PRIVATE COMPUTING
THE TRUSTED DIGITAL ASSISTANT

PROEFSCHRIFT

ter verkrijging van
de graad van doctor aan de Universiteit Twente,
op gezag van de rector magnificus,
prof. dr. F.A. van Vught,
volgens besluit van het College voor Promoties
in het openbaar te verdedigen
op vrijdag 28 juni 2002 te 13.15 uur.

door
Tage Stabell-Kulø
geboren op 24 januari 1963
te Kristiansund, Noorwegen

Het proefschrift is goedgekeurd door
prof. dr. S.J. Mullender, promotor.

Private Computing

The Trusted Digital Assistant

Tage Stabell-Kulø

Ph. D. Dissertation

University of Twente

This dissertation was typeset with LaTeX in Charter and Euler, with symbols from $\mathcal{AMS}$. The simple figures were drawn, then scanned at 1200 dpi, and converted to JPG with GIMP. The complicated once were programmed in METAPOST. The cover was drawn by Marianne Kulø. The dissertation was in its entirety written and typeset using NetBSD.

ISBN 90-365-1762-1

PrintPartners Ipskamp B.V.
Postbus 333
7500 AH Enschede
Nederland

Preface

Over the last 10 years, I have continously been working on projects related to private computing. The projects to which I have been affiliated have varied in focus, in size and in scope. This dissertation reports from the various projects, and thus represents a decade of research.

Over the years, I have had the pleasure of working with quite a few people. Fortunately, not all of them have let me be solely in control of our joint projects. In fact, some have put up quite some resistance when I have tried to lead the project in the direction I believed was best. Today, I believe this stubborn resistance benefited not only the projects *per se*, but probably also, in the end, this dissertation. The price to pay for cooperation is that I am not the sole author of every single publication this dissertaion is based on. However, I am the first author on all but one. On that last one, the authours are in alphabetic order. Thus, my claim is that this dissertation reports from my contributions.

My ambition has been to present only work that has been validated by peer review. To that end, at the very beginning of each chapter there is a section identifying the underlying publication(s). All chapters, except those in the introduction (Chapters 1 – 3) and Chapter 7, are based on articles that has been published. A few of them has simply been printed, but most have been published.

Because I report from a decade of continous work, some of the results presented here are quite dated. However, at the time they were published, they were not.

Acknowledgements

Beste Sape,

Op een paar weken na is het nu tien jaar geleden dat je mij uitnodigde om te beginnen aan mijn promotie. Nu, na een decenium met zijsporen, twijfel, verontschuldigingen, en onzekerheid moet ik met verbazing constateren dat je gelijk hebt gekregen. Of, met andere woorden: Je hebt meer vertrouwen in mij gehad dan ikzelf, en dat verbaasd me.

Ik hoop dat het geduld dat jij door de jaren heen met mij hebt gehad, mij zal helpen mijn geduld te bewaren als ik zit te wachten tot mijn studenten terugkomen van hun zijsporen. Ook als de wachttijd oploopt tot een jaar of tien.

Laat ons vaker dineren samen!

Molto tempo fa, a Enschede, feci il mio primo incontro con gli Italiani. Mentre parlavamo della vita, dell'universo e di tutto il resto nella "Hall C", intravidi uno stile di vita diverso dal mio. Mi chiesi come poteva essere vivere in Italia, quando potevo descrivere l'Olanda come un "eldorado burocratico". Iniziai a capire durante una cena a San Giuliano Terme: ricordo vividamente che avevamo dato appuntamento alla miglior tagliata e, come tocco finale, alla grappa più pregiata. Le fondamenta di questa dissertazione furono gettate mentre stavo mangiando pasta a "The Monsters". Le fondamenta del mio futuro furono gettate mentre stavo mangiando pasta a "La Casa dei Norvegesi". Ne sono grato.

Det er mange som mener de har betalt en høy pris for at denne avhandlingen skulle bli ferdig, nesten ti år efter at den ble påbegynt. Av dem som med rette kan fremme krav om tilbakebetaling er naturligvis alle de andre deltagerne i de prosjektene jeg har fått være med i; det er beklageligvis for mange til å ramse opp alle. Blant dem som nok ønsker betaling er mange ansatte og studenter ved Institutt for Informatikk som gjennom de siste 17 årene i praksis, mer eller mindre frivillig, har lagt forholdene til rette (og betalt prisen) for mine sprell.

Fremst i rekken av kreditorer står helt åpenbart dem jeg har arbeidet sammen med: Arne [66, 67, 68, 126, 69], Feico [124, 39], Gianluca [126], Gunnar [45], Per Harald [96, 123, 95], Ronny [123] og Terje [45, 46, 125, 124]; navnene er sortert alfabetisk, men de burde muligens vært sortert efter antall referanser, antall krangler, antall grappa, antall utkjørte kilometer, eller noe. Uansett, med seg har de studenter og ansatte som har vært involvert i PASTA på en aller annen måte. I tillegg kommer Anders som har unngått å skrive sammen med meg så langt, men som nok ikke slipper unna; han har altså bare betalt og ikke fått noe igjen.

Uansett hva jeg måtte mene om min egen fortreffelighet, så er det i bunn og grunn Hustruen som har gjort denne avhandlingen både mulig og nødvendig; hun har dermed fått som fortjent. Da stiller det seg anderledes med Marine, som gjennom mesteparten av oppveksten har hatt avhandlingen svevende over seg som en syvende far i huset. Jeg håper det ikke skal skje igjen.

Den som utviste mest tålmodighet men samtidig var mest utålmodig var likevel Farmor; jeg tror du hadde likt å se resultatet. Avhandlingen dedikeres til deg.

Financial support has been received from The University of Twente, The Netherlands, The University of Tromsø, Norway, EU through the BROADCAST and PEGASUS projects, and the Research Council of Norway through the GDD, ARCTIC BEANS and PESTO projects.

Contents

Chapter 1

Introduction

"Private information is practically the source of every large, modern fortune."

Oscar Wilde, "An ideal husband", act two.

Computers are becoming omnipresent and ever more information is processed electronically; our times have long been described as one with "ubiquitous computing" [142]. Computers, small and "invisible" as well as large and powerful will be at our disposal. One effect will be that information stemming from all parts of life, from private as well as professional activities, is processed, stored, updated, correlated and used for a variety of purposes. Some of these might very well be undesirable (from a user's point of view, that is). For example: a provider of mobile telephone services knows the distance between its base stations, and the time customers use to travel between them. Based on this information, the provider can deduce how fast customers drive. This information might be beneficial to insurance companies, which could adjust premiums according to their customers' driving style. Or, as has become customary in some countries, providers are subpoenaed to provide information on the whereabouts of the mobile phones belonging to customers. The opposite is also taking place: In a high-profile criminal case in Norway[1], one of the accused claimed to have an alibi because

[1] In the so called "Orderud case", a married couple and their adult daughter were murdered. Their son, his wife, her half sister, and the half sister's boyfriend were later convicted of assisting in the murder; at the time of writing an appeal is pending.

data from the provider showed that her phone had not been at the scene of the crime. There are countless other possibilities of such (mis)use of information. As long as there is profit to be gained, such opportunities must be expected to be exploited. Those who are worried about intrusion into their private sphere must acquire the expertise necessary to protect themselves; switching off the mobile phone while speeding is an example. Refraining from speculating on the likelihood of a merger between an insurance company and a provider of mobile telephony, we assert that there are many such constellations where providers of different services might blend their services to gain profit in new or existing markets. A question related to system design arises from this: How should systems be constructed to enable users to muster enough information about the system to fend of attacks on their privacy? Part of such an investigation is related to user interfaces; how can and should information about the system be presented to the user? We believe users wants to participate in the decision making, being in the decision loop as it were.

In addition to a vast array of entirely new services and products made possible by private computers and networking in concert, some old concepts will be transformed to a new form. For example, it is likely that in the future a large percentage of private communication will be electronic, and the traditional envelope will be replaced by the *cryptolope*.[2] A cryptolope is a cryptographic envelope, as it were, designed to provide a countermeasure to the ease with which (electronic) data can be gathered, searched and analyzed. The cryptolope is harder to open than the envelope but this is a result of the hundreds of years that separate the two inventions. While contemporary envelopes provide (some) privacy by their sheer number, cryptolopes achieves the same goal by ensuring that it requires a substantial effort to gain access to the content. Cryptolopes will probably become as common as envelopes, but an important problem must be solved before they can be put to their full potential: How, where, and by whom (what) should cryptolopes be "sealed"? Obviously, the entity that seals a cryptolope must be trusted not to "peek inside", or, if it has to, not to leak information about the contents. Restraining a system component from "peeking" necessitates examination of the design of the system, as well as paying close atten-

[2]The word "cryptolope" is a trademark owned by IBM, but nevertheless used here to convey the amalgam of new and old technology. The IBM Cryptolope Technology is an ingredient in IBM e-Business and Content Management Solutions.

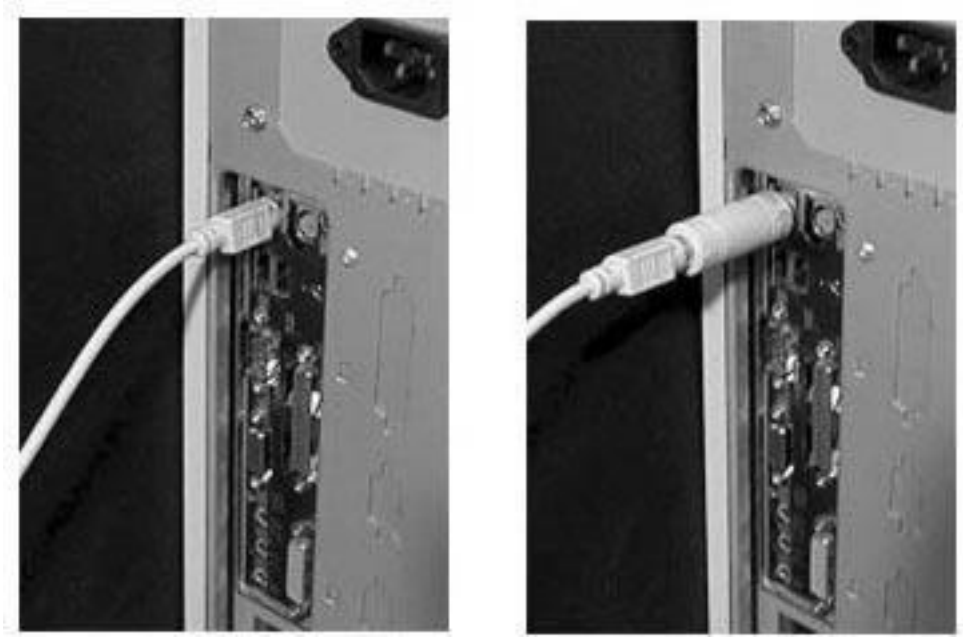


Figure 1.1: Non-intrusive device to capture all keystrokes.

tion to implementation issues. Doing so seems to be an aspect of the question posed above: How should systems be constructed, in order to make it possible for users to protect themselves? Essentially the same question can be posed by asking if there are components that should not be used in a system aspiring to offer security in some form.

As an example, consider a device known as "Key Katcher". The manufacturer[3] describes the product as follows:

> This is a device that can be connected to a keyboard to record all keystrokes. It has a changeable password, keyword search, enable/disable option, and stores URLs. Records more than 65,000 keystrokes and does not require any software. Monitor unauthorized access to your computer or your network. Use it to troubleshoot or make fixes by tracing back through a users command sequence.
>
> Key Katcher plugs in between your keyboard and your computer. A micro-controller interprets the data, and stores in-

[3] Allen Concepts, Inc., Chandler, AZ, USA. The "Key Katcher" costs in the order of US$ 80.

formation in the non-volatile memory (which retains the information even when there is a loss of power.) This means that the Key Katcher device can be unplugged, and the information will not be lost.

To access the recorded data, you simply type your password in a text editor and the Key Katcher comes to life. A menu is displayed with options to erase data, view data, search data for keywords, change password, or disable the device.

The device can be seen in Figure 1.1. It should not come as a surprise that such devices exists, and that they are readily available.

Analogous to the privacy enjoyed at home, users also need a private place in distributed systems where they retreat to when they want; in order to read or write private email, for example. Or, more to the point, to encrypt and decrypt. Today, most users are at home when they engage in private exchanges of email. In such a setting, the private machine provides all the flexibility and power one could possibly desire. Both regarding processing power in a trusted environment, and for storage of secrets. However, a growing number of users compute while on the move. What they need is a machine that can act as the trusted haven they have at home, but a portable one.

The computational model in which private machines play a central rôle will be denoted *private computing* to differentiate it from *personal computing*. The latter term is derived from to the term "personal computer" which is a desktop machine.

We will name this the *Trusted Digital Assistant* (TDA). Two possible sanctuaries spring to mind: The palm-top sized PDA and the smartcard.

– A contemporary *Personal Digital Assistant* (PDA) is a palm-top sized, privately owned machine. The accompanying software is geared towards private needs (diary and phone lists) rather than business (spreadsheets).

  In particular, the blend of attributes implies that while the personal computer is there for you when you need it, the private computer is not there for anybody else. It is impossible to give a rigorous definition of which characteristics a computer must have in order to be a PDA. In general it is the combination of software focused on private tasks, and hardware focused on portability and ease of use.

Also, for a PDA to be as useful as we envision, the machine should be portable. If a private machine is to be a sanctuary for sensitive data and computations, having the machine at hand is paramount. In addition, most machines require that physical access to them is restricted if they are to remain secure.

PDAs are available from a wide spectrum of possible solutions. In any case, PDAs are small, and being small implies that they are "meagre": they have few resources, low bandwidth communication, low display resolution, small (if any) keyboard, and so on. As an example, a popular contemporary PDA, the PalmPilot from 3Com, has a version of the well known Motorola 68000 processor (Dragonball), a display resolution of a mere 160 X 160 pixels, and no keyboard at all (it utilizes handwriting recognition and converts strokes into characters and commands). Even a simple task, such as providing a password, is a nuisance because stroking is considerably slower than typing. However, at only 120 grams and 114 X 77 mm it does fit nicely in a shirt pocket.

– A properly manufactured *smartcard* can, for all practical purposes, be considered a tamperproof device where secrets can be stored. The typical secret is an encryption key. The general idea is that a digital signature made by the card should be more trustworthy then one made with a key stored on some other medium where the key is accessible to the user, and others.

  From a Turing-machine point of view, smartcards are obviously "real" computers, complete with communication channels for input and output [139]. The situation is somewhat murkier seen from a user's point of view. Because the smartcard does not have any communication channels that are directly accessible to humans (such as a display), even though they are "real" computers it is not obvious how they can be applied in that rôle. If we maintain the notion that a private machine is one that is unavailable to others, a smartcard is probably as far as one can come. It is off-line most of the time, and has very limited support for sharing of resources. Storing one's keys on a card will protect them properly, but we note that the card is used as a storage device rather than as a computer in the proper meaning.

  There are important obstacles to use a smartcard as a safe haven

for private data and computations The crux of the matter is: When the card is inserted into a card reader, how can the user know what is sent to the card for processing? Even though the card is a computer, the areas where it can be trusted is not sufficient to build a private computing sphere.

The desire to include TDAs in distributed systems raises two issues. First there is the question of what constitutes a TDA, how can such a machine be built, what functionality must it have, and so on. Second, quite a few engineering challenges arise: how to find a balance between the services users would like to have access to, and the ones that can be reasonably supported on small devices, while, at the same time, keeping an eye on privacy. The investigation of these two issues is the topic of this dissertation.

In the context of private computing, security in general and privacy in particular is a personal matter. We believe the user in some cases will have a desire to withhold information from the system he is using, while using it. The example with mobile phones and speeding highlights this aspect. Furthermore, security has traditionally revolved around the systems' integrity just as much as around the users' desire for privacy. As an example, consider a traditional Unix system. When Unix is observed from the users' point of view, it is impossible for the user to construct any type credentials that will persuade the system to let an outsider obtain access to any resources (except by giving the password away). In particular, if a Unix user is off-line, it is impossible to grant access to any data of his (short of surrendering his password). This is in fact true both inside and outside of Unix; that is, it is impossible to construct such credentials regardless of whether the user is logged onto the system or not. Notice that we are not discussing whether a user can give his data away (by sending an email or by writing a program that answers queries), or give other users at the same system access to his data, but rather that the system is "closed" to outsiders.

Unix is a centralized system and has not been designed to support users that are off-line, and it would be unreasonable to claim that this is a fault in the design. But this is true also for many distributed systems. Take for example any system where authentication is based on Kerberos [127]. The original design is becoming quite dated, but the design represents in a sense an extreme point on a scale where private computing—when it comes to trust and control of resources and information—represents the other extreme. Also in regard to Kerberos,

it is impossible to create credentials outside the realm of Kerberos that are valid within. As with Unix, it is impossible to delegate authority to an outsider not recognized (or acknowledged) by the system. In other words, users have not been given any means to delegate access rights to their own objects.

Such a regime is enforceable only when the sole means for users to access the system is through channels (normally workstations) controlled by the administrative body that controls the system itself. Both Unix and Kerberos force users to trust the workstation (by giving it their password). While we might label Unix as a traditional, centralized system, we label Kerberos as a building block in a *distributed* system with *centralized* control. We believe none of these models are well suited for distributed systems with private machines.

Some technologies are particularly prone to lay the foundations of centralized design in distributed systems; biometrics is one of them. The prospects of carrying definite means of authentication at all times is obviously convenient. However, there are quite a few negative aspects to this. First and foremost is that biometric information can not be changed. That is, I can change my password and delete a private key, but I can not change my iris. The result is that it is hard, if not impossible, to end a relationship with a service provider. Furthermore, basing authentication on biometrics marginalizes the user to a degree unseen in other technologies. An introduction to the field can be found in [10, Chap. 13].

In contrast to the design of these two centralized systems, consider a user armed with a TDA, and assume that he trusts it. Because he has a machine he trusts, he is able to perform calculations, and he can, for example, demand two-way authentication as part of the login process; when the user identifies himself, the system should do likewise. Furthermore, with a TDA, a user should be able to create delegation certificates without interacting with the system where the certificates are to be used. We believe that in systems supporting private computing, it is reasonable to expect that the user, not the system, should decide which credentials that are deemed "good enough" for access to the user's resources. However, when users have the possibility to decide for each and every transaction which credentials to delegate to, it is also necessary that the monitor is able to interpret the intentions, and verify that correct and valid credentials are presented. Such a regime might be called an "on the fly" security policy. Quite some machinery

is probably necessary to realize such policies; an infrastructure for dissimilation and validation of public keys and delegation certificates just for starters. In addition, we believe that users equipped with TDAs will interact with many distributed systems, all of which might be under different administrative domains. A single, global, security policy is not possible, and probably not even desirable.

When users are given the possibility to construct their own certificates, a particular aspect of credentials must be considered: The quality. We believe that, in the private sphere, users will not only create their own credentials and value them highly, users will also value different sources differently. That is, when there are multiple sources for credentials, users might deem their "quality" different. The quality of credentials are considered only in the cases where the credential is (cryptographically) valid; the quality is an attribute augmented to the credentials by the user, not by the system. For example, to access a document created as part of a professional relationship, a user might choose to accept credentials originating by an administrative body (their common employer). It is reasonable also to expect that in order to grant access to a private letter, a different set of credentials might be requested, even if the requestor is the same one. The quality of credentials is a piece of information that obviously can not be determined by a machine unless it is already mapped down to one from a set of discrete values by the user. Adding quality—in addition to validity—to the attributes of credentials poses yet another challenge for system designers.

A TDA is not, by no means, a panacea in privacy matters, but we believe that users will trust their TDA more than they do machines not under their (physical) control. At the very least, we believe that users will find it less hard to trust machines under their physical control then to trust machines controlled by others. Furthermore, we also believe there are all reasons to assume that a single-user machine under the users' physical control is more trustworthy that other machines that might be available.

Because users neither can encrypt nor decrypt without assistance, it is necessary to trust some part(s) of a system in order to interact securely with others. The part(s) one chooses to trust becomes the Trusted Computing Base, or just TCB [35]. A somewhat imprecise description of what a TCB is can be those components that can betray the user, without the user noticing before it is too late [85]. It is important to know exactly what constitutes the TCB, as elements outside of

it normally can not inflict as much harm on the user as those elements inside.

On one hand, having in the TCB a private computer that is trusted not only for a limited domain (such as file servers are, for example), but rather in general, enables users to firmly place themselves in the center of their own computing environment, delegating authority over their resources on a per-use manner and interact securely with service providers. On the other hand, if a private machine is to be truly useful, it must be so small that it can be brought along conveniently, which limits its applicability. Taken together, we believe that the net effect is one where the TCB will vary in both time and space. To meet one challenge the user will trust some (remote) server to act on his behalf, to meet another a different set of servers are relied upon. This leads us to take a closer look at how to design and implement a variable-size TCB. That is, one that differ in size depending on the context, but where a TDA is a central element.

1.1 Problem statement

To summarize the introduction, we have claimed the following:

- Private, PDA-style of computers will be commonplace; we have not made any decisive claims relating to the technology needed to create a TDA.
- Private computing will pose a shift in the security and privacy concerns of users.

After asserting that they are (or will become) true, we have observed that systems' design and implementation will face new challenges due to new types of demands and priorities. We are led to pose the following questions:

1. How does the integration of private machines influence the design of distributed systems related to security and privacy?
2. How can users with private, trusted devices be included in the decision loop when it comes to sharing, delegation, authorization, and so on?

This dissertation will revolve around these questions and shed light on them from different angles.

1.2 Methodology

In general, security can not be analyzed by running an implementation and harvesting the results, as is possible with performance, for example. The reason is simple: The actual security of a system hinges not only on the implementation, but also on its design; security is guarding against the unknown. The best we can hope for is to analyze subsets of the system. In particular, we can analyse (parts of) the design in order to gain a better understanding of the assumptions needed for the design to be sound; a design that is sound is both secure and useful. Such understanding will pave the way for a more secure implementation. The issues that we care to analyse are derived from the nature of the object we study: security in a distributed system.

The questions we have posted can in some sense be answered by implementing a system that demonstrates that some aspect can be efficiently designed and implemented. However, rather than labor to enumerate them, borders will be drawn, both theoretical and practical, within which a viable system must be designed. Protocols to be used between the machines are analyzed by means of a logic of authentication, and trust relations between principals by means of a theory of authorization and delegation.

Evaluation by means of implementation can rightly be criticized for only providing a single point in a large design space. However, the lack of resources experienced by PDAs places strict limits on what can be realized, and an implementation will serve as an proof of concept. Also, an implementation forces us to a throughout study of the interaction between different components in a system. This can yield an understanding not available by other means.

1.3 Outline and main results

This dissertation deals with security in general and privacy in particular, in systems where private machines are used to represent their owners.

The following is an outline of the text.

Chapters 2 and 3: Security is made possible by, among other things, the application of encryption. To ensure that this dissertation is "self contained" in this respect, these two chapters very briefly present cryptography and tools to analyze aspects of security in

distributed systems. Cryptographic tools make it possible to build secure systems, while the analytical tools make it possible to study system components; protocols in particular. Some themes that are discussed are shared- and public key encryption, hashes, digital signatures and infrastructure to distribute certificates of different types. There is little focus on the inner workings of the cryptographic tools that are used, and Chapter 2 contains a detailed description of the assumptions made about them. Chapter 3 is concerned with logics for the analysis of protocols and a theory for authentication and delegation.

Chapter 4: Designing systems to encompass private machines requires a different approach to central issues such as control and management. It will be argued that systems encompassing private machines give rise to new semantics to well known terms such as *trust* and *conflicting updates*. We claim that this is an aspect of private computing in general. A design strategy, named the *open-end argument*, is presented and discussed.

Chapter 5: In order to argue that the open-end argument is viable we will present a distributed system named File Repository (FR). FR offers distributed storage of files, without being a traditional distributed file system, while having a user-centric design. FR has been designed with with private computing in mind, and to that end it offers to the user mechanisms to control his own security regime.

A per-user security regime requires the system to provide mechanisms for each and every user to decide which, if any, third parties he trusts, which (types of) credentials he is willing to accept for authentication, and the set of servers he trusts to perform tasks (replication, storage, authentication) on his behalf. In addition, a lax security regime of one user should not jeopardize the privacy of others.

FR has served as our research vehicle, and the system is in a constant state of flux. We present the current design, which is under implementation. The relevance of FR in this context is that it shows that in private computing, striving for transparency is not a viable strategy. This reflects upon security in that transparency is not a viable strategy there either.

Chapter 6: The discussion in Chapter 3 highlights the necessity of making precise (and correct!) assumptions about participants, their intentions and abilities. Although precise and correct assumptions underpin any reasoning on the security of a system, actually picking the right assumptions is hard. For example, one should not assume without careful consideration that a private key is indeed kept secret and used with prudence. We discuss two different settings where realizing (e.g. implementing) assumptions is surprisingly hard. An aspect of assumptions is discussed in this chapter: the gap between intent and ability. It is our view that this gap is an abyss and the security of systems which encompass private computers must be seen in this light. In essence, this chapter is concerned with pure engineering aspects of designing systems with security in mind. Both at the operating system level, and authentication protocol level. First, the task of deleting a file (and its contents) is scrutinized. It is demonstrated how difficult even this seemingly simple task is, and the example is used to elaborate on the semantics of good encryption keys and the problems arising from having transparency as a design goal.

Regarding protocols, the focus is on a particular disturbing scenario: What happens when a user loses his trust in some formerly trusted principal? In general, the situation can be described as follows: if a user tomorrow loses trust in someone he trusts today, is he certain that his former trust cannot be misused?

In centralized systems the problem has a different flavor. There, a user who loses trust in the system simply refrains from using it. The (owner of the) system can claim that the user continues to use it for as long as he wishes, and there is little the user can do to prevent it. On the other hand, in most cases the association between the user and the system is rather weak, and the fact that the system can send messages it claims originate from the user is fairly obvious. In distributed systems with autonomous participants, one would expect that the separation between systems obliterated any such dependency. Users represented by a machine they trust should be able to erect a wall separating themselves from any system they use; especially in such a way that any future misuse can be avoided.

Protocols are scrutinized with this in mind, and the findings are

described: some protocols are not secure in this respect. Based on this observation, a method is developed to analyse protocols for such defects. It is shown that the defect is intrinsic to shared-key cryptography, but introducing public-key cryptography does not automatically solve the problem. It is shown that the design *per se* might render valid assumptions incorrect. In particular, regardless of how careful a user might be, protocols designed to protect his privacy implicitly assume the perpetual honesty of other parties; an assumption one should not make without careful consideration. Shared key cryptography is an attractive option for performance reasons, and designers should beware of the problems associated with them.

Chapters 4 and 5 are concerned with how one goes about designing systems where private computers are integrated in the design. The claim is that following the open-end argument is a valid and viable design strategy, and that certain constructs and designs should be avoided.

Chapter 7: FR has been designed to be a cog in a private computing regime. Private computing is not constrained by the systems that supports users in the same way as more traditional designs. With the potential of becoming a central building block in private computing, FR must support "on the fly" security policies. The problem is amplified by our desire to use contemporary TDAs, not equipped with the computational power it turns out that is needed to achieve our goals; integrating smartacrds also pose some interesting issues. Uncommon signature schemes are necessary to facilitate such functionality.

The implementation is discussed in some detail. The main contribution is an exploration of off-line delegation.

Chapter 8: A principal is a participant in distributed systems capable of doing "more" than just sending messages. Principals must be able to make statements; a statement is data accompanied by a digital signature that vouches for its origin. It is the signature that sets statements aside from other strings claiming to originate from the user. Public key technology is but one means to achieve signatures.

This chapter is concerned with what constitutes a TDA. It is fairly

obvious that there is a smooth transition between a large palm-top and a small laptop (and from there to more powerful computers). A more interesting case is the transition between a small palm-top and the smartcard. To highlight the functionality required by a TDA, we discuss how digital signatures can be created by means of a smartcard. It will be shown that (semi) trusted third-parties, complex protocols and a fair amount of user involvent is necessary to achieve what can be done quite easily with a TDA. It is also shown that the resulting signatures are less strong (more assumptions must be accepted). An exploration into the assumptions that are made reveals that TDAs provide the crucial functionality lacking in a smartcard: a secure bi-directional channel between the user and the card. The main result is a method to securely sign a message by means of a smartcard in a hostile environment.

To verify that the method is usable, an implementation has been conducted; some implementation details are discussed. The implementation demonstrates that the method we have devised indeed can be utilized.

Exploring further the border between a smartcard and the implications of the term TDA, it seems prudent to ask: with what does a smartcard need to be augmented in order to be able to be the foundations of a private computing sphere? The only aspect of a TDA that is not shared by a smartcard is the secure channel between the machine and its user. Although it is obvious in principle that a keyboard and a display must be added to the card, a question of implementation remains. A novel implementation is presented.

The contribution is that light is shed on the area between the contemporary TDA on one hand and the standardized smartcard on the other. Implicitly, what constitutes a TDA is also discussed.

Chapter 9: Conclusions are drawn from the investigations and their relevance discussed.

1.4 Notation

The following notation will be used; it is adopted (with a few modifications) from [136]. As will become evident later, the notation captures

many of the assumptions we make on the cryptographic tools we apply. Here, we only discuss the notation, and return to the actual assumptions in Chapter 2.

K_A**:** The encryption key K is known only to A, a principal of some sort (a human, a process, a machine, or what have you). It is normally assumed that the key is *secret*. A key is secret if it is (believed) to be known only by the "owner" and those the owner trusts in this matter.[4] K_{AB} is used to indicate that the key is known to both A and B, and as such represents a bidirectional encryption channel between them.

$\{X\}_K$**:** The datum X is encrypted with the key K; the key is explicitly not assumed to a shared key. If a key K is used for encryption, there must exist a key $\tilde{K}$, called the *complement* of K, by which the message can be decrypted. In shared-key system $K = \tilde{K}$ while this is not the case for public-key systems.

$\{X, Y\}_K$**:** the datums X and Y are encrypted together. In addition to being "hidden" by encryption, it is not possible to determine, without access to the key $\tilde{K}$, the relative size of X and Y.

(K_A, K_A^{-1})**:** A public-private key-pair, where the private key K_A^{-1} is known only to A. Usually the public key (K_A) is publicly known, but need not be. We need not assume that

$$\{\{X\}_{K_A}\}_{K_A^{-1}} = \{\{X\}_{K_A^{-1}}\}_{K_A}$$

although this is sometimes true. It generally is the case that $K_A^{-1} = \tilde{K}_A$, but the two notations are used to enable us to distinguish a decryption key from the secret part of as public-secret key-pair.

Public-key cryptography is used for three purposes:

Encryption: $PK_\psi(A, K)$ represents the key K, known only by A, supposed to be used for encryption; that is, sending encrypted messages to A.

Signature: $PK_\sigma(A, K)$ represents a key for signatures (and verification of signatures).

[4] Normally it will also be assumed that those that are trusted to know secrets that might be used for authentication refrain from using them for this purpose.

Key agreement: $PK_\delta(A, K)$ is a key for the exchange of encryption keys.

Regardless of how a public-secret key-pair is used, K_A is the public part while K_A^{-1} is the secret part. Not all systems can be used to implement all three types of functionality.

When $PK_\sigma(A, K)$ the notation $\{X\}_K$ is used to show the signature by K on X, but possessing $\{X\}_K$ does not imply possessing X. When we want to indicate that X and $\{X\}_K$ is available *together* we shall write $[X]_K$.

$\langle X \rangle$**:** A *message* containing X. In some cases it is necessary to differentiate between the message M and the contents Y of that message. For example, $M = \langle \{Y\}_K \rangle$ captures the fact that Y was encrypted with K, and sent as message M. In general, *messages* are never encrypted because an encrypted message is indistinguishable from a random string of bits at the receiving side.

$A \rightarrow B : m$**:** A message $\langle m \rangle$ is sent from A to B. The notation could be $B \overset{a}{\leftarrow} A : m$ (where m is the name of the channel) to indicate that B received m on a channel which is known to originate at A [24]. However, no assumptions are made on message delivery since A has no *a priori* knowledge about who will actually receive m. $A \rightarrow B : m$ indicates what A actually does (sending a message to B) rather than indicating that B receives it.

In addition, when a message is sent from one participant to another, it is not assumed that the communication channel has any interesting properties such as providing secrecy, integrity or (ordered) delivery.

1.5 The terms security and privacy

Unfortunately, the terms privacy and security are somewhat imprecise, and the lack of precise definitions inevitably leads to unclear answers to many question regarding these issues. Laboring to find a rigorous definition is probably not productive and will resort to discuss some general aspects. A rough divide of the field will result in the following [102, 17, 118]:

Authentication: Verifying an identity claimed by a participant.

Access Control: Protection of resources according to a policy.

Audit Trail: Logging of activities.

Confidentiality: Ensuring that unauthorized participants donot obtain access to information supposed to be protected.

Integrity: The property that information remains unaltered.

Availability: Resources are supposed to be available upon request, according to some policy.

Nonrepudiation: Ensuring that claims can not be disavowed.

The essence of privacy is to avoid that systems leak private information.

There are essentially two ways to maintain privacy (in addition to refraining from using systems):

1. Ensuring that those one trusts are trustworthy. This approach requires social engineering.

2. Keeping zealous control over information.

The latter is less hard to realize than the former, and users must rely on security to protect themselves.

We will not be concerned with logging (audit trails), availability or nonrepudiation as we regard them as outside the scope of this work. In this light, we can say that security is to ensure that the following two attributes are maintained for every object.

- Confidentiality

- Integrity

Subscribing to this definition implies that if security is maintained, and those a user trusts with information are trustworthy, privacy will not be breached. According to Merriam–Webster's Collegiate Dictionary (Tenth Edition) the term *privacy* is *the quality or state of being apart from company or observation* and also *freedom from unauthorized intrusion*. In our setting with distributed systems that may leak information, we will widen the definition somewhat, to mean *what one has when information (regarding oneself) that is given away (be it explicit or implicit)*

is used solely for the purpose for which it was intended [143]. Several issues arises from such a definition, among them:

– Privacy (as a term and definition) is rendered meaningless unless it is used in some context.

