance, which reduces the amount of computations he has to perform during the protocol considerably.

Figure 8.6 depicts this protocol.

Security

Before stating (and proving) lemmas on the soundness, completeness and being zero-knowledge of the scheme, we will shortly analyse it.

The set-up of the scheme seems to be that of a classic public-key cryptosystem, but the public key has a very particular feature: it is not only derived from the secret key s, but also from a secret exponent from the center (d). So no one can himself compute his public key. The same secret exponent is applied to the user's identity, so P and ID are connect by the relation R:

$$P^e ID = h^{-s} \bmod n.$$

Therefore anyone can compute $h^{-s} \bmod n$, but no one can compute s starting only from P and ID unless discrete logarithms modulo a composite number can be computed.

The crucial point is that P has not to be certified, since if an impostor substitutes $\tilde{P}$ to the public key of the user with identity ID, he also has to find a number $\tilde{s}$ such that R holds with $\tilde{P}$ and $\tilde{s}$:

$$\tilde{P}^{\tilde{s}} ID = h^{-\tilde{s}}.$$

Two strategies are possible. The impostor can first choose $\tilde{P}$, but does then need to solve $h^{-\tilde{s}} = \tilde{P}^e ID \bmod n$, which is the discrete logarithm problem. The second option is to choose $\tilde{s}$ first, but then $\tilde{P}^e = h^{-\tilde{s}} ID^{-1} \bmod n$ has to be solved for unknown $\tilde{P}$. This is the problem of inverting RSA, which in practice is as hard as factoring.

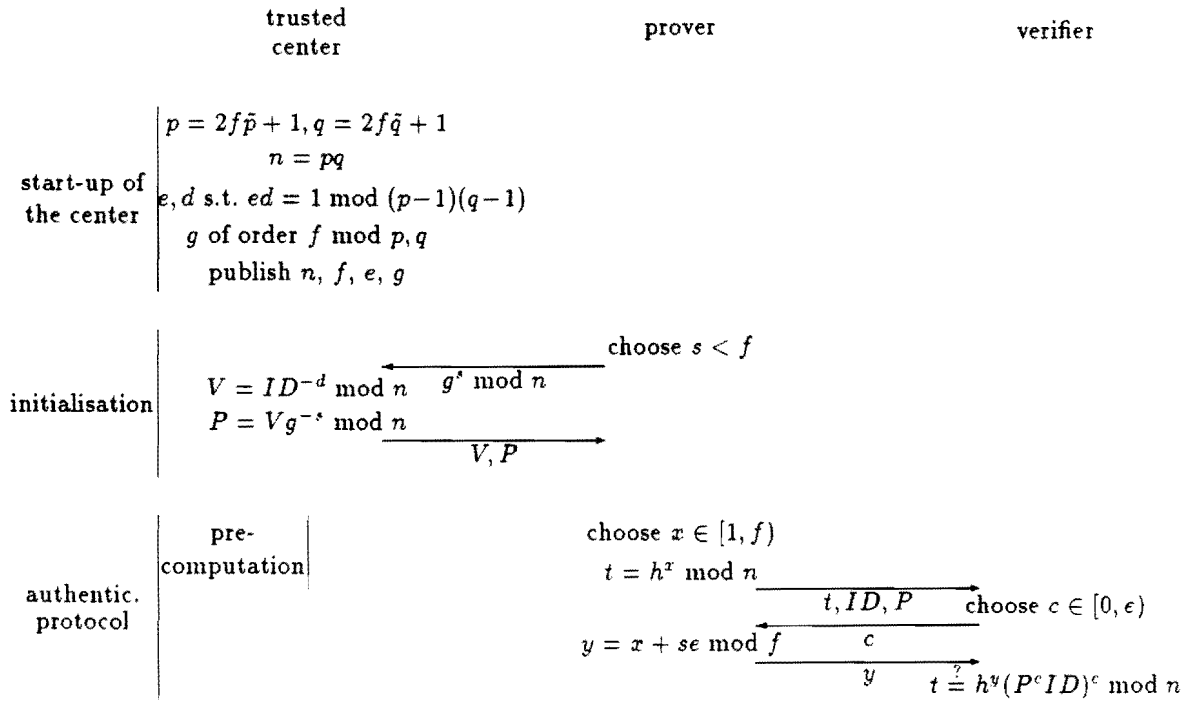


Figure 8.6: Girault-Pailles authentication protocol

Now we will consider the requirements on the modulus n:

- First we will give an algorithm that can factor n if it does not hold that $\tilde{p}, \tilde{q} >>> f$:

  step 1 Compute $\tilde{p} + \tilde{q} \bmod f$ starting from n and f. This can easily be done using that

  $$n = 4f^2\tilde{p}\tilde{q} + 2f(\tilde{p} + \tilde{q}) + 1,$$

  and thus

  $$\tilde{p} + \tilde{q} = \frac{n-1}{2f} - 2f\tilde{p}\tilde{q}.$$

  Denote $\tilde{p} + \tilde{q}$ by $\tilde{S}$.
  Since $(n-1)/(2f)$ can easily be computed and is an integer, this equation gives us $\tilde{S} \bmod f$. If f is large enough, $\tilde{S}$ can be found from $\tilde{S} \bmod f$ by exhaustive search.

  step 2 Compute $S := 2(f\tilde{S} + 1)$, which equals $p + q$ since

  $$p + q = 2f\tilde{p} + 1 + 2f\tilde{q} + 1 = 2(f(\tilde{p} + \tilde{q}) + 1) = 2(f\tilde{S} + 1).$$

  step 3 Compute p and q by using their product n and their sum S:

  $$p, q = \frac{S}{2} \pm \sqrt{\left(\frac{S}{2}\right)^2 - n}.$$

  This algorithm does not work if $\tilde{S}$ is much larger than f, which discourages an exhaustive search in step 1. This can be achieved by choosing $\tilde{p}$ and $\tilde{q}$ much larger than f (a more exact requirement can be found in the section "Complexity").

