Budapest University of Technology and Economics

Mitigating the attacks of malicious terminals

PhD dissertation

István Zsolt BERTA
MSc in Technical Informatics

Supervised by
Prof. István VAJDA
Department of Telecommunications

2005

Abstract

Smart cards, having no user interface, are unable to communicate with the user directly. Communication is only possible with the aid of a terminal, which leads to several security problems. For example, if the terminal is untrusted (which is a very typical scenario), it may perform a man-in-the middle attack.

I have created a formal model for dealing with untrusted terminals, and developed mathematical proofs on the limitations of a user in an untrusted terminal environment. Unfortunately, these limitations are too severe, so the attacks of malicious terminals cannot be fully eliminated. Thus, I elaborated solutions to mitigate the problem:

I have developed a protocol that takes advantage of the biometric abilities of the user and thus allows sending authentic messages from untrusted terminals. I have also developed a framework for the user to review signatures made in untrusted environment, and to revoke unintended signatures.

Alulírott Berta István Zsolt kijelentem, hogy ezt a doktori értekezést magam készítettem és abban csak a megadott forrásokat használtam fel. Minden olyan részt, amelyet szó szerint, vagy azonos tartalomban, de átfogalmazva más forrásból átvettem, egyértelműen, a forrás megadásával megjelöltem.

Budapest, ...

.....
Berta István Zsolt

A dolgozat birálatai és a védésről készült jegyzőkönyv a későbbiekben, a dékáni hivatalban elérhető.

Contents

Summary

Mitigating the attacks of malicious terminals

A human user would like to send sensitive messages to a remote partner through an insecure network. In order to access the network, she needs to have a terminal at her disposal. If this terminal is untrusted it may perform a man-in-the middle attack.

If the user protects her messages using a trusted smart card, then the untrusted terminal is unable to access the cryptographic keys stored in the card. However, the user still cannot control the operations the terminal performs with the card, so she still does not have any control over her cryptographic key.

I have elaborated a formal model for the above probelm. According to my model, the human user is unable to perform strong cryptographic operations. This means, she cannot encrypt long documents and she cannot compute a message authentication code with a security that the untrusted terminal cannot easily breach.

Within my model, I have proven the following statements:

- If the user is unable to securely encrypt a long document in one step, a remote partner cannot help her in establishing an encrypted channel.
- If the user cannot securely authenticate a long document in one step, a remote partner cannot help her in establishing an authenticated channel.

Within my model, I have proven that the user cannot solve the above problem, so I sought solutions outside the boundaries of my model. I have elaborated two practical solutions for mitigating the problem of sending authenticated messages:

- I proposed a framework for the revocation of unintended digital signatures that were initiated by malicious terminals.
- I proposed a solution where the user authenticates her messages using her biometric abilities. Later on, the biometric signature of the user is protected by the digital signature of the smart card, so the two signatures prove the authenticity of the message together.

Both of my solutions can be used by non-professional users too.

Összefoglaló

Védekezési lehetőségek a rosszindulatú terminálok támadásai ellen

(Mitigating the attacks of malicious terminals)

A felhasználó, egy mindennapi ember, üzenetet kíván küldeni egy távoli félnek. Ahhoz, hogy a hálózaton kommunikálhasson, terminálra van szüksége. Ha csak nem biztonságos terminál áll a rendelkezésére, kiszolgáltatottá válik annak man-in-the-middle támadásaival szemben.

Ha a felhasználó egy megbízható chipkártya segítségével védi üzeneteit, akkor a kártyán tárolt kulcs nem kerülhet a terminál birtokába. Ugyanakkor, a felhasználónak továbbra sincsen lehetősége annak ellenőrzésére, hogy a terminál milyen műveleteket végez a chipkártyával, vagyis az azon tárolt kriptográfiai kulccsal.

Formális modellt dolgoztam ki a fenti problémára, amely szerint a felhasználó nem képes erős kriptográfiai művelet végrehajtására, vagyis nem képes sem hosszú dokumentumokat titkosítani, sem hosszú dokumentumokra dokumentumra kriptográfiai ellenőrző összeget (MAC) számítani úgy, hogy azt a terminál ne tudná jelentős valószínűséggel sikeresen támadni.

A fenti modellemben a következő két állításra adtam formális bizonyítást:

- Ha a felhasználó nem képes a dokumentumot egy lépésben titkosítani, akkor egy távoli fél sem képes neki segítséget nyújtani a titkos kommunikáció felépítésében.
- Ha a felhasználó nem képes a dokumentumot egy lépében hitelesíteni, akkor egy távoli fél sem képes neki segítséget nyújtani a hiteles kommunikáció felépítésében.