  We believe that in most cases the "intension of use" is external to the system per se. For example, it is not part of the GSM standard that the provider should not reveal to outsiders its customers driving habits.

– Information without an explicit purpose should not be used.

  In particular, for every bit of information, it is impossible to define privacy unless it is explicitly made clear what purpose the information has. In many cases, the less these intentions are defined, the more private one would wish the information to be held. In other words, if the intentions are not known, one wishes to err on the side of prudence.

  It follows that if a user ensures that no information is implicitly leaked from his activities, that in itself assists to ensure that his privacy is not breached. The threat of speeding while subscribing to a GSM service is an example of this; The fact that anonymous subscriptions is an option with many service providers does not alter the validity of the argument.

The upshot of all this is that users which rely on their private computers will probably have to protect their privacy themselves rather than relying on service providers.

Chapter 2

Cryptography

2.1 Introduction

Cryptography is very different from security. At best, cryptography is one of a large set of tools and engineering techniques one can use to achieve security. Encryption can be implemented on any general purpose computer (e.g., it can readily be made available in software) while security requires users to act in certain ways. Even worse, as it will become evident, obtaining security requires that users are willing and able to understand the rather complex and subtle relationships that exist between different entities—be it trust relations between principals, relations between the keying material in public-key cryptosystems, or between integrity and confidentiality.

This chapter features an overview of the cryptographic techniques that are employed in both the text, in systems that are discussed, and in implementation efforts we will present. In addition we give a short overview of smartcard technology in Section 2.5. A smartcard can be regarded as a tamper proof device, and as such it can be used as a building block in secure systems. However, as always, it is not a "silver bullet".

The disposition is aimed at readers with a good understanding of computer science, but lacking insight into cryptography. The presentation is terse as the aim is not to educate but rather to establish a common vocabulary and clarify our assumptions. There are many excellent tutorials on this material, see for example [10, 50, 93, 128, 115,

119, 106].

The cryptosystems that will be discussed are secure in a *computational* sense rather than an *information-theoretical* sense. By this we mean that is it believed to be computationally *infeasible* rather than information-theoretically *impossible* to violate the assumptions that will be put forth. This infeasibility might stem from either the sheer size of some language (key space) or from the number of calculations needed to perform some task (factor a large composite number). Obviously, development in technology will alter which computational tasks are considered infeasible. Presently, cryptosystems that, on average, require 2^{90} calculations or more, are considered safe for more than 10 years [21]. That is, having made a conservative estimate of the technological development in the coming ten years, it has been concluded that, ten years from now, 2^{90} calculations will still be considered infeasible.

If we for a moment allow ourself to be somewhat informal, we can say that the term *infeasible* denotes just that: A task that can not be performed in practice regardless of how many resources are made available. As discussed above, it is believed to be infeasible to execute 2^{90} calculations, more or less regardless of how they are implemented. A more formal view is also possible: we can define the term *feasible* to mean "computable in polynomial time and space" and *infeasible* to mean "not computable in polynomial time and space". This implies that a function f is infeasible to compute if it belongs to the class **FNP** (the class of problems as hard as those in **NP**, but that exists as functions rather than decision problems). A function f is feasible if it belongs to the class **FP** (the functions that can be computed in polynomial time). In the case of shared-key cryptography the length of the key is normally fixed (even if one can choose between several different lengths), and the size of the key space is given as a function of the length of the key (in bits). In most cases the key space will grow lineally with the key (viewed as a number) and thus exponentially with the length of the key. For public-key encryption this is often not so and we rely on the exponential growth in the number of calculations needed to invert the one-way function when the key is made longer (bigger).

Regardless of how one defines "infeasible", there are two ways to view cryptography. One is to view cryptography as manipulation of symbols where security properties are expressed formally, the other as operations on bit strings where security properties are expressed

in terms of computational complexity [4]. All of our analytical tools regard cryptography as manipulation of symbols.

This chapter will present shared- and public-key cryptography, digests, and digital signatures, with the corresponding problems on which they rest. An existing system is presented to demonstrate how cryptographic technology is put to use.

First comes shared-key encryption in Section 2.2; IDEA will serve as example. Then, public-key encryption in Section 2.3, starting with one-way functions (with MD5 as example) in Section 2.3.1. Then, RSA and the Diffie-Hellman exponential key exchange can be presented as examples of public key cryptosystems. In Section 2.3.3 some light is shed on elliptical curves as a newer algebraic setting for cryptosystems; we apply these methods in a later chapter. The notion of digital signatures is discussed in Section 2.3.5. PGP will be used to demonstrate an implementation of software providing security.

Cryptography is commonly used as an integral part of communication protocols. In most situations, it is assumed that all encrypted messages are properly engineered [54, 57]. Furthermore, it is assumed that users are able to generate good encryption keys (and nonces) when required, although it is acknowledged that the notion of "good" depends on both the purpose and the arbiter. Producing random data is an implementation issue that depends to a large degree on the actual system in use. Modern computers come with high-quality random number generators built in; entropy is often obtained from the thermal noise (Johnson noise) generated by a resistor [72]. Hunting for random numbers has been a tedious task, but that seems to come to an end.

2.2 Shared-Key Cryptography

Shared-key (or, symmetric) encryption is the traditional (pre 1976) form for encryption [75]. Both parties know a secret (key) and it is used as input to a transformation. The transformation is at least a surjection from one language to another (if the languages have the same size, the transformation is a bijection); this is required if decryption is to be unique. Secrecy should not rely on knowledge of the transformation in itself, mostly for practical reasons as the transformation then becomes part of the key (which is to be kept secret). When the algorithms is (logically) part of the key, it becomes very hard to remember the key

without resorting to written sources. When the only requirement for security is that the key is kept secret, it is simple to understand how to maintain security in such cryptosystems. However, distributing keys between the parties requires a secret channel, and the cryptosystem is no more secure than the channel for key distribution. Distributing keys is the major difficulty when using shared-key ciphers in isolation.

A shared (secret) key represents an encryption channel. This channel is assumed to have the following two properties:

Secrecy: Access to encrypted messages is denied to anyone who does not have access to the correct key. In particular:

- Given an encrypted message $\{M\}_K$ it is infeasible to determine M, for all M, without prior knowledge of K; e.g., the cryptosystem is immune to ciphertext-only attacks.
- Given any sequence of encrypted messages $\{M_i\}_K$ for any sequence of messages $M_1, M_2, \ldots$, it is possible to determine neither K, nor N from $\{N\}_K$; e.g., the cryptosystem is immune to known-plaintext attack.

Integrity When receiving a message $\langle\{M\}_K\rangle$, after processing and decryption, it can be determined whether M has been fully recovered; no altering will pass undetected, including truncation. It is also assumed to be infeasible to construct $\{M\}_K$, for all M, without knowing K.

Most shared-key ciphers transform one block of data into another block of the same size. The size of a block varies, but 64 bits is common. Because the transformation is a bijection, identical blocks of data will be transformed to identical blocks of encrypted data. Unless precautions are taken, this reveals information about the encrypted data; cryptanalysis is performed by exploiting such leaks of information. To remove this weakness, *chaining* can be applied. One possibility is cipher-block chaining (CBC). Assume there are t blocks of data: $x_1, x_2 \cdots x_t$. Encrypting x_i yields c_i. Prefix the data by a block of random data, called *initialization vector* (IV). Regard the IV as c_0. Then, for each block x_i, xor x_i with c_{i-1} before encryption. Each encrypted block c_i, except c_0 which is random data, will depend on the preceding block. During decryption, single bit error in block c_i will render x_i invalid, and introduce a single bit error in x_{i+1}. It is an advantage that the chaining

is "self restoring"; notice, however, that any checksum applied to the decrypted data will fail, and give an indication that the incorrect key has been used.

Regarding encryption in general we will make some broad assumptions:

- Given $\{X\}_K$, we assume it is infeasible to precisely determine the length of X; in an implementation the length of X can be obscured both by padding and by compression.
- Given $\{X, Y\}_K$ it is infeasible to determine, without access to the key K, the relative size of X and Y.
- We assume that $\{X\}_K$ and $\{Y\}_K$ are equivalent in the sense that without access to K, it is infeasible to determine whether $X = Y$. Actually, this assumption is related to implementation issues rather than cryptography because, obviously, encrypting X yields a different result than encrypting Y. However, by using a random initialization vector and CBC mode when encrypting, this assumption can be made to hold.

These assumptions can be summarized as "perfect encryption" [22].

Shared-key ciphers achieve their secrecy by diffusing the data out over the encrypted data, and by making the relationship between the key and the cipher text as complex as possible. Complex combinations of transposions and substitutions are repeatedly transforming the data under control of the key.

As an example of a shared key cryptosystem, what follows is a presentation of IDEA (see below); much of its fame stems from it being used in PGP [82, 6, 149]. It is a block cipher, encrypting 64-bit blocks of data with a 128-bit key. The same algorithm is used for both encryption and decryption.

First, the key is used to generate 56 sub-keys. A block of data is split into four 16-bit sub-blocks. There are eight identical rounds. In each round, each sub-block of data is xor'ed, added and multiplied with six new sub-keys and the other three sub-blocks of data. In the ninth round, one of the remaining four sub-keys is applied to each sub-block. The operations involved in the algorithm map nicely down to those available on contemporary general purpose computers, and the algorithm lends itself to fast implementations even on 16-bit machines. The performance is excellent: on (an old) 486DX2-66 data can

be de/encrypted at 1.7 Mb/s, while on a (somewhat more) modern machine, a 180 MHz PentiumPro, data can be de/encrypted at 16 Mb/s. A hardware implementation is available, it de/encrypts at 300Mb/s.

2.3 Public Key Cryptography

In the mid seventies it was understood that there are some mathematical phenomena that can be exploited in a fascinating way [37, 36]. This section describes the ideas on which public key cryptography is founded. Much of this material is mathematical in its nature, but focus is on the concepts rather than on the proofs. This presentation is built on [97].

Invented in 1976, public key encryption has made it possible to use encryption in new settings, mainly by making key maintenance less a burdon. In short, public-key encryption offers two interesting features [110]:

- An existing channel that provides integrity (the sender of data on the channel is known) but not privacy, can be converted into one that also offers privacy. This is called key exchange.
- An existing channel that provides integrity can be used to add secrecy and/or integrity to any other channel. This is called digital signature.

In the next sections we will briefly discuss both "ordinary" one-way functions and those with a trapdoor built in. Then we will present some public-key cryptography proper.

2.3.1 One-Way Functions

Public-key cryptosystems rely on the existence of one-way functions. A function f is one-way if it satisfies the following conditions:

1. f is injective (one-to-one). Notice that even though the "hash functions" (or "digital digest functions") are often said to be one-way functions, they are one-way in a very different meaning of the word; we will return to this class of functions below.

2. $f(x)$ is at most polynomially longer or shorter than x. Notice that there is no reason why elements in the preimage of f have to be the same length as in the image.

3. f can be computed in polynomial time on the length of the input.

4. The function f^{-1}, the inverse of f, is not in **FP**. That is, there exists no polynomial-time function which, given y can find an x such that $f(x) = y$, or determine that such a value does not exist. f^{-1} must then be a member of **FNP**.

Obviously, finding a function that is in **FP**, but where the inverse is not in **FP** but rather in **FNP**, is only possible if the two classes are different. This can only be the case if **P** $\neq$ **NP** and the discussion that follows will hinge on **P** being different from **NP**.

To construct a *working* public-key cryptosystem, one needs a particular kind of one-way function: The *trapdoor one-way function*. A trapdoor one-way function is a one-way function as described above where there exist some information that enables the holder to perform the task that is infeasible without. The best known example is that of factoring. Given two "large" primes p and q and the composite number $n = pq$, any operation which is easy when one knows p and q, but infeasible without, could be used as foundation for a trapdoor one-way function (because factoring is so hard). Factoring is a function (not a decision problem), and it is believed to be in the class **FNP** (it is not in **NPC**, but nevertheless a "hard" problem [49]).

As an example of a one-way function with a trapdoor, we will describe the Diffie-Hellman problem. It is well known that given a (large) prime p and the Galois field $GF(p)$ (that is, the numbers $\{0, 1, \ldots, p-1\}$ under arithmetics modulo p), it is simple to calculate

$$Y = \alpha^X \mod p, \text{ for } 1 < X < p - 1$$

where α is a fixed primitive element of $GF(p)$ (that is, the powers of α generate all the non-zero elements $\{1, 2, \ldots, p-1\}$ in some order). It is believed to be infeasible to find X given Y and α. However, given X and α it is easy to find

$$Y' = \alpha^{XX'} \mod p, \text{ for } 1 < X < p - 1$$

So while finding the discrete logarithms is infeasible, knowing X or (X') amounts to a trapdoor. The Diffie-Hellman exponential key exchange is built on this problem [37].

Assume Alice wants to communicate secretly with Bob.[1] As a one-time set up Bob has published appropriate prime p and generator α of Z_p^* (that is, of $GF(p)$ as explained above). Alice chooses a random secret $x, 1 < x < p-1$ and calculates her part of the key $K_A = \alpha^x \bmod p$. She then sends the first message:

$$\text{Message 1} \quad A \rightarrow B \;:\; K_A$$

Upon receipt, Bob chooses a random number $y, 1 \leq y \leq p-1$, computes his part of the key $K_B = \alpha^y \bmod p$, and sends the reply:

$$\text{Message 2} \quad B \rightarrow A \;:\; K_B$$

The secret, shared key is $K_{AB} = (\alpha^y)^x = (\alpha^x)^y$; all arithmetic done modulo p. Both Alice and Bob can easily calculate K_{AB} as explained above.

Diffie-Hellman can also be used to send encrypted messages, not only to establish a secret channel. Assume Bob has published the three numbers α, p, and $K_A = \alpha^x \bmod p$ as his public key. In order to encrypt the message M, Alice generates a random number y and calculates $K_B = \alpha^y$ and $K = (\alpha^x)^y \bmod p$. The key K can be used as an encryption key to create $\{M\}_K$. Alice sends the message $\langle K_B, \{M\}_K \rangle$. Because Bob knows his public key, he can calculate K, and decrypt the message.

Secrecy is based on

$$K_{AB} \neq (\alpha^x)^{\alpha^y} \neq (\alpha^y)^{\alpha^x}$$

That is, to find K_{AB} one needs either x or y. An attacker must thus find x from Message 1 (or, obviously, y from Message 2), which amounts to finding the discrete logarithm (that is, given $K_A = \alpha^x$ find x. If p is 1000 bits long, about 2000 multiplications are needed to find K_{AB} from K_A and y. An attacker, who must find x from α^x, is faced with something in the order of 2^{100} operations. No easier solution is known. On the other hand, it has not been proved that solving the discrete logarithm is the only way to obtain access to K_{AB}.

2.3.2 RSA

RSA is based on a similar mathematical problem as Diffie-Hellman. Alice chooses two "large" primes p and q, she finds the product $n = pq$

[1] Alice and Bob was introduced in [109] as the first couple in cryptography.

of these primes, and chooses an exponent e. She publishes (n, e) as her public key. When Bob wants to send her a message M he calculates $C = M^e \mod n$. This exponentiation is easy, and it is also a one-way function. If p, q and e are chosen with some care, finding M from C is infeasible. However, Alice knows the factors p and q and she can easily find a number d such that

$$(M^e)^d = M^{ed} \equiv M$$

(all arithmetic modulo n). That is, she has some material enabling her to open the trapdoor; the pair (d, n) is called Alice's private key.

The security of RSA depends on the difficulty of factoring n, because even if n is (part of) the public key it is infeasible to find p and q (requires factoring), and thus infeasible to find M, and infeasible to find d given e (which is the other part in the public key). To model a known plain-text attack, assume a known message M and the corresponding ciphertekst C. To find d the attacker must solve $M = C^d \mod n$, which is finding discrete logarithms; a well known hard problem. A ciphertext-only attack is equivalent to solving $C = M^e \mod n$ where (only) C is known, which is equivalent to taking roots modulo a composite number with unknown factorization.

In order to make RSA secure n has to be large, typically 1024 bits or more. A software implementation of RSA, run on a Sparc-II, uses 0.93 sec. to encrypt (sign) with a 1024 bits modulo [115].

2.3.3 Elliptical Curve Cryptography

A signature made with RSA is as many bits as the composite number in the key (called n in most context). Typically, n will be between 512 and 2048 bits long. That is, anything encrypted with RSA will be as long as the number n in the public key. Elliptical curves provide an alternative algebraic setting for cryptosystems where the public keys can have fewer bits.

Whether finding discrete logarithms is infeasible or not depends not only on the size of the field, but also on the "structure" of the underlying (cyclic) group. To that end, the group of an elliptic curve defined over a finite field has interesting features.

En elliptical curve can be drawn. In this interpretation, a number is represented as a point $P = (x, y)$ on the curve, and the number $-P$ is defined as $(x, -y)$ (flipping over the x-axis). A non-vertical straight

line that intersects an elliptical curve, does so in two more places (this is a property of the curve). Adding two numbers (points) is defined as drawing a line from one to the other, yielding the negative of the sum. That is, the line through P and Q intersects the curve in $-R$, and R can be found by flipping over the x-axis. Adding a point to itself is also addition, but with only one point. This is defined as drawing the tangent to the curve in the point, and the point where the tangent crosses the curve is then again the negative of the resulting number (that a non-vertical tangent to the line will cross the line in another point, is another property of the curve). Having defined the $+$ operation, and the number 0 as a point to which all lines are vertical (infinity), we have an algebra.

Assuming that we only consider points with integral (x, y) components, we can construct a curve over the field created by a large prime; the curve would then form a group. Then, informally, the *elliptic curve discrete logarithm problem* can be stated as follows: Given two numbers Q and P on a curve E, where $Q = iP$, find i. In other words, more or less the same mathematical problem as finding discrete logarithms, but in a different algebraic setting. For details, please consult [92].

Some attacks that are known on "ordinary" discrete logarithms are not applicable in the setting of elliptical curves. Therefore, when public key encryption is realized in this setting, the result of encryption can be much shorter than, for example, RSA keys (and still provide comparable security). It is believed that a cyclic subgroup of size 2^{160} provides a secure setting for a cryptosystem. A public key is a number and consists of two points on the curve, thus a key is twice the size (number of bits) as the size (in bits) of the group.

2.3.4 Hash function

One more cryptographic technique is required in order to construct cryptosystems with high performance: The digital hash. A hash function f must have the following properties:

Compression: f maps an input of arbitrary but finite length to an output of fixed length. The output is typically 128 or 160 bits.

Ease of computation: f is easy to compute.

Preimage resistance: Given only $f(x)$ it is infeasible to find x.

2nd-preimage resistance: Given x it is infeasible to find a second input x' so that $f(x') = f(x)$.

Collision resistance: It is infeasible to find x and x' so that $f(x) = f(x')$. Notice two degrees of freedom.

Because f maps a long input down to a short one, it is obvious that the function is surjective, and thus not a one-way function as described above in Section 2.3.1. In other words, it is obvious that no message-digest function actually is collision resistant.

Given a block cipher, a hash function can easily be constructed. Use the key 0 (zero), split the input data into the block size of the cipher and encrypt each block in CBC mode. The encrypted last block depends on all previous blocks, and if the usual assumptions hold for the cipher, a hash function has been implemented.

When f offers preimage resistance, proving that one has the digest is as secure as showing the original data. Furthermore, when f offers 2nd-preimage resistance, the digest of data will be proof that the data was available when the digest was taken; when the data is secret, this can be used for authentication. The first use of one-way functions was to authenticate users' passwords against a digest of the password stored in a file [144].

MD5 will be our example of a dedicated hash function. In MD5, the length of the input is added to the input itself (effectively making the hash depend (also) on the length of the input), and then padded to a multiple of 512 bits. The data is then divided into a number of 512-bit blocks, which again is divided into 16 32-bit sub-blocks. The algorithm has four rounds, in which both logical (logical and, xor, or and not), non-linear (sine) and shifting is applied to the data. The size of the input is limited to 2^{64} bytes. The speed of MD5 was reported to be in the 100 Mbps range [138].

2.3.5 Digital Signatures

A signature on a (paper) document creates a tie between the document and the person. Because signatures (supposedly) are unique, the signature is also undeniable. However, when important documents are to be signed, witnesses and/or a notary public must be used as well. The value, and limitations, of a written signature as evidence is well un-

derstood. In particular, signatures are understood to be evidence, not proof [110].

The second contribution of public-key encryption is that of digital signature (the first, as explained, is key exchange). In relation to private computing, there is an particularly interesting aspect to keep in mind: A digital signature is bound tightly to the signed document and only loosely to the signer, while a handwritten signature is bound tightly to the signer and only loosely to the signed document.[2] The implication is that when it is established that a digital signature is (cryptographically) valid, it is still an open question who actually made it. Compared to "ordinary" signatures, there is an fundamental difference, in that "ordinary" signatures can not be valid and produced by "someone else" at the same time. Also, it is hard to imagine how to arrange for a written signature to be detached from the document, while still being valid.

Because (cryptographic) validity does not vouch for any binding to a person, technical issues related to the protection of encryption keys is of crucial importance to privacy.

2.3.6 Conclusions

This dissertation is not concerned with cryptography, and we will take a "block box" approach. To that end, we will assume the following:

- It is infeasible to find an encryption key by means of cryptanalysis, be it shared- or public key cryptography.
- Inverting a hash function is infeasible.
- Unless explicitly noted, a good source of randomness is available.

As with all cryptographic tools, *caveat emptor*!

2.4 Putting it all together: PGP

Pretty Good Privacy (PGP) can be used as an example of how ciphers are put to use. Although PGP is very well known, we include a short description here because we will use the format of one of its packages as example later.

[2] As was first observed by Matt Blaze at AT&T (stated by Carl Ellison in the SPKI mailing-list).

PGP is a program intended to provide users access to high quality encryption [149]; the format used by PGP is well documented [6, 31]. In general, PGP provides users with the ability to sign and encrypt (and decrypt) with shared key or public key ciphers. PGP has an the distinguishing feature that it explicitly does not assume any infrastructure for exchanging and verifying public keys. Authentication of messages is based on well-known public keys [11].

PGP uses IDEA as a shared key cipher, up to version 2.6.3 solely, MD5 for hashing, and RSA for public-key encryption. A message encrypted with a public key can be used as an example of how PGP encodes messages; it is described in great detail in [6].

PGP uses the term "package" to indicate data of some type. For example, a package containing a message encrypted with a public key consists of information used to ensure integrity together with information on the key that is used, followed by an IDEA key encrypted with the RSA key, followed by the original message encrypted with the IDEA key. Integrity of the RSA package is ensured by appending a 16-bit checksum of the encrypted IDEA key to the key itself before RSA encryption. Integrity of the IDEA package, which contains the original message, is ensured by prefixing the data with 64 bits of random data followed by the last 16 bits of random data again. Upon decryption with IDEA, the 16 bits starting at bit 49 up to bit 64 should be identical to the bits 65 to 80. Furthermore, the random material ensures that the cipher feedback chaining is properly initialized. Taking it all together, the format of a message encrypted with a public key is as follows[3]:

$$I_{KP}, \{KS\}_{KP}, \{R, r, X\}_{KS}$$

where KP is a public key and I_{KP} the "name" of the key, KS is a fresh shared key, R random data, r the last two bytes of R, and X the data that is to be protected by encryption.

PGP has become very popular indeed. We believe the main reason is that no infrastructure is needed to use PGP. In fact, PGP can be viewed primarily as a format and as software to support the reading and writing of this format. Because of this PGP can and must be used according to the users' needs. To make this point, we can contrast PGP to PEM [88].[4] In PEM, all messages must be signed; the standard dictates the usage of the tool.

[3] There are many more fields, such as version numbers; these have left out.

[4] We will return to PEM in Chapter 4.

The success of PGP is simply the lack of policy (and hence infrastructure). We believe this is an important issue, and will return to the practice of dictating policies several times throughout the dissertation.

2.5 Smartcards

In many settings where security is deemed necessary, in particular where authentication is required, smartcards are used to implement tamper resistant storage of secrets. It can be holding a security token used for authentication, possibly in combination with a password, or hold valuables in the form of call-time on a telephone or numbers of trips taken on a bus; it can be need to collect data, or store data that is to be kept secret (encryption keys). In the first two applications, modifications to the data must be controlled, in the last, access in general; the tamper-resistance property of the card enables both types of applications [104, 41, 61].

What is commonly called smartcards are governed by an international standard [73]. In technical terms, a smartcard is a Contact Integrated Circuit card (IC). Plastic cards with a magnetic strip have been with us for many years and they have had a strong influence on the size and shape of IC cards. At the time of their conception, the dominance of magnetic strip cards was so absolute that it was felt that all IC cards would have to carry a magnetic stripe also. Physical standards of such cards were embodied in ISO 7810 and ISO 7811 standards. These standards have been adopted within ISO/IEC 7816 and remain the most relevant standards appropriate to contact IC cards. The ISO/IEC 7816 standard comes in six parts, with two additional parts in progress.

Basically there are two aspects that make the smartcard interesting, in addition to their almost ubiquitous presence in society. First, the contemporary processors in such cards are general-purpose machines programmable in general purpose languages such as C or Java. The implication is that applications can be rapidly developed, debugged and deployed. In particular, cards supporting applications written in Java take advantage of the “sandbox” security model offered by the language runtime system [130].

Second, contemporary smartcards are manufactured in such a manner that the card is tamper resistant. Some successful attacks have been reported [12, 13], but manufacturers are also steadily improving the

cards We assert that although *tamper resistance* is technically correct, for all practical purposes, contemporary smartcards are *tamper proof*. Or, in other words, attacks on systems which employ smartcards will probably mainly be directed against other components than the card (such as protocols). An intermediate attack—attacking protocols running on the card—has also been reported [76].

On their applicability, it suffices to mention that as of this writing, cards are available with 64 Kb of memory available for applications and data, and the cards come with embedded processors to speed up modular arithmetic.

Following the authors of [116], employing smartcards implies that borders must be drawn between a larger than usual set of rôles. Among them we find:

Cardholder: The party who has day to day possession of the card.

Data owner: The party that own (controls) the data on the card. In the case of a pre-paid subscription card it will be the service provided.

Terminal: The device which provides the card with an interface to the "world". This might or might not be the service provider.

Card issuer: Having issued the card, the issuer most likely controls the operating system on board, and hence the possibility to control the applications running on the card.

In addition one could choose to consider also:

Card manufacturer: By implementing the hardware all sorts of attacks are possible.

Software manufactured: In many cases the card will run third-party software (such as Java).

Having all these rôles in the system makes many new attacks feasible. In particular the terminal is a crucial point in any design incorporating smartcards: we will return to this issue later.

2.6 Conclusion

Ubiquitous computing gives rise to ubiquitous encryption, which is probably good for privacy. However, encryption requires advanced software,

and is funded on advanced mathematics. It has been made quite clear that the security of encryption hinges on a large number of details.

Chapter 3

Reasoning about Security in Distributed Systems

In more or less all computer systems, users must identify themselves to the system before they are allowed to carry out operations (consume resources). Typically, they type a password, proving to the system who they are. After the user has thus logged in, it is up to the system to keep track of the originating user for the stream of commands processed by it.

In centralized systems, this is relatively straightforward; commands from different users arrive on different wires from different terminals. In distributed systems, this is much harder: a single network wire can carry commands from many different users. The machines at the far end of the network can (and do), of course, label commands with the name of the originating user, but remote machines cannot always be trusted.

The solution is usually to require that the commands arrive on an encrypted stream, and to assume that only the originating user (and the system processing the commands) has access to the key. This introduces several new problems. One is that keys and users must be matched up securely — mistakes result in a breach of security. Another is that human users cannot encrypt and decrypt data by themselves; they must trust a machine to do that for them. A third, more technical, problem is that it is unsafe and inefficient to use a user's identifying key to encrypt a command stream. In practice, the identifying key (and other keys) are used to derive a *session key* with which the command

stream is encrypted. This further complicates the key-to-user mapping that systems must do.

Unless explicitly stated otherwise, we will assume that possible attackers are *active*. This model of attack(er) was discussed by Needham and Schroeder [99] and has become the standard against which systems should be able to defend themselves (and their users); this model is often named "Dolev-Yao" after the first thorough analysis [40]. An active attacker is assumed not only to retain copies of messages that travel on the network. He is assumed to be a legitimate user who is able to establish a connection to any other user, and insert a message into any existing connection.

Broadly speaking, there are six different types of attacks that can be mounted against a protocol:

Bug: Any bug, either in software itself, in the application of (cryptographic) tools, or in other parts of the system can be exploited; the weakness in WEP is a recent example [47, 129].

Freshness: The attacker replaces a field with the same field from an earlier session.

Type: The attacker replaces a field with one of a different type.

Binding: The attacker uses ambiguities in the binding between keys and participants to substitute one key for another.

Parallel Session: The attacker replaces a field with one from a simultaneous and parallel session.

Oracle: Using a legitimate participant in exchange of messages, so that the computations performed by the user can be used to obtain useful information.

Any secure system must defend itself (and its users) against these lines of attack, and any tool to analyse systems should take all of them into account. Hard to detect errors will normally occur when a weakness in one area is exploited in another.

This chapter will discuss tools used to reason about security in distributed systems; it is the usage of these tools later in the text that makes them worth presenting. Section 3.1 discusses a family of modal logics used to investigate whether authentication protocols meet their

goals, and to understand the assumptions that underpin them. We will discuss three such logics, the BAN logic [27], the GNY logic [58], and the newer SVO logic [136, 131].

When reasoning about security in a distributed system, messages are assigned meaning, depending on their contents, which keys have been used (for which purpose) and so on. A theory of authentication and delegation is needed to describe the meaning of messages and the rôles participants have in a system at a given time. We review such a theory which is presented in Section 3.2.

At the end of this chapter, in Section 3.4, some issues are discussed that are related to infrastructure.

3.1 Logics for Authentication

Systems interact by sending messages. In order to be understood, messages must be part of a protocol, which can be viewed as a specification of the format and order of messages. A *cryptographic protocol* is a protocol that uses cryptographic machinery to establish certain properties of messages, such as integrity, secrecy and origin. These properties, in their turn, enable the recipients to draw conclusions on the *meaning* of the messages. A class of protocols called *authentication protocols* is particularly interesting. These protocols can, for example, be used to exchange a session key between two parties, to establish mutual authentication, maybe both these two at once, establishing the presence of a participant, or all three at the same time. These protocols are usually quite short, fewer than ten messages is normal [87][1]. However, it is surprisingly difficult to construct protocols that meet the goals the protocol was meant to meet [14, 3, 15]. One of the reasons is the difficulty of expressing precisely what the goals are. The remedy we will discuss here, is logics of authentication, with which protocols can be analyzed for correctness. No more than some core ideas are presented, not the logics in full [131, 27, 58, 136]. Surveys of formal approaches to protocol design and verification can be found in [91, 60, 30].

[1] An unpublished but continuously updated survey of authentication protocols is available as [33].

3.1.1 The BAN logic

The BAN logic is a logic for authentication protocols, and it enables an analysis of *beliefs*. The logic consists of a notation to capture interesting aspects of authentication protocols such as that some data is fresh, that a key is secret, and so on. For example, $P \overset{K}{\longleftrightarrow} Q$ denotes that K is a shared key between P and Q, and that the key is "good". What it constitutes for an encryption key to be "good" is outside the scope of an authentication protocol because it depends on the actual cipher, but it must normally be secret. What the term secret implies is a question of semantics, more about this later. The logic has 18 logical postulates detailing what beliefs can be derived from already held beliefs, in combination with messages that are received. For example: if P believes that K is a good key for communication between P and Q, and P sees X encrypted with K, then P should believe that Q has done the encryption; the interpretation (semantics) is that Q has once said X:

$$\frac{P \textbf{ believes } P \overset{K}{\longleftrightarrow} Q,\ P \textbf{ sees } \{X\}_K}{P \textbf{ believes } Q \textbf{ said } X}$$

It is here implicitly assumed that P, upon receiving the message $\langle\{X\}_K\rangle$ is able to deduce that K is the correct key to decrypt this message, and that he has not sent the message himself, in order for him to conclude the message was sent by Q. In order to verify that the message is legitimate, the message must contain redundant information [54].

The logic assumes one can state the relevant *beliefs* all participants have about the system (including the other participants). When a participant believes that a key can be used to secretly communicate with another participant, it must be *assumed* that the key is secret; the question is whether the other party knows about it or not. New beliefs are derived from the messages that are received, according to the interference rules of the logic. The goal of a protocol is usually to ensure that the participants obtain beliefs about the final state of the system (knowledge about secret keys, for example). Because the logic is concerned with authentication, it does not aim to reveal information leakage of any kind. Perfect encryption is assumed.

Most authentication protocols aim at getting in a situation where all (usually two) parties believe that the other parties are present, and that a session key has been distributed, and that all parties believes the key

is good. This is expressed in this way (for two parties P and Q):

$$\mathrm{P}\ \textbf{believes}\ \mathrm{P} \overset{\mathrm{K}}{\longleftrightarrow} \mathrm{Q} \quad \text{and} \quad \mathrm{P}\ \textbf{believes}\ \mathrm{Q}\ \textbf{believes}\ \mathrm{P} \overset{\mathrm{K}}{\longleftrightarrow} \mathrm{Q}$$
$$\mathrm{Q}\ \textbf{believes}\ \mathrm{P} \overset{\mathrm{K}}{\longleftrightarrow} \mathrm{Q} \quad \text{and} \quad \mathrm{Q}\ \textbf{believes}\ \mathrm{P}\ \textbf{believes}\ \mathrm{P} \overset{\mathrm{K}}{\longleftrightarrow} \mathrm{Q}$$

This summary has briefly mentioned two limitations of the logic. First, it is assumed in the postulates themselves that the participants are honest. This has created some concern: The following can be found in [5]:

> It might be quite appropriate to base an analysis in this logic on the initial assumption that participants believe other participants are honest, but it does not seem to be a good idea to accept honesty as a basic principle.