- As usual, $p-1$ and $q-1$ must have a large prime factor to thwart some well-known attacks. Since $p-1 = 2f\tilde{p}$, and $q-1 = 2f\tilde{q}$, this requirement is satisfied automatically by the above construction.

- It could occur that the revelation of g renders easy the factorisation of n. No proof is known that it does not, but for $u < f$:

$$gcd(g^u - 1 \bmod n, n) = 1,$$

and

$$gcd(g^f - 1 \bmod n, n) = n,$$

so no factor of n can be obtained by such greatest common divisor computations.

The authentication protocol itself is similar to Schnorr's one, except that n is composite, and its factors only are known to the trusted center. Hence, its security analysis is quite similar.

The lemma that achieves completeness of the scheme is as usual easy to verify:

Lemma 12 *If A and B follow the protocol, B always accepts the proof as valid.*

A fraudulent A can guess a challenge c and compute $(P^eID)^c$ $(= h^{-sc})$, choose a y at random, and use them to compute

$$t = h^u(P^eID)^c$$

to send to B in step 1. If B in step 2 indeed sent c, A replies with the y chosen. Since A computed t as to satisfy B's check, B will accept the proof. The probability that A guesses the correct c is $1/e$.
Similarly to the proof in [S90a], it can be proven that A has no better winning strategy. Thus, the scheme is also sound:

Lemma 13 *Assume that A does not know s. Then if B follows the protocol, he will accept the proof as valid with probability bounded by $1/e$*

Intuitively, B does not learn anything from s because the y he receives from A in step 3 is "scrambled" by the random number x, and computing x from t is not feasible unless discrete logarithms can be computed efficiently. By formalizing this, it can be proven that the protocol is zero-knowledge.

Lemma 14 *The protocol is zero-knowledge.*

Complexity

As usual, the complexity of the protocol is divided in three aspects:

Time

A has to compute t and y. The computation of t can be done in a precomputation, and costs about $\frac{3}{2}\log_2 f$ modular multiplications ([FP91]). The computation of y consists mainly of computing the product of the $\log_2 e$-bit number c and s. The public exponent e can have any value, so it is advantageous to choose it as small as possible, but still achieving an acceptable high security level $1/e$. Note that if e is chosen smaller, it suffices to repeat the protocol several times to achieve the required level of security. Summarising: A must perform $\frac{3}{2}\log_2 f$ modular multiplications in a precomputation step, and not even one modular multiplication during the protocol.

B has to perform one exponentiation modulo an f-bit number and one modulo an e-bit number, and has to multiply the results, which adds up to $\frac{3}{2}(\log_2 f + \log_2 e)$ modular multipications.

Communication
During the proof, ID, P, t, c and y have to be transferred. If ID is neglected, this are $2\log_2 n + \log_2 f + \log_2 e$ bits.

Space
The prover needs to store the $\log_2 f$-bit s in a secret ROM, and the P in a ROM which does not have to be secret. During the proof, a $\log_2 f$-bit RAM is needed.
The verifier needs a RAM of $\log_2 n + \log_2 e + \log_2 n$ bits to store t, c and $P^e ID$.

The authors of the article recommend a 750-bit n. If the exponent e is chosen about 20 to 30 bits, one protocol execution is enough to have a security level of 2^{-20} to 2^{-30} bits, while it is still small enable fast and easy computations in the authentication protocol. For signatures, 75 bits are recommended. One can either choose g to be small, which makes it easy to generate primes p and q such that g is primitive in GF(p) and GF(q), or choose $h = g^e$ to be small to facilitate the prover's and verifier's computations. In the last case, the center derives g from h by computing $g = h^d$. Finally, a 150-bit f (and thus also s) are sufficiently safe.

8.6.2 Girault-Pailles signature scheme

protocol description

When the authentication scheme is used to make a signature scheme, a hash function *hash* is needed to replace B's challenge, similarly to the previous protocol descriptions. Thus, *hash* hashes an arbitrary length input to a $\log_2 e$-bit output.

To sign a message M, the following steps are performed by A:

step 1 A chooses at random $x \in [0, f)$ and calculates $t = h^x \bmod n$.

step 2 A computes $c = hash(t, M)$.

step 3 A computes $y = x + sc \bmod f$, and sends ID, P, M, and the signature (c, y) on M to B.

B carries out the following protocol to verify the signature:

step 1 B calculates $\tilde{t} = h^y (P^e ID)^c \bmod n$.

step 2 B computes $\tilde{c} = hash(\tilde{t}, M)$.

step 3 B checks that $\tilde{c} = c$.

It is clear that Lemma 15 holds true:

Lemma 15 *If A and B follow the protocol, B always accepts the signature as valid.*

The length of the signature (c, y) is $\log_2 f + \log_2 e$.

8.7 Comparision

8.7.1 Overview

The table below shows the security and complexity of the schemes discussed in the previous sections. "SROM" is used as an abbreviation for "Secure ROM".

	security	# modular multiplications	communication (in bits)	space (in bits)	signature (in bits)
Fiat-Shamir	2^{-kt}	$t(k+2)/2$	$t(2\log_2 n + k)$	prover $k\log_2 n$ SROM $\log_2 n$ RAM verifier $(k+1)\log_2 n + k$ RAM	$k(\log_2 n + t)$
Guillou-Quisquater	$\frac{1}{c}$	$3\log_2 c + 1$	$2\log_2 n + \log_2 c$	prover $\log_2 n$ SROM $\log_2 n$ RAM verifier $\log_2 n + \log_2 c$ RAM	$\log_2 n + \log_2 c$
Ong-Schnorr	2^{-kt}	prover t (pre) $t(k+2)/2 - 1$ verifier $t(k+2)/2 + 1$	$(k+2)\log_2 n + kt$	prover $k\log_2 n$ SROM $k\log_2 n$ ROM $\log_2 n$ RAM verifier $(k+1)\log_2 n + kt$ RAM	$\log_2 n + kt$
OS small integer variant	2^{-kt} $(t \geq 4)$	prover t (pre) $\frac{5}{2}t + 1$ verifier $t(k+2)/2 + 1$	$(k+2)\log_2 n + kt$	prover $64k$ SROM $k\log_2 n$ ROM $\log_2 n$ RAM verifier $(k+1)\log_2 n + kt$ RAM	$\log_2 n + kt$
Schnorr	2^{-t}	prover $\frac{3}{2}\log_2 q$ verifier $\frac{3}{2}\log_2 q + \frac{1}{4}t$	$2\log_2 p + \log_2 q + t$	prover $\log_2 q$ SROM $\log_2 p$ ROM $\log_2 q$ RAM verifier $\log_2 p + t$ RAM	$\log_2 q + t$
Girault-Pailles	$\frac{1}{e}$	prover $\frac{3}{2}\log_2 f$ (pre) < 1 verifier $\frac{3}{2}(\log_2 f + \log_2 \epsilon)$	$2\log_2 n + \log_2 f$ $+ \log_2 \epsilon$	prover $\log_2 f$ SROM $\log_2 n$ ROM $\log_2 f$ RAM verifier $2\log_2 n + \log_2 e$ RAM	$\log_2 f + \log_2 \epsilon$

Remark:
In the table we left the communication of the prover's identity string ID and the certificate of his public key (in the Schnorr scheme) out.

8.7.2 Choices for the parameters

Usually a 512-bit n is considered to be sufficiently large to factor, but since factoring this one n would break the whole system for all users, a very good safety margin is desirable. Therefore we choose a 750-bit n in the Fiat-Shamir-like and the Girault-Pailles scheme. In the article of Schnorr, a size of 512 bits for the prime p, and a 140-bit prime q are proposed. But to enable a fair comparision of the schemes, we will enlarge the size of p to 750 bits, and the size of q proportionally to 200 bits. For the f in their scheme, Girault and Pailles recommend a 150-bit integer.
When we choose a security level of 2^{-20} for authentication, and substitute all these values in the table, we get the following overview of security and complexity for the schemes considered:

	security	# modular multiplications	communication (in bits)	space (in bits)	signature (in bits)
Fiat-Shamir	2^{-20} ($kt = 20$)	$10 + t$	$1500t + 20$	prover $750k$ SROM 750 RAM verifier $751k + 750$ RAM	$750k + 20$
Guillou-Quisquater	2^{-20}	61	1520	prover 750 SROM 750 RAM verifier 770 RAM	770
Ong-Schnorr	2^{-20} ($kt = 20$)	prover t (pre) $9 + t$ verifier $11 + t$	$750k + 1520$	prover $750k$ SROM $750k$ ROM 750 RAM verifier $750k + 770$ RAM	770
OS small integer variant	2^{-20} ($kt = 20$; $t \geq 4$)	prover t (pre) $\frac{5}{2}t + 1$ verifier $11 + t$	$750k + 1520$	prover $64k$ SROM $750k$ ROM 750 RAM verifier $750k + 770$ RAM	770
Schnorr	2^{-20}	prover 300 verifier 305	1720	prover 200 SROM 750 ROM 200 RAM verifier 770 RAM	220
Girault-Pailles	2^{-20}	prover 225 (pre) < 1 verifier 255	1670	prover 150 SROM 750 ROM 150 RAM verifier 1520 RAM	170

8.7.3 Concluding remarks

Zero-knowledge proofs promise to be quite useful for authentication protocols. Besides the schemes introduced here there are several other suggestions that deserve further study. Examples are techniques based on elliptic curves ([BC90]), permuted kernels ([S90b]), or error correcting codes ([S89]).