This is clearly visible in the *nonce-verification rule*, which says that *if* P believes that X is fresh, *and* that Q once have said X, *then* Q must have said X recently—in particular, after P came to believe that X is fresh—and if Q have recently said X, then Q still believes X:

$$\frac{\mathrm{P}\ \textbf{believes fresh(}X\textbf{)},\ \mathrm{P}\ \textbf{believes}\ \mathrm{Q}\ \textbf{said}\ \mathrm{X}}{\mathrm{P}\ \textbf{believes}\ \mathrm{Q}\ \textbf{believes}\ \mathrm{X}}$$

X might be a session key, and this rule enables P to believe the key to be good in that Q now also believes in the key. However, the sender need not believe in the contents of messages, and the logic thus requires honesty in order to analyse protocols. A remedy is presented in [5] where honesty is removed by reformulating the two rules which rely on it.

Second, there is no machinery to capture information leakage. Take for example the protocol in [100], where a good session key is sent in a message where it is signed rather than encrypted while at the same time, the author is able to construct a proof the protocol ensures that the participants gain confidence in the key; it is claimed to a flaw in the logic; the claim is rebutted in [28]. The argument is that because exposing a secret key violates the assumption that the key is secret, and any arguments based on this flawed assumption must, inevitably, lead to false assertions. The crux of the problem is that the assumption A **believes** $A \overset{\mathrm{K}}{\longleftrightarrow} \mathrm{B}$ is violated by A with the sending of a signed message that contains the (previously) secret key. When the analysis

assumes that a key is good (secret), it is then also assumed that the protocol will maintain the secrecy of the key. In other words, when the protocol is used to analyse secrecy, it is implicitly assumed that the goals hold before the analysis begins. This is so because information leakage can not be modeled in the logic. One can view this as a question about the semantics of the logic [132].

To sum up, an analysis with BAN is only meaningful when the following assumptions are made:

- All participants are trustworthy. In particular, they keep secrets.
- All encryption is "good".
- It can be verified that a message decrypted correctly; presumably by including redundant and identifiable material before encryption.
- Every participant is able to recognize messages sent by himself.

And the main drawbacks of BAN are then:

- There is no difference between "seeing" a message and understanding its contents.
- There is no way to alter beliefs.
- Trust can not be taken into consideration.

The shortcomings of the BAN logic lead us to consider another approaches which differ somewhat, the so called GNY logic.

3.1.2 The GNY logic

The GNY logic, as BAN it is named after its authors, is seen as a new approach within the same framework as BAN [58].

In contrast to BAN, GNY distinguishes between what one possesses from what one believes in. In particular, the contents of a message might imply different "beliefs" depending on context. That is, the contents of a message and the information implied by the reception of the message is treated separately. The construct of *message extension* enables a participant, having believed that a message is genuine, to choose whether to believe the senders' belief [58, sec. 6]. It is thus left

to the receiver to choose whether to trust the sender not to send messages it does not believe in. Also, in the GNY logic one can reason about messages that are transmitted in clear text as all messages are treated similarly. In general GNY contains fewer assumptions and it can be seen as a more general approach since it does not exclude as many types of protocols.

However, in order to obtain these favorable aspects, GNY is applied to a much lower level of abstraction than BAN. In fact, while BAN deals with the meaning of contents of messages, GNY deals with actual bit strings (and it is then inferred what the message contains). One consequence is that the number of rules increases from 18 to 50, and the proofs also increase in complexity. As an example, the second rule for interpretation of messages (**I2**) says that if, for a participant P, all of the following conditions hold: (1) P receives a formula consisting of X concatenated with S, encrypted with a public key and with a not-originated-here mark; (2) P possesses S and the corresponding private key; (3) P believes that his public key is his own; (4) P believes S is a suitable secret between himself and Q; (5) P believes that X concatenated with S is recognizable; (6) P believes that at least one of S, X, or $+K$ is fresh, then P is entitled to believe that (1) Q once conveyed the formula X concatenated with S; (2) Q once conveyed the formulate X concatenated with S and encrypted with the public key (3) Q possesses the public key:

$$\frac{\begin{array}{c}\mathsf{P}\ \textbf{sees}\ *\{\mathsf{X}, <\mathsf{S}>\}_{+\mathsf{K}}, \mathsf{P} \in (-\mathsf{K}, \mathsf{S}),\\ \mathsf{P}\ \textbf{believes}\ \mathsf{K}_{+\mathsf{K}}\mathsf{P}, \mathsf{P}\ \textbf{believes}\ \mathsf{P} \overset{\mathsf{S}}{\longleftrightarrow} \mathsf{Q},\\ \mathsf{P}\ \textbf{believes}\ \phi(\mathsf{X}, \mathsf{S}), \mathsf{P}\ \textbf{believes fresh(}\mathsf{X}, \mathsf{S}, +\mathsf{K}\textbf{)}\end{array}}{\begin{array}{c}\mathsf{P}\ \textbf{believes}\ \mathsf{Q}\ \textbf{said}\ (\mathsf{X}, <\mathsf{S}>), \mathsf{P}\ \textbf{believes}\ \mathsf{Q}\ \textbf{said}\ \{\mathsf{X}, <\mathsf{S}>\}_{+\mathsf{K}}\\ \mathsf{P}\ \textbf{believes}\ \mathsf{Q} \in +\mathsf{K}\end{array}}$$

The complexity of the axioms are forbidding; and it serves as a warning on the complexity that it has been shown that this particular rule contains a redundant premise (S need not be part of the second premise (that is in $\mathsf{P} \in (-\mathsf{K}, \mathsf{S})$), and both unsoundness and incompleteness have been found [112]. Because BAN is too high level and GNY is too complex, we will use a tool named SVO logic; to be presented next.

3.1.3 The Syverson–van Oorschot logic

The SVO logic, named after the authors, is based on the BAN family of logics [136, 131]: The BAN- [27], GNY- [58], AT- [5], and VO-[140] logics. The GNY and AT logics add constructs to overcome problems with BAN, while the VO logic adds rules to reason about key-agreement protocols. The SVO logic captures the essential features of the former, while not being so complex. In addition, the idealization step required in the BAN logic is made less of a burden. In particular, what to infer from a message (or component of a message) must be explicit.

The core idea is that a message can not be *comprehended* before it can be tied back to some other comprehensible message (such as one in clear text). For example, if a participant sees Y and Y is the hash of some other message X, the meaning of Y must be treated as unknown unless X is known. Only when X itself is at hand can the conclusion be drawn that Y is the hash of X. Beliefs, then, are essentially defined to be messages and components of messages that are comprehensible. SVO also separates the three different applications of public-key cryptography: Encrypting, signing and key-agreement (Diffie-Hellman). Regarding the concept of freshness, SVO makes it clear that datums need to be "genuinely fresh". If X is fresh, then so is $F(X + Y)$, but $F(X * 0 + Y)$ is not. When freshness is obtained by combining with fresh material, rather than the simple use of just including a fresh datum in a message, then the function must be considered as well.

In BAN, the first step of an analysis is to idealize the protocol. This step is error prone, mainly because it is difficult to separate the intention of the analyser from the beliefs the reception of a message should give rise to. For example assume the following message (from the Needham-Schroeder protocol [99]): $\langle\{N_b - 1\}_{K_{ab}}\rangle$. The reception of this message should make the receiver believe that the key K_{ab} is "good". However, it is not enough to receive a message that is interpreted to have this meaning as the receiver must also believe, preferably explicitly, that he has received the key in the first place in order to consider it to be a likely candidate. In BAN, such premises are implicit, in SVO they are not.

Protocol analysis consist of four sets of premises: First comes the initial ones regarding well known public keys, belief in the freshness of nonces (generated by this participant), understanding of the effect of verifying signatures, and so on. Assume a server S sends a message

to A containing a timestamp, the name of another participant B and a session key good between A and B:

$$S \rightarrow A : \{T_s, B, K_{ab}\}_{K_{as}}.$$

The first type of assumptions would be

$$A \textbf{ believes fresh(}T_s\textbf{)}$$

$$A \textbf{ believes } A \overset{K_{as}}{\longleftrightarrow} S$$

and so on.

Second are premises reflecting the receipt of messages in a protocol. In our example, this would be

$$A \textbf{ received } \{T_s, B, K_{ab}\}_{K_{as}}.$$

Third are premises reflecting what is comprehended by each participant of the messages he receives. In particular, random numbers such as keys and nonces must be assumed to be understood as such; an unknown (message)component is denoted '$*$'. In our small example:

$$A \textbf{ believes } A \textbf{ received } \{T_s, B, *\}_{K_{as}}$$

It is assumed here that time stamps have a known and verifiable format.

Last come the premises reflecting the interpretation that a receiver attaches to a received message:

$$A \textbf{ believes } (A \textbf{ received } \{T_s, B, *\}_{K_{as}}) \supset$$

$$A \textbf{ received } \{T_s, B, A \overset{K_{ab}}{\longleftrightarrow} B, \textbf{fresh(}K_{ab}\textbf{)}\}_{K_{as}}$$

Note that this is one possible interpretation that explicitly has been made clear. This is to a large extent the replacement of the idealization in the BAN logic.

From the premises we will derive various goals, such as proving that the messages exchange a session key, just as in the BAN logic. We will apply the SVO logic to a protocol later in the text.

3.2 Delegation

The File Repository (a distributed system to be described later) is used to store (and replicate) files. Access-control decisions are made based

on certificates. Authority to perform operations on files is usually delegated from the owner to his TDA, which again delegates some authority to FR and others. We need tools in order to reason about delegation and access control.

In order for the TDA to delegate authority over a file to FR, the TDA must *speak for* its owner. Furthermore, unless the TDA—again, ultimately the user—wants the receiver to have complete control over the files, e.g., giving them away as it were, this delegation must be limited in some way. For example, it can have a limited lifespan, or that delegation requires cooperation between several parties.

Delegation is performed by creating *certificates*. For example, the message $\langle [A, B, C, F, T]_C \rangle$ might be a certificate allowing A to hand a copy of the file F, which belongs to C, to B, before time T. It is obvious that the interpretation of messages will depend on the context; for example, how the identity of B is to be determined. Below is a brief overview of the theory presented in [85, 2], by means of a short example.

Assume a secret key is stored off-line, and used to make statements on some other key that is on-line; for example stating who owns the key. Also assume that a service is to be implemented to provide on-line verification of the statements previously made by the off-line key. If a client would like to have verified that a statement is (still) valid, what should the messages in the protocol contain to reach this goal? The client sends a request to the on-line server asking if a key is (still) valid. The on-line server searches its lists of revoked keys, and if the key in question is not found on any of the lists, the service creates a certificate stating this fact and sends it to the client. The challenge is to ensure that the messages "say" what we expect of them. Take the message sent from the on-line service: The message is to convey that the online service has not found any proof that the key has been revoked. That is, the online service states that the key in question is just as good alone as with the support of the online service.

Security precautions such as requiring an online verification can naturally be built into a system (hard coded as it were). But if one wants a flexible security regime all these certificates must be written in a language that can easily be parsed. Furthermore, it must be possible for the client to ensure that enough "evidence" has been gathered; to ensure that the requirements set forth by the certification authority has been met.

Following the example, we must start by asking how the off-line server requires that its certificates be validated by the online service. The participants are the off-line server S, the online server O, the user U and his key K_U. The server S states that K_U belongs to (speaks for) U, if O vouches for the statements K_U makes

$$S \textbf{ says } K_U \wedge O|K_U \Rightarrow U$$

The online service now makes the statement discussed above:

$$O|K_U \textbf{ says } K_U \Rightarrow O|K_U$$

which by the axioms of the theory makes it possible for the client to assert that the key K_U "speaks for" U. In addition, the client has gathered the evidence that he has obtained the necessary certificates, and verified the validity of the requested keys.

Although all proofs have been left out, it is evident even from this small example that this theory is aimed at a higher level of abstraction than the logics for authentication discussed above. In fact, the ability to authenticate is assumed in this theory, and there is no interest in the encoding of messages, not even encryption, except to indicate that encryption is used to provide channels on which messages appear.

3.3 Principals

A *principal* is an identifiable participant in a distributed system [85]. Often a principal will be represented by a public key, and uniquely named by a digital hash of the key or a number of bits from it; in PGP the "key identifier" of a public key is the 64 least significant bits of n. In the setting with private computing a principal will often be a human. In these cases, a range of means for identifying the person might be available; the identification will now be moved to the binding between a key and a uniquely representation of the human (such as a name). These binding are of interest in themselves, and many infrastructures have been set up to deal with them, see below.

How a participant is identified and upgraded to a principal will differ between systems. Usually the crucial decision will be normally be based on some (set of) assumptions. For example that a secret key is controlled by a principal, or that the secret key itself *is* the principal. A principal might also be a group, either by means of delegation or as a general construct[85, 134].

3.4 Infrastructure

The difficulty of distributing (secret) keys inherent to shared key encryption is replaced with the difficulty of authenticating (public) keys in public-key encryption. To be precise, public key cryptography can be used to turn a communication channel without any interesting properties into one which offers integrity, assuming that another channel which offers integrity already exists [110]. This second channel must exist in order to verify the public key. Without binding of some attribute (name, for example) to a public key, nothing of value can be concluded from the verification of a digital signature. Therefore, unless the user community is so small that manual verification and revocation is feasible, some infrastructure is required to support the use of public key encryption.

Such an infrastructure could amount to the following:

Finding a key: Sending an encrypted message to a user requires the sender to determine which key to use. Unless the sender is well acquainted with the receiver, this task can be daunting.

Locating a key: When a signed message is received, verification of the signature requires locating the proper (public) key. This problem is less hard than the one of finding a key because the key in this case can be uniquely determined.

Verification: In either case, it must be established wether the key (still) is valid. The semantics of validity will depend on the application.

In most cases the above difficulties are sought to be resolved by a Trusted Third Party (TTP), formerly denoted as Certification Authority (CA) [80]. In settings where the user community is very large, as in electronic commerce, TTPs seem to be indispensable [48]. In addition to the issues of consistency and performance, important issues related to privacy arise from such infrastructures [25]. The best known technology for building such infrastructures is the X.509 certificate framework [32]; it has evolved over the years and is at present available as X.509v3.

A more flexible approach is available through SPKI[43, 44].[2] SPKI specifies a syntax for specification of names, keys, certificates, and other

[2]At the time of writing, the structure of SPKI has been formalized in [44], while the accompanying representation is only available as an "Internet Draft". The draft is

objects necessary in an infrastructure for authentication and authorization. In addition there are rules for the merging of chains of certificates, and examples from a wide variety of settings where SPKI can be applied.

When building secure applications, designers are faced with constructing, sending, receiving, parsing and understanding messages, together with the profound issues of naming [98]. Messages usually contain encrypted material, nonces, time stamps and other non-textual material making both debugging and understanding difficult. Also, as will become evident later, human verification of certificates (or, more to the point, scrutiny of the data they contain) is at times necessary. This task is close to impossible without formatting and "pretty printing". To this end, SPKI specifies also a syntax that can be uniquely parsed. In this respect SPKI describes a language that informally can be described as "PostScript of security".

In particular, the SPKI draft specifies how to uniquely encode certificates (and other material) so that signatures can be generated and verified regardless of operating system and programming language, a format suitable for transport (based on "base64" from [23]), and a suggestion of how to present such material in a "human readable" form; the latter is named "advanced format" and will be used throughout this text.

As a small example, below is the advanced representation of a sequence consisting of a certificate, followed by a signature. The certificate says that the key with hash "`iCxd0R...`" asserts that the name "Test Name on Key" is associated with the key named as "subject". The signature consists of a hash of the certificate, followed by the "name" of the signing key (the same as is used in the certificate), followed by the signature.

```
(sequence
  (cert
    (issuer
      (name
        (hash md5 |iCxd0RAnBQA6z9GHhSMjaA==|)
        Test Name on Key))
    (subject
      (hash md5 |LY+G0R2DtB8u8hgEVOgoHg==|)))
  (signature
```

currently known as "draft-ietf-spki-cert-structure-06.txt"; we will refer to this draft as "the SPKI draft".

```
(hash md5 |9O715rOzoM5oOTZ2iPZIbQ==|)
(hash md5 |iCxdORAnBQA6z9GHhSMjaA==|)
|KzDO4BqnyCwHWKuB5DOjhoSAQKFpK9O7HST2
 cwilOAGCckgEuIsR8EGDP4IBshLr|))
```

It should be noted that SPKI is geared towards authentication and authorization, but the language is flexible and can be used to represent many types of objects. As a small example, below is the format of a public key encrypted message, as given in [6] with the corresponding implementation in SPKI. These two messages are identical to the one presented in Section 2.4 (page 30). The PGP public-key encrypted packet containing a shared-key is represented as the following SPKI message (notice the usage of the name of the key rather than the key itself; part of SPKI is to reduce the chain such certificates creates):

```
(PGP-public-key-enc-msg
  (version spki-pgp-v1.0)
  (public-key-encrypted shared-key
    (name Test Name on Key)
    (ciphertext |i56JOa4EiOj6Vv3DFwkomtGME.....|)))
```

This message would be followed by a conventionally encrypted data, encoded as follows:

```
(PGP-conventional-enc-msg
  (version spki-pgp-v1.0 )
  (encrypted
    (3des-cipher CBC |HIpoGgY/kgI=|
      |I5E5Dtk9K3NHhNglU7zlXGpRthFhjC6K+j...|)))
```

The conventionally encrypted packet must be transmitted together with the public-key encrypted encryption key.

3.5 Conclusion

In general, whether a protocol is secure or not, is undecidable. Even if only finite protocols are considered, and restrictions are placed on the generation of nonces and encryption keys, security is still undecidable [42]. The logics for authentication all make it quite clear that secrecy is a property that can not be analyzed; for a discussion see [100, 28, 133].

In order to discuss with precision the issues related to security in distributed systems, powerful tools are needed. There are logics to analyse whether authentication protocols meet their goals, and under which assumptions. Furthermore, after authentication has been done, the question of authorization remains. In order to determine what constitutes a user, a prerequisite to grant access, delegation to session keys and other types of keys must be possible. To analyse such settings, a theory of delegation is needed.

Chapter 4

The Open-End Argument

The Open-End Argument was presented in Tage Stabell-Kulø, Feico Dillema and Terje Fallmyr: *The Open-End Argument for Private Computing*, in Proceedings of the ACM First Symposium on Handheld, Ubiquitous Computing, October 1999 [124].

4.1 Introduction

We have designed and implemented a system around the use of PDAs; the system itself will be described in Chapter 5. During that endeavor we have come to realize the need for attention to which policies are set at the time of the design, and which policies are left for the user to decide. Users' ability to exercise control depends on whether the system is structured so that no important decision is taken without first consulting him. Moreover, important information for making decisions should be made available in spite of the engineering tradition of information hiding.

4.2 Open-End Argument

A traditionally layered system shields the user from the intrinsic details of the system, and does not require him to be competent to act according to the information the system might present about its internal state. Removing such requirements from the user is generally considered an advantage, in particular for making make the system user-friendly. Systems based on this model are designed to be transparent. The "trans-

parency design principle" states that users should be shielded as much as possible from the inner workings and state of the system they are using (see, for example [59]).

This principle is not only applied to end-systems, but also more generally to layers of abstraction inside the system. While this design principle is important for building abstractions and has been applied successfully to many system designs, its rigid application suffers from some serious problems.

The end-to-end argument [113] argues against placement of functional components and services at low levels of abstraction in a computer system, unless it is needed by all clients of that layer and if it can be completely implemented by that layer. In general, a layer cannot handle all types of errors, and will in some situations enter undersireable states, such as for instance by blocking or even fail by crashing. Such errors and exceptional events in one layer require intervention from the layer above, where the peer entities may utilize their position to resolve the situation below. This may propagate up through the layers until it reaches the topmost layer, which implements the interface towards the user. In the end, the user is confronted with a malfunctioning system.

With the advent of truly personal and mobile computing and communication devices, we believe the system and user model as described earlier are rendered ineffective as design goals, for at least three reasons:

– As discussed in Chapter 1 we believe that "real world" private assets will be stored in PDAs, owned and controlled by individual users. Such assets can be diaries, keys used for authentication and/or access control, and possibly electronic money. Users will want zealous control over such resources; before valuables are handed to the system, the system must be trusted. Trust and trust relations are crucial components in any secure system. Trust is a *feeling*, based on personal experience, social context and personal perception. It cannot be measured or quantified by the system in any way and is often difficult or impossible for a user to quantify (and specify to the system) *a priori*. Hence, it can not be used in algorithmic computations by the system for making decisions and evaluations. In essence, acknowledging the non-computability of trust in the personal sphere implies that systems designed with the traditional system and user model, cannot support the user's

perception of trust and cannot make decisions based on a user's trust and trust relationships.

– The user will view the system as providing him with a set of services for processing, storage, retrieval and communication of (private) data. Traditionally, the system decides for the user how such services treat the user's data. In addition, the system often is the only designated authority when it comes to making qualitative judgments (based on policies specified by the system maintainer/owner). This makes sense in traditional environments where no part of the system is really owned by the actual user and the user's data is not considered (at least by the owner of the system) as his private property. When systems are to process a user's truly private data, the user will want to be in control of the system rather than being "just a user", as the user is the only one capable of making qualitative judgments regarding his private data.

– The failure mode will not be uniform. PDAs often operate at the limits of what can be achieved by current technology and operational problems due to resource shortage must be considered part of normal operation. Because PDAs typically are powered by batteries, communicate over wireless links and may roam in hostile environments, several types of failure are likely to occur much more frequently than in non-PDA based systems. Almost all failures will occur in what is normally considered as "infrastructure", and users are not supposed to interact with problems at such a low level. However, when operating a PDA each and every user must be involved also in these issues in order to keep the system usable even when parts of the system fail as part of almost normal daily operation.

There is a common denominator in these three aspects. Distributed systems are designed as layers of abstraction, with peer entities communicating with each other over the network. The user, however, often has no peer in computer systems (in the "other end" so to speak) that can replace the user for making certain decisions. We have an *open-ended* setting when the users rôle is controlling rather than only using the system. Taken together, systems that encompass PDAs do not fit elegantly into the classical model of distributed systems design, where computation may be distributed, but where authority and competence

to make decisions is centralized.

Rigid application of the transparency design principle tends to structure the design in ways that causes two problems:

- A quest for transparency tends to push complexity down into the lower layers because more functionality is needed to handle ever more exceptions. As a consequence, hiding the state of the system from users may in itself result in increased system complexity. Increased complexity often leads to more complex failure modes of the system, which in effect may be increasingly difficult to hide from users, and so on. For example, NFS (Network File System [114]) tries to shield users from the fact that some files are remote. It provides location transparency, and users are presented with a uniform interface for local and remote files. However, an application that issues a write operation on what seems to be a local file can get an error message like "Remote Host Unavailable", or simply block (depending on implementation details). Maintaining transparency over such failures would require the addition in the system of significantly more (and more complex) machinery, like (write) caching and consistency control, and so on.

- Application of the transparency principle seems to structure the system such that the control users have over the system is reduced. This is especially true when it is combined with a rather pessimistic user model in which users are not assumed to be competent to make any decisions based on the state of the system. In such systems, problems that the system cannot handle transparently and which the typical user of the system is not assumed to be able to handle either, are left to be resolved by the more knowledgeable system manager or help-desk support personnel. The position of the user in the system can then be said to be on the "edges" of the system. When the user is placed at the edge of the system, avoiding the potential confusion caused by strange error messages in NFS (example above), is no requirement for the system design. However, proper handling of failures is of greater concern in PDA-based systems as argued above.

End-to-end and similar style arguments have led to the idea of "stupid operating systems", "stupid networks" and "stupid processors" [105]. We introduce the idea of "stupid PDAs" in order to discuss the structure of systems that support owners of such machines. A stupid PDA

is designed to be simple enough for its owner to feel confident he understands how the it operates (on his behalf!) and for him to feel able to control it. It keeps its owner well-informed about relevant aspects of its (internal) state at all times, rather than hiding it from him. The essence of what is known about the state of the system should be communicated. Only then is it possible for the user to understand what is going on, and intervene if necessary.

Private computing implies that the owner of a private computer is in full control of it. For the owner to be confident that he fully controls the activities of and access to this private computer, systems need to be explicitly designed for it. "User friendliness" achieved only by applying the transparency design principle is insufficient or even inappropriate for such systems. Hiding the true activities and state of the system from the owner inevitably result in reduced confidence on his ability to control his private machine.

Confidence is a human feeling and is therefore difficult or impossible to quantify by others. Like concepts such as "trust" and "competence" it is defined by a user's beliefs, experiences and personality (amongst others). These notions are inherently qualitative in nature and must be quantified in a meaningful way in order to be used in a computation. The notion of trust can serve as an example. Alice might trust Bob in some circumstances, but not in others. Furthermore, whether she trusts Bob depends on the credentials he conveys with his requests to. She may accept an authentication performed by a Kerberos server for access to her work-related documents, but require a completely different set of credentials for access to private documents. Alice makes a qualitative assessment of the information presented to her (e.g.: "Is this signature 'better' than the other, and do I find it 'sufficient' for this particular request?). Such assessments are well known in "the real world", where different credentials (i.e. papers) are needed for different purposes; a passport is needed in some situations, an ID card suffices in others. Different papers that "say" essentially the same thing, but with different quality. Again, it is the notion of private computing using PDAs that introduces such qualitative assessments into the system. This forces the designer to either ignore the user, or to ensure that the user can control the system.

Often a system is considered easy-to-use if it requires little *a priori* knowledge and little training from the user. Even though competent users cannot be assumed for many system designs, we may assume

that even ignorant users have the ability to learn. Unfortunately, many system designs hide much of their inner workings (by design), so that it is very difficult for a user to learn how to control and manage it better. We argue that especially in the context of private computing, a system that is not designed to let a users learn to control and manage it, can not be considered easy-to-use as users are prevented to learn how to deal with situations the system itself can not resolve transparently. In order to support user learning and user control, a system supporting private computing should provide the user with all the system information he needs and/or desires.

We believe that a different approach is required when designing systems to support private computing. In addition we argue that only the user can and should be the judge of what is important for him to control. This can not be left for the system to decide. We call this line of argument *the Open-end argument*:

Open-end argument: The system should be designed in such a way that in all situations where qualitative assessment of information is needed to make a decision, the user is consulted.

This argument guides a system designer in deciding when to follow the transparency design principle and when to violate it by informing and consulting the end-user.

4.3 Discussion

We will return to the open-end argument many times, and discussing it in a different setting demonstrates that it has applicability also outside the security domain. Here the discussion is related to consistency.

During a network partition it is impossible to know from within the minority partition, whether it is safe to proceed with a task that alters shared data [51]. In fact, it is even impossible to know whether a datum has become shared. The user, on the other hand, may be capable of understanding the situation and *evaluate* the risk, and, more importantly, understand the consequences of his actions. Also, the user might use channels outside the system to gather information for understanding and evaluation.

Consider the following scenario: Alice and Bob are sharing a file on a distributed file system; they are employed at different sites. On

one fine Sunday afternoon Alice decides to write, she contacts her local file server and requests a write-lock on the file. The response is that a copy of the file is available (after all, it is replicated) but the network is partitioned (between the sites of Alice and Bob) so that no guarantees regarding either consistency or correctness can be given by the system. Alice is left to make the decision how to proceed based on her extra-system knowledge, e.g. about the work habits of Bob and the amount of work involved in merging possible conflicts later. It is obvious that she has no guarantee from the system whatsoever, but *she* decides whether to proceed or not. She might use the telephone to establish a de facto lock by means of social engineering, for example, to reduce the risk of inconsistencies. By exploiting her understanding of the situation she is able to take the "correct" action even though seen from the systems' point of view, she has created two inconsistent copies of the file. The point is, the *system* did not force her to try to circumvent its services, but rather made it possible for her to make progress even when she was in a minority partition. She obtained progress at the risk of conflicting updates, but at her discretion.

This example demonstrates the tradeoffs that are possible in a particular setting. We believe the open-end argument provides guidance when design choices has to be made. In a later chapter we will describe a distributed storage infrastructure designed with the open-end argument.

4.4 Conclusion

A machine that is portable and trusted can be part of daily life, and its owner will soon come to depend on it. Successfully including such machines in a static infrastructure requires that the design acknowledges the special properties TDAs have, and the special way TDAs can, and probably will, be used.

We claim that systems structured in accordance with the open-end argument will be different from systems targeted at support for static systems only. Or, in other words, systems that aim at including TDAs should be designed with extra care. Decentralization of control, management and authority inevitably leads to new semantics of terms such as "conflicting updates" and "trusted".

Discussing this in a setting with concurrency control reveals that

an open-ended system shows different characteristics than systems designed using more traditional system design guidelines. It was shown that this also applies to security issues such as access control.

In the larger picture it has been argued that transparency is intrinsically woven into the end-to-end design principle. Unfortunately, security is ill served with transparency as a guiding principle, and private computing even more so. Seen from a security point of view, behind transparent (security) services all sorts of problems lurk. Not only functional annoyances—such as not being able to grant others access to one's own resources—but potentially more dangerous ones such as not being in control over in which manner private data is transported, kept, safeguarded, and so on.

We believe that systems that are designed with the open-end argument in mind will be better suited to give users the full benefit from the PDAs they have. It is, however, hard to prove that we are right, and we are faced with a methodological problem. As discussed in Section 1.2, the best we can hope for, is to design a system and use an implementation as an argument for the viability of the approach. To that end, the system we have designed is named File Repository FR; it is described in the next chapter.

Chapter 5

File Repository

The system described in this chapter, the File Repository (FR) has been presented on several different occasions. The primary reference is Tage Stabell-Kulø and Terje Fallmyr: *User controlled sharing in a variable connected distributed system*, in Proceedings of the seventh IEEE international Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE'98) [125], but also on several other occasions [46, 45, 68]. The ideas from FR are now being perused in PESTO [39].

The trend towards ubiquitous computing adds the requirement that users should be able to roam between many different environments. Independent of what administrative domain they roam into, they want access to their personal resources while using resources available in that domain. As a general rule, we can say that policies, mechanisms and resources are different in each environment and system, and a lot of machinery between systems is necessary in order to maintain and achieve important nonfunctional aspects like security and safety across systems. However, creating this machinery is often impossible because policies and available mechanisms in the different domains are too diverse. We believe that ubiquitous computing requires a platform for distributed systems and applications that makes no assumptions on the environment the user currently is in.

A reasonable interpretation of the open-end argument states that many of the nonfunctional aspects in a distributed system can only be reasonably quantified by the user (for example, who and what to trust). Qualitative assessment of information should be made by users only,

and the system should be designed to make this feasible for them to do.

Engineerings aspects aside, the challenge of private computing is to design a system that allows its users to span other systems in such a way that they retain ownership and control over their own information, without requiring administrative control over these systems. That is, we would like a system where individual users can enforce their policies on their data, even in environments that they do not own or controls.

We have applied the open-end argument while designing FR, resulting in a system that provides a novel set of mechanisms. Furthermore, the mechanisms that are offered, carry few guarantees. In particular, any guarantees that would include to override the user's decision are ruled out. Which guarantees that are in fact offered, and their implications, will be diuscssesd below.

5.1 Overview

FR's main design goal is to incorporate trust relations and ownership into a distributed system. At the same time, we do not want to add or require ownership and trust relationships of their own that have no real-world basis or justification. In essence, to achieve this goal we have separated trust relations from administrative relations. We will present a few important aspects of FR before we discuss the applicability of the open-end argument.

- FR has facilities that enables users to control their data and enforce their real-world ownership. The user is the only one who can specify security and safety policies *and* the only one who decides who is trusted to enforce these. A user can delegate authority to other nodes trusted by him, but he need not do so. If the user does not authorize some part of the system to speak on his behalf on a certain matter, FR will involve the user himself in the decision process to resolve questions concerning it.

- Experience shows that users are often unwilling to pay any non-trivial price for prevention of a risk, for as long as they have not been negatively affected by it [86]. This is well documented in the security and safety engineering literature as a very common cause for security breaches and calamities [8, 101]. We will simply assume that disasters *will* happen; nodes will fail, trust will

be violated, subtle mistakes will be made and accidents will occur. Rather that trying to avoid such threats, FR supports recovery by storing and replicating all authorized updates and by recording who authorized them. Notice that logging is not meant as a measure against threats, but to restore the integrity of the system after the security breach has been detected.

– Using FR, user Bob can share the content of his files with Alice, regardless of whether Alice is known to FR or not. Furthermore, he can delegate authority to Alice in the manner he feels appropriate; in particular: public-key cryptography is not a fundamental building block in the system. Furthermore, Bob decides who is trusted to enforce his access control policies.

– It is well known that write sharing of data in typical file systems is relatively rare, see e.g. [78]. FR assumes further that the more private data is, the less likely it will be shared between different users. However, we assume that data shared between different instances of *the same* user, e.g. at different locations, using different machines, or in different rôles, will be much more common.

  Sharing of data in a distributed system with unreliable communication requires some form of consistency control to resolve conflicting concurrent updates. Consistency can be enforced at the system-level or the application-level. A system-level implementation would require the user to trust a subset of the system to run a consistency control protocol on his behalf. We avoids such requirements by separating replication control from consistency control.

– We believe that a key to empowering users, is to separate storage and availability of *data* (encrypted content) from accessibility to *content*. To make the distinction clear, assume that Alice owns some storage space. She controls this resource and she decides who may make use of it. But when she grants Bob the right to use some of this storage, the *system* should not require of him that he makes the *content* available to her as well.

  Sometimes a user will have no other option however, than to trust another user with content in order to make use of his resources, even though there might not be a basis for such trust. For example, a user roaming into unknown territory carrying merely

his PDA may find himself lacking enough trusted resources to accomplish his task. In such settings, editing a file on an untrusted machine should not require trust in other infrastructures, like the network links between that machine and the user's machine at home. In addition, it should not put the privacy of any other content of the user at risk. FR allows a user to put his privacy at risk in order to make work progress, while minimizing the risk involved.

– Disconnected operation seems to remain a normal mode of operation for private machines, regardless of the ever growing network coverage. Also, low bandwidth communication (like GSM data) prevails. We acknowledges this, and FR supports both disconnected and semi-disconnected operation by applying asynchronous communication and computation throughout.

To sum up, the design of FR carefully separates the different tasks an infrastructure must support. Because of this, a user can place responsibility for the different tasks on different machines, possibly under a variety of administrative domains.

Following the open-end argument, is should be obvious that not all parts of a large, distributed system are equally trusted by all users (and probably not equally trustworthy either). To capture this and facilitate the implementation of these trust relations in the system, the design of FR offers the user to separate the system into the following three sets:

Trusted Storage Base (TSB): The set of servers a user trusts to store replicas of data (encrypted content). These servers are the storage service providers of the user. A user may define a different TSB for different sets of data.

These servers are trusted not to delete the files they store; any modification will be detectable by anyone who can obtain access to the content (unencrypted data).

Trusted Replicator Base (TRB): The set of servers that have been given authority over a part of the user's storage resources. The servers in a TRB are trusted to enforce a replication policy on behalf of the user. Each member of a TRB is responsible for the distribution of replicas to some subset of servers in a TSB. Together they make up a directed distribution graph with edges leaving only servers

in the TRB and ending in servers in the TSB. A user may define a different TRB for different sets of data.

These servers are trusted to run an algorithm to distribute (encrypted) data.

Trusted Access Base (TAB): The set of servers that have been given authority over a part of the user's *content*. The members of a TAB are trusted to enforce an access control policy on behalf of the user. A user may define a different TAB for different sets of data. Typically, TAB members are also members of the TRB.

These servers are trusted not to divulge the content in files.

We believe that by giving the user the ability to divide the infrastructure in these three sets, we have made it simpler for him to design an environment according to his preferences.

The separation in space offered by FR is matched with an separation in time to support disconnect operation:

Replication: The distribution of replicas is performed using an asynchronous request-response protocol. Request and response messages may be separated by a period of disconnected operation. A desirable side-effect is that email can be used to transport messages, using existing infrastructure.

Access Control: Authorization for access to content and storage can be acquired at a different time than their actual usage, allowing for work progress when disconnected from the relevant trusted access base.

The infrastructure is not responsible for consistency control. That task is separated from replication control and is left to the applications and the user himself. In addition, the infrastructure is assumed to offer best-effort service only. Monitoring and control of the quality of the actual service is left to the user and his (management) applications.

To sum up: FR keeps different responsibilities separate, such that the user can allocate them to different, but possibly overlapping, parts of the infrastructure. This design facilitates disconnected operation, as well as very fine grained access control. We believe this is in line with the open-end argument.

5.2 Related Work

The challenges of systems that contain small mobile nodes have been addressed in several other projects, and many systems address the problem of variable supply of resources in mobile computers. The duality of system and application control, captured in the term application aware adaption is addressed in projects like Odyssey [103], Rover [74] and Barwan [26]. Projects like INRIA's Project SOR [16] target at minimizing user annoyance when disconnected, and seek solutions in providing transparency to mobility by identifying and using locally available resources. In general, the projects tend to address problems related to system issues and how to deal with the situation when there is insufficient resources to continue in the current state. We are not aware of any project that addresses the issue of providing the user with proper information about internal state and the ability to fully control the mobile machine.

The persistent storage architecture in the OceanStore system [81] has similar goals and ambitions as FR, but relies solely on public key cryptography for update integrity checking at clients and is therefore less flexible. OceanStore does provide end-to-end security, does not trust the infrastructure with unencrypted content and does not rely on a trusted computing base for this. It does not, however, allow the user to specify what part of the infrastructure he trusts, and for what. OceanStore defines a class of servers that are trusted to carry out protocols on behalf of its users and defines the quality of its persistent storage service, where FR lets the user specify these.

The Bayou [137], Coda [78] and Ficus [63] distributed file systems target mobile clients in the system, use replication to improve availability while weakening consistency and introduce specialized conflict resolution schemes. They allow disconnected operation under the assumption that conflicting updates are rare, but do little to assist the user in avoiding conflicts like FR. In FR, it is assumed that conflicting updates of users relatively anonymous to each other will be rare, while conflicting updates of users that know each other well (e.g. workgroup members, or even the same person in different rôles at different locations) may be common but can be avoided or resolved at the user-level instead of at the system-level (e.g. by a phone call to your co-worker). These systems focus on efficient and transparent hoarding/caching for availability of data during disconnected operation. Hoarding techniques,

however, are not suitable for nodes with little amount of storage resources like PDAs. FR assumes that semi-disconnected operation will be more common than full disconnected operation, and primarily stores encryption keys on such nodes which facilitate use of untrusted computing resources in its environment.

5.2.1 Coda

It might seem as if FR has striking similarities with Coda [78, 77]. However, the design philosophies behind the two systems are almost antithetical, and this becomes visible when parts of the system fail.

The most important observation underpinning the design of Coda, is that conflicting updates on files are rare. That is, if a file is taken offline for a day or two, chances are overwhelmingly large that a modified version can be copied back in place without any conflict. In fact, the authors report that within a week, the probability that two different users modify the same file is less than 0.2% [77]. The smooth working of Coda relies on this being true. Coda is designed after a model where the "laptop" is in focus. Users have copies of their files on the laptop and bring the machine home in the afternoon. It is to be expected that files in a user's home directory remain unaltered. If, however, both copies of a file are updated, Coda does not resolve the conflict. In fact, because Coda does not have any knowledge about the files' contents it *can not* resolve the conflict. Instead, Coda generates an archive containing both files; it is left as a task for the user to decide what to do afterwards (how to merge the files).

In a system with private machines and where each user owns several machines, the assumption that conflicts do not occur will inevitably lead to failures. In fact, in FR it is *assumed* that conflicting updates will be common rather than the exception. In accordance with the open-end argument, FR strives to detect potential conflicts (and report these to the user) rather than hide potential risks (as Coda does). In fact, in [78] it is noted that *involuntary* disconnection caused by failures are no different than *voluntary* disconnections, but that users expectations and extent of user cooperation are likely to be different. There is a fundamental difference between Coda and FR in that FR will try its outmost to inform users about the state of the system while Coda does the opposite; only in the case of total failure is the user informed about the (lack of) progress.

Furthermore, Coda explicitly denotes the copy on a user's machine as second-class, while denoting the copy on the server as first-class. In FR, the open-end argument dictates that when *the user* owns the file, the copy the users chooses to denote as "first class", by definition it is just that. In our view, it is inappropriate to overrule a decision made by the user rather than trying to implement it. These differences between Coda and FR are explained by the different system and user models each has been designed for. We believe that the open-end argument, which underpins the design of FR gives a more flexible system, and one that better suits the need of users with TDAs.

5.2.2 Secure File System

The Secure File System (SFS) is a secure, distributed file system without centralized control [89]. Having a globally available file system as its goal, it does not make sense to assume global trust, globally consistent security policy, or a homogenous user community. One effect of such an approach is that one can not rely on any configuration being stored (and kept consistent) on clients.

The approach used in SFS is to embed in the name of files the location of the file, together with the public key of the server. Assuming common protocols, with the information available in a filename a user can obtain its content regardless of his location. Embedding the name and location of a file in the name relieves the user from maintaining local configuration.

For convinience SFS uses NFS [114] to obtain integration in the system where clients run. This works well and the performace is satisfactory, but at the cost of requiring a kernel-level implementation.

SFS and FR shares many design goals, and their functionality overlaps in some areas. But the simple model for locating a file offered by SFS inhibits replication. One of the main goals of FR was to support replication both for availability and reliability. In our view the tradeoffs in SFS have resulted in a well designed, but which is less versatile system than FR.

5.2.3 Authentication

As an extreme example of a service that is not user-centric, we can consider the design of Kerberos [127, 79]. Kerberos itself is an authen-

tication service, and a system might employ it for authentication of its users. The setting would normally be a traditional client-server one, but more esoteric applications can of course be envisioned. In a traditional setting, the system would require a so-called *ticket* in order to grant access to services. Unless you are a user acknowledged by Kerberos, it is impossible to obtain a ticket, and thus impossible to obtain access to resources. This also makes it impossible to delegate access to resources to users not known to Kerberos. That is, it is impossible to generate credentials outside the realm of Kerberos that will be valid within. Kerberos is in widespread use, and it is proper to note that it is naturally not a weakness in Kerberos itself, and we only use it as an example of a design philosophy; Kerberos is a large system (component), and one need to be aware of the setting in which Kerberos is intended to be applied [19].

Snowflake provides end-to-end naming and authorization across administrative boundaries [71]. It is not as "key-centric" as SPKI (see Section 3.4) and allows for other principals than public keys only (like local channels, shared key secure channels) as FR does.

5.2.4 Taos

When it comes to authentication, we have build FR on the same theoretical foundations as Taos [146]. In both systems, principal are the only source of authority, and authority is transferred by means of credentials made up by certificates. We believe this is a sound approach. But where Taos uses the Echo file system [20], and creates a centralized system out of distributed setting, FR uses a fully asynchronous model which suits the model of private machines much better.

5.3 Design

The overall design of FR is based on the concept that each user makes up his own separate administrative domain, based on his private computer. If a user wants his data to be replicated, the owner of some remote computer must delegate authority over storage resources. Authority is proven by credentials that authenticate a channel that speaks for the user. Requests that arrive on this channel will be honored within the rules then have agreed upon.

Because the design is based on the open-end principle, it has been a goal to ensure that no situation can arise where the user is unable to progress if he so desires. Obviously, if the system is partitioned altering data without having access to a quorum might cause inconsistensies.

5.3.1 Storage and Identification

Each version of a file is identified by a *globally unique identifier* (GUID) which is a random number (128 bits long in our current implementation). Due to the random selection from an immense name space, a new GUID can be generated locally with virtually no risk of an identical name existing anywhere in the system. It is even infeasible for a malicious user to correctly guess an existing name. Regarding the uniqueness of a randomly generated, 128 bit long name in a system, we refer to the discussion in Section 2.2 on shared-key chiphers.

In addition, using such a large flat name space for identification has the advantage that it obviates the need for a global name service because all names are assumed to be unique. There is no need for consistency control of this name space itself.

The different versions of a file are linked together into an file update history tree. Each version of a file keeps a reference to the previous version; its parent. When two updates are concurrent the two versions will have the same parent; a branch has been created in the version tree. How to deal with a brach is left to the user to decide.

This scheme also facilitates the recovery from unauthorized updates, and separation of replication from consistency control.

5.3.2 Communication

We have accepted up front that disconnected operations will occur, and hance very few assumptions are made about the underlying communication infrastructure. Transport agents are only assumed to be capable of implementing an unreliable, untrusted, best-effort transport services for messages over some transport channel. The communication infrastructure thus supports:

– Disconnected operation by separating creation and storage of a message and its transport, and by asynchronous communication.

- Separation in time of file version creation (storing information about it) and file version storage (storing its actual content).
- Failure analysis and recovery by storing all protocol command messages.
- Synchronization and consistency control by applications.

We believe this is sufficient to build a well functional distributed system. We also believe that no service that cannot be guaranteed has been promised, and no "strange" errors should be reported (such as the "host not found" error in NFS).

5.3.3 Policies

Separate access control to (unencrypted) content and access control to storage resources allows for the separation of content sharing and storage sharing. Access to storage resources at a node implies access to *encrypted* content at that node.

A user that has acquired the right to use resources at a remote node can delegate these rights to the nodes of the trusted replication bases he has specified. Actually, the user's task is merely to acquire the initial authorization from the remote node and to specify the replication policies for his files. The required delegations matching the distribution graph can then be derived from these policies and can be performed by the system without further involvement of the user. In addition, users can delegate storage authorization to third parties.

To sum up, before a user can send requests to a node it needs to require authority over storage at the node, and a channel that speaks for him. The channel is the result of a delegation, either from another channel or from some kind of service contract. Once use of resources has been authorized, a user can send requests to the node for processing.

5.3.4 Replication Policy

A replication policy specifies a set of replicas and a set of replicators. The set of replicas forms the trusted storage base (TSB) and the set of replicators forms the trusted replicator base (TRB) for the files subject to

this policy. The TRB is always a subset of the TSB for practical technical reasons (see below).

The replication policy is by defaultencrypted so that information about the members of the trusted replication base (TRB) and the trusted storage base (TSB) is only made available on a need-to-know basis. Nodes in the TRB are informed of their membership of the TRB and of the subset of nodes they are expected to distribute content to. Members of the TSB are granted merely the ability to check whether they are (still) member of this TSB. Hence, nodes can not derive the TSB and TRB from their view on the replication policy if not explicitly granted. Or, in other words, the size of the system and the full set of participating nodes need not be known by any other party than the user himself.

Although this in itself does not prohibit discovering such information by untrusted parties using for example traffic analysis, it may significantly increase the amount of (technical, legal) resources needed to complete such a task successfully. By limiting the number of vertices in the distribution graph appropriately, some degree of protection against traffic analysis can be obtained. More elaborate schemes are possible, if deemed necessary [53]. By making one part of the TSB public while keeping another secret, publication infrastructures can be build that are resilient to denial of service attacks, in the spirit of e.g. the "Eternity Service" proposed in [9].

5.3.5 Consistency

During a network partition it is impossible to know from within the minority partition, whether it is safe to proceed with a task that alters shared data [51]. Traditional, pessimistic consistency control algorithms strive to offer *single-copy* semantics, i.e. users see only a single copy at all times [84, 34]. Such strong consistency requirements are therefore impossible or very expensive to meet in our target environment, where full connectivity might never happen. The consequence of this is that merging of conflicting updates may be required upon reconnection.

Whether concurrent updates represent a conflict is application dependent; e.g. an append-only unordered log will never see conflicts. It is therefore our view that the system is not in a position to decide what represents a conflict and how potential conflicts should be handled. The user, on the other hand, may be able to *evaluate* the risk of

disconnected operation and understand the possible consequences of his actions. It should be *his* task to decide whether being able to make work progress now is worth the risk of conflict resolution work later. It is not the task of the storage system to make such an assessment, and it will therefore not deny a user service based on some built-in notion of how to preserve correctness for application content. The system's task is merely to distribute updates. It is left to the application to define a consistency policy which determines how replicas are presented to the user. This is fully in line with the open-end argument.

Applications can obtain the information necessary to determine the current state of any version of a file, such as the number of current replicas, number of updates to a file, who made these updates, and whether the object is shared.

FR allows an application to create an update, while delaying storing its content. This can be used to implement advisory locking schemes that work well even in a semi-partitioned network; the created update without its content functions as the advisory lock. This feature can also be used to keep track of activities in the system, that is logging, without having to store the actual data.

5.3.6 Access Control Policy

An access control policy specifies whom should be granted access rights, and specifies what credentials the user deems necessary and sufficient for such rights to be granted. All files under one policy is protected by the same encryption key. This key is only made available to the members of the trusted access base specified for the policy.

An update will be authorized if it arrives on a channel that speaks for a user who has authority over storage. Access control thus resolves to access control of secret cryptographic keys. Or, in other words, FR is built on traditional shared-key cryptography rather than public-key. The advantage is that a shared key can be generated from a password, giving the user a wider range of settings where FR can be applied.

Turning now to technicalities, we first discuss how a user can delegate authority to other users, and then we describe how a channel that speaks for the user is established.

Authorization

Authentication and delegation of authority is based on certificates encoded and interpreted according to the rules set forth by SPKI [43]. SPKI certificates tie a delegation to a (hash of a) key. Authority can usually be propagated, making up delegation chains.

The authority expressed in a certificate describes the resource by naming the access control policy governing it. Authority may be limited by explicitly listing the GUIDs of individual file versions from this set. Certificate chains are presented to members of the TAB which verify them and match the authority expressed in them against the ACL of the access control policy.

SPKI supports specification of restrictions (validity conditions) in certificates and ACL entries, including various on-line tests. In FR, an on-line tests can name another resource (file) by GUID. The on-line access decision for that resource forms than the validity condition, which can be used to implement several useful features:

- Group membership tests, where the membership list is the tested resource which may be under control of a different user than the owner of the ACL.
- Certificate Revocation List (CRL) tests, where the CRL does not need to be controlled by the certificate issuer (but for example by the owner of the ACL; the certificate verifier).
- On-line verification and confirmation by a user, by listing the user (his private computer) as resource to be tested. This is most useful for cases where a user decides to use an untrusted node for a task. Instead of denying the user service, the system forwards the request to his private computer which presents it to the user himself, such that he may confirm his decision securely to the system and make work progress.

Secure Authorized Channels

Each version of a file is encrypted with its own unique key; its *version key*. The set of version keys of a file, one for each of its versions, are themselves encrypted with a so-called *family key* which is unique for that file. An authorized update to a file is a file version encrypted with

a unique version key combined with this version key encrypted with the family key of the file.

The family key of a file is only made available to members of the trusted access base; family keys are encrypted with the key of the access control policy that applies to the file. In order for a user to construct a new authorized version of a file it needs a newly generated version key for it and this version key needs to be encrypted by the family key. As the family key is only available to the TAB for the file, he needs to make a request to a member of the TAB to encrypt the new version key with the family key for him. Access control agents thus grant read and update access to files, by granting access to existing and newly generated version keys.

Basically, the access control agent delegates authority to the version key, that can be used once. This version key acts as the already authorized channel for the update. This means that obtaining authorization for an update is separated from the actual injection of such an update into the storage system. This is of value for two reasons:

1. Checking credentials may be a relatively time consuming operation. Our scheme allows this work to start long before its result is needed for work progress, which may significantly decrease perceived latency.

2. A user may obtain authorization to update when connected and can use it later even if disconnected from the TAB (but connected to other nodes of the storage infrastructure).

A potential problem with this scheme is that users may request (many) such version keys and store them for future use; use them even after their access rights have been revoked! Limiting or prohibiting delayed use of authorizations would prevent this, but it would also reduce the advantages of the scheme enumerated above. A better solution would allow detection of such unauthorized updates such that they can subsequently be removed from the system by some form of garbage collector. This is facilitated by separating creation of a update from storing its data content.

Creation of a new file version involves then storing its GUID, version key (encrypted with the family key) and the credentials that were used to authorize it. Storage of the update involves then adding the data encrypted with the version key to this meta information about the new

version. The task of the TAB is to create updates it authorizes, while the user may later store its encrypted data. If the credentials used to authorize the creation of an update get revoked before its data is stored, the update can be rendered harmless by storing nothing as its data as part of the revocation process. This scheme has two additional merits:

1. It reduces the requirements on random number generation at the client side, as GUIDs and keys are generated by nodes trusted to enforce access control policies (i.e. TAB members).

2. It can be used as basis for concurrency control schemes (as described in section 5.3.5).

A node storing data for the user may not be trusted with the cryptographic keys needed to discriminate between authorized updates and unauthorized junk. Removal of unauthorized junk may be needed as part of a recovery process, i.e. after a node trusted to store and distribute updates got compromised and sent junk to other members of the TSB.

5.3.7 Local and Global Naming

At the system level content is identified by its GUID, and is located using its replication policy. The owner of a given file will always have authority over the replication policy and he can always locate a copy of a file. Users without access to the replication policy will need some form of location service. Typically, a location service will not be part of FR proper.

Building (persistent) publication infrastructures like Globe [141] on top of FR with (relatively) anonymous sharing of content, will require a more structural approach to location services.

5.4 Discussion

Our design builds on the notion of private computing, is based on the open-end argument and has a user-centric view in that all authority in the system originates from individual users. The access control and replication mechanisms do not mandate any hierarchical, fixed or static structure on administrative domains. This makes our system inherently

suitable for building personal ad-hoc infrastructures for sharing and cooperation between individuals, but it is certainly not limited to such. Individual administrative domains can be used as building block to construct larger domains using delegation. This requires cooperation from the individual users; a user can not force another to delegate any of his authority to him. Enforcing mandatory policies within the system would require all nodes to be under full control of a single authority. But even then, users may evade the mandated controls using channels outside the system. Extending the reach of the mandated policy to include such channels as telephone, paper notes and floppy-disks is infeasible in all environments but the most restricted ones (intelligence agencies, the military, criminal organizations and the like).

We believe that the lack of mandatory transfer of authority is not a weakness in our design, but reflects that in most real-world environments, user cooperation is a requirement to enforce a shared or 'global' policy. However, cooperation from non-malicious users may be 'bought' by making the use of shared resources conditional to such cooperation. For example, a company could define a storage policy for its file servers that allows only storage of files of employees for which it has been delegated the rights to define the TAB, TRB and/or the TSB. In addition, it could setup its communication infrastructure such that only servers under its control may be reached through it.

Even though we did not list it as a design goal, our design is well suited for the construction of infrastructures that are to a high degree resilient to (distributed) denial of service attacks similar to the "Eternity Service" [9] and "The XenoService" [147]. Denial of service is targeted at destroying a certain resource or at exhausting the resources of the service providers. Replication of content together with logging assists in preventing the former. Resilience to resource exhaustion attacks can be gained by protecting the use of the resources and/or by ensuring more resources are available than an attacker is able to consume. Our design supports resource protection by separating access to storage from access to content and its asynchronous protocol. Its graceful degradation during periods of semi or full disconnected operation limits the damage of a successful network denial of service attack. In combination with the ability to turn members of a TSB (kept secret untill under attack) into a member of the TAB with merely the exchange of a single cryptographic key, the amount of available resources can be increased dynamically to counter the resources consumed by the attacker.

From a security point of view, the ability to separate storage of data from control over content opens new possibilities. First and foremost it implies that safety and privacy can be addressed separately; the degree of replication can be increased to obtain better availability without having to consider the trustworthiness of new participants. Second, by leaving to the user community to decide what authentication credentials that are considered "good enough", FR places very few demands on the community; we regard that as an advantage. Third, by making almost no assumptions on the communication media, the design and construction of transport agents are relieved from any privacy concerns. We believe this cultivate good engineering.

5.5 Conclusions

FR's target to support private computing is responsible for most of its distinguishing properties: a flexible distributed storage system simple enough to support resource-poor devices. FR allows its users to specify what part of the infrastructure to trust and assumes all other parts are untrusted. It supports incorporation of the real-world personal and business trust relationships into the system, while not dictating or assuming such relationships in the design itself. Safety and recovery mechanisms are designed together with security mechanisms, providing ease of management and resilience against user error and violation of trust.

A dual-level encryption scheme makes read and update access control possible without relying on public-key cryptography. Public-key cryptography is supported for delegation and authorization. It is designed to perform well in networks that are semi-partitioned by supporting off-loading work to nearby, better-connected machines. Support for acquisition of authorization before actual use of it provides solid and secure support for disconnected operation.

The design of FR demonstrates the effects of applying the open-end argument in full.

Although the system has been implemented in several versions, it has mostly been used as a research vehicle. That is, whenever a new idea has surfaced, FR has been used as environment for a prototype. The result is that a variety of experiments have been performed on FR, and many aspects of it's design have been altered numerous of

times. The students that have contributed to this body of understanding includes (year of graduation shown in parentheses): Kjetil Kolin (M. Eng. 1995), Ingeborg Østrem Hellemo (M. Eng. 1996), Dag-Frode Olsen (M. Eng. 1996), Gaute Moxnes (M. Sc. 1997), Håkon Haugli (M. Eng. 1997), Ken Are Johnsen (M. Eng. 1997), Heidi Villmones Hundhåla (M. Eng. 1998), Ronny Håvard Arild (M. Sc. 1998), André Risnes (M. Eng. 2000), Christine Lund (M. Eng. 2000), and Geir Egil Myre (M. Sc. 2001).

Chapter 6

Intent and ability

The section on removal of information is based on Tage Stabell-Kulø: *Security and Log Structured File Systems*, printed in ACM Operating System Review, April 1997 [121], while the section on stability of beliefs is based on Tage Stabell-Kulø, Arne Helme and Gianluca Dini: *Detecting Key-Dependencies*, published in Proceedings of the Third Australasian Conference on Information Security and Privacy (ACISP'98), July 1998 [126].

This chapter is focused on the difference between the will to do something, and ability to do so; the will and ability to take necessary security precautions in particular. The rather long Section 6.2, is concerned with the removal of private information. We discuss deleting information stored in files in Section 6.2.1, and closing communication channels in Section 6.2.2. Concluding remarks can be found in 6.3.

6.1 Introduction

To be worthy of trust, one has to be both honest, and be able to do what one is trusted to do. The latter might be harder than the former. As an example of the difference between will and ability, consider signing a document. When the document is presented on paper, and the signature is expected to be written with a pen, the setting and modes of failure are simple to understand, and it is reasonable to expect that a signature is trustworthy. It is unlikely that any attempt to disavow a signature on technical grounds will succeed; a statement such as "I wrote

10 but the pen changed it to 20" seems silly. It is also worth noting that if a signature is valid, it has been written by the "owner"; a valid signature created by someone else is a contradiction in terms. However, in a setting where programmable devices and public-key encryption makes up the infrastructure (rather than pen and paper), assuming a division between will and ability is far from silly [65]. For starters, one can ask what constitutes the document: The file that contains it, or the rendering of the document on a screen. In addition, it is no longer the case that a valid signature must have been created by the "owner", as the signature is based on knowledge rather than ability.

Understanding whan can go wrong can be a difficult task, see for example [120, 8, 7].

6.2 Closing a Session

In principle, any message that flows through a communication network can be recorded by eavesdroppers. Recording a message implies that the contents of the message can be revealed at any later time, even after both the sender and the intended receiver of the message have destroyed their copy. The contents of a message cease to exist when no copy of the message exists in the system. It is obvious that two communicating partners are unable to enforce that the messages exchanged between them cease to exist. If, however, the messages were encrypted, discarding the encryption key will efficiently place the content of the messages out of reach, which is most cases will be sufficient. If the key is discarded by all participants, we can regard the session as properly closed.

Distribution of session keys among communication partners is a task that is accomplished using authentication protocols. A closer look at such protocols reveals, not surprisingly, that many are constructed in such a way that the session key is conveyed to the parties by means of messages. If the session key has been sent in a message, encrypted using some long-term key, then the session key does not cease to exist before the long-term key is destroyed. The term *dependency* will be used to describe the relationship that comes into existence between keys, when one key is encrypted by another, e.g., when the session key is encrypted by a long-term key. The effect of key-dependency is that the long-term secrecy of the session depends on the secrecy of the

long-term key. It also influences the quality of the session key. The longer a long-term key is in use, the higher the risk of compromise, and the session key is exposed to the same risk through the dependency. When a key-dependency arises from a protocol the assumption that a key *is* secret is transformed into an assumption that the key will *remain* secret. Thus, the protocol alters the assumptions, or, the way by which the assumptions are used, alters them.

Consider the Needham-Schroeder protocol outlined below:

$$\begin{array}{lllll} \text{Message 1} & A \rightarrow S & : & A, B \\ \text{Message 2} & S \rightarrow A & : & \{K_{AB}, B, \{K_{AB}, A\}_{K_{BS}}\}_{K_{AS}} \\ \text{Message 3} & A \rightarrow B & : & \{K_{AB}, A\}_{K_{BS}}, \{A, N\}_{K_{AB}} \\ \text{Message 4} & B \rightarrow A & : & \{N+1\}_{K_{AB}} \end{array}$$

The protocol description is slightly simplified, see [99] for more details.

In the protocol, the session key K_{AB} is sent in messages, encrypted with both K_{AS} and K_{BS}, in Messages 2 and 3, respectively. When, as here, a short-term key (K_{AB}) is encrypted with a long-term key (in fact two keys, both K_{AS} and K_{BS}), a dependency is created between the short- and long-term keys. The implication is that the session based on K_{AB} is not properly *closed* before all the three keys K_{AB}, K_{AS} and K_{BS} have been discarded. The secrecy of the session does not solely depend on the session key (which are under the control of the clients) but also on the long-term secrecy of the keys K_{AS} and K_{BS}. The long-term privacy of A and B thus rests on the honesty of S as the protocol is in progress (e.g., S discards K_{AB} as soon as Message 2 has been sent) *and* the management of S after the protocol is terminated. Or, in other words, in addition to the traditional assumptions made on the quality of the cryptographic tools, assumtions must also be made about future events (the management of S).

6.2.1 Removing a file

If a session is kept open (in the sense discussed above) because the key is stored in a file at one of the endpoints, the session can not be considered closed before the file has been deleted. However, properly deleting a file might be quite hard.

One of the main objectives of a file system is to hide details of physical storage, and users are not (in general) concerned with which physical disk block that actually was used to store data. However, if the user

wants to firmly delete a file he must ensure that the physical disk blocks on which the file is stored, are properly overwritten. It is well known that deleting the name of the file does not remove the contents of the file. On some systems, MS-DOS (systems derived from it) in particular, files can simply be "undeleted" and the entire contents recovered. On more sophisticated systems, such as UNIX, this is not the case, and more effort is required in order to recover data from a deleted file. In both systems the chances of success decrease with time, as measured in operations that modify the file system; the details are left out for brevity.

All this is well known, and quite some effort has gone into designing software helping users to (try to) discard their private data before the files is deleted. In particular, it is also well known that overwriting the original data with zeroes is insufficient in systems that provide compression as a means to increase the effective size of disks, see, for example [29]. This is so because a series of zeroes will usually compress better than the original data it was supposed to overwrite. The new version of the file is thus smaller than the old one, and (the end of) the original data is left unaltered on the storage medium when the file is deleted. Subsystems such as compression are usually designed with great care and with compatibility in mind, and their presence can not reliably be detected by applications. Hence, the utmost care must always be taken. To this end, programs such as PGP (see Section 2.4) write random data when overwriting, based on the fact that random data is not (significantly) compressible.

Most file systems, among them are notably the UNIX FFS [90] and MS-DOS, will reuse the physical blocks originally assigned to a file. It has been shown that a substantial performance gain can be achieved by using a log structured file system (LFS) [111]. LFS obtains its performance by lazy garbage collection and by avoiding seeking for disk blocks. The key issue is that when data is written to a file it is always written to a physical location on the disk that optimizes the performance of the file system. The functionality of reclaiming old disk blocks is performed when needed.

In LFS, it is not possible to overwrite data in a file because new disk blocks are always allocated. Although no file system API guarantees that the same physical disk blocks will be used, on many systems this can be assumed. On LFS, the opposite is true.

From a situation where the user could quite safely assume that the

original data would be erased by overwriting, the assumption on LFS is that the data almost certainly remains on disk (although outside the file system proper). This change is important, but can not be noticed by users that do not possess considerable insight into the workings of their system; most users are unaware of how their file system is implemented. Furthermore, this change is not detectable by tools the user might rely on, and a change in systems software can significantly alter the usefulness of users' tools to protect their privacy. Users in systems with LFS are particularly vulnerable because they are unable to remove data from their file system.

Which file system is employed in a computer is only one of countless many implementation issues relating to security. The most important aspect is that security to a large extent becomes dependent on implementation. Which, again, implies that the gap widens between the will to do "the right thing" and the ability to actually do so.

Backup

If encryption keys are stored in files, and backup have been made, properly deleting the key becomes even harder. Related to backup, special care must also be taken regarding public keys and their secret counterpart, especially when RSA and similar systems are used. In RSA, the secret key has two purposes: To decrypt messages encrypted by the public key, and to create digital signatures. That is, the same key is used both as a decryption key and signature key. The problem is that there is good reason to back up the decryption key, for example in order to inspect security logs at a later point in time. But there does not seem to be good reasons to back up the signature key without taking precautions to safeguarding it.

In our view, backup and similar system-related security threats are best dealt with by making the TCB as small as possible. To that end, we have designed into FR a strict separation of the TCB of encryption keys and data, making it possible to have different TCB on a per file basis (see Chapter 5 for details). Basically, this is an argument for the usefulness of a TDA.

6.2.2 Analyzing dependencies

We consider it harmful that sessions can not be closed solely by the participants. In fact, we believe each and every participant should be able to close the sessions in which he participates. Under the assumption that all participants are honest, protocols should have the property that any session closes properly.

We have developed methods to analyze protocols with this in mind. The central idea is to model the problem of locating key-dependencies between keys as determining the connectivity of a directed graph. More precisely, a graph describing a key distribution protocol contains nodes representing either data or transformations. Edges leading to datum nodes represent dependencies while edges leading to transformation nodes represent input to the transformation. The graph can be inspected in order to detect key-dependencies that render sessions open.

Modeling key-dependencies as edges of a graph is closely related to the methods described in [57], where a graph is built to detect the weakest (shortest) path between passwords that can be guessed (or text that can be verified) and a session key. Here, a similar approach is used to detect key-dependency properties in authentication and key-distribution protocols.

The set of nodes in the key-dependency graph $\mathcal{G}$ is defined as follows:

N1. Graph $\mathcal{G}$ has one node for each message, for each message component, and for each key necessary to decrypt the message. For instance, if message $m = \langle x, y \rangle$ is considered, then $\mathcal{G}$ contains nodes for m, x and y. Moreover, if a conventional cryptosystem and the message $m = msg\{x, y\}_k$ are considered, then $\mathcal{G}$ contains one node for the message m itself, one node for each message component (i.e., one for x and one for y), and one node for the key k. Similarly, if a public-key cryptosystem and the message $m = \langle\{x, y\}_k\rangle$ are considered, then graph $\mathcal{G}$ contains one node for each message component, x, y, one node for the message m itself, and one node for the private key k^{-1} (the decryption key corresponding to the public encryption key k).

N2. The graph $\mathcal{G}$ has one node for the computation that a principal has to perform in order to obtain the key (or other material) on the material received through messages, in its clear-text form, or lo-

cal information.[1] Moreover, the graph contains one node for each argument of the computation and one for the result. For instance, if the computation $x = f(x_1, \ldots, x_n)$ is considered, then graph $\mathcal{G}$ contains one node for f, one node for x and one node for each $x_i, i = 1, \ldots, n$.

Notice that by **N1**, if a graph is built from two messages containing the same datum, e.g., the messages $m_1 = \langle a, x\rangle, m_2 = \langle b, x\rangle$, the resulting graph will have five nodes (m_1, m_2, a, b, x) as x is one datum transmitted twice.