Appendix A

A.1 Rabin primality test

A composite number passes one round of the algorithm below with probability less than $1/4$, while a prime always passes. The Rabin primality test *rabintest* ([D92]) is the following:

step 1
Set n equal to the integer to be tested for primality, and write $n = 1 + 2^h \cdot a$, where a is odd.

step 2
Generate a random integer b, $1 < b < n$, called the base.

step 3
Set $j = 0$ and $z = b^a \bmod w$.

step 4
If $z = 1$ go to step 8.

step 5
If $z \bmod n = n - 1$, go to step 8.

step 6
Set $j = j + 1$; if $j < h$ set $z = z^2 \bmod n$ and go to step 5.

step 7
fail (n is not a prime).

step 8
pass (n is composite with probability less than $1/4$).

Note that t iterations of the algorithm do not necessarily imply that the probability that a composite number passes is less than $\frac{1}{4}^t$. The error probability depends on the distribution with which the candidate primes are chosen.

A.2 Probprime and probprimeinc

The algorithm *probprime* below ([D92]) generates a probable prime number p chosen at random from the interval I, such that $p-1$ is divisible by v and $gcd(p-1,e)=1$. Chosing $v=e=1$ results of course in a totally random prime in I.

We make the following assumptions:
- The Rabin primality test described above in A.1 is available, named *rabintest*, called with input n and producing output "pass" or "fail"
- A good random number generator *randomchoice* is available that is called with an interval I as input and returns a random odd number from I (see for example [MS91]).
- A table that contains all odd primes less than a fixed number r is available.
- A function *gcd* for computing the greatest common divisor of two integers is available.

step 1
If $I=[a\ldots b]$, define $\tilde{I}$ by $\tilde{I}:=[\frac{a}{2v}\ldots\frac{b}{2v}]$.

step 2
Compute n as $n=2v\cdot randomchoice(\tilde{I})\ +\ 1$.

step 3
If n is divisible by a prime less than r, or $gcd(n-1,e)\neq 1$ go to 2.

step 4
Set $i=0$.

step 5
Set $i=i+1$ and do *rabintest*(n); if *rabintest*(n) = fail go to 2.

step 6
If $i<t$ go to 5.

step 7
output n.

In [DL91] the probability that this procedure outputs a composite number is analysed in the cases where the interval is of the form $[2^{k-1}\ldots 2^k]$ for some k, and $e=v=1$, and in [D92] is a table showing some results. This probability appears to be (much) less than $\frac{1}{4}^t$. For example, for a 300-bit n, 10 iterations give already an error probability of at most $\frac{1}{4}^{42}$.

An alternative to *probprime* is *probprimeinc* ([D92]), generating a probable prime number p chosen at random from the interval I by incremental search, such that $p-1$ is divisible by v and $gcd(p-1,e)=1$.
Probprimeinc chooses at random an odd starting point n_0 such that n_0-1 is divisible by v and examines $n_0, n_0+2v,\ldots$
This algorithm is more economical in its use of random bits, and testdivision by small primes can be done much more efficiently: first compute the residue of n_0 modulo each small prime in the table. Each time the current candidate is increased by $2v$, $2v$ is added to the residue modulo each of the small primes and it is tested that none of the residues becomes zero. In [BDL91] is shown that the optimal r equals

$$r=\frac{R}{D\cdot\log(R/D)},$$

where R is the time needed to do one Rabin test, and D the time needed to divide a candidate prime number by a prime less than r.

The algorithm is as follows:

step 1
If $I = [a \ldots b]$, define $\tilde{I}$ by $\tilde{I} := [\frac{a}{2v} \ldots \frac{b}{2v}]$.

step 2
Compute n as $n = 2v \cdot randomchoice(\tilde{I}) + 1$, and initialize testdivision.

step 3
Set $n = n + 2v$, and if now n is not in I, go to 2.

step 3
If n is divisible by a prime less than r (use optimized testdivision), or $gcd(n-1, e) \neq 1$, go to 2.

step 4
Set $i = 0$.

step 5
Set $i = i + 1$ and do $rabintest(n)$; if $rabintest(n) =$ fail, go to 3.

step 6
If $i < t$, go to 5.

step 7
output n.

If one accepts an upper limit on the number of candidates to be examined, the probability that the output is a composite number can be estimated in the case $v = e = 1$. This error probability is dependent on the number of candidates. In [D92] a table can be found.

A.3 Provprime

We will give a recursive algorithm for generating provable primes proposed by Maurer ([D92], [M90]). It is based on the following number theoretic result by Pocklington:
If
- $n-1 = FR$ with $q_1, \ldots, q_t$ the distinct prime factors of F,
- there exists a number a such that $a^{n-1} = 1 \bmod n$, and for all i, $1 \leq i \leq t$, $gcd(a^{\frac{n-1}{q_i}} - 1, n) = 1$, and
- $F > \sqrt{n}$,
then n is a prime.

This suggests the following algorithm *provprime* for generating a random prime in some interval $[low \ldots high]$:

step 1
Generate recursively $q_1, q_2, \ldots$ where $q_1 \geq q_2 \geq \ldots$ until their product F is larger than $\sqrt{high}$. Set t equal to the number of q's that are generated.

step 2
Choose at random an even number R and compute n as $n = FR + 1$.

step 3
Choose at random a number a such that $a^{n-1} = 1 \bmod n$.

step 4
Set $i = 1$.

step 5
Compute $gcd(a^{\frac{n-1}{q_i}} - 1, n)$.
If the outcome is not equal to 1, go to 2.

step 6
If $i < t$ set $i = i + 1$ and go to 5.

step 7
output n.