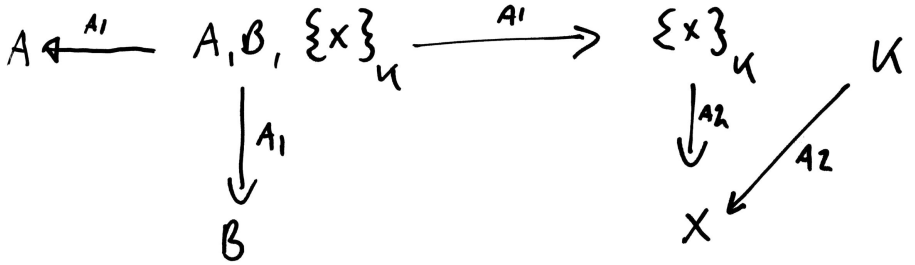
The set of arcs in $\mathcal{G}$ is defined as follows

A1. Let m be a message with n components, $m = \langle m_1, \ldots, m_n\rangle$. Then, graph $\mathcal{G}$ contains one arc from m to each $m_i, i = 1, \ldots, n$.

A2. Shared key encryption $m = \{x\}_k$ is characterized by a pair of arcs, the one from m to x and the other from k to x. Public-key encryption can be characterized similarly: if $m = \{x\}_k$ is considered, then graph $\mathcal{G}$ contains one arc from m to x and one arc from k^{-1} to x.

A3. If a computation $x = f(x_1, x_2, \ldots, x_n)$ is considered, then graph $\mathcal{G}$ contains a set of arcs as follows: one arc from every x_i to f, $(i = 1, \ldots, n)$, and one arc from f to x.

For instance, the message $\langle A, B, \{X\}_K\rangle$, where K is a shared key, yields the following graph. Each edge is labeled with the rule that applies to it.



After the graph has been constructed according to the rules **N1–N2** and **A1–A3**, it is reduced using the following rules.

[1] This computation is of course different from the computation that a principal has to perform in order to build up a message.

R1. For each distinct path from any node corresponding to a long-term key to the session key, mark all nodes on the path.

R2. Remove all unmarked nodes.

The result is that information about key-dependencies is transformed to edges in the graph.

6.2.3 Examples

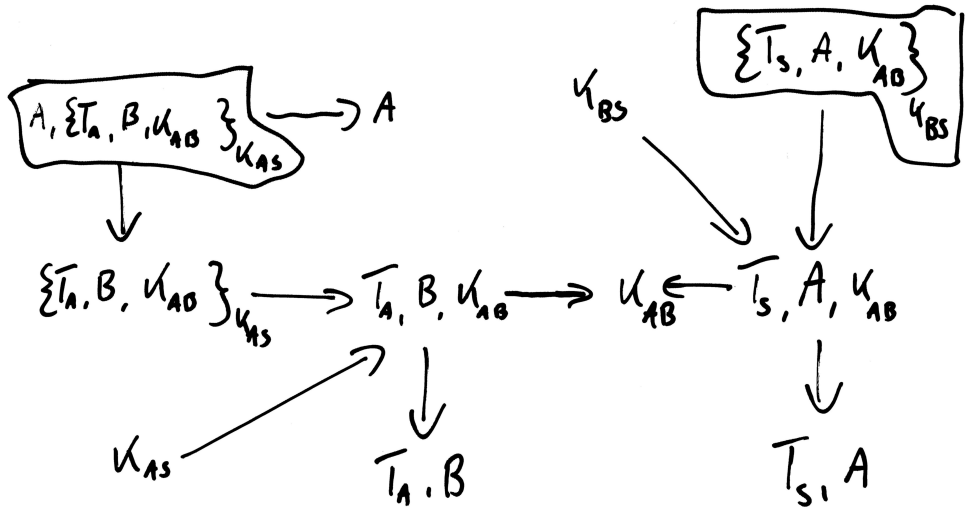
In this section five well-known protocols are analyzed by means of the method described in the previous section. As will be shown, these protocols give rise to a varying degrees of key-dependencies.

Wide-Mouthed-Frog Protocol

First the Wide-Mouthed-Frog protocol [27], a relatively simple protocol which involves three parties. In this protocol, the two parties A and B each have a secret key, shared with the authentication server S, K_{AS} and K_{BS} respectively. This protocol consists of only two messages.

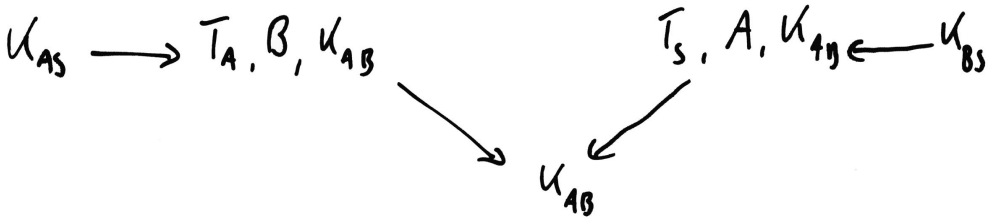
$$\begin{array}{lll} \text{Message 1} & A \rightarrow S & : \; A, \{T_A, B, K_{AB}\}_{K_{AS}} \\ \text{Message 2} & S \rightarrow B & : \; \{T_S, A, K_{AB}\}_{K_{BS}} \end{array}$$

When following the procedure outlined above, the following graph is obtained:



The two messages that were sent have been framed for clarity. In addition, each edge is labeled according to the rule that applies to it.

The long-term keys are K_{AS} and K_{BS}. Applying the rules **R1–R2** yields the following graph:



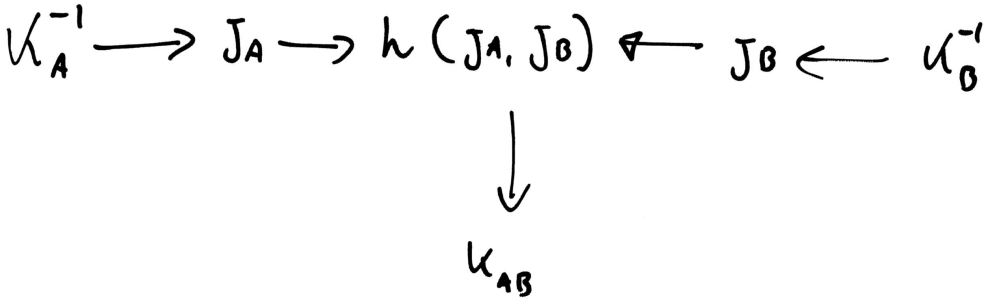
The graph shows that key K_{AB} depends on either one of two other keys, K_{AS} and K_{BS}. Thus, knowing *either* of the latter two will make it possible to recover K_{AB}, provided that the attacker has a recording of the protocol and the session. Consequently, in order to close a session based on K_{AB}, both K_{AS} and K_{BS} must be discarded. However, both keys are known to S, which implies that A and B do not control when the session will be closed.

Node-to-node channel

Consider the protocol to set up a node-to-node channel between the two nodes A and B in a distributed system [85]. The essence is that both A and B invent a random number, the numbers are exchanged, and the session key is constructed as a function of them. A and B are assumed to have public keys K_A and K_B, known to the other party, and both are competent to invent good random numbers. The protocol is slightly simplified, see [85] for a complete description.

$$\begin{array}{lll} \text{Message 1} & A \rightarrow B & : \{J_A\}_{K_B} \\ \text{Message 2} & B \rightarrow A & : \{J_B\}_{K_A} \end{array}$$

The session key K_{AB} is then found as a hash of J_A and J_B. Building the graph, removing the nodes that are marked public, and let $h()$ represent the hash function, produces the following graph:



When inspecting this graph, it becomes clear that the session key depends on two values, which again depend on two different keys. In particular, a single edge, not two as in the Wide-Mouthed-Frog protocol, leads to the session key. In order to find K_{AB} one must find *both* J's and they each depend on different keys. Thus, to decrypt the session protected by K_{AB} both K_A^{-1} and K_B^{-1} (assuming both J's are discarded) need to be compromised.

It can be concluded that the node-to-node channel protocol achieves a better result than the Wide-Mouthed-Frog protocol. The main reason is the way public-key cryptography is used. The public key in a public-private key-pair gives rise to a one-way channel leading to the holder of the private key. The one-way property implies that if A sends a datum to B through the channel represented by B's public key, and A dutifully discards the datum, there is no way to regain the datum without B's participation. This is used in the protocol by sending parts of the key on the unrelated channels. It can also be seen as a separation of the issues of authentication and conveying a secret.

SSL 3.0

SSL is a protocol designed to be used in a variety of circumstances and with a variety of security environments, and with a variety of cryptographic tools. This protocol is widely used, in particular by Web-browsers. SSL can be used in settings where both the client and server have public keys and mutual authentication is desired. With some simplifications (for example, only one method for hashing), the protocol

can be described as follows:

$$
\begin{array}{lll}
\text{Message 1} & C \rightarrow S & : \; C, N_C, T_C \\
\text{Message 2} & S \rightarrow C & : \; N_S, T_S, K_S, \{N_C\}_{K_S^{-1}} \\
\text{Message 3} & C \rightarrow S & : \; K_C, \{P\}_{K_S}, \{H(M + H(Z + M))\}_{K_C^{-1}}, \\
 & & \quad H(M + H(Y_C + M)) \\
\text{Message 4} & S \rightarrow C & : \; H(M + H(Y_S + M))
\end{array}
$$

In the protocol description, T_S and T_C are the time stamps, N_S and N_C are 28-byte nonces, and K_S and K_C are the public keys of the server and client, respectively. The keys are sent together with X.509 certificates making claims on the user-key binding [32]. P is the 46 bytes called "pre-master-secret", the function H is MD5 [107], M is the master-secret derived from the pre-master-secret by combining the pre-master-secret with N_C and N_S plus some padding, and hashing the result. Z is the concatenation of Message 1 and Message 2, Y_C is the concatenation of Z and the number 1129074260, Y_S is the concatenation of Z, Message 3 and the number 1397904978. In essence, the parties sign each others nonces.

When processed according to the graph reduction rules, the following is obtained:

Inspection of the reduced graph reveals that the secrecy of the master secret depends solely on the secrecy of K_S^{-1}, which again implies that the client is unable to close the session based on the master secret. Although SSL is based upon public-key cryptography, its behavior with respect to key dependencies is weaker that the Wide-Mouthed-Frog protocol. In the latter, the "users" have the possibility to close the session by changing the key they share with the server. In SSL, this is not possible.

At this point it it warrant to point out an essential detail: It is true that a malicious server can hold on to session keys forever. This is captured by the fact that the client must trust the server, and that the server is part of the client's TCB. However, trusting the server *now* has important implications in that when a connection has been set up by means of SSL, the server has to be trusted also in the future.

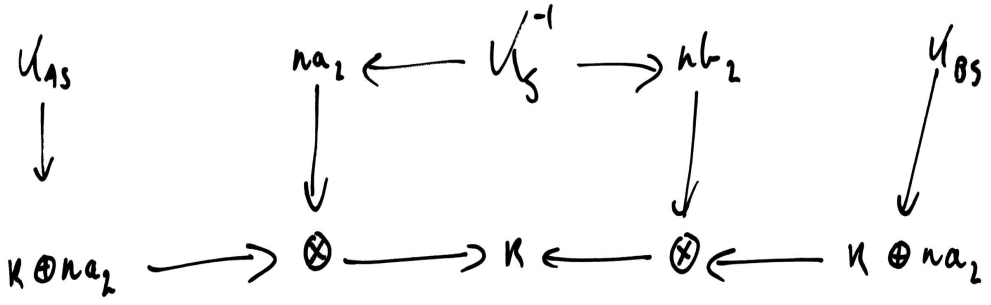
Demonstration Protocol

In [57], quite a few protocols are described, and in the following, the *Demonstration Protocol* is studied in more detail. It consists of eight messages sent between two principals A and B and a security server S. The last three messages form an exchange of nonces for verification, and are left out of the protocol description:

$$\begin{array}{lrcl}
\text{Message 1} & A \rightarrow S & : & \{A, B, na_1, na_2, \{ta\}_{K_{AS}}\}_{K_S} \\
\text{Message 2} & S \rightarrow B & : & A, B \\
\text{Message 3} & B \rightarrow S & : & \{B, A, nb_1, nb_2, \{tb\}_{K_{BS}}\}_{K_S} \\
\text{Message 4} & S \rightarrow A & : & \{na_1, k \oplus na_2\}_{K_{AS}} \\
\text{Message 5} & S \rightarrow B & : & \{nb_1, k \oplus nb_2\}_{K_{BS}}
\end{array}$$

The symbol $\oplus$ denotes the bit-wise exclusive-or operation, the datums prefixed by n's are nonces, the key K_S is the public key of S, the keys K_{AS} and K_{BS} are shared between A (and B) and S, and k is the session key. The protocol is slightly simplified—confounders are left out—see [57] for the details. In the graph, the nodes denoted with $\otimes$ represent a computation as required by **N2**. In this protocol, the computation is in fact bit-wise exclusive-or, but regarding it as a general computation does not alter the graph.

The algorithm produces the following graph:



Observe that there are two edges leading to k, indicating that k depends on two sets of keys. However, the secret key K_S^{-1} is a member of both sets. The outcome from the node-to-node protocol is better in this respect. Furthermore, all endpoints leading to k in the graph (K_S, K_{AS} and K_{BS}) are known to S and neither are discarded after Message 5 has been sent (after which S no longer takes part in the session). On the other hand, compromise based on K_A (or K_B) alone is not enough.

Compared to the Wide-Mouthed-Frog protocol, the outcome is better because compromise of one of the user's key is not enough to endanger the privacy of the session. The outcome is better than that of SSL, in that if A and B both change the key they share with S, the session is closed, while in SSL the session key depends on K_S^{-1} alone.

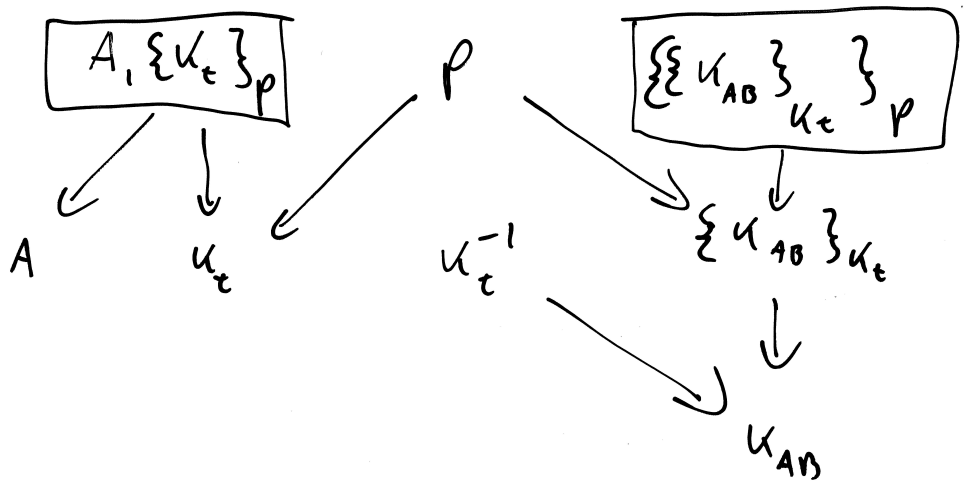
Encrypted Key Exchange

From the previous examples, it is clear that key-dependencies arise when session keys are encrypted with long-term keys. Using a fresh, temporary public key avoids the key-dependency issue. As an example, the Encrypted Key Exchange [18] is described.

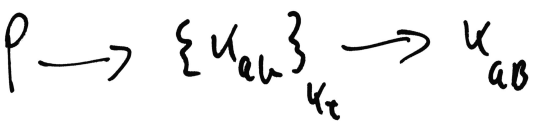
Let A and B be the two parties, P a shared secret, K_t a public key with K_t^{-1} as the secret counterpart, and K_{AB} a session key. The protocol consists of five messages, of which the last three are for mutual verification of the key; they are left out. The first two messages are:

$$\begin{array}{lll} \text{Message 1} & A \rightarrow B & : \; A, \{K_t\}_P \\ \text{Message 2} & B \rightarrow A & : \; \{\{K_{ab}\}_{K_t}\}_P \end{array}$$

The protocol gives rise to the following graph:



Reduction results in the following graph:



First, notice that the secret, temporary, key is not included in the graph because it is not a long-term key. Second, inspection of the graph reveals that holding key P is not sufficient to obtain the K_{AB} because it comes to depend also on the key K_t^{-1}, which is temporary. The graph, in its reduced form, captures this fact by depicting a path from P to K_{AB} which contains encrypted material whose decryption key is not depicted. In other words, the graph captures the essence in the protocol, that shared and public key encryption complement each other.

6.2.4 Discussion

The analysis of five protocols reveals a clear relationship between the use of shared-key encryption and key-dependencies. Without a pre-arranged secret channel it is hard for participants to verify that a session key indeed is correct [56]. This becomes evident in protocols based on shared keys, where the session key *must* be exchanged over the shared channel as there is no other alternative. In such settings, a key-dependency is inevitable. This can be argued for as follows: Assume two peer principals wanting to communicate, and exchange a session key, by the means of a security server. Based on the messages sent, B must decide on the same session key as A. This is only possible if A can assume that B's actions are deterministically based on the input (the messages sent by A and S). If C knows the algorithms that B follows, and can read the channel to and from B, C will be able to achieve the same result as B. Thus, it is impossible to avoid key-dependency in a system solely depending on shared key encryption. The above analysis verifies this.

It can be argued that a dishonest participant can hang on to a session key forever, thus preventing any channel from closing properly. This is true, but this scenario violates the assumption that the participants are honest.

By considering how dependencies arise it becomes evident that to improve the situation with shared keys there must not exist an unencrypted path in the graph from long-term keys to the session key(s). That is, a cryptographic channel must exist that is not transmitted. Today, public key cryptography is the most common choice for such channels, but more exotic possibilities are possible (see for example [94]). However, as became evident in the analysis of SSL, protocols using public keys also gives rise to key-dependencies if not applied with care.

In systems based on public key cryptography, and where a trusted third party is used to ease authentication, one can separate the issues of authentication from the exchange of a session key. In particular, one can leave it to the users to handle the latter. This approach, for more or less this reason, is taken in [85, Footnote 10] (and in [146]). When the server is not engaged in issuing the session key, no key-dependency arises on keys known to the server. It is, however, regarded as good engineering practice to involve a server in this process, see [27] and [3, example 11.2]. With the case of TDAs with few resources, this practice may well come into fashion again, especially if smartcards are used as TDAs.

Whether a protocol is secure or not is not only undecidable (see page 35 for a discussion), but obviously depends also on how "security" is defined [64]. Dependency between keys has not been seen as a problem in theoretical models of security, hence can not be detected.

Although the protocols analyzed in this section were not designed with *forward secrecy* in mind [38, 62, 21]. it is still important to point out that key-dependency vulnerabilities do exist in them. The design of SSL, for example, is considered sound for authentication purposes, but as shown in this section, can be vulnerable to attacks exploiting key-dependencies.

Dependencies do not enlarge the TCP. However, protocols that give rise to dependencies alter the semantics of the rôles the principals have in the TCB. It is particularly troublesome that the protocol alters an assumption about the world as it is (that a principal *is* honest) into an assumption about the future (that the principal will *remain* honest). This is very different from *beliefs* about the secrecy of a key. A key-exchange protocol can be poorly designed (or implemented) and reveal secrets, the protocol will make an assumption invalid (and thus effect everything that depends on it being true). Seen in the light of BAN and GNY, the findings in this section can be viewed as yet another aspect of the semantics of the logics [5, 133]. The usual goal of semantics is to make it possible to see the logic as part of a system, and statements proven to be true in the logic has some (more or less) well defined aspect of truth in the real system. In fact, having only an intuitive understanding of the semantics directly opposes the usual goal of having semantics at all [132].

6.3 Conclusion

This chapter has demonstrated engineering problems (deleting a file), and some problems related to design (dependency). Interesting enough *per se*, but they are merely pieces of a larger picture. In creating a secure system, the design and engineering must match. Designing a complex system does no good if it is impossible to realize the system without making horrendous assumptions; as shown even assuming that the contents of a file can be properly deleted is an assumption one should not make without careful consideration.

Furthermore, protocols themselves alters underlying assumptions (from *being* secret to *remaining* secret). This finding could have been predicted based on the discussion of the logics for analyzing such protocols in Chapter 3. Both assumptions and predicates are stable, in that once they have become true they remain true.

Taken together, the two main results in this chapter points towards a TCB which is as small as possible. Having as the TCB only a TDA would, for example, render void the problem of deleting files. The next section will show just how one can integrate a TDA into a distributed system to achieve better security.

Chapter 7

Offline Delegation

This chapter is based on a published paper: Arne Helme and Tage Stabell-Kulø: *Offline Delegation*, in Proceedings of the Usenix Security Symposium, August 1999 [69]. The idea has been presented on several occasions [66, 67, 68].

When private machines are incorporated into distributed systems, private control over private data becomes an issue which should be solved. Rather than solely protecting centralized resources from unauthorized access, protecting the interests of the individual also becomes necessary. In particular, users will want to decide whom can access their data, and when. Or, in other words, they want to be the sole authority over authorization for access to his data. As described in Section 5.3.6, FR has machinery built in that enables the user to authorize anyone by providing him with the correct encryption key. In this case, the user has taken it upon himself to authenticate any access; this would then be a question of trust. In this chapter we will stude a slightly more complex setting: one where the user has instructed FR not to divulge any file without requesting proper credentials. Assume furthermore that the user is off-line, but has his TDA at hand: How can he convey credentials in this situation?

The basis for our interest in this particular problem, is that armed with a TDA, each and every user can challenge the centralized model of authentication and access control by taking control of his own resources. In essence, a TDA provides the user with the possibility of creating his own TCB; as should be possible, according to the open-end argument. FR is in no position to deny the user the freedom of delegat-

ing authority whenever he desires.

This chapter examines a problem that is rather specific to the blend of mobile computing and security: how to handle the circumstances when there is no connectivity between user (and his TDA on one side) and the infrastructure on the other, but the user nevertheless have a desire to make security-related decision or action. In particular, how a certificate can be conveyed when (electronic) communication can not be established.

7.1 Introduction

The setting is one with FR, which does not require that those requesting access to files are registered as users in any way. Credentials should make FR believe that the file owner (or someone speaking for him) is the principal requesting access. An integrated part of FR's design is that authority over files can be delegated freely. Generally speaking, a certificate names a file and a user, together with an access right (read, write or both), and a time of expire. The syntax and semantics of certificates will be discussed in detail below.

Assume that Alice and Bob are having a conversation over the phone. Alice decides to grant Bob access to a file of hers. The problem is, how can she convey a certificate to him as part of the conversation without compromising her own security. One solution is that Alice creates a delegation certificate off-line and conveys it orally to Bob. Bob later presents the certificate to FR when requesting access to the file in question. If FR deems that Alice created the certificate and it is (still) valid, access will be granted.

Making off-line delegation possible requires that two distinct problems are solved. First, it must be possible to generate a valid certificate by means of a TDA without having access to FR. Second, it must be possible to convey the certificate orally. The latter rules out every binary representation of certificates in so far that it is unlikely that anyone will be able or willing to read hundreds of digits over the phone. Similar arguments also rules out the use of digital signature schemes relying on very long signatures (RSA, for example, with 1024–2048 bits in the signature) The problem is denoted "off-line delegation of access rights" (initially described in [66]).

7.2 Design requirements

Design of an off-line delegation mechanism must carefully balance trade-offs concerning convenience and availability against security. This section explores the design space, and in particular requirements for the delegation certificates.

A solution must fulfill a few overall design criteria:

1. A delegation (i.e. a certificate) should not enable any principal to impersonate the delegator or delegatee.

2. The credentials must form valid and meaningful access rights. In particular, all objects (principals, machines and files) must be unambiguously named.

3. Certificates must be secure in the sense that they should not pose a security risk; the cryptographic tools that are used should have sufficient strength. Hence, the authority granted by a certificate should not be transferable, and a certificate should only be valid once.

To ease the task of orally transferring access rights the following strategy is used. In essence, of all the information in a typical certificate structure only the digital signature constitute binary data. Because delegation certificates contain a wealth of information that easily can be exchanged orally, such as dates, file names, domain names, and so on, the digital signature is only a fraction of the data that has to be exchanged. In this scenario, the key issue is to simplify and ease the exchange of the signature bits.

Consider again the scenario involving Alice and Bob. She tells him which file it is, where it is located, the access rights she wants to delegate, together with the certificates' creation and expire times. On her TDA, Alice builds the certificate and signs it with her private key. She then reads the actual signature bits to Bob. Bob is now in possession of sufficient information to reconstruct the certificate on his TDA.

Later, Bob can submit the certificate to the file server together with credentials verifying his own identity. Upon receiving the certificate the file server will verify that it is signed by Alice, that it is Bob who is issuing the request, that the delegated access rights do not violate any overall file server policy, and, if all this is true, send the reply back to

Bob. Notice that we are explicitly not discussing the case where communication can be established with FR as many other solutions exists in that case.

In order to delegate access rights based on electronically generated certificates, the amount of data that needs to be exchanged is critical. For example, it is awkward to convey numbers in the order of 1024–2048 bits as part of a conversation between humans. When using public-key cryptography, it is essential that the data to be exchanged can be minimized as much as possible. A challenge is thus to use a digital signature scheme that has small signatures. Decreasing the signature length means to either use smaller key components or to use a cryptosystem with a denser key space. We have opted for the latter, and chosen a cryptosystem based on elliptic curves.

In addition to the design requirements mentioned above, it must be ensured that each certificate that is generated is unique. Rather than having a serial number in a certificate, it is an assumption that it takes more than one second to generate a certificate. Thus, with a one second granularity on time stamps, including the time of creation in the certificate will ensure uniqueness among certificates from a particular user.

In Chapter 4 we claimed that the open-end argument is applicable in settings where a TDA is "included". The design of a distributed (file) system such as FR is a typical example. The file system should take into consideration that users (with TDAs) will want to make access-control decisions over their resources—files in the case of FR—also when it is inconvenient for FR.

7.3 Certificate creation

Basically, the owner creates a digital signature on data that vouches for his delegation of access rights for a particular file to another user. This signature, together with the contents of a matching certificate, gives the delegatee access to the file. Thus, the certificate components that easily can be conveyed orally are the following:

- The names of the file, the owner and the requester,
- Time of creation and expire of the certificate (where the time stamp by assumption is unique),

– Specification of delegated access rights,

The oral delegation, thus, ends with the exchange of the signature bits that, together with the other information exchanged, enables the requester to (re)construct a machine readable representation of the delegation certificate. These certificates, obviously, must be readable not only by computers but also by humans. To facilitate this, certificates are encoded in the advanced format from SPKI [108].

In short, to access a file the requester signs the delegation certificate and transmits the result to the file server.

$$\begin{array}{lll} \text{Message 1} & A \rightarrow B: & [A, B, F, AC, T, S]_{K_A} \qquad (= X) \\ \text{Message 2} & B \rightarrow S: & [B, S, X]_{K_B} \\ \text{Message 3} & S \rightarrow B: & [S, B, \mathrm{data}]_{K_S} \end{array}$$

In the protocol description shown above, A and B represent the users Alice and Bob, S is the file server, F is the name of the file in question, AC is the delegated access rights, and T is the time stamp. K_A^{-1} is the private key belonging to A. Message 1 (or X, in short) is the delegation certificate. The field $data$ represents the effect of the invoked operation—typically the result of a file operation request—and is the file data returned to the file requester. The protocol does not distinguish between conventional (networked) and oral transfer of the first message—in both cases the same information is presented to the server in Message 2.

7.4 Implementation details

The off-line delegation mechanism is hosted on the 3Com Palm Pilot PDA and on UNIX workstations. In addition to FR itself, the implementation consists of a library for cryptographic functions, a library to parse and generate SPKI-like objects, a library that implements PGP, and a graphical user interface running on the TDA.

We have made an prototype implementation of off-line delegation The interesting part of an implementation is whether an a prototype, on contemporary hardware, will perform well enough for the concept to be useful. It is, in practice, very hard to estimate the time it will take to run complex algorithms such as the creation of digital signatures. In providing an implementation, our aim has been to establish that the

solution does not introduce an intolerable delay, or overly complicated procedures for the user. As a contemporary PDA, we have chosen the Palm Pilot.

In addition to necessary modifications to FR itself, the implementation consists of a new library for cryptographic functions, a new library to parse and generate SPKI-like objects (to be discussed below) and a graphical user interface running on the TDA. In addition we have written software to obtain the full functionality of PGP in the form of a library. This library was subsequently ported to the PDA.[1]

All keys, protocol messages and delegation certificates are encoded as S-expressions using the formats described in the SPKI documentation [108]. As an example, here is the first message in the protocol. It contains the delegation certificate and is encoded in SPKI as shown in Figure 7.1.[2] It is worth noting that the SPKI encoding shown in the figure is the format we choose (see [69] for details), and not the more modern (and standardized) format found in [44] (see Section 3.4 on page 46 for a discussion).

The delegation certificate contains a 256 bits Nyberg-Rueppel signature which is the only piece of information in the certificate that cannot easily be transferred orally; notice that the `Object-Hash` can be generated at the receiving side as well as on the sending side.

The crypto library provides functions to create digital signatures on data and to operating system specific functions to handle cryptographic data/contexts in different operating system environments. The elliptic-curve crypto implementation is based on the algorithms for fast operations in finite fields described in [145]. The implementation has been tuned to fit the limited processor and memory resources on the Palm Pilot platform. Currently, it uses a finite field of order $GF(136)$ with fast operations on element in a smaller subfield of order $GF(8)$. The order of the fixed point on the curve is a 241 bit prime number. This yields Nyberg-Rueppel scheme digital signatures with a total length of 256 bits for both the r and s components of the signature.

The bandwidth of a conversation is low. To that end, the core of an off-line delegation system is thus the dual ability to generate a valid certificate on a TDA and making it "readable" in a form which is simple to both "transmit", and "receive".

[1] This port was performed by Per Harald Myrvang, as part of his M.Sc. thesis.

[2] Several years after our implementation effort, two RFCs were issued that standardized the format of certificates. The language is now known as SPKI [43, 44].

```
( FR-Offline.Message1:
  ( Protocol-version: FR--Offline-v1.0 )
  ( Offline-Delegation-Cert:
    ( EC-Signature:
      ( Algorithm: ECNR-with-SHA1 )
      ( Signing-Key: |B3ZbHXKyBwQ=| )
      ( Object:
        ( Delegate-From: Arne Helme )
        ( Delegate-To: Tage Stabel-Kulo )
        ( File: /etc/passwd )
        ( Expire: 1999-01-08T14.33.12.000+0000 ) )
      ( Object-Hash: |0w3m2YI6xThw9yO6fHyvWg==| )
      ( Galois-Field: #88# )
      ( R: |1ebYMZ4iVpxQEc8VOfaZKQ==| )
      ( S: |RbQ5uLLfd9+MbnqPphQl/A==| )
) ) )
```

Figure 7.1: Encoding of a message in SPKI.

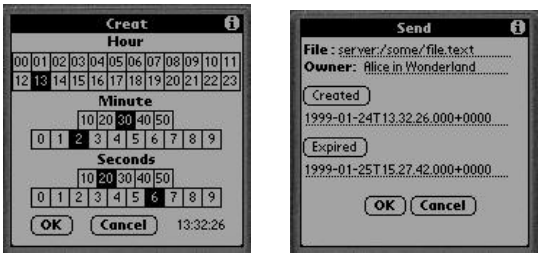
In general, making a certificate consists of first entering into the application the necessary information, and then generate it. The following information is needed:

Filename: It is the users' sole responsibility to ensure that filenames are unique within the scope of a server; the name of the file is prefixed by the name of a server.

Delegatee: As in all systems where users are represented by their public key, names must be unique within a system. Or, more precisely, the name appearing in user-key certificates must be unique.

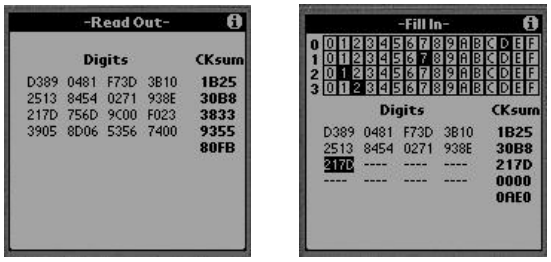
Expire: Certificates should expire. In order to ease the process of entering date and time, calendars are used to find a date (with today suggested as default). The granularity in the time stamp is one second; as described earlier this still ensures uniqueness. Figure 7.4 shows how the time is entered into the application; the user taps the boxes with the stylus. There is an similar form for entering the date.

Figure 7.4 are screen shots of the applications running on the PalmPilot. The images shows how general information is entered into the



(a) Setting the time (b) Information

Figure 7.2: Elements from the user interface: forms that are identical on both the sending and receiving side.



(a) Sending (b) Receiving

Figure 7.3: Sending and receiving a certificate

application.

The idea is that while Alice enters information into the fields she talks to Bob; Bob enters the same information. The software has been designed to be used this way, that is, the order in which information is entered when certificates are created is matched on the receiving side. By going over the fields together, they build up the certificate. When she has entered information, Alice will sign the data with her private key, generating a certificate.

As explained above, the signature is a string of 256 bits. On the sending side, the bits are presented to Alice as 16 4-digit hexadecimal numbers. Screen shots of the application are shown in Figure 7.3(a). Notice the checksum given in the right hand column; it facilitates a crude error detection scheme. She can now read them to Bob, one group at the time.

When Bob receives the numbers, he needs a means to enter them as fast as Alice reads. To facilitate this, a dedicated form is presented.

By tapping on the screen of the TDA he is able to enter data ("receive" as it were). The design is such that users can receive fast enough for the system to be usable. See Figure 7.3(b) for a signature that has been partly received.

The checksum is automatically calculated as data is entered. Although only a proper verification of the signature can determine whether it was properly transferred, the checksum is used to give Alice and Bob some confidence.

FR was augmented with the UNIX version of the crypto library, and the elliptic-curve public keys were added to the access control lists. The format we choose, shown in Figure 7.1 is ad-hoc, because no standard has yet emerged on the representation of elliptic curve parameters in SPKI.

7.5 Conclusions

The single most intriguing issue with the concept of TDAs is the possibility for each and every user to have a TCB which does not include resources controlled by others. Keys and credentials can safely be stored in a TDA. Private computers that are trusted can act on their owners' behalf in conducting secure delegation of access rights. In the scenario just described, FR is included in the TCB only when the file itself is in question (because it is stored in FR). In particular, when it comes to the usual usual question of binding a public key to a human and making signatures, the TCB consists only of the TDA.

Each delegation certificate issued is assumed to be unique. Uniqueness is guaranteed by numbering and the included time stamp provided by the user and by enforcing that at most one certificate can be generated per second. The server uses uniqueness information when updating the revocation list of a file to determine whether the same delegation certificate has already been used.

Time stamps are used as an additional source of information for revocation purposes. Because individual users can specify access policies, the correctness of the time-stamp encoded into each delegation certificate depends entirely on this user's ability to determine the current time. However, time stamps are only used to catch up with old certificates. In any case, using time to discard once-only delegation certificates is not entirely without risks (for a discussion, see, for example,

[55]).

Granting users the right to delegate access to their own resources implies losing control over what constitutes a user. When the owner of a file can grant access to whatever method of authentication he desires, it is impossible for the system to guard against poor passwords, for example.

The goal has been to show, in practical terms, which changes the inclusion of TDAs implies. In this case, no traditional concept of users are longer possible, and a scheme based on requests appearing on channels must be designed and implemented.

Chapter 8

Smartcard versus PDA

This chapter is based on two published papers. The method for secure signing with smartcards has been published as Tage Stabell-Kulø, Ronny Arild and Per Harald Myrvang: *Providing authentication to messages signed with a smart card in hostile environment*, in Proceedings of the USENIX workshop on smartcard technology, May 1999 [123] and Tage Stabell-Kulø: *Putting smartcards to use in a user-centric system*, in Proceedings of the second ACM Symposium on Handheld, Ubiquitous Computing, September 2000 [122]. Norwegian patent no. 20004206 has been granted on the novel method of augmenting smartcards.

This chapter is focused on what constitutes a TDA; the main question is whether a smartcard can be used as a TDA, or if more powerful (and usually larger) machines are necessary.

We consider the least we can require from a TDA: The creation of digital signatures in an untrusted environment; a machine that can not assist in signing data is merely a memory prosthesis, and not a TDA as we envision it.

First, a method to create signatures by the smallest TDA possible (the smartcard) is presented and discussed. It is argued that signing is not a question of processing power, but rather of infrastructure. The majority of such an infrastructure has been designed and build; the implementation is discussed.

Our second approach is to augment smartcards with a display and a keyboard, thus making the user able to safely communicate with the card, that is, in effect making the smartcard into a more "traditional" PDA. The problem is to construct a smartcard that conforms to the

specifications (e.g., is still a smartcard) while holding a keyboard. A suggestion of how this can be done is then presented.

8.1 Digital Signatures in Untrusted Environments

In order to be trusted to be able to make trustworthy digital signatures, a smartcard must be supported by some infrastructure outside the card proper. Unlike many other classes of hardware, smartcards do not have the ability to communicate securely with the user. Deprived of means to keep the owner informed, the positive properties of smartcards are difficult to utilize. In our view, to be useful as an extension of the users' private sphere, a machine must at least have enough functionality and resources to create trustworthy digital signatures (to speak for the user, as it were). A less resourceful machine can merely act as a memory prosthesis, helping the owner remembering addresses and phone numbers.

Smartcards are designed to be tamper resistant, and as such they seem ideal as a minimal machine. However, trustworthy digital signatures can not be created by smartcards alone, simply because the user can not observe (and verify) what is given to the card for signing.

In the design of modern distributed systems, there are two forces that pull in opposite directions. One is users' desire to include computers into their private sphere, for a simple task such as keeping a diary up-to-date or a more complex one such as keep digital money. The other is users' desire for privacy.

Systems that aim at reaching users in their private sphere must be prepared to span more than one administrative domain. A crucial issue in such systems is that it must be possible for them to uniquely identify users. It must be possible for service providers (and others) to determine from whom a request or statement originates. In fact, we dare to say that any system, aiming at being ubiquitous or not, in which users can not make proper identifiable statements does not have a strong binding between human users and build-in concept of a user. In particular, without deterministic identification of statements (requests) charging for services becomes difficult.

Users have access to networking in most settings; both in friendly environments (for example at home) and possibly unfriendly (an Inter-

net café). In order to be able to make identifiable statements also in settings where the infrastructure is controlled by others, the user must be able to make statements without having to trust the infrastructure. A statement is some data, an ASCII string for example, accompanied by a digital signature. It is the signature that sets statements aside from other strings claiming to originate from the user. Public key technology is but one mean to achieve signatures. In any case, a statement can only be made on a trusted machine. This is so because a user can neither verify a signature's (cryptographic) validity, nor whether a signature is on some data at hand.

It is fairly obvious that when the user is armed with a laptop-type machine that he trusts, he can construct his message and sign it by whatever means he prefers on the laptop. The TCB is confined to the laptop itself. The signed statement can then be released to a possible hostile environment for transport to its destination. After signing, the statement can safely be disseminated because it is protected by some cryptographic property. Denial of service is always possible, but any alteration of the statement will be detected.

Not only laptops, but also most contemporary palmtop, such as the Palm Pilot, can be used to sign statements; the performance of a palmtop is not up to that of a laptop, but hardware with cryptographic functionality can easily be added to most such devices. A palmtop would be used in the same manner as a laptop: Show the string on the screen, and create the signature. Conceptually, a palmtop is identical to a laptop in this manner. They belong to the same "class" of hardware, because each has both a screen and (some form of) input channel.

The smartcard belongs to a different class. The distinguishing aspect is that a smartcard does not have a secure channel to (and from) the user. In other words, there seems to be a conceptually important border dividing the smartcard (and comparable devices) from "real" computers. In the context of security, a border between machines that can securely cerate a digital signature, and those that can not, is an important concept. For system designers, the mere existence of such a border implies that systems which aims at including smartcards will ultimately be less versatile than those that do not. It is apparent that in one sense, what constitutes the border is the lack of a secure channel. When using a smartcard, all messages to or from the card must pass through some machinery, and this machinery (whatever it might constitute) can alter the message at will. In other words, trusting a smartcard is in itself

not enough to create a digital signature. Focusing on the convenience of the smartcard, it is prudent to ask whether it is at all possible to build an infrastructure to compensate for the lack of a communication channel. That is, designing an infrastructure that spans any number of administrative domains but makes no assumptions on the trustworthiness of the terminal the user might be using when asking for a string to be turned into a signed statement by his card.

The rest of this section is structured as follows: In Section 8.1.1 the setting is described in more detail, and it is explained why smartcards in themselves makes the user less autonomous. Then an online service is described that, together with a secret number and a one time pad, enables users to create digital signatures on strings. Based on the method that is to be described, Section 8.1.3 contains a discussion centered around trust and trust relations. It is shown how a user by means of the described solution can take control over his own signatures. Related work is discussed in Section 8.1.5.

8.1.1 Overview

It is difficult to produce a trustworthy digital signature on data in a hostile environment, even armed with a smartcard that can create one by means of some public-key technology within the card. To see why, assume a user A sitting in front of a machine M with the smartcard residing in a smartcard reader connected to M. If A wants to sign some message X, he has no means to verify that M actually gives X to his card; Neither can he prevent M from retaining his PIN, nor that M presents multiple messages to the card to sign. The consequence is that A can not use the card without trusting M just as much as he trusts the integrity of the card itself. But if he trusts M, he could have used M to sign rather than involving a smartcard in the first place. When cards are used to create digital signatures, the card is normally used to ensure that the *user* is under control (he must bring his card, and can only use machines chosen by the owner of the system), rather than enabling the user to build a versatile digital personality.

There are many settings where one desires to sign data with a smartcard, where the environment might be hostile. For example a point-of-sale terminal, or during a visit to an “Internet café”. Using a computer laboratory at a university is another example. In general, any environment where one does not want to include the terminal in the TCB, for

whatever reason.

The problem is that there is no authenticated "channel" from the card to the user. The card is unable to "tell" the user what it has been asked to sign, and the user can not verify that the message X has been received for signing; the problem is well known [1, 52, 148].

In general, data integrity relies on either secret information or authenticated channels [93]. In other words, when using smartcards without any authenticated channels, some sort of secret information is needed. The user A is unable to *encrypt* for secrecy anything with his smartcard without trusting M, because all messages to the card must pass through M. Therefore, focus is solely on creating digital signatures. That is, secrecy can not be obtained at all in this setting (unless, as before, the user trusts the machine M, in which case the problem of encryption becomes trivial). This is in itself an limiting factor on the systems that can be build with smartcards.

The setting is that the user A has some data, an email perhaps, that he wants to sign, using the secret key stored in his card without having the key leaving the card. He would instruct the software running on M to send the data to the smartcard reader, insert his card, and having the signature returned in order to be attached to the email. The problem is that M might give any data it desires to the card, and the card will sign it. Unless A can verify public-key signatures in his head, he has no means to judge whether M is trustworthy or not.

In fact, what might seem to be a single problem really poses three distinct challenges:

1. How can A ensure that the correct data has been signed?
2. How can A verify that the signature is valid?
3. Is it possible for a third party to conclude that A has verified that the correct data has been signed?

The last item is required if the signatures A makes with his card are to have a non-trivial value.

Concerning the first question, only A knows the answer, because only he knows what he intended to have signed; the fact that M also happens to know is of no relevance because M is not trusted. The user must thus be involved in the verification of the signature at the "message meaning" level. Or, in other words, no solution to this problem

can be envisioned without involving the user in some way, after the signature has been created.

In a realistic scenario, the possibility of A verifying the signature himself can be ruled out. This implies that a third party must verify the signature. Such a third party should take the form of an online service, in order to better enable the user to timely obtain an answer the second question. This, however, raises a new obstacle: How can this server, called S, communicate with A over a channel that provides integrity? Our contribution is a working method to solve this particular problem.

Turning now to the third challenge, it will become evident that A in advance can sign a certificate that, together with the credentials created during progress, enables others to conclude that the data indeed was signed by A's card, with A's consent.

Our solution consists of three parts, an on-line service, a small one-time pad (OTP) and a secret. An OTP is a perfectly secure method for encryption. To see why, assume that Alice desires to send the message $\{1,2,3,1\}$ to Bob. If they share the encryption key 5 (and the algorithm is to add with the secret key), the encrypted message becomes $\{6,7,8,6\}$. It it obviously a problem that it is evident that the first and last component are equal, and the first, second and third component are sequential in the original "alphabet"; recall that any reasonable cryptosystem should assume that the algorithm (addition in this case) is known and only the encryption key (5) kept secret, see Chapter 2 for details. A remedy would be for Alice and Bob to share a "pad" with numbers (all of them secret), which reads $\{4,5,1,7\}$, for example. Using the pad as key rather than the number 5, but still using addition as algorithm, the encrypted message becomes $\{10,6,9,8\}$; it is no longer possible to know anything about the contents of the original message, except the length. Furthermore, if a pad is only used once (hence the name One-Time Pad), perfectly secure encryption has been achieved. Without knowing the key (the secret pad) it is impossible to derive the original message or key regardless of the computational power at hand, assuming the numbers on the pad are truly random. To be precise: There is no information in the encrypted data that can reveal neither the key nor the original message [117]. The One-Time Pad is an old invention, details can be found in [75, 93].

The scenario we have described can be pictured as shown in Figure 8.1. We will now describe the messages that constitutes the proto-

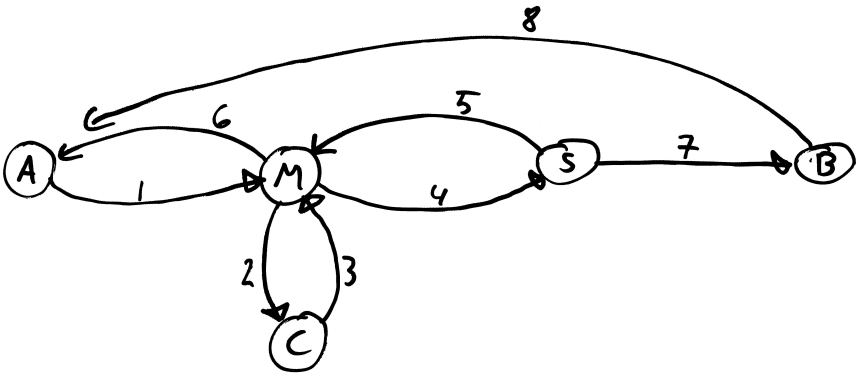


Figure 8.1: The protocol run

col, and in some detail argue for content of each message.

- The machine M is not trusted. Thus, M is not the logical sender or recipient of any message (even though the actual hardware will be used to send messages). From a logical point of view, M is part of the communication infrastructure and not trusted more (or less) than any other component.

  In this light, the only principals of interest are the user A, his smartcard C and the server S and the remote user B (to be described below).

- The user has some data X, which typically is a string of characters (i.e., a text). A inserts his card (into the smartcard reader attached to M) and instructs M to transfer the data to the card.

  $$\begin{array}{lrcl} \textit{Message 1:} & U \rightarrow M & : & X \\ \textit{Message 2:} & M \rightarrow C & : & X \end{array}$$

  There is no need for M to actually send X as a digital hash would suffice.

- The card accepts the message and signs X, creating $\{X\}_{K_C^{-1}}$. Notice that the card has no means to verify that X actually originates from A. Two questions must be answered:

  1. Is the signature valid?

2. Has the correct data been signed?

The online service can be used to verify the signature's validity; the signed data is sent to O.

$$\begin{array}{lrcl} \textit{Message 3:} & C \rightarrow M & : & \{X\}_{K_C^{-1}} \\ \textit{Message 4:} & M \rightarrow S & : & [X]_{K_C^{-1}} \end{array}$$

Recall that $[X]_{K^{-1}}$ implies the message X together with a signature, while $\{X\}_{K^{-1}}$ is a detached signature.

– The crux of this solution is that S can send back to A a transformation f of the data it has verified.

Assume that A and S share a secret OTP and a secret number. After verifying the signature on X, S will create two new message as follows.

Using the OTP, a new message $Z = f(X)$ is constructed, and sent to A. The function f is a digital hash (one-way function) as discussed in Section 2.3.1.

$$\begin{array}{lrcl} \textit{Message 5:} & S \rightarrow M & : & Z \\ \textit{Message 6:} & M \rightarrow U & : & Z \end{array}$$

Z is the message X transformed for integrity under the OTP. The nature of the transformation is discussed below.

Furthermore, if the signature is valid, S constructs a certificate asserting this fact; the content of this certificate will be discussed later. Having created the certificate, S sends it to B.

$$\textit{Message 7:} \quad S \rightarrow B \quad : \quad [C, X, f(X, f(Y))]_{K_O}$$

A random number Y is associated with each OTP. Y is known only to A, but $f(Y)$ is known also by S. If S finds that the signature on X is valid and made by C, S will sign a certificate stating this fact; the certificate will include $f(Y)$.

– When Z is received by A, he can without much effort (and without using M to anything but display Z) verify that $Z = f(X)$. Because Z is a transformation of X, A can conclude that the content was

what he intended to sign, and that his trusted server S has verified the signature. A now releases Y by sending it to B.

$$\textit{Message 8:} \quad A \rightarrow B \ : \ Y$$

To sum up, the online service verifies the signature made by the smartcard, and the user acknowledges the actual text by releasing Y.

There are eight messages in total:

$$\begin{array}{lll}
\textit{Message 1:} & A \rightarrow M & : \ X \\
\textit{Message 2:} & M \rightarrow C & : \ X \text{ from } P \\
\textit{Message 3:} & C \rightarrow M & : \ \{X\}_{K_C^{-1}} \\
\textit{Message 4:} & M \rightarrow S & : \ [X]_{K_C^{-1}} \text{ from } C \\
\textit{Message 5:} & S \rightarrow B & : \ [C, X, f(X, f(Y))]_{K_O} \\
\textit{Message 6:} & S \rightarrow M & : \ \langle X \rangle_{OTP} \\
\textit{Message 7:} & M \rightarrow A & : \ \langle X \rangle_{OTP} \text{ from } O \\
\textit{Message 8:} & A \rightarrow B & : \ Y
\end{array}$$

Messages 6 and 7 contain the string of digits S constructed based on its copy of the OTP. Because the OTP is secret, the string is X combined with a secret. The construction $\langle X \rangle_{OTP}$ is from BAN.

The next section presents a detailed description of the one-time pad that is required, a closer look at the messages that are sent and, most important, a careful analysis of the logical meaning of each message and of the certificates that are required to conclude that X was signed by C with the consent of A.

8.1.2 The One-Time Pad

We do not assume that A has any significant computational resources at hand (the machine M can not be trusted). It is also unreasonable to assume that any user can verify digital signatures without the help of a computer. It is for these two reasons a secure channel must be constructed from S to A, on which a message can be sent. That is, A needs to receive from S some information that convinces him that the correct message was signed by the smartcard. This information must be a function of the message X in order for A to know that the correct text has been signed. In addition, A must be convinced that the message he receives comes from S. Taken together, the channel must provide

authentication (because authentication implies integrity [93]). Cleartext attacks are indeed a threat because M knows X.

If, on the other hand, X was unknown to M, then A and S could share a list L of random numbers, each number L_i of L being as long as X. S would verify the signature on X, calculate $Z = X + L_i$ and send the result to A. A would be able to calculate $Z - L_i$ and verify that the card had signed X. This cipher would (also) be perfectly secure.

In the protocol we have designed, each OTP contains two small tables. The first contains random numbers, as one would use to create a one-time pad. However, in this case secrecy is not a goal (X is known to M anyway) but rather integrity. This is so because if 'A' and a random number yields 12 then 'B' must have yielded 13; it is obvious that this makes a simple attack possible. This goal is achieved by incorporating an additional table. It is a permutation of the characters; it is named a *substitution table*.

In the OTP used by A and S, the alphabet that is available are all the upper-case characters, space (denoted as '␣'), dot ('.'), the digits and the two symbols $ and @; 40 characters in all. These characters are matched with a table of random numbers, assigning a random number to each character. The example on page 162 shows two examples of tables; each has six rows:

Letter: The alphabet available to the user

Subst: The substitution table; each character from the alphabet is replaced by the corresponding number from the substitution table.

X: In this row the user writes his message

OTP: The number representing each character is added (modulo 40) to the corresponding element in the One Time Pad.

Z: The result.

Y: A secret number, see below.

When A receives $Z = f(X)$ from S, he would want to verify the result. In order to do so, he proceeds as follows.

1. Count the number of characters in the message, and prepend this number (as a string) to the message.

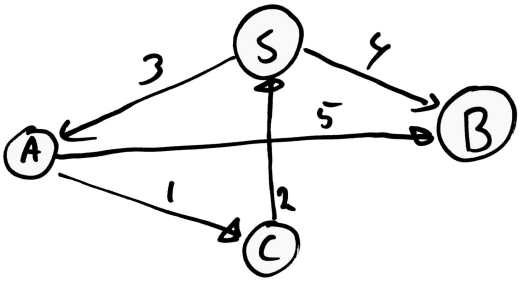


Figure 8.2: The protocol run with M removed

2. Write the string in the table (in the row marked X) above the random numbers.

3. For each character, add the ordinality of the character (taken from the substitution table) with the random number. The addition must be done modulo 40 (the number of characters).

An example of a table which is filled in is shown in the Appendix (on page 161). If A sees that Z indeed is the correct transformation of X, he will release Y. The certificate generated by S contains $f(Y)$, but Y is only known to A. In other words, by releasing Y, A makes it known that he supports the certificate issued by S.

8.1.3 Analysis

In this section we will analyse the proposed protocol to give a better understanding of its merits. The analysis is on three separate levels. First we use the SVO logic to analyse the messages that are sent and received, to show which believes the messages warrants. Second, at a higher level of abstraction, we use the theory of authentication to discuss the certificates needed for the user user A to have his statements authenticated and their content authorized. Then, at the end, we will discuss the trust relations inherit in the protocol, and argue informally that no single adversary can do any harm (except denial of service).

In general we can attach a special rôle to the machine hosting the smartcard reader. The machine is not under the users' control, and it is not trusted. Hence, we can consider M to be part of the communication infrastructure rather than an participant in the protocol. This way we

need not make any assumptions on its behavior. The simplified protocol is shown in Figure 8.2.

This protocol has different properties than the many key-exchange protocols in that the two important parties A and B does not act symmetrically or in response to receiving messages from the other party. Before considering the position of B we will briefly discuss the other three principals:

C**:** The situation is simple because C will sign anything that is presented to it. C does not attach any special meaning to the material it signs, and relies on the environment it runs ni (smartcard) to protect the secret key.

S**:** The server S is trusted by A to be honest, and assist him in verifying the signatures made by C.

First there are initial assumptions about the environment:

S1 S **believes** $S \overset{K}{\longleftrightarrow} A$

S2 S **believes** $PK_\alpha(C, K_C)$

S3 S **received** $[X]_{K_C}$

The assumption **S1** captures the shared nature of the one-time pad, while **S2** manifests that S believes K_C to be the signing key of C. S is expecting a single message. The message is expected to contain X together with a digital signature on X by C; C must embed some means for S to identify the sender. All this is shown in **S3**.

Then, what S might comprehend:

S4 S **believes** S **received** $[X]_{K_C}$

That is, S believes he will be able to determine that the correct message has been received.

Finally, we include premises corresponding to the meaning S is supposed to attache to the message he receives:

S7 S **sees** $[X]_{K_C} \supset$ S **believes** C **said** X

If S sees a signed message from C, he will believe that it was sent by C.

When **S3** holds, based[1] on **S4**, **Ax1**, **Ax3**, **MP**, and NEC, **S7** holds as well. Having been convinced that C has signed X and sent the message, S generates two new messages, one for A and one for B. It is worth noting that S is in the same position as C: None of them have any reason to believe that X is fresh. In particular, the message might be a replay. For this reason S can not vouch for the freshness of X.

A**:** In our setting the principal A is human. As such he is not a principal in the traditional sense, since he is unable to generate a digital signature. And for this reason it is probably unwarranted to reason formally on his behavior. Nevertheless we will do so, ignoring the fact that a user might choose to proceed with his actions regardless of the warning issued by the system. One could say that we take an optimistic view on the users' will and ability.

Here are the initial assumptions:

A1 A **believes fresh(**X**)**

A2 A **received** $\{X\}_K$

A3 A **believes** $A \overset{K}{\longleftrightarrow} S$

A4 A **believes** $((S$ **says** B **says** $X) \supset$ B **says** $X)$

Recall that contrary to the notation in BAN, $\{X\}_K$ means any transformation under the control of a key; in this case it is the one-time pad. To ensure that **P1** is warranted, A should either include a nonce or a serial number in the message; knowing that X is fresh in the sense that it was created now, is not interesting. The semantics of SVO states that **fresh(**X**)** *iff* for all principals P and all points in time prior to epoch, P **said** X has never been true. Hence, A must ensure that X is unique, or **P1** is not true.

P4 says that A will believe S when S says that C has said something. It is crucial that A is convinced that **P1** holds; he is the only one able to ensure that this is the case.

A is able to comprehend a single message:

[1] See Appendix B for a compiled list of axioms.

P5 A **believes** A **received** $\{X\}_K$

Because A knows X he is able to comprehend a message coming from S which contains a transformation of it.

Finally, we include premises corresponding to the assumed meaning that A should attach to the message he receives:

P6 A **believes** (A **received** $\{X\}_K \supset$
S **says** C **said** X)

We now state the goal:

G1 A **believes** C **said** X

The rationale is that if A believes that his smartcard has just signed the message he wanted it to sign, he will release the secret number that will bind him to the message (and the signature).

Upon receiving Message 3 (see Figure 8.2) A can verify by calculating from his one-time pad, that the message has been encrypted correctly. That is, he can assert that **P2** holds. When **P2** holds, **P6** will hold as well, based on **P1**, **P3**, **P4**, **Ax1**, **Ax3**, MP and NEC.

P4 and **P6**, based on **Ax1** and MP, will enable A to reach **G1**.

The only point of concern is **P4**. Assuming that some other principal has jurisdiction should not be done without careful consideration. We will return to the justification of this belief below.

B**:** The principal B is not assumed to be human. In particular, it rests on B to verify credentials that are supplied with the request from A; if B is human he must ensure that he has access to sufficient computational resources.

There are two possible desirable outcomes:

- B **believes** A **said** X
- B **believes** A **says** X

The difference in semantics is tied the the existence of a "current run". See Appendix B for details.

The problem in our setting is that in this protocol there is no exchange of messages between A and B, and as such no clear notion of a "session". We will now analyse the protocol to expose precisely what is needed to achieve the two goals.

First there are initial assumptions about the environment:

B1 B **believes** $PK_\alpha(S, K_s)$

B2 B **believes** $PK_\alpha(C, K_c)$

How B get to known the signature keys of S and C is outside the scope of the protocol;

B will receive two messages

B3 B **received** $\{X\}_{K_C^{-1}}, [C, X, H(X, H(Y))]_{K_S}$

B4 B **received** Y

Then, what B might comprehend:

B5 B **believes** B **received** $*_2, [C, X, *_1]_{K_S}$

B6 B **believes** B **received** $*_3$

The components $*_1, *_2$ and $*_3$ are unknown until some assumptions are made on their interpretation.

P7 B **believes** B **received** $[C, X, *_1]_{K_S} \wedge$
B **believes** B **received** $*_3 \supset$
B **received** $[C, X, H(X, H(Y))]_{K_S}$

P8 B **believes** B **received** $[C, X, *_1]_{K_S} \wedge$
B **believes** B **received** $*_2 \supset$
B **received** $\{X\}_{K_C^{-1}}$

The function H is assumed to be well known and effectively one-way function. Having received $*_1, *_2$ and $*_3$, B can verify that **P7** and **P8** are true. We can now proceed with our proof.

1. B **believes** B **received** $[C, X, H(X, H(Y))]_{K_s}$
   by B5, B8, Ax1, MP.
2. B **believes** S **said** $C, X, H(X, H(Y))$
   by 1, B1, Ax3, Ax1, NEC, and MP.

3. B **believes** C **said** X
   by B6, B2, Ax1, and MP.

We can get no further without making new and more powerful assumptions. In particular, we must make assumptions on the relation between A, C and S. A reasonable one might be:

B **believes** S **said** $C, X, H(X, H(Y)) \wedge$ B **sees** $Y \supset$
B **believes** A **said** X

There are two issues for us to consider:

1. How to defend against replay (there is nothing to make B believe X is fresh).
2. Which credentials should A present to B to convince B that such a strong assumption is reasonable.

We will now look at each of them in turn.

Freshness

The protocol and associated one-time pad must be regarded as a framework within which many possible solutions can be constructed. In particular, as discussed there is a need to convince B that the message is fresh. There are two possible approaches depending on what is required: A time stamp can be added to the messages in a variety of ways, or B can be engaged in the protocol and given the opportunity to have a nonce added to various messages. We will now explore these possibilities.

Adding a real-time component to the messages can be done in a variety of manners:

1. B can assume that S is trustworthy in this matter.

   If B adds to its set of initial beliefs that

   B **believes** S **controls** Ts

   then B is able to conclude that message 5 is fresh within some predetermined time frame.

2. C can include the time in the message.

   Contemporary smart cards do not have an on-board clock (because there is no power available on the card); this will probably change in the future. When a clock becomes available, C can include a time stamp in the signature it generates.

   Even without a clock, C is able to time stamp messages by relying on a trusted service by relying on a (mutual) trusted service. If such a service is available C could issue a nonce and obtain a time stamp on it.

   In addition, there are several other possibilities. If A and B share a common trusted service, C can contact it to obtain the time (by means of a nonce). There are technical difficulties in implementing such an approach, mainly in that the machine holding the card reader must be able and willing to forward messages generated by the card.

3. A includes the time in the message.

   A and B can agree, a priori, that each message needs to start with a string identifying the time when it was signed. This string would be 12 characters long, for example 200109241330 representing 13:30 September 24th, 2001. Such a solution will also require that some time of expire has been agreed upon; this can be embedded in the credentials that must accompany the messages that are exchanged in this protocol.

   Adding 12 characters (and a probably a trailing space) to the message obviously decreases the available space for the message.

   It is worth remembering that the protocol we are discussing is centered around the user, and having A include the time of day is the only means in which the user can be involved.

The second approach is to let B participate in the exchange of messages. Either C or S (or both) can contact B and offer to include a nonce in the message to be signed.

1. C includes a nonce provided by B; this is included in the signature and therefore included in Message 2.

   Under the assumption that B believes his own nonces to be fresh, B should believe that the signature on the message is fresh. This

does not at all vouch for the freshness of X itself. However, under the assumption that Y is released by A *after* S has generated $H(X, H(Y))$, it should again be possible for B to conclude that X is fresh because the signing must have happened before Y was released [83].

2. S can interact with B prior to signing to obtain a nonce that can be included. The line of arguments will be identical to one above.

Both of these solutions carry with them a problem identification: How is C (or S) to known that B is the intended recipient? There is nothing in the protocol that facilitates this. One can, for example, envision that A augmented the message sent to C with the name of B. On the other hand, the protocol has been devised to make it feasible for A to sign an arbitrary message by means of his smartcard, and whether the message is fresh or not might or might not be an issue of concern.

When discussing the issue of freshness, it is necessary to have the semantics in mind. Ideally we would like B to be convinced that A **says** X rather than "just" A **said** X. The semantics ties the difference between the two on whether there exists an epoch or not. In the protocol as we have discussed it, there is not, and hence no way to establish **fresh(**X**)**. And no way to make B believe that A **says** X.

We end this section by pointing out that our line argument is based on the assumption that there is no communication between A and B prior to the first message being sent. If A and B has some extra-system communication channel they can agree on a nonce, for example, that is to be included in X. It B adds the premise that B **believes fresh(**X**)** it is possible to derive B **believes** A **says** X.

Trust

We have shown that the protocol fulfills its goals, when regarded from the point of view of the believes the messages can reasonable lead to. Here we will explore what is required to make any of the assumptions untrue.

What has been described is how a user A can sign a message; what follows is a description of how a receiver verifies that a signed message is valid. Assume a user Q receives a message $\langle Y, [X]_{K_C} \rangle$ from A. Assume furthermore that Q believes that C belongs to A. Upon receiving the message, Q contacts S and asks for the certificate that O should have

Message	Meaning
X	X
$[X]_{K_C}$	C **says** X
$[C, X, H(X, H(Y))]_{K_O}$	$O\|C$ **says** X, $O\|Y$ **says** X
Y	Y

Table 8.1: Messages and their interpretation

generated. Obtaining it, Q has all he needs to conclude that X was signed by C, that O has verified that the signature was in order, and that A has verified that the correct data was signed. The four datums that are available to Q is shown in the left column of Table 8.1. Informally, the fact that A has released Y is proof that A has verified the signature. Below is a more formal view of the system, expressed in the theory from [85].

If Q is to act upon X he would need a certificate, signed by A (or a principal Q believes speaks for A), asserting that possessing the four items together vouches for the conclusion that X originates from A. Because the machine M is not trusted, A does not control C, and to assume $C \Rightarrow U$ (i.e., C **speaks for** U) is unwarranted. The intention of A is that no-one will hold him responsible for any message X unless the following conditions are met:

- X is signed by C.
- The signature made by C is verified by O. O must say that C have said X.
- O must tie (the secret) Y to the signed message. This enables A to accept the signature by releasing Y.
- Y is available.

All this is captured in the following certificate

$$P \textbf{ says } (C \wedge O|C \wedge O|Y) \Rightarrow P \tag{8.1}$$

Because Y is secret, Q is unable to satisfy the certificate (8.1) unless A releases Y. In practice, S could in addition act as an on-line verification for the validity of C in that A would make C issue C **says** $(S|C \wedge C) \Rightarrow C$. See [85] for details.

With these credentials, the axioms and interference rules set forth in [85], it follows that U **says** X. Note that the use of Y give the message the properties of a *transaction authentication* as defined in [93]; message authentication and the use of time-variant parameters (timeliness or uniqueness).

As can be seen from (8.1) it is a prerequisite for certificate verification that S says that A says X. However, A does not want to include S in his TCB. S has not been given Y by A, but rather $f(Y)$. When S quotes A as saying X it might turn out that S is mistaken; this is in fact correct in the cases where a message has been given to C for signing without As consent. In other words, when B collects credentials he might or might not be able to locate Y. In such a situation there are two possibilities: Either A has not released it (he has detected an attack) or Y has been delayed or deleted as part of a traditional denial-of-service attack. In general, because A is at the mercy of the machines he is using, there is no way to defend A against denial of service.

For a user, creating a trustworthy digital signature is impossible unless he trusts the hardware he is using. As described earlier signing becomes a problem when the communication channels leading to and from the smartcard is controlled by some other entity. In a distributed setting, the server S is a *trusted third party* in that the user A trusts it to act according to the protocol (i.e., not to certify that a signature is good if it is not). On the other hand, S is not able to deceive A without colluding with the machine M. When S and M collude, M can feed a false message to the card and let S send an erroneous message back to the user. The important issue is that acting in isolation, S can not deceive A. In the same manner as S can not deceive A alone, neither can B, nor C. It can be concluded that no principal is in a situation to make A release Y, without colluding with some other principal.

The online server S is central to the security of the system. If it colludes with M, signatures can be created without A's consent. However, it is to be expected that O is under the control of the user. That is, the user can have O placed in a trusted environment (at home, for example) because there is no reason why the online server need to be part of an infrastructure controlled by others.

Concern might be raised against the solution in that the combination of the card and a set of one-time pads represents a threat; losing them together would make the finder able to sign messages. However, access to the card might be secured with a PIN as is done with contemporary cards. This way, theft would not represent a threat.

If a server such as S is not available, the functionality offered by S can be implemented within C. In such case one would store the OTPs on the smartcard together the the secret key. C would then be able to first sign X and then generate $Z = f(X)$. Contemporary smartcards do not have sufficient memory to store a significant number of OTPs, but will surely change with time. We believe the above analysis will be correct regardless of whether C and S is physically within the same machine or not.

8.1.4 Implementation details

The security of the system hinges on three properties. The first is that one component in each sum is a random number. Randomness ensures the resulting list of numbers are random. No amount of calculation or number of previous messages can give information necessary to alter the text. Second, each OTP and substitution table can only be used once. Third, text can not be appended to the string.