Maurer shows that if the q's are large, nearly any choice of a will suffice for proving primality of n if n is prime.

The algorithm can be speeded up in various ways:

- After step 2., test division on small primes should be used. Since all candidates are of the form $n = FR + 1$ for fixed F, one can translate the condition that none of the small primes divides n into a condition on R. Concretely, if $n = FR + 1 = 0 \bmod p$, then $R = -F^{-1} \bmod p$. So we can precompute $-F^{-1}$ modulo each small prime used for test division and check for every candidate for each p if $R = -F^{-1} \bmod p$ ([M90]).
- When a candidate has passed test division, a Rabin test with base 2 (see Annex A.1) should be done. This base gives the most efficient Rabin test possible, and it excludes virtually all composites. Furthermore, if n passes the test it is implicitely checked that $2^{n-1} \equiv 1 \bmod n$. It is therefore advantageous to choose $a = 2$ in step 3 ([D92]).
- Finally, the improvement in [BLS75] of Pocklingtons result can be used. Here the following theorem is proved:

Given $n = FR + 1$, suppose we have an a satisfying Pocklingtons conditions. Let $\tilde{R}$ be the odd part of R, and $\tilde{F}$ be $\frac{n-1}{\tilde{R}}$. Let r and s be defined by $\tilde{R} = 2 \cdot \tilde{F} \cdot s + r$, where $1 \leq r < 2 \cdot \tilde{F}$. Suppose $\tilde{F} > \sqrt[3]{n}$. Then n is prime if and only if $s = 0$ or $r^2 - 8s$ is not a square.

This refined condition is somewhat more computationally costly to verify, but this makes little difference in practice if the Rabin test is used before. Experience shows that the above result is only used for the final candidate, and the extra computations to find $\tilde{R}$, $\tilde{F}$, r and s, and perhaps a square root computation take negligible time compared to the exponentations. Furthermore, only 5% of the integers x have all primes less than $\sqrt[3]{x}$, and this will be detected after generation of the first prime. Although it biases the distribution of the primes generated slightly, this small percentage can safely be neglected. This simplifies the code and saves time compared to the original version for the circa 30% of the integers with largest prime factor less than $\sqrt{x}$.

A.4 Strongprime

The algorithm *strongprime* below ([D92]) generates a prime p chosen at random from the interval I based on a seed s, such that it can be used in RSA with public exponent v satisfying the last two constraints.

To define some notation, let r, s and t be primes that divide $p-1$, $p+1$ and $r-1$ respectively. *Strongprime* uses the procedure *probprime* (or *probprimeinc* which makes the scheme more efficient) of Annex A.2. The procedure *randomchoice* that is used in these two procedures chooses primes based in a scheme initialized by the procedure *initrand* on a random seed s passed as a parameter. It is possible to use provable primes which can be produced by the algorithm of Annex A.3 instead of probable primes produced by *probprime* or *probprimeinc*.
Furthermore, fixed parameters c_1 and c_2, based on the interval I, have to be chosen to control the size of $p-1$, $p+1$ and $r-1$. If $I=[a\ldots b]$, a possible choice for them such that r, s, and t are the maximal allowed size where there is still a good change of finding a prime in I with the right properties is:

$$c_1 = \frac{1}{2 \cdot bitlength(a)}, \text{ and } c_2 = \frac{1}{2 \cdot \sqrt{bitlength(a)}}.$$

At the other extreme, the constant 2 in both formulas can be replaced by a larger number such that r, s and t are of the minimal required size.
The algorithm *strongprime* is the following:

step 1
If $I=[a\ldots b]$, define I_1 by $I_1 := [c_1\sqrt{a}\ldots c_1\sqrt{b}]$, and I_2 by $I_2 := [c_2\sqrt{a}\ldots c_2\sqrt{b}]$.

step 2
Compute t as $t = probprime(I_1, 1, 1)$.

step 3
Compute s as $s = probprime(I_2, 1, 1)$.

step 4
Compute r as $r = probprime(I_1, t, 1)$.

step 5
Compute p_0 as $p_0 = s^{r-1} - r^{s-1} \bmod rs$.

step 6
If p_0 is even, then set $p_0 = p_0 + rs$.

step 7
output $probprime(I, p_0, v)$.

A.5 Computations of discrete logarithms and factoring of a "hard integer"

The table below gives information on the computation time needed for computing discrete logarithms over GF(p) and GF(2^n), and the factorization of an integer that is the product of two primes of about the sime size.

	discr. log over GF(p)	discr. log over GF(2^n)		integer factorization
asymptotics	$\exp(\sqrt{\log(p)})$	$\exp(\sqrt[3]{n})$		$\exp(\sqrt{\ln(n)\ln(\ln(n))})$
largest comp.	p 224 bits	$n = 503$		n 365 bits (400 MIPS years)
sugg. param.	p 512 bits	$n = 993$	$n = 1186$	$n = 512$
projected running time	5,000,000 MIPS years	4000 MIPS years	100,000,000 MIPS years	500,000 MIPS years

Note that 1 MIPS year is the amount of computation performed in a year by a 1 Million Instructions Per Second machine ($\approx 3 \cdot 10^{13}$ instructions).