Obviously, the length of the string that can be transmitted (and verified) in this manner is restricted by the length of the pad. However, the pad can be made as long as one desires and the amount of work to verify a message increases linearly with length. Another way to increase the task of verification is to increase the alphabet length (now being 40). If this length is increased, the OTPs and corresponding substitution tables must be increased accordingly.

The application of the signing procedure described here lies primarily in signing short messages. We wanted to verify the design by means of an implementation. Although the protocol was described as having five different participants (see Figure 8.1 on page 111), an implementation merges the on-line server (denoted O in the figure) and the users' server (denoted S). Furthermore, the machine that holds the card reader has a very simple protocol to adhere to (simply forwarding the messages it receives). In fact, the machine only needs to forward to the card the message to be signed, and to the user the message it receives from the card. All the other messages can be handled by the

user through a browser or some other means. We first discuss an implementation in general, before dwelling with details in our.

- On the smartcard, the application must accept the message to be signed, perform the signing with the appropriate public key, and return the result. We use the Cyberflex Open Series, manufactured by Schlumberger. The card conforms to the JavaCard 2.0 specification and communicate by means of the T=0 protocol (as per ISO 7816-3 [73]). The card supports both applets and applications (classes with a main-method); programs are called Cardlets in JavaCard terminology.

  To minimize communication, only a secure hash of the message is sent to the card, although the card supports signing the entire message. The Cardlet signs the hash with the installed secret key, and returns the signature.

- The server receives the message and the signature created on the smart card. After verifying that the signature is indeed calculated from the hash of the message, the server looks up the table of OTPs, selects the first unused one, and using the substitution table, the message and the actual OTP, calculating the resulting "encrypted" string. This is the returned to the caller (which might or might not be the machine the user is using; this is not a threat to security except that it makes denial of service possible). This software is trivial, and can be implemented in any language one might fancy.

- On the machine holding the smart card some form of user interface must be available; the system itself might very well solely support an API making it possible to write any interface one might desire (such as part of an CGI script). The message is collected and sent to the card for signing, and the signature retrieved. A driver for the T=0 protocol is needed; many are available.

Our implementation differs from the above in that we chose a slightly different approach, linking secure signing with access control. That is, instead of assuming that the user is sitting in front of a machine, we have designed and built a device that makes it possible to obtain a signature from *the user* that he wants access (not only from his smartcard) without giving him access to a machine proper. The device has a standard reader for smartcards, a small keyboard, and a display. All three

Figure 8.3: Device for secure signing; Manufactured by Ken-Arne Jensen, based on a design by this author. In the background, a cradle for the PalmPilot.

devices are controlled by a machine in the room by means of a two serial lines; the keyboard and display are controlled by one (RX and TX respectively) while the card reader is controlled by the other.

The user might prefer to prove his presence by some other means than by signing a message with the key stored in his smartcard; in particular, he might prefer to use the key he has in his PDA. To facilitate this, a cradle for the PalmPilot is also installed.

The actual setup is a machine which controls the lock of a door, the cradle, and the special purpose device to facilitate secure signing with a smartcard according to the protocols described earlier. The setup is shown in Figure 8.3. The door can be unlocked by applying 12V, controlled by a relay, which again is controlled by the voltage supplied by the parallel port on a machine.

The device has a standard reader for smartcards, a small keyboard, and a 80×2 back lit LCD. All three devices are controlled by a machine in the room, and five different methods are available for unlocking the door (in addition to the physical key):

1. Proof by knowledge: a Traditional PIN-code; the code is simply typed on the keyboard.

2. Proof by possession: Having a smartcard that can be identified (returns a valid encrypted challenge). The key needed to unlock the card is known to the machine.

3. A combination of the two: The PIN is used to derive an DES key, which is then used to challenge the card, or a combination where the key to the card is known and the user-card binding is done by the machine.

4. Proof by providing a PDA which can sign with a secret key. We have chosen the PalmPilot, and a cradle for it is installed next to the device holding the smartcard (see the figure). Signing can either by by a shared key (as explained in [96]) or with a secret key.

5. Secure signing with a smartcard as described earlier.

Since the keyboard does not have all 40 keys needed to sign messages with text, the system can only be used to sign numerical challenges. Although this is less powerful than described, it is sufficient to prove the identity of the person outside.

Experiments show that verification performed by the user is initially done at a speed of approximately 7-8 seconds per digit. Speed increases as one gets accustomed to the calculation. Experienced users spend approximately 3-4 seconds per digit. Slightly slower than typing, but in our opinion worthwhile.

8.1.5 Related work

We know of no other system where smartcards are used to place the user in control (as opposed to control the user). Storing encryption keys is a prime application for smartcards, but, as argued, it places users at the fringe of the system. As an example, smartcards can be integrated into Kerberos to further marginalize the user [70].

Smartcards give rise to a different set of trust-relations than does "traditional" computers; this is discussed in [116].

Our solution is in principle a Message Authentication Code (MAC). MACs are well covered in the literature, see for example [93, 128, 119]. Most types of MACs, such as MD5 [107], are surjective, and require quite some computation to be secure (mapping one language onto a smaller while being a one-way function; see Section 2.3.1 for details).

The use of unconditionally secure MACs are described in great detail in [128, Chapter 10] with the use of orthogonal arrays (OA). These OAs seems, however, to be infeasible to work with for humans compared

to substitution tables and OTPs that only require the use of elementary arithmetics.

8.1.6 Discussion

Smartcards are commonly employed to ensure that users are firmly placed at the fringes of the system. Because smartcards lack a communication channel to the user, the system that uses them is in full control. In general terms, the rôle assigned to users is one where they present their PIN whenever requested to do so. In our view, such a rôle is rather old fashioned. The challenge is to make it possible to enjoy the benefits of tamper resistant hardware with a standardized form factor, while building user-centric systems.

In our view, the crux of controlling one's own private computing environment, is the ability to create digital signatures in a secure manner. It has been shown here that users can achieve secure authentication of messages signed with a smartcard even in hostile environments, by using a partial trusted server together with a substitution table and a one-time pad. The applicability of the proposed solution lies in short messages with small character sets.

Admittedly, the solution is cumbersome, and of limited use as it is. However, it proves that, from a security point of view, there is no border separating smartcards from more powerful machines. Or, in other words, user-centric systems can be built also if one wants to place secrets on smartcards.

The most important ramification of this result is that it demonstrates how systems can be constructed where holders of smartcards benefit from the properties of smartcards. A different view is that it is also possible to construct a system where a user can retain their (digital) identity, as defined by their encryption keys, over a wide range of equipment.

8.2 Augmenting smartcards

So far, the discussion has been focused on infrastructure necessary to securely sign a message by means of a smartcard. As became evident, what is lacking is secure channels from the card to the user, and those channels were implemented by means of a one-time pad. In this section

a different approach is described, deviating from the standard in order to obtain new functionality.

In Section 1.4 it was made explicitly clear that we do not assume that communication links have any interesting properties such as providing secrecy, integrity or (ordered) delivery. However, some channels do indeed have such properties. In settings with private machines, it is reasonable to expect that "messages" that are printed on the device' screen are regarded not only as fresh, but also as authenticated. It is also evident that such a message is fresh regardless of whether any component of the message is fresh in the technical sense. This obviously also holds for input.

Analyzing protocols with messages in clear text is not less hard than analyzing other protocols, and we need to look at the logics. However, the logics need to be augmented in order to add a notion of clear-text channels with special properties [1]. We will briefly review the additions to the logics; the original contribution was to augment BAN and we will follow that line here even though we use SVO through.

First some notation:

P **said**$_L$ X**:** The principal P once said X on a link named L

P **sees**$_L$ X**:** P sees X on a link named L

timely(L)**:** All messages on the channel L are known to have been sent recently

$\overset{L}{\prec} P$**:** All messages arriving on L is known to have been sent by P, or a principal trusted by P

The intuitive semantics is that a message is fresh if you see that it is printed on a display.

In order to infer new beliefs from reception of messages the BAN logic must be augmented. Two new rules for inference is introduced:

- The origin of a message can be inferred if it arrives on a channel about which assumptions are made:

$$\frac{P \textbf{ believes } \overset{L}{\prec} Q, P \textbf{ sees}_L X}{P \textbf{ believes } Q \textbf{ said}_L X}$$

– If it is believed that a channel is timely (all messages on it are fresh), "said" can be promoted into "believe"

$$\frac{P \textbf{ believes timely}(L), P \textbf{ believes } Q \textbf{ said}_L\ X}{P \textbf{ believes } Q \textbf{ believes } X}$$

In [1] these new rules are used to reason about protocols in settings with smartcards. Here, the augmented logic can be used to analyse protocols with messages sent in clear text on timely and identifiably channels.

In essence, because the user A is unable to sign or encrypt without assistance, and any devices used to assist the user increases the TCB, we would like to have an authenticated channel that provides secrecy between the user and the device. With the formalism presented above, we want a channel L that makes it possible for the card to assume **timely**(L), and subsequently $\overset{L}{\prec} U$.

8.2.1 Secure Channels to and from a smartcard

The previous section showed the applicability of a keyboard and display on a smartcard. This section will describe how a card can be manufactured to include both a display and a keyboard. In brief, the question is: How can a card be inserted into a card reader while retaining enough space to accommodate a keyboard and a display, while being constrained by the standards that determines how large the card can be [73].

8.2.2 General characteristics

We propose a card which deviates from the standard. The costs are obvious, but we believe the gains might very well be worth the change. In short, a card is proposed, which, when from the reader, is a standard one. On "upside" of the card one finds a small display and 12 keys.

Seen from above the card resembles a simple pocket calculator, as shown in Figure 8.4.[2] The user interface (parts the user can interact with) consists of two elements:

Display: The card sends messages to the user by writing on the display.

[2]The figures in this section have been copied from the patent application, and has not been altered.

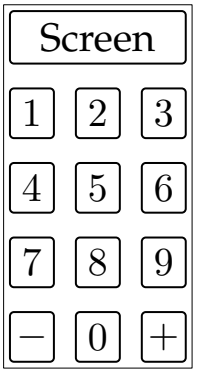


Figure 8.4: A card seen from above

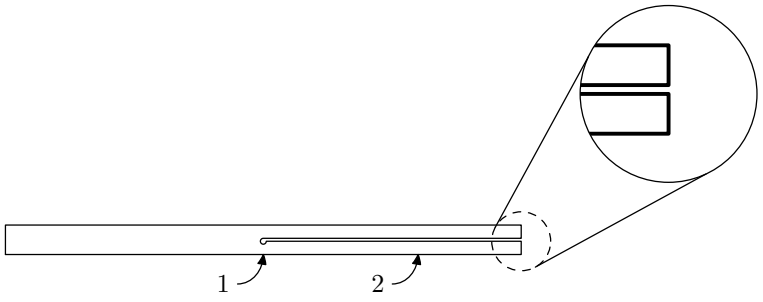


Figure 8.5: A card seen from the side

Keypad: To enable the user to send numeric messages, amounts and PIN code in particular, the keys for the ten digits are provided. In addition, two keys are provided to facilitate acceptance (on the figure symbolized by the key "+") or refusal (symbolized by "−").

Assuming communication with the reader is possible (to be described below), an application can, for example, request the card to sign a nonce to prove that the correct user is present. The card will prompt the user for the PIN, using it to decrypt the secret key stored on the card. With the secret key decrypted, the nonce can be signed. The PIN does not have to pass through an untrusted reader. A solution along these lines are described in [1].

The card consists of two major parts. The major part is the one hosting the display and the keys, while the minor part, hinged to the major part, constitutes a "flap". Mounted on the flap is the processor,

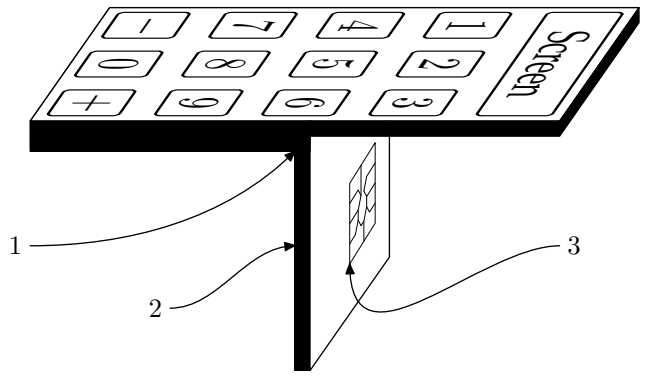


Figure 8.6: A card seen in perspective

and it has a standard form factor as described in the standard. The processor is connected to the display and keys by means of conductors embedded in the card. In Figure 8.5 the modified card is seen from the side, and the flap is in the "up" position. It can be noted that the card itself is twice as thick as the flap, which has the prescribed form factor. Arrow 1 points at the hinge, arrow 2 at the flap.

When a card is modified, the crucial question is compatibility. Figure 8.6 shows the card in perspective with the flap in the "down" position. The keyboard and display are clearly visible on top of the card, and it is shown how the flap can be placed downwards. Again, arrow 1 points to the hinge and arrow 2 points at the flap proper. In this position the electrical contacts for the processor are visible; arrow 3 points at the contacts.

Notice that the contacts are positioned on the now standard "lower location" on the flap in according to ISO 7816/2. That is, one half of the card is according to the form factor described in the standard. In particular, this modified card can be used in all readers where the card is inserted only partially.

A dedicated reader has been designed; a cut through this reader is shown in Figure 8.7. The idea is that the card can now be inserted vertically into the reader. When the card is full inserted, with the flap into the slot in the reader (arrow 4 point to the slot), the card will lay flat on the surface of the reader (arrow 5), while the contacts on the flap are placed firmly against the contacts in the reader (positioned where arrow 6 points).

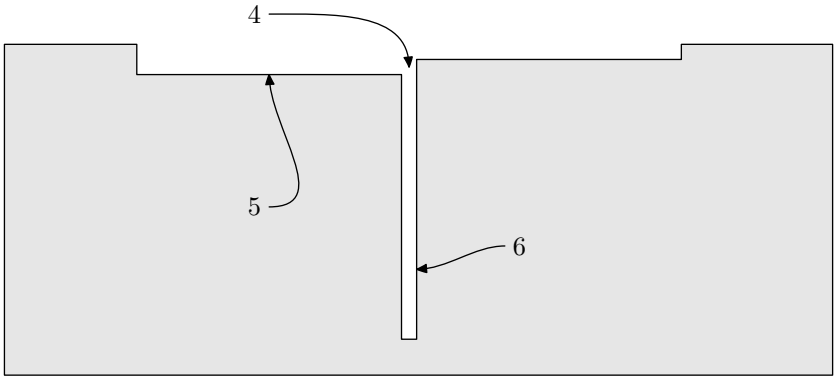


Figure 8.7: Cross section of a reader

8.2.3 Related work

Our patent is not, by far, the first to be concerned with the communication channel between the user and personal computing devices. Assuming that that the device has a communication channel, one can reason about protocols on the channel [1]. The problem remains, however, of how to implement the communication channel.

It is the combination of three issues that makes our approach novel:

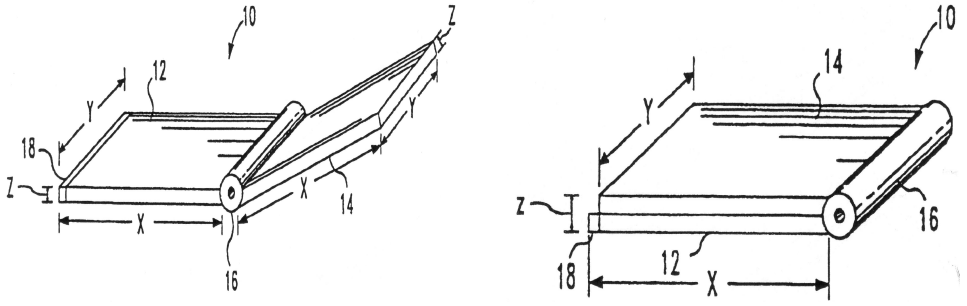
1. Making it possible to fold the card; this is both for convinience and for attaching it to a reader.

2. Attaching a keyboard to a smartcard.

3. Maintaining the formfactor of the standardized smarcard to the extent possible.

We will now discuss these issues separately.

United States Patent no. 5.373.147 describes a "Folding Electronic Card Assembly". The patent itself is limited to PCMCIA cards, but the idea can obviously be extended to other card-like devices. As shown in Figure 8.8(a), the card can be inserted into a Type I or Type II PCMCIA slot. The part outside the machine can, for example, contain a radio transmitter for use with a wireless local area network. When folded, as shown in Figure 8.8(b), the card will fit within a Type III or Type IV PCMCIA slot.



(a) Unfolded (b) Folded

Figure 8.8: A card that can be folded.

In this invention the folding is used to accomodate the card into several different standardized readers. In some senses, this is also our approach, although we only aim for one reader. We must assume that in both inventions the technical solution for the manufacturing of the hinge will be similar.

A very different approach is taken by UK Patent no. 2333926 which describes a device that can be folded, and to which one attaches a smartcard with an integrated keyboard. The device is designed as a GSM phone and as such not of interest to us. In this invention it is the device (phone) that can be folded and not the smartcard. The keyboard that is needed on a telephone is mounted on a smartcard and can be detached; the device is shown in Figure 8.9. The smartcard (item no. 909 with electric contacts placed as indicated by no. 904) slides onto the device proper. The contacts then engages onto the contacts on the device (no. 915).

The smartcard necessary for a GSM subscription is portable, and implementation of applications can be envisioned to use the keyboard and display available on the card. This invention demonstrates that integrating a keyboard onto a smartcard in itself is not novel. The problems of how to manufacture the keyboard so that it will fit on the card will be the same for both inventions.

German patent no. DE 4205615A1 constitutes a very different approach to the problem of obtaining a communication channel between the device and the user. Here the device does not host a smartcard but is rather a portable reader where a smartcard can be inserted. The device can be folded for convinience, but the folding is not part of the working of the device. The device is shown in Figure 8.10.

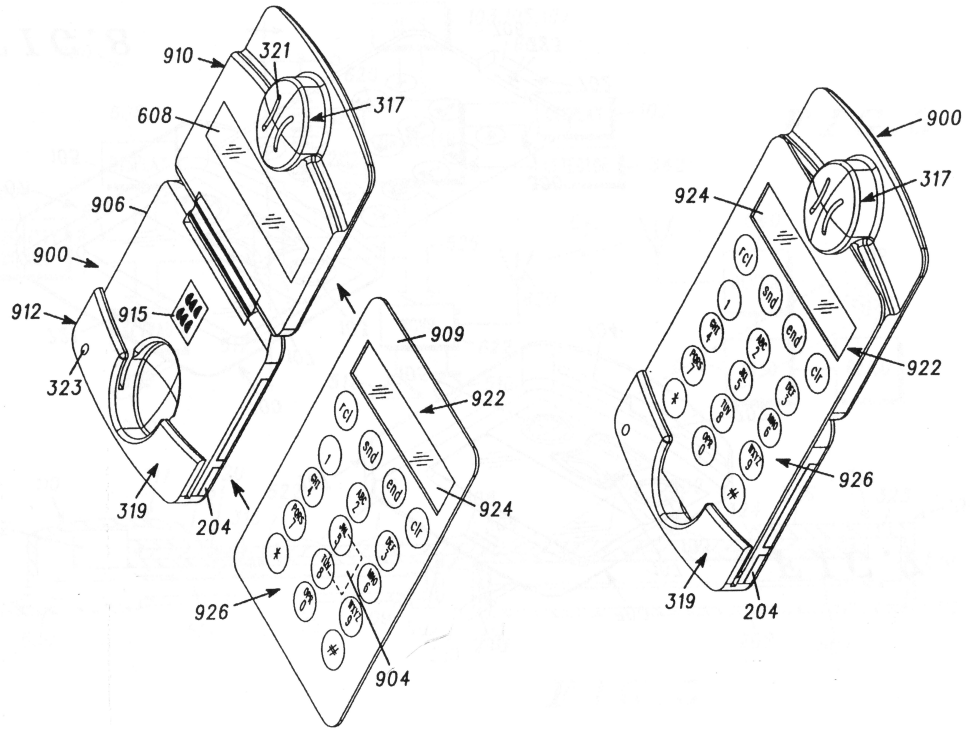


Figure 8.9: Smartcard with keyboard, and custom designed reader.

The idea is that a user can insert his smartcard (item no. 9) into his own reader (item no. 13) so that the card will engage the contacts of the reader (item no 10). Secure communication between the user and the card is now possible by means of a trusted reader. The patent describes how the card, for example, can be "loaded" with a PIN so that when the card later is inserted into an untrusted reader there will be no need for the card to obtain the PIN again. This patent does not discuss whether such mode of operation requires changes to the software on the card or in the banking system, but we believe this to be the case.

8.2.4 Discussion

Being tamper resistant, smartcards are the prime target for implementations of electronic commerce. Few other platforms enjoy such widespread acceptance of both the form factor and programming interface. However, lacking a secure channel to the user, the applicability of smartcards

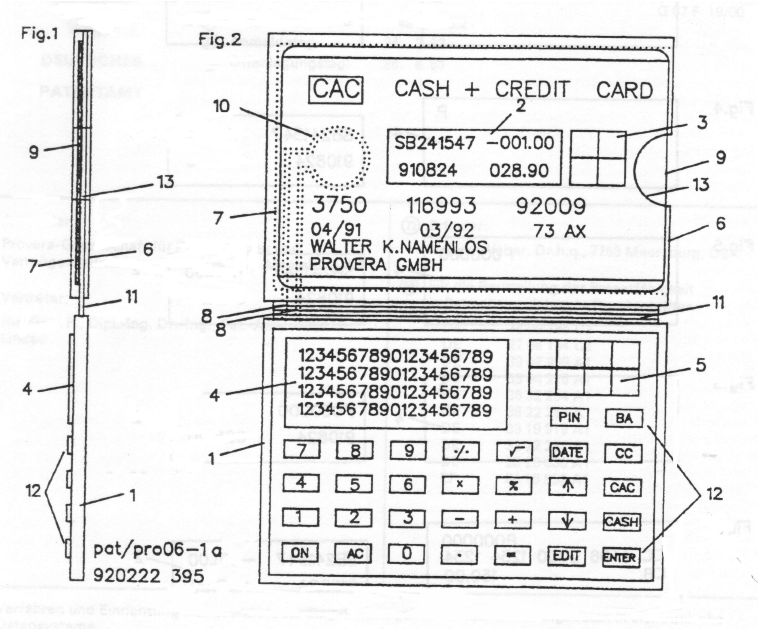


Figure 8.10: Card that can be folded, with user interface.

must be traded against the privacy of the user. The suggested change to the card, which to a great extent maintains compatibility, will enable applications that better guard privacy.

None of the ideas that are used in our approach are new but the combination is.

8.3 Conclusions

It might seem as if a PDA is different from a smartcard. However, because smartcards are a general purpose computers, at least seen from a computational point of view, it is possible to communicate with such machines in just the same manner as one would with any other machine. In particular, it is possible to exchange messages by means of channels that offer integrity, such as those a one-time pad represents. Establishing this fact by means of an working example effectively blurs the border between smartcards and PDAs. How smartcards best are put to use in distributed systems remains to be seen, but the contributions made here makes it quite clear that the border between a smartcard

and a PDA is one of convenience, not of real significance. In particular, because there is no important border between the two "types" of machines, we can envision a range of equipment spread out between the contemporary PDA and the standardized smartcard. These machines would be able to offer basically the same services, but with a varying degree of convenience.

In this chapter we have merely established the fact that there is a borderless area between the traditional PDA and the smartcard. There is probably room for considerable amounts of ingenious engineering; we have not indulged in those challenges.

Chapter 9

Conclusions

Our investigation has focused on private machines, and their inclusion in distributed systems. Basically, we have presented three lines of argument. First, we presented the open-end argument, and, based on it, we elaborated on how to close sessions and the stability of beliefs. Our main point was that the price to pay for transparency is the ability for users to control the system. Second, FR and off-line delegation was used as examples of how the inclusion of TDAs influences systems, and how the open-end argument can be put to use. Third and last, the nature of the TDA was explored by demonstrating how smartcards can be used as TDAs, and how to add functionality to a smartcard to turn it into a TDA.

In the introduction, our discussion on the use and applicability of private machines led us the pose the following two questions:

1. How does the integration of private machines influence the design of distributed systems?

2. How can a private, trusted device be included in the decision loop when it comes to sharing, delegation, authorization, and so on?

We have discussed a few issues related to this.

From our discussions it has become apparent that distributed systems with an ambition to include private machines must have (at least) three properties that will set them aside from more traditional systems. The

first is related to off-line delegation, the second to what constitutes a user (principal), and the third to transparency as a design goal.

With a private computer at hand, users will inevitably desire to grant access (delegate authority) to objects under their control. In the common case, the certificate is generated by the owner in cooperation with infrastructure (a file server, for example). With communication in place, transferring the certificate is also simple; the receiver can then exercise it at will. Delegation should be possible also when the participants are not connected to the infrastructure. This situation can arise either because the server in question is unreachable, or because the participants are isolated (off-line). Furthermore, we believe delegation should also be possible when there is no (electronic) connectivity at all. The single most important prerequisite for off-line delegation is that authority is expressed by means of certificates. We have demonstrated how this can be done under these conditions. The proposed solution, that works without connectivity, will obviously also work when communication is possible.

Taking a user-centric position, it is natural that the policy governing access to resources owned by the user, for example files, should be controlled by the user himself. Files are but an example, because the general model where a monitor evaluates requests can be used on many types of resources. Access to a room (by means of unlocking a door) would be implemented in just the same manner, with the same type of certificates. Our line of arguments have applicability also outside the scope under which they have been presented.

When authentication is based on certificates, we are free to again take a user-centric view. Users are free to delegate authority over their own resources to whatever channels they desire. However, it is an open engineering question how to design a system the will gracefully handle "downgraded" security. That is, designing and implementing a security regime where users can estimate the security risk of delegating authority to (security wise) weak channels (such as a password). For example, a user can delegate to a password authority to access a file for a limited time span (that is, anyone with the password are authorized). Before the certificate expires, an infrastructure for revocation is necessary to invalidate it, but that again must be encoded in the certificate in the first place (unless one is willing to accept a default security policy, which was the reason to start using certificates in the first place). It is an engineering challenge to find the tradeoffs in the design of a

particular system.

To us, it is evident that if a distributed system is to include private machines, it must be designed in accordance with the open-end principle. In fact, one could argue that a system where the user (and his machine(s)) is not the focal point, does not embrace private computing at all. We believe the term ubiquitous computing must imply private computing, and that the open-end argument should be the guiding principle for all such endeavor. We can envision an ubiquitous system where the user did not involve himself in any manner that was important to him. That is, the system could turn the light on and off in his office, provide him with stock quotes regardless of his whereabouts, and so on. But if he were to entrust the system with private information to any interesting extent, we are convinced that he would require control over it.

If we turn to design philosophies behind distributed systems, we have noted that layered design lends itself to transparency. Not that transparency is a bad thing, but in the mix of private computing and security there are qualitative assessments to be made as well as verifying (cryptographic) correctness. On the other hand, electronic commerce, almost by definition, seems to require some sort of authentication of strangers and it is hard to envision such services without the assistance of a (trusted) third party. Here, transparency indeed has its place. Again we see that private computing places new demands on distributed systems in that they require a different design strategy.

An old saying goes that security can not be added to a system as an afterthought; this probably also holds for TDAs.

Because it is impossible to give a rigorous definition of what a TDA is, we resort to the requirement that a TDA must (at least) be able to create a digital signature in a safe manner. Safe from the user's point of view, that is. We fail to see how a TDA can be realized without being at least this powerful.

We have discussed two types of machines that can be used as basis for a TDA: The smartcard and the PDA. Of these, the smartcard is particularly interesting because the large installed infrastructure backing this technology. It is obvious that the PDA can act as a TDA, as long as the software on it is designed with this in mind. The interesting aspects arise in the interaction between a PDA and supporting infrastructure.

Here, the challenges one faces are alike, regardless of whether the TDA has been build no the PDA or smartcards.

However, smartcards offer additional challenges, because machinery of some sort is needed for users to communicate with smartcards. This typically marginalizes the user. The problem is, basically, that nothing ties the signed text to the user. In particular, the user would always be able to claim that his intensions were to sign a different text, and no-one would be able to produce any convincing argument against the validity of this claim. What this means is that the electronic identity based on smartcards would be much "weaker" in a sense; in stark contrast to the weight which one would be able to place on the signature *per se*. Smartcards have a real advantage over other possible technologies: They are standardized, widespread, and supported by a large and prosperous industry. It is thus an engineering challenge to put them to work.

With marginal resources at hand, users will (have to) rely on remote servers to do part of the work. It is then necessary for these servers to be designed and implemented with the user as the focal point. In particular, such services must be built with the assumption that there is no such thing as a default security policy to which users must adhere. In our view, the most interesting aspect of this observation is that user themselves will have the power to grant access to to their own resources by whatever means they desire. As another example, it was shown that if a regime is put in place where the user (represented by his TDA) controls the system rather then the other way around, then the system must also be designed in such a way that sessions can be closed at the users' discretion. This also reflects onto transparency as a design principle; transparency and security is not a good mixture.

Because smartcards can be shown to be equivalent to a TDA, there does not seem to be a lower bound on how a TDA can be realized; we will naturally assume that we are considering computers at least as powerful as the Turing machine. Thus, it is a question of convenience whether one wants to utilize smartcards in a system rather than a more general PDA when implementing the TDA. The advantages of a tamper-proof device should probably not be underestimated in system design. Contemporary smartcards are virtually tamper proof, and it would be a pity to employ them in a setting where that positive aspect is not fully exploited. Seen in this light, smartcards have no place unless complemented by other, trusted machinery. In concert, a smartcard on which

secrets are stored securely and a trusted machine which acts as mediator between the user and the card, constitutes a powerful platform for guarding the privacy, while being flexible at the same time.

— * —

Bibliography

[1] ABADI, M., BURROWS, M., KAUFMAN, C., AND LAMPSON, B. Authentication and delegation with smart-cards. *Science of Computer Programming 21*, 2 (Oct. 1993), 93–113.

[2] ABADI, M., BURROWS, M., LAMPSON, B., AND PLOTKIN, G. A Calculus for Access Control in Distributed Systems. In *Proceedings of Advances in Cryptology—Crypto'91* (1992), Springer-Verlag, pp. 1–23.

[3] ABADI, M., AND NEEDHAM, R. M. Prudent engineering practice for cryptographic protocols. *IEEE Transactions on Software Engineering 22*, 1 (Jan. 1996), 6–15.

[4] ABADI, M., AND ROGAWAY, P. Reconciling two views of cryptography. In *Proceedings of the IFIP International Conference on Theoretical Computer Science (IFIP TCS2000)* (Sendai, Japan, Aug. 2000).

[5] ABADI, M., AND TUTTLE, M. A Semantics for a Logic of Authentication. In *Proceedings of the 10th Annual ACM Symposium on Principles of Distributed Computing* (Aug. 1991), pp. 201–216.

[6] AKTINS, D., STALLING, W., AND ZIMMERMANN, P. PGP message exchange format. RFC 1991, The Internet Society, Aug. 1996.

[7] ANDERSON, R. J. Liability and computer security: Nine principles. In *Computer Security (ESORICS 94)* (1994), vol. 875 of *Lecture Notes in Computer Science*, Springer Verlag, pp. 231–245.

[8] ANDERSON, R. J. Why Cryptosystems fail. *Communications of the ACM 37*, 11 (Nov. 1994), 32–40.

[9] ANDERSON, R. J. The Eternity service. In *Proceedings of the 1st International Conference on the Theory and Applications of Cryptology* (1996).

[10] ANDERSON, R. J. *Security Engineering*. John Wiley & Sons, Inc., 2001. ISBN 0-471-38922-6.

[11] ANDERSON, R. J., CRISPO, B., AND LEE, J.-H. *The Global Internet Trust Register*. MIT Press, 1999. ISBN 0-26251-105-3.

[12] ANDERSON, R. J., AND KUHN, M. Tamper resistance — a cautionary note. In *Proceedings of the 2nd Workshop On Electronic Commerce* (Oakland, California, Nov. 1996), USENIX Association, pp. 1–11.

[13] ANDERSON, R. J., AND KUHN, M. Low cost attacks on tamper resistant devices. In *Proceedings of 5th International Workkshop on Security Protocols* (Apr. 1997), vol. 1361 of *Lecture Notes in Computer Science*, Springer Verlag, pp. 125–136.

[14] ANDERSON, R. J., AND NEEDHAM, R. M. Programming Satan's Computer. In *Computer Science Today — Recent Trends and Developments*, J. van Leeuwen, Ed., vol. 1000 of *Lecture Notes in Computer Science*. Springer-Verlag, 1995, pp. 426–440.

[15] ANDERSON, R. J., AND NEEDHAM, R. M. Robustness Principles for Public Key Protocols. In *Proceedings of Advances in Cryptology—Crypto'95* (1995), vol. 963 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 236–247.

[16] BAGGIO, A. System support for transparency and network-aware adaptation in mobile environments. In *ACM Symposium on Applied Computing special track on Mobile Computing Systems and Applications* (Atlanta, Georgia, USA, Feb. 1998). Also available as a research report: INRIA Research Report 3408, April 1998.

[17] BASHIR, I., SERAFINI, E., AND WALL, K. Securing network software applications. *Communications of the ACM 44*, 2 (Feb. 2001), 29–30.

[18] BELLOVIN, S., AND MERRITT, M. Encrypted key exchange: passord-based protocols secure against dictionary attacks. In *Proceedings of the 1992 IEEE Computer Society Conference on Research in Security and Privacy* (1992), IEEE Computer Society, pp. 72–84.

[19] BELLOVIN, S. M., AND MERRIT, M. Limitations of the Kerberos authentication system. *ACM Computer Communications Review 20*, 5 (1990), 119–132. A version was published on the 1991 Usenix Winter conference.

[20] BIRRELL, A. D., HISGEN, A., JERIAN, C., MANN, T., AND SWART, G. The Echo distributed file system. Techical report 111, Digital Equipment Corporation Systems Research Center, Palo Alto, CA, Sept. 1993.