Bibliography

[I7498-2]
ISO 7498-2: 1989
OSI Information processing systems - Open systems Interconnection - Basic Reference Model - Part 2: Security architecture.

[I7816-1]
ISO 7816-1: 1987
Identification cards - Integrated circuit(s) cards with contacts - Part 1: Physical characteristics.

[I7816-2]
ISO 7816-2: 1988
Identification cards - Integrated circuit(s) cards with contacts - Part 2: Dimensions and location of the contacts.

[I7816-3]
ISO/IEC 7816-3: 1989
Identification cards - Integrated circuit(s) cards with contacts - Part 3: Electronic signals and transmission protocols.

[I9564-1]
ISO 9564-1: 1990
Banking - Personal Identification Number Management and Security - Part 1: PIN Protection Principles and Techniques.

[I9594-8]
ISO 9594-8: 1990
Information processing systems - Open Systems Interconnection - The Directory.

[I9735]
ISO 9735: 1988
Electronic data interchange for administration, commerce and transport (EDIFACT) - Application level syntax rules (amended and reprinted 1990).

[II9796]
ISO/IEC 9796: 1991
Information technology - Security techniques - Digital signature scheme giving message recovery.

[II9798-1]
ISO/IEC 9798-1: 1991
Information technology - Security techniques - Entity authentication mechanisms - Part 1: General Model.

[II9798-2]
ISO/IEC 9798-2: CD 1992
Information technology - Security techniques - Entity authentication mechanisms - Part 2: Entity authentication using symmetric techniques.

[II9798-3]
ISO/IEC 9798-3: WD 1993
Information technology - Security techniques - Entity authentication mechanisms - Part 3: Entity authentication using a public key algorithm.

[II10118-1]
ISO/IEC 10118-1: DIS 1992
Information technology - Security techniques - Hash functions - Part 1: General.

[II10118-2]
ISO/IEC 10118-2: DIS 1992
Information technology - Security techniques - Hash functions - Part 2: Hash functions using an n-bit block cipher algorithm.

[I11166-1]
ISO 11166-1: CD 1991
Banking - Key Management by means of asymmetric algorithms - Part 1: Principles, Procedures and Formats.

[II11770-3]
ISO/IEC 11770-3: WD 1992
Key Management - Part 3: Key Management Mechanisms Using Asymmetric Cryptographic Techniques.

[A92]
J.C. Anderson, *Responses to NIST's proposal*, Communications of the ACM, Vol.35, No.7, pp.41-52, 1992.

[AMV89]
G.B. Agnew, R.C. Mullin, S.A. Vanstone, *Digital Signatures for the CA34C168 Data Encryption Processor*, Document 834168.MD603.02, University of Waterloo, Waterloo, Ontario, Canada, Calmos CA34C168 Application Notes, August 1989.

[ANSI81]
Data Encryption Algorithm, X3.92, American National Standards Institute, 1981.

[BB78]
B. Blakley and G.R. Blakley, *Security of Number Theoretic Public Key Cryptosystems Agains Random Attack*, Cryptologica. In three parts: Part I: Vol.2(4), pp.305-321, oktober 1978; Part II: Vol.3(1), pp.29-42, January 1979; Part III: Vol.3(2), pp.105-118, April 1979.

[BB79]
G.R. Blakley and I. Borosh, *Rivest-Shamir-Adleman Public Key Cryptosystems Do Not Always Conceal Messages*, Comp. & Math. with Appl., Vol.5, pp.169-178, 1979.

[BC90]
A. Bender and G. Castagnoli, *On the Implementation of Elliptic Curve Cryptosystems*, Proceedings of Crypto '89, Lecture Notes in Computer Science Vol.435, Springer-Verlag, pp.186-192, 1990.

[BDL91]
J. Brandt, I. Damgaard and P. Landrock, *Speeding up Prime Number Generation*, Proceedings of Asiacrypt '91, Springer verlag, 1991.

[BGM91]
Barents, Gasille and Mout, *Trusted third parties and similar services*, TEDIS Programme, report for the Commission of the European Communities DG XIII, October 1991.

[BLS75]
J. Brillhart, D.H. Lehmer and J.L. Selfridge, *New Primality Criteria and Factorizations of* $2^m \pm 1$, Math. Comp. 29, pp.620-647, 1975.

[BS89]
Bach and Shallit, *Factoring with Cyclotromic polynomials*, Math. Comp. 52, pp.201-219, 1989.

[CEC89]
Commission of the European Communities, *The legal position of the member states with respect to electronic data interchange*, TEDIS report, 1989.

[CCITT88]
CCITT, *Data Communication Networks Directory*, Recommendations X.500 - X.521, 1988.

[D83]
D.E. Denning, *Cryptography and Data Security*, Addison-Wesley Publ., 1983.

[DL91]
I. Damgaard and P. Landrock, *Improved bounds for the Rabin primality test*, manuscript, submitted to Math. Comp., 1991

[D92]
I. Damgaard, *RSA Key Generation*, Danish input for TR on generation of primes and modulus, SC27/W62 N133, March 23, 1992.