[21] BLAZE, M., DIFFIE, W., RIVEST, R. L., SCHNEIER, B., SHIMOMURA, T., THOMPSON, E., AND WIENER, M. Minimal Key Length for Symmetric Ciphers to Provide Adequate Commercial Security. A Report by an Ad Hoc Group of Cryptographers and Computer Scientis, Jan. 1996.

[22] BLUM, M., AND GOLDWASSER, S. An efficient probabilistic public-key encryption scheme which hides all partial information. In *Proceedings of Advances in Cryptology—Crypto'84* (1984), vol. 196 of *Lecture Noets in Computer Science*, Springer verlag.

[23] BORENSTEIN, N. S., AND FREED, N. MIME (multipurpose internet mail extensions) part one: Mechanisms for specifying and describing the format of internet message bodies. RFC 1521, The Intnernet Society, Sept. 1993.

[24] BOYD, C., AND MAO, W. Designing Secure Key Exchange Protocols. In *Third European Symposium on Research in Computer Security (ESORICS 94)* (Brighton, United Kingdom, Nov. 1994), vol. 875 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 93–105.

[25] BRANDS, S. *Rethinking public key infrastructures and certificates – building in privacy*. PhD thesis, Technische Universiteit Eindhoven, Sept. 1999. ISBN 90-901-3059-4.

[26] BREWER, E., KATZ, R. H., AMIR, E., BALAKRISHNAN, H., CHAWATHE, Y., FOX, A., GRIBBLE, S., HODES, T. NGUYEN, G., PADMANABHAN, V., STEMM, M., SESHAN, S., AND HENDERSON, T. A network architecture for heterogeneous mobile computing. *IEEE Personal Communications Magazine 5*, 5 (Oct. 1998), 8–24.

[27] BURROWS, M., ABADI, M., AND NEEDHAM, R. M. A logic of authentication. *ACM Transactions on Computer Systems 8*, 1 (Feb. 1990), 18–36. Also available in the Proceeding of the 12th Symposium on Operating System Principles, Litchfield Park, AZ, USA, 3–6 Dec. 1989. Published as *ACM Operating System Review*, Vol. 23, No. 5, pp. 1–13, December 1989. Also presented in *Proceedings of the Royal Society of London*, Series A, 426:233–271, 1989.

[28] BURROWS, M., ABADI, M., AND NEEDHAM, R. M. Rejoinder to Nesset. *ACM Operating System Review 24*, 2 (Apr. 1990), 39–40.

[29] BURROWS, M., JERIAN, C., LAMPSON, B., AND MANN, T. On-line data compression in a log-structured file system. In *Proceedings of the Fifth International Conference on Architectural Support for Programming Languages and Operating Systems* (1992), pp. 2–9.

[30] BUTTYÁN, L. Formal methods in the design of cryptograpic protocols. Technical Report SSC/1999/38, Swiss Federal Institute of Technology (EPFL), Lausanne, Switzerland, Nov. 1999.

[31] CALLAS, J., DONNERHACKE, L., FINNEY, H., AND THAYER, R. OpenPGP message format. RFC 2440, The Internet Society, Nov. 1998.

[32] CCITT. Information Technology — Open Systems Interconnection — The Directory: Authentication Framework. CCITT Recommodation X.509, ISO/IEC 9594-8, Dec. 1991.

[33] CLARK, J., AND JACOB, J. A survey of authentication protocol literature. Available from http://www.cs.york.ac.uk/~jac.

[34] DAVIDSON, S. B., GARCIA-MOLINA, H., AND SKEEN, D. Consistency in partitioned networks. *ACM Computing Surveys 17*, 3 (Sept. 1985), 341–370.

[35] DEPARTMENT OF DEFENSE. DoD 5200.28-STD: Department of defense (DoD) Trusted Computer System Evaluation Criteria (TCSEC), 1985.

[36] DIFFIE, W. The first ten years of public key cryptography. In Simmons [119], ch. 3, pp. 135–175. ISBN: 0-87942-277-7.

[37] DIFFIE, W., AND HELLMANN, M. E. New Directions in Cryptography. *IEEE Transactions on Information Theory IT-22*, 6 (Nov. 1976), 644–654.

[38] DIFFIE, W., VAN OORSCHOT, P., AND WEINER, M. J. Authentication and authenticated key exchanges. In *Designs, Codes and Cryptography* (1992), vol. 2, Kluwer Academic Publishers, pp. 107–125.

[39] DILLEMA, F. W., AND TAGE STABELL-KULØ. The Pesto storage architecture. Work in Progress session at Middleware 2001, Nov. 2001.

[40] DOLEV, D., AND YAO, A. C. On the security of public key protocols. *IEEE Transactions on Information Theory IT-29*, 2 (Mar. 1983), 198–208.

[41] DREIFUS, H., AND MONK, T. *Smart Cards – A Guide to Building and Managing Smart Card Applications*. IEEE Computer Press, 1997. ISBN 0-471-15748-1.

[42] DURGIN, N., LINCOLN, P., MITCHELL, J., AND SCEDRO, A. Undevidability of bounded security protocols. In *Proceedings of the Workshop on Formal Methods and Security Protocols* (Trento, Italy, July 1999), N. Heintze and E. Clark, Eds.

[43] ELLISON, C. SPKI requirements. RFC 2692, The Internet Society, Sept. 1999.

[44] ELLISON, C. M., FRANTZ, B., LAMPSON, B., RIVEST, R., THOMAS, B., AND YLONEN, T. SPKI certificate theory. Rfc, The Internet Society, Sept. 1999.

[45] FALLMYR, T., HARTVIGSEN, G., AND STABELL-KULØ, T. Supporting Mobile Users in a Variable Connected Distributed System: the PASTA Approach. Presented at NIK'95, Nov. 1995.

[46] FALLMYR, T., AND STABELL-KULØ. QoS applied to security in mobile computing. Workshop on Mobility and Network Aware Computing, Zürich, Switzerland, Sept. 1997. Held in conjunction with the Sixth European Software Engineering Conference and Fifth ACM SIGSOFT Symposium on the Foundations of Software Engineering. No proceedings. Names of authors in alfabetic order.

[47] FLUHRER, S., MANTIN, I., AND SHAMIR, A. Weaknesses in the key scheduling algorithm of RC4. Eighth Annual Workshop on Selected Areas in Cryptography, Aug. 2001.

[48] FROOMKIN, A. M. The essintial role of trusted third parties in electronic commerce. *Oregon Law Review 75*, 1 (1996), 49–115.

[49] GAREY, M. R., AND JOHNSON, D. S. *Computers and intractability. A guide to NP-completeness*. W. H. Freeman and Company, 1979. ISBN 0-7167-1044-7.

[50] GARRETT, P. B. *Making, breaking codes; an introductino to cryptology*. Prentice-Hall, 2001. ISBN: 0-13-030369-0.

[51] GIFFORD, D. K. Weighted voting for replicated data. In *Proceedings of 7th Symposium on Operating Systems Principles* (Pacific Grove, California, Dec. 1979), ACM Press, pp. 150–62.

[52] GOBIOFF, H., SMITH, S., TYGAR, J. D., AND YEE, B. Smart Cards in Hostile Environments. In *Proceedings of the Second USENIX Workshop on Electronic Commerce* (Oakland, CA, Nov. 1996).

[53] GOLDSCHLAG, D., REED, M., AND SYVERSON, P. Onion routing. *Communication of the ACM 42*, 2 (Feb. 1999), 39–41.

[54] GONG, L. A note on redundancy in encrypted messages. *ACM Computer Communication Review 20*, 5 (Oct. 1990), 18–22.

[55] GONG, L. A Security Risk of Depending on Synchronized Clocks. *ACM Operating Systems Review* (Jan. 1992).

[56] GONG, L. Increasing availability and security of an authentication service. *IEEE Journal on Selected Areas in Communications 11*, 5 (June 1993), 657–662.

[57] GONG, L., LOMAS, M. A., NEEDHAM, R. M., AND SALTZER, J. H. Protecting Poorly Chosen Secrets from Guessing Attacks. *IEEE Journal on Selected Areas in Communications 11*, 5 (June 1993), 648–656.

[58] GONG, L., NEEDHAM, R., AND YAHALOM, R. Reasoning about Belief in Cryptographic Protocols. In *Proceedings of the IEEE 1990 Symposium on Security and Privacy* (Oakland, California, May 1990), pp. 234–248.

[59] GOSCINSKI, A. *Distributed Operating Systems, The Logical Design*. Addison-Wesley, 1991.

[60] GRITZALIS, S., NIKITAKOS, N., AND GEORGIADIS, P. Formal methods for the analysis and design of cryptographic protocols: A state-of-the-art review. In *Proceedings of the IFIP Working Conference on Communications and Multimedia Securit* (1997), vol. 3, pp. 119–132.

[61] GUILLOU, L. C., UGON, M., AND QUISQUATER, J.-J. The smart card. A standardized security device dedicated to public cryptology. In Simmons [119], ch. 12.

[62] GUNTHER, C. An identity-based key-exchange protocol. In *Proceedings of Advances in Cryptology—Eurocrypt'89* (1989), vol. 434 of *Lecture Notes in Computer Science*, Springer Verlag, pp. 29–37.

[63] HEIDEMANN, J. S., PAGE, T. W., GUY, R. G., AND POP EK, G. J. Primarily disconnected operation: Experience with Ficus. In *Proceedings of the Second Workshop on the Management of Replicated Data* (November 1992).

[64] HEINTZE, N., AND TYGAR, J. D. A model for secure protocols and their composition. *IEEE Transactions on Software Engineering 22*, 1 (Jan. 1996), 16–30. Special section from 1994 *IEEE Symposium on Security and Privacy*.

[65] HELME, A. *A System for Secure User-controlled Eletronic Transactions*. PhD thesis, Informatica, Universiteit van Twente, Enschede, the Netherlands, Aug. 1997.

[66] HELME, A., AND STABELL-KULØ, T. Off-Line delegation in a File Repository. In *1996 DIMACS WorkShop on Trust Management in Networks* (Rutgers University, Sept. 1996). Work presented at workshop, but no proceedings. Names of authors are in alfabetic order.

[67] HELME, A., AND STABELL-KULØ, T. Security Functions for a File Repository. Memoranda Informatica 96–17, University of Twente, Enschede, The Netherlands, Nov. 1996. Names of authors are in alabetic order.

[68] HELME, A., AND STABELL-KULØ, T. Security Functions for a File Repository. *ACM Operating Systems Review 31*, 2 (Apr. 1997), 3–8. Names of authors are in alabetic order.

[69] HELME, A., AND STABELL-KULØ, T. Offline Delegation. In *Proceedings of the 8th USENIX Security Symposium* (Washington, D.C., Aug. 1999), The USENIX Association, pp. 25–33. ISBN 1-880446-28-6.

[70] HONEYMAN, P., AND ITOI, N. Smartcard integration with Kerberos V5. In *Proceedings of the Usenix workshop on smartcard technolog* (Chicago, IL, USA, May 1999).

[71] HOWELL, J., AND KOTZ, D. End-to-end authorization. In *Proceedings of the Fourth Symposium on Operating Systems Design and Implementation (OSDI 2000)* (Oct. 2000), pp. 151–164.

[72] INTEL CORPORATION. *Intel 82802AB/AC Firmware hub*, Nov. 2000. Document Number: 290658.

[73] INTERNATIONAL ORGANIZATION FOR STANDARDIZATION (ISO), TECHNICAL COMMITTEE / SUBCOMMITTEE: JTC 1/SC 17 (IDENTIFICATION CARDS AND RELATED DEVICES). *ISO/IEC 7816: Identification cards – Integrated circuit(s) cards with contacts*. International Organization for Standardization (ISO), 1 rue de Varembé, Case postale 56, CH-1211 Genève 20, Switzerland, 1998.

[74] JOSEPH, A. D., TAUBER, J. A., AND KAASHOEK, M. F. Mobile computing with the Rover toolkit. *IEEE Transactions on Computers: Special issue on Mobile Computing* (Mar. 1997), 337–352.

[75] KAHN, D. *The Code-Breakers: The story of secret writing*. Macmillan Publishing Company, New York, USA, 1967.

[76] KELSEY, J., SCHNEIER, B., AND WAGNER, D. Protocol interaction and the chosen protocol attack. In *Proceedings of the International Workshop on Security Protocols* (Apr. 1996), Lecture Notes in Computer Science, Springer Verlag, pp. 91–104.

[77] KISTLER, J. J. *Disconnected operations in a distributed file system*, vol. 1002 of *Lecture Notes in Computer Science*. Springer Verlag, 1996.

[78] KISTLER, J. J., AND SATYANARAYANAN, M. Disconnected operation in the Coda file system. *ACM Transactions on Computer Systems 10*, 1 (Feb. 1992), 3–25. See also [77].

[79] KOHL, J., AND NEUMAN, C. The Kerberos network authentication service (v5). RFC 1510, The Internet Society, Sept. 1993.

[80] KOHNFELDER, L. M. Towards a practical public-key cryptosystem. Master's thesis, MIT Laboratory for Computer Science, May 1978.

[81] KUBIATOWICZ, J., BINDEL, D., CHEN, Y., EATON, P., GEELS, D., GUMMADI, R., RHEA, S., WEATHERSPOON, H., WEIMER, W., WELLS, C., AND ZHAO, B. OceanStore: An Architecture for Global-scale Persistent Storage. In *Proceedings of ACM ASPLOS* (Nov. 2000), ACM.

[82] LAI, X., AND MASSEY, J. L. A proposal for a new block encryption standard. In *Proceedings of Advances in Cryptology—Eurocrypt'90* (1991), I. Damgård, Ed., vol. 473 of *Lecture Notes in Computer Science*, Springer Verlag, pp. 389–404.

[83] LAMPORT, L. Time, Clocks, and the Ordering of Events in a Distributed System. *Communications of the ACM 21*, 7 (July 1978), 558–565.

[84] LAMPORT, L. On interprocess communication, part I and II. *Distributed Computing 1*, 1 (1985).

[85] LAMPSON, B., ABADI, M., BURROWS, M., AND WOBBER, E. Authentication in distribued systems: theory and practice. *ACM Transactions on Computer Systems 10*, 4 (Nov. 1992), 265–310.

[86] LEVESON, N. G. *SAFEWARE: System Safety and Computers*. Addison-Wesley, 1995.

[87] LIEBL, A. Authentication in Distributed Systems: A Bibliography. *ACM Operating Systems Review 27*, 4 (Oct. 1993), 31–41.

[88] LINN, J. Privacy enhancement for internet electronic mail: Part I: Message encryption and authentication procedures. RFC 1421, The Internet Society, Feb. 1993.

[89] MAZIÈRES, D., KAMINSKY, M., KAASHOEK, M. F., AND WITCHEL, E. Separating key management from file system security. In *Proceedings of the 17th ACM Symposium on Operating System Principles* (Dec. 1999), pp. 124–139. Operating Systems Review, volume 34, no. 5.

[90] MCKUSICK, M. K., JOY, W. N., LEFFLER, S. J., AND FABRY, R. S. A fast file system for UNIX. *ACM Transactions on Computer Systems 2*, 3 (1984), 181–197.

[91] MEADOWS, C. Formal Verification of Cryptographic Protocols: A Survey. In *Proceedings of Advances in Cryptology—Asiacrypt'9* (1995), vol. 917 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 133–150.

[92] MENEZES, A. J. *Elliptic Curve Public Key Cryptosystems*. Kluwer Academic Publishers, 1993. ISBN: 0-7923-9368-6.

[93] MENEZES, A. J., VAN OORSCHOT, P. C., AND VANSTONE, S. A. *Handbook of applied cryptography*. The CRC Press series on discrete mathematics and its applications. CRC Press, 1997. ISBN 0-8493-8523-7.

[94] MERKLE, R. C. Secure communication over insecure channels. *Communications of the ACM 21*, 4 (Apr. 1978), 294–299.

[95] MYRVANG, P. H. Obol: a light-weight protocol description language. Submitted for publication, Apr. 2002.

[96] MYRVANG, P. H., AND STABELL-KULØ, T. The PASTA doorkeeper. NIK'99, pp 15-17, Nov. 1999.

[97] NECHVATAL, J. Public Key Cryptography. In Simmons [119], ch. 4, pp. 177–288. ISBN: 0-87942-277-7.

[98] NEEDHAM, R. M. Names. In *Distributed Systems*, S. J. Mullender, Ed. ACM Press, 1989, ch. 5, pp. 89–102.

[99] NEEDHAM, R. M., AND SCHROEDER, M. D. Using Encryption for Authentication in Large Networks of Computers. *Communications of the ACM 21*, 12 (Dec. 1978), 993–998.

[100] NESSET, D. M. A critique of the Burrows, Abadi and Needham logic. *ACM Operating System Review 24*, 2 (Apr. 1990), 35–38.

[101] NEUMANN, P. G. *Computer-Related Risks*. ACM Press, 1995.

[102] NEUMANN, P. G. Security and privacy issues in computer and communication systems. In *The computer science and engineering handbook*, A. B. Tucker Jr., Ed. CRC Press, 1996, ch. 89, pp. 1910–1913.

[103] NOBLE, B. D., SATYANARAYANAN, M., NARAYANAN, D., TILTON, J. E., FLINN, J., AND WALKER, K. R. Agile application-aware adaption for mobility. *ACM SIGOPS Operating Systems Review 31*, 5 (Dec. 1997), 276–287. in: SIGOPS '97. Proceedings of the sixteenth ACM symposium on Operating systems principles, pages 264-275.

[104] RANKL, W., AND EFFING, W. *Smart Card Handbook*, 2 ed. John Wiley & Sons, 2000. ISBN 0-471-98875-8.

[105] REED, D. P., SALTZER, J. H., AND CLARK, D. D. Active networking and end-to-end arguments. *IEEE Network 12*, 3 (May 1998), 69–71.

[106] RIVEST, R. L. Cryptography. In *Handbook of Theoretical Computer Science*, J. van Leeuwen, Ed., vol. A. Elsevier Science Publishers B.V., P.O. Box 211, 1000 AE Amsterdam, The Netherlands, 1990, ch. 13, pp. 717–755.

[107] RIVEST, R. L. The MD5 message-digest algorithm. RFC 1321, The Internet Society, Apr. 1992.

[108] RIVEST, R. L., AND LAMPSON, B. SDSI—A Simple Distributed Security Infrastructure, 1996. Working document (Version 1.1).

[109] RIVEST, R. L., SHAMIR, A., AND ADLEMAN, L. A Method for Obtaining Digital Signatures and Public-key Cryptosystems. *Communications of the ACM 21*, 2 (Feb. 1978), 120–126.

[110] ROE, M. *Cryptography and evidence*. PhD thesis, Clare College, University of Cambridge, UK, 1998.

[111] ROSENBLUM, M., AND OUSTERHOUT, J. K. The design and implementation of a log-structured file system. In *Proceedings of the 13th Symposium on Operating System Principles* (Oct. 1991), pp. 1–15.

[112] SAFAVI-NAINI, R., MATHURIA, A. M., AND NICKOLAS, P. R. Some remarks on the logic of Gong, Needham and Yahalom. In *Proceedings of the International Computer Symposium* (NCTU, Hsinchu, Taiwan, Dec. 1994), vol. 1, pp. 303–308.

[113] SALTZER, J. H., REED, D. P., AND CLARK, D. D. End-to-end arguments in system design. *ACM Transactions on Computer Systems 2*, 4 (Nov. 1984), 277–288.

[114] SANDBERG, R., GOLDBERG, D., KLEIMAN, S., WALSH, D., AND LYON, B. Design and implementation of the Sun Network Filesystem. In *USENIX Association summer conference proceedings* (Portland, Oregon, USA, June 1985), USENIX Association, pp. 119–130.

[115] SCHNEIER, B. *Applied cryptography. Protocols, Algorithms, and source code in C*, second ed. John Wiley & Sons, Inc., 1996. ISBN 0-471-12845-7.

[116] SCHNEIER, B., AND SHOSTACK, A. Breaking up is hard to do: Modeling security threats for smart cards. In *Proceedings of the USENIX workshop on smartcard technology* (Chicago, Illinois, USA, May 1999), USENIX Association, pp. 175–185.

[117] SHANNON, C. E. Communication theory of secrecy systems. *Bell System Technical Journal 28* (Oct. 1949), 656–715.

[118] SHIREY, R. Internet security glossary. RFC 2828, The Internet Society, May 2000.

[119] SIMMONS, G. J., Ed. *Contemporary Cryptology, The Science of Information Integrity*. IEEE Press, 1992. ISBN: 0-87942-277-7.

[120] SIMMONS, G. J. Cryptanalysis and protocol failures. *Communications of the ACM 37*, 11 (1994), 56–65.

[121] STABELL-KULØ, T. Security and log structured file systems. *ACM Operating Systems Review 31*, 2 (Apr. 1997), 9–10.

[122] STABELL-KULØ, T. Putting smartcards to use in a user-sentric system. In *Proceedings of the 2nd ACM Symposium on Handheld, Ubiquitous Computing* (Sept. 2000), P. Thomas and H. W. Gellersen, Eds., vol. 1927 of *Lecture Notes in Computer Science*, Springer Verlag, pp. 200–210.

[123] STABELL-KULØ, T., ARILD, R., AND MYRVANG, P. H. Providing authentication to messages signed with a smart card in hostile environment. In *Proceedings of the USENIX workshop on smartcard technology* (Chicago, Illinois, USA, May 1999), USENIX Association, pp. 93–99.

[124] STABELL-KULØ, T., DILLEMA, F., AND FALLMYR, T. The open-end argument for private computing. In *Proceedings of the ACM First Symposium on Handheld, Ubiquitous Computing* (Oct. 1999), H.-W. Gellersen, Ed., vol. 1707 of *Lecture Notes in Computer Science*, Springer Verlag, pp. 124–136.

[125] STABELL-KULØ, T., AND FALLMYR, T. User controlled sharing in a variable connected distributed system. In *Proceedings of the seventh IEEE international Workshop on Enabeling Technologies: Infrastructure for Collaborative Enteprises (WETICE'98)* (Stanford, California, USA, June 17–19 1998), IEEE Computer Society Press, pp. 250–255.

[126] STABELL-KULØ, T., HELME, A., AND DINI, G. Detecting key-dependencies. In *Proceedings of the Third Australasian Conference on Information Security and Privacy (ACISP'98)* (Brisbane,

Australia, July 1998), C. Boyd and E. Dawson, Eds., vol. 1438 of *Lecture Notes in Computer Science*, Springer Verlag, pp. 356–366.

[127] STEINER, J. G., NEUMANN, B. G., AND SCHILLER, J. I. Kerberos: An authentication system for open network systems. In *Proceedings of the Winter 1988 Usenix Conference* (Feb. 1988), pp. 191–201.

[128] STINSON, D. R. *Cryptography – Theory and practice*. The CRC Press series on discrete mathematics and its applications. CRC Press, 1995. ISBN 0-8493-8521-0.

[129] STUBBLEFIELD, A., IOANNIDIS, J., AND RUBIN, A. D. Using the Fluhrer, Mantin, and Shamir attack to break WEP. Technical Report TD-4ZCPZZ, AT&T Laboratories, Aug. 2001.

[130] SUN MICROSYSTEMS, INC. Java Card 2.0 Language Subset and Virtual Machine Specification. Revision 1.0 Final, Oct. 1997.

[131] SYVERSON, P., AND CERVESATO, I. The logic of authentication protocols. In *Proceedings of 9th International Conference on Cooperative Information Systems* (Trento, Italy, Sept. 2001), C. Batini, F. Giunchiglia, P. Giorgini, and M. Mecella, Eds., vol. 2172 of *Lecture Notes in Computer Science*, Springer Verlag.

[132] SYVERSON, P. F. The use of logic in the analysis of cryptograpgic protocols. In *Proceedings of the 1991 IEEE Computer Society Symposium on Research in Security and Privacy* (Los Alamitos, California, USA, 1991), IEEE Computer Society Press, pp. 156–170. A corrected discussion of many of the issues in this paper appeared in [133].

[133] SYVERSON, P. F. Knowledge, belief, and semantics in the analysis of cryptographic protocols. *Journal of Computer Security 1*, 3 (1992), 317–334.

[134] SYVERSON, P. F., AND STUBBELINE, S. G. Group principals and the formalization of anonymity. In *Proceedings of the World Congress on Formal Methods in the Development of Computing Systems* (Sept. 1999), J. M. Wing, J. Woodcock, and J. Davies, Eds., vol. 1708 of *Lecture Notes in Computer Science*, Springer Verlag, pp. 314–333.

[135] SYVERSON, P. F., AND VAN OORSCHOT, P. C. On unifying some cryptographic protocol logics. In *Proceedings of the 1994 IEEE Computer Society Symposium on Research in Security and Privacy* (Los Alamitos, California, USA, May 1994), IEEE Computer Society Press, pp. 14–28.

[136] SYVERSON, P. F., AND VAN OORSCHOT, P. C. A unified cryptographic protocol logic. CHACS Report 5540-227, Naval Research Laboratory, Washington, USA, 1996. Parts of this paper appeared in prelimary form in [140] and [135].

[137] TERRY, D. B., THEIMER, M. M., PETERSEN, K., DEMERS, A. J., SPREITZER, M. J., AND HAUSER, C. H. Managing Updates in a Weakly Connected Replicated Storage System. In *Proceedings of the 15th ACM Symposium on Operating Systems Principles* (Copper Mountain Resort, Colorado, December 3–6 1995), pp. 172–183.

[138] TOUCH, J. Report on MD5 performance. RFC 1810, The Internet Society, June 1995.

[139] VAN EMDE BOAS, P. Machine models and simulations. In *Handbook of Theoretical Computer Science*, J. van Leeuwen, Ed., vol. A: Algorithms and Complexity. The MIT Press, 1990, ch. 1, pp. 1–66. ISBN: 0–262–22038–5.

[140] VAN OORSCHOT, P. C. Extending cryptograpic logics of beliefs to key agreement protocols (extended abstract). In *Proceedings of the First ACM Conference on Computer and Communication Security* (Nov. 1993), pp. 232–243.

[141] VAN STEEN, M., HOMBURG, P., AND TANENBAUM, A. Globe: A Wide-Area Distributed System. In *IEEE Concurrency* (Jan. 1996), pp. 70–78.

[142] WEISER, M. The Computer for the 21st Century. *Scientific American* (Sept. 1991), 66–75.

[143] WESTIN, A. *Privacy and freedom*. London: Bodley Head, New York, USA, 1970. ISBN 0-370-01325-5.

[144] WILKES, M. V. *Time-sharing computer systems*. American Elsevier, New York, USA, 1972.

[145] WIN, E. D., BOSSELAERS, A., VANDENBERGHE, S., GERSEM, P. D., AND VANDEWALLE, J. A Fast Software Implementation for Arithmetic Operations in $GF(2^n)$. In *Proceedings of Advances in Cryptology—Asiacrypt'91*, K. Kim and T. Matsumoto, Eds., vol. 1163 of *Lecture Notes in Computer Science*. Springer-Verlag, Nov. 1996, pp. 65–76.

[146] WOBBER, E., ABADI, M., BURROWS, M., AND LAMPSON, B. Authentication in the Taos operating system. *ACM Transactions on Computer Systems 12*, 1 (Feb. 1994), 3–32.

[147] YAN, J., EARLY, S., AND ANDERSON, R. The XenoService - A Distributed Defeat for Distributed Denial of Service. In *ISW 2000, IEEE computer society, Boston, USA* (Oct. 2000).

[148] YEE, B. S., AND TYGAR, D. Secure Coprocessors in Electronic Commerce Applications. In *Proceedings of The First USENIX Workshop on Electronic Commerce* (New York, July 1995).

[149] ZIMMERMANN, P. *The official PGP user's guide*. The MIT Press, 1995. ISBN 0-262-74017-6.

Appendix A

A specialized One-Time Pad (OTP) was described in Section 8.1.2. The OTP has six rows:

Letter: The alphabet available to the user

Subst: The substitution table; each character from the alphabet is replaced by the corresponding number from the substitution table.

X: In this row the user writes his message

OTP: The number representing each character is added (modulo 40) to the corresponding element in the One Time Pad.

Z: The result.

Y: A secret number

On the following page, two OPTs are shown; one of them is filled out with the string "GIVE TAGE@ACM.ORG $500." to show how one would go about to send an encrypted message.

Letter	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	␣	.	$	@
Subst	05	27	13	32	03	21	16	22	00	08	26	06	04	07	18	39	30	15	19	09	37	23	24	38	17	25	14	20	10	02	31	33	34	35	12	01	36	28	11	29
X			2	3	G	I	V	E	␣	T	A	G	E	@	A	C	M	.	D	R	G	␣	$	5	0	0	.															
OTP	31	25	08	32	02	16	38	18	19	13	17	01	37	38	20	24	00	33	10	01	24	34	37	11	01	05	08	14	15	29	03	03	18	39	30	05	10	22	24	14
Z	04	17	38	11	35	34	20	05	03	35	30	23	02	04	12	17	13	00	37	35	15	02	16	29																
Y	0x8bde94b630f1504b																																							

Letter	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	␣	.	$	@
Subst	05	27	13	32	03	21	16	22	00	08	26	06	04	07	18	39	30	15	19	09	37	23	24	38	17	25	14	20	10	02	31	33	34	35	12	01	36	28	11	29
X																																								
OTP	31	25	08	32	02	16	38	18	19	13	17	01	37	38	20	24	00	33	10	01	24	34	37	11	01	05	08	14	15	29	03	03	18	39	30	05	10	22	24	14
Z																																								
Y	0x8bde94b630f1504b																																							

Figure 9.1: Two One-Time Pads, one of them filled in

Appendix B

For ease of reference we have included the 21 axioms found in the SVO logic [131], and a few detailed descriptions of semantics. The logic was first described in [136], but corrected in [131]; we use the latter.

Axioms

The axioms are referred to as Axn in the text. Modus Ponens and Necessitation is referred to as MP and NEC, respectively

Modus Ponens: From φ and $\varphi \rightarrow \psi$ infer ψ.

Necessitation: From $\vdash \varphi$ infer $\vdash P$ **believes** φ.

Belief Axioms

1 P **believes** $\varphi \wedge P$ **believes** $(\varphi \rightarrow \psi) \rightarrow P$ **believes** ψ

2 P **believes** $\varphi \rightarrow P$ **believes** P **believes** φ

3 P **believes** $\varphi \rightarrow P$ **believes** $(P$ **believes** $\varphi)$

4 $\neg(P$ **believes** $\varphi) \rightarrow P$ **believes** $(\neg P$ **believes** $\varphi)$

Source Association Axioms

5 $(P \overset{K}{\longleftrightarrow} Q \wedge R$ **received** $\{X^Q\}_K) \rightarrow (Q$ **said** $X \wedge Q$ **has** $K)$

6 $(PK_\alpha(Q, k) \wedge R$ **received** $X \wedge SV(X, k, Y)) \rightarrow Q$ **said** Y

Key Agreement Axioms

7 $((PK_\alpha(P, k_P)) \wedge (PK_\beta(Q, k_Q))) \rightarrow P \overset{F_0(k_P, k_Q)}{\longleftrightarrow} Q$

8 $\varphi \equiv \varphi[F_0(K, K')/F_0(K', K)]$

Receiving Axioms

9 P **received** $(X_1, \ldots, X_n) \rightarrow P$ **received** X_i

10 $(P$ **received** $\{X\}_{K^+} \wedge P$ **has** $K^-) \rightarrow P$ **received** X

Here K^+ and K^- are used to abstractly represent cognate keys, whether for symmetric or assymetric cryptography.

11 P **received** $[X]_K \rightarrow P$ **received** X

Poesession Axioms

12 P **received** $X \rightarrow P$ **has** X

13 P **has** $(X_1, \ldots, X_n) \rightarrow P$ **has** X_i

14 P **has** $(X_1 \wedge \ldots P$ **has** $\wedge X_n) \rightarrow (P$ **has** $F(X_1 \wedge \ldots \wedge X_n))$

The function F is meta-notation for any function that is effectively a bijection (e.g., collsion free hashes) and such that F^+ and F^- is computable in practicee by P.

Comprehension Axiom

15 P **believes** $(P$ **has** $F(X)) \rightarrow P$ **believes** $(P$ **has** $X)$

Saying Axioms

16 P **said** $(X_1, \ldots, X_n) \rightarrow (P$ **said** $X_i \wedge P$ **has** $X_i)$

17 P **says** $(X_1, \ldots, X_n) \rightarrow (P$ **said** $(X_1, \ldots, X_n) \wedge P$ **says** $X_i)$

Freshness Axioms

18 **fresh(**X_i**)** $\rightarrow$ **fresh(**$X_1, \ldots, X_n$**)**

19 **fresh(**$X_1, \ldots, X_n$**)** $\rightarrow$ **fresh(**$F(X_1, \ldots, X_n)$**)**

It must be infeasible to compute F without all the X_i.

Jurisdiction and Nonce-Verification Axioms

20 $(P$ **controls** $\varphi \wedge P$ **says** $\varphi) \rightarrow \varphi$

21 $($**fresh(**X**)** $\wedge P$ **said** $X) \rightarrow P$ **says** X

Symmetric Goodness Axiom

22 $P \overset{K}{\longleftrightarrow} Q \equiv Q \overset{K}{\longleftrightarrow} P$

Semantics

In Section 8.1.3 we refer to the semantics of SvO; the relevant parts are shown below.

Freshness

A message is fresh if it has not been part of a message sent prior to the current epoch. It is sufficient but not necessary for freshness that a message be unseen prior to the current epoch. A principal might generate a message earlier and not send it until the epoch begins. Truth conditions are thus in terms of 'what has been said' rather than 'what has been seen'.

$$(r, t) \models \textbf{fresh(}X\textbf{)}$$

iff, for all principals P and all times $t' < 0$, $(r, t') \not\models P$ **said** X.

Saying

$$(r, t) \models P \textbf{ said } X$$

iff, for some message M, at some time $t' \leq t$ in r, P sent M and X is a said submessage of M for P at (r, t'). This gives the truth conditions for P having said X at some point in the past. We also characterize what it means for P to have said X in the current epoch (typically taken to mean since the initial point of the current protocol run).

$$(r, t) \models P \textbf{ says } X$$

iff, for some message M, at some time $0 \leq t' \leq t$ in r, P sent M and X is a said submessage of M for P at (r, t').