[DH76]
W. Diffie and M.E. Hellman, *New Directions in Cryptography*, IEEE Trans. on Information Theory, Vol. IT-22 (6), pp.644-654, 1976.

[DP89]
D.W. Davies and W.L. Price, *Security for Computer Networks*, John Wiley & sons, 1989.

[DS91a]
M. De Soete, *Public Key Algorithms*, Proceedings of the ESAT course "State of the Art and Evolution of Computer Security and Industrial Cryptography", KU Leuven, Springer Verlag, to appear.

[DS91b]
M. De Soete, *Smart Cards and their Applications*, Proceedings of Compsec 91, Elsevier Science Publishers Ltd., pp.147-154, 1991.

[DS93]
M. De Soete, *The Key To Open EDI: Digital Signature*, presented at EEMA conference, January 1993.

[EG85]
T. El Gamal, *A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms*, IEEE Trans. on Information Theory, Vol.31-4, pp.469-472, 1985.

[FFS88]
U. Feige, A. Fiat and A. Shamir, *Zero-Knowledge Proofs of Identity*, Journal of Cryptology 1, pp.77-94, 1988.

[FP91]
W. Fumy and A. Pfau, *On the Complexity of Asymmetric Smart Card Authentication*, Smart Card 2000, ed. D. Chaum, Elsevier science publishers B.V., Amsterdam, The Netherlands, pp.181-190, 1991.

[FS87]
A. Fiat and A. Shamir, *How to prove yourself: practical solutions to identification and signature schemes*, Advances in Cryptology: Proceedings of Crypto '86, Lecture notes in Computer Science Vol.263, Springer-Verlag, pp.186-194, 1987.

[GGM84]
O. Goldreich, S. Goldwasser and S. Micali, *How to construct random functions*, 25th symposium on Foundations of Computer Science, October 1984.

[GMR85]
O. Goldreich, S. Micali and C. Rackoff, *The knowledge complexity of interactive proof systems*, Proceedings of STOC '85, pp.291-304, 1985.

[GP90]
M. Girault and J.-C. Pailles, *An identity-based scheme providing zero-knowledge authentication and authenticated key exchange*, working paper ISO/IEC JTC1, 1990.

[GQ88]
L.C. Guillou and J.J. Quisquater, *A practical zero-knowledge protocol fitted to security microprocessors minimizing both transmission and memory*, Proceedings of Eurocrypt '88, Lecture Notes in Computer Science Vol.330, Spinger-Verlag, pp.123-128, 1988.

[ICC88]
International Chambers of Commerce, *Uniform rules of conduct for interchange of trade data by tele-transmission*, Paris, 1988.

[K87]
N. Koblitz, *A Course in Number Theory and Cryptography*, Graduate Text in Mathematics 114, Springer-Verlag New York Inc., 1987.

[L91]
X. Lai, *Detailed Description and a Software Implementation of the IPES Cipher*, Signal and Information Processing Laboratory, ETH Zurich, November 8, 1991.

[LLMP90]
A.K. Lenstra, H.W. Lenstra Jr., M.S. Manasse and J.M. Pollard, *The number field sieve*, Proc. of STOC 90, pp.564-572, 1990.

[LM91a]
A.K. Lenstra and M.S. Manasse, *Factoring with two large primes*, Advances in Cryptology, Proceedings of Eurocrypt '90, Springer-Verlag, pp.72-82, Berlin 1991.

[LM91b]
X. Lai and J.L. Massey, *A Proposal for a New Block Encryption Standard*, Advances in Cryptology, Proceedings of Eurocrypt '90, Springer-Verlag, pp.189-404, Berlin 1991.

[LMO91]
B.A. LaMacchia and A.M. Odlyzko, *Computation of discrete logarithms in prime fields*, Advances in Cryptology, Proceedings of Eurocrypt '90, Springer-Verlag, pp.616-618, 1991.

[M90]
U.M. Maurer, *Fast generation of RSA products with almost maximal diversity*, Advances in Cryptology - Eurocrypt '89, Lecture Notes in Computer Science 434, Springer Verlag, pp.636-647, 1990.

[MPW92]
C.J. Mitchell, F. Piper and P. Wild, *Digital Signatures*, Contemporary Cryptology, The Science of Information Integrity, edited by G.J. Simmons. IEEE Press, New York, pp.325-378, 1992.

[MRW89]
C. Mitchell, D. Rush and M. Walker, *A Remark on Hash Functions for Message Authentication*, Computers & Security Vol.8, pp.55-58, 1989.

[MS90]
S. Micali and A. Shamir, *An improvement of the Fiat-Shamir Identification and Signature Scheme*, Proceedings of Crypto '88, Lecture Notes in Computer Science Vol.403, Springer-Verlag, pp.244-247, 1990.

[MS91]
S. Micali and C.P. Schnorr, *Efficient, Perfect Polynomial Random Number Generators*, Journal of Cryptology Vol.3, pp.157-172, 1991.

[N92]
J. Nechvatal, *Public Key Cryptography*, Contemporary Cryptology, The Science of Information Integrity, edited by G.J. Simmons. IEEE Press, New York, pp.177-288, 1992.

[NBS77]
Data Encryption Standard, Federal Information Processing Standards Publication 46, National Bureau of Standards, US Department of Commerce, January 1977.

[NIST91]
A proposed Federal Information Processing Standard for Digital Signature Standard (DSS), National Institute for Standardisation and Technology, August 1991.

[NIST92a]
A proposed Federal Information Processing Standard for Secure Hash Standard (SHS), National Institute for Standardisation and Technology, January 31, 1992.

[NIST92b]
American National Standard X9.30-199X, *Public key cryptography using irreversible algorithms for the financial services industry part 1: The Digital Signature Algorithm (DSA)*, American Bankers Association, USA, februari 18, 1992.

[OO90]
K. Ohta and K. Okamote, *A Modification of the Fiat-Shamir Scheme*, Proceedings of Crypto '88, Lecture Notes in Computer Science Vol.403, Springer-Verlag, pp.232-243, 1990.

[OS91]
H. Ong and C.P. Schnorr, *Fast Signature Generation with a Fiat Shamir - Like Scheme*, Proceedings of Eurocrypt '91, Lecture Notes in Computer Science Vol.473, Springer-Verlag, pp.432-440, 1991.

[P85]
C. Pomerance, *The quadratic sieve factoring algorithm*, Advances in Cryptology, Proceedings of Eurocrypt '84, Lecture Notes in Computer Science Vol.209, Springer-Verlag, pp.169-182, 1985.

[R78]
R. Rivest, *Remarks on a Proposed Cryptanalytic Attack of the M.I.T. Public Key Cryptosystem*, Cryptologica, Vol.2(1), pp.62-65, January 1978.

[R91]
R.L. Rivest, *The MD4 Message Digest Algorithm*, Proceedings of Crypto '90. Lecture Notes in Computes Science Vol.537, Springer-Verlag, pp.303-311, 1991.

[R92]
A. Rossnagel, *Digitale Unterschriften und Verfassungsverträglichkeit*, Kommunikation & Sicherheit, edt. H. Reimer and B. Struif, TeleTrusT Deutschland e.V., Bad Vilbel - Darmstadt 1992.

[RD91]
R. Rivest and S. Dusse, *The MD5 Message-Digest Algorithm*, Network Working Group, Internet draft [MD5-A], July 10, 1991.

[RSA78]
R. Rivest, A. Shamir and L. Adleman, *A method for obtaining digital signatures and public key cryptosystems*, Comm. of the ACM, Vol.21, pp.120-128, 1978.

[S85]
A. Shamir, *Identity-based cryptosystems and signature schemes*, Proceedings of Crypto '84, Lecture Notes in Computer Science Vol.196, Springer-Verlag, pp.47-53, 1985.

[S87]
R.D. Silverman, *The multiple polynomial quadratic sieve*, Math. Comp., V.48, pp.329-339, 1987.

[S89]
J. Stern, *An Alternative to the Fiat-Shamir protocol*, presented on "Kryptographie", Oberwolfach, 1989.

[S90a]
C.P. Schnorr, *Efficient Identification and Signatures for Smart Cards*, Proceedings of Crypto '89, Lecture Notes in Computer Science Vol.435, Springer-Verlag, pp.239-252, 1990.

[S90b]
A. Shamir, *An Efficient Authentication Scheme Based on Permuted Kernels*, Proceedings of Crypto '89, Lecture Notes in Computer Science Vol.435, Springer-Verlag, pp.606-609, 1990.

[S92]
Ernest F. Brickell and Chris Holloway, Proceedings of *Securicom 92*, an exceptional tutorial day on: "Electronic signature algorithms and protocols", CNIT Paris - La Defense - France, March 17, 1992.

[SN77]
G.J. Simmons and J.N. Norris, *Preliminary Comments on the M.I.T. Public Key Cryptosystem*, Cryptologica, Vol.1(4), pp.406-414, October 1977.

[SP89]
J. Seberry and J. Pieprzyk, *Cryptography, An Introduction to Computer Security*, Advances in Computer Science Series, Pretence Hall, Australia, 1989.

[T91]
Das TeleTrusT-Zertifikat, Version 1, Working version of TeleTrusT AG4, edited by W. Schneider, GMD, June 4, 1991.

[T92a]
TeleTrusT Object identifiers Register, Draft version 1 of TeleTrust AG4, edited by Klaus Truöl, GMD, Januari 24, 1992.

[V92]
K. Vedder, *Smart Cards*, Proceedings of the Sixth Annual European Computer Conference on Computer Systems and Software Engineering, The Netherlands, pp.630-635, May 4-8, 1992.

[vH92]
E. van Heijst, *Special Signature Schemes*, doctoral thesis, University of Technology Eindhoven, July 6, 1992.

[vO91]
P.C. van Oorschot, *A comparision of public-key cryptosystems based on integer factorization and discrete logarithms*, Proceedings of Crypto '90, Lecture Notes in Computes Science Vol.537, Springer-Verlag, pp.576-581, 1991.

[vO92]
P.C. van Oorschot, *A Comparision of Practical Public Key Cryptosystems Based on Integer Factorization and Discrete Logarithms*, Contemporary Cryptology, The Science of Information Integrity, edited by G.J. Simmons. IEEE Press, New York, pp.289-322, 1992.