Bebizonyítottam, hogy modellemben a felhasználó nem képes megoldani a problémát, így a modellem határain kívül kerestem megoldást a probléma enyhítésére. Két gyakorlati megoldást dolgoztam ki a hiteles üzenet küldésének esetére:

- Olyan keretrendszert dolgoztam ki, amelyben a felhasználó a csaló terminálok által kezdeményezett aláírásokat – szigorúan szabályozott feltételek között – visszavonhatja.
- Olyan megoldást dolgoztam ki, amelyben a felhasználó saját maga, biometriai képességei segítségével hitelesíti az üzenetet. A felhasználó “biometriai aláírását” később a chipkártya kriptográfiai aláírása is megvédi, és a dokumentum hitelességét e két aláírás együttesen igazolja.

Mindkét megoldás alkalmas rá, hogy laikus felhasználók is igénybe vegyék.

Acknowledgements

First of all, I would like to gratefully thank my supervisor, Prof. István Vajda for his helpful guidance and continuous support. Without his help and inspiration the results presented in this thesis would have never been reached.

I am also grateful to Dr. Levente Buttyán for his valuable comments and suggestions, and for encouraging me to further investigate the perils of untrusted terminals.

I would also like to thank Boldizsár Bencsáth and all the people who established the creative atmosphere in the Laboratory of Cryptography and Systems Security.

Finally, I would like to thank my family – my father, my mother and my sister – for their continuous love, support and encouragement, without them my work would have never been completed.

Part I
The Problem

1 Introduction

Electronic commerce applications require participants to send sensitive information over a network. For example, if Alice would like to buy some goods from Bob via the Internet, she needs to send her order and payment information (e.g. her credit card number). Generally, the Internet is an insecure medium, so whenever sensitive information is transmitted it needs to be protected. Cryptographic protocols provide a widespread way to protect the confidentiality and integrity of such information.

Cryptographic protocols are usually described between abstract hypothetical entities like 'Alice' and 'Bob'; in practice they are usually processes running on computers. As long as electronic commerce is performed between computers, both parties can authenticate each-other using challenge and response mechanisms, and can exchange information through an encrypted and authenticated channel.

However, it is a particularly interesting case when Alice is a human. A sole human needs a computer, a *terminal* to take part in such a protocol. On the one hand, she needs the terminal to connect to the network. On the other hand, she needs the terminal to perform complex cryptographic operations like encryption or digital signature. If a protocol participant is a human, it is implicitly assumed that she has a computer at her disposal. It is also assumed that she trusts her terminal for following the cryptographic protocol and for not mounting an attack.

Within this dissertation that practical scenario is considered when the above statement is not true, and the terminal is untrusted, i.e. Alice assumes that her terminal performs an attack.

1.1 An example of an attack

Let us consider Protocol 1, when the human user Alice would like to send a digitally signed message to Bob. Alice has a terminal that stores her private key. She types her message using the keyboard of the terminal.

Protocol 1. – The protocol for trusted terminals

Step 1 Alice $\rightarrow$ Terminal: message m

Step 2 Terminal $\rightarrow$ Bob: message m signed with the private key of Alice

---

If we assume that the terminal is not malicious, the protocol is secure. However, if the terminal is malicious (either because it is infected by a virus or because it is under the control of an intruder from the network), it may perform an obvious attack: it can replace message m with m' where m' is a message that Alice would not sign. [Schneier and Shostack, 1999]

Many countries have laws that accept certain versions of digital signatures to be equivalent with handwritten ones (e.g. [EU Directive, 1999], [Hungarian Law, 2001]), so the untrusted terminal may make Alice sign an arbitrary legally binding statement.

A key problem with Protocol 1 is, that the private key of the user is stored by the terminal. This way, the terminal may abuse the private key of the user and sign an arbitrary message whenever it chooses.

In the following example, user Alice has a smart card that stores her private key. A smart card is a trusted, tamper-resistant microcomputer with the size of a plastic card. The private key of Alice is generated on the smart card, and never leaves it. If Alice would like to sign a message with her card, she sends the message to the card and receives the signature in return. [Berta and Mann, 2000a] A smart card is protected by a PIN code, so a thief who steals the card cannot sign messages in the name of Alice. Having introduced the smart card, Protocol 1 can be extended as follows:

Protocol 2. – Storing the private key on a smart card

Step 1 Alice → Terminal: message m, PIN code

Step 2 Terminal → Card: message m, PIN code

Step 3 Card → Terminal: message m signed with the private key of Alice

Step 4 Terminal → Bob: message m signed with the private key of Alice

---

This way, if the terminal is stolen, the attacker cannot extract the private key of Alice from it. However, a terminal that is malicious at the time of the signing can still mount an attack. The malicious terminal can still replace the message m with message m' Alice would not sign. Moreover, it can sniff the PIN code of the user in Step 1, so it can initiate signature operations if the card is accessible.

Although PIN codes are useful against e.g. card theft, they provide little protection against the threat of untrusted terminals, so their use is not discussed in this work.

1.2 An attack tree of the problem

Attack trees were proposed by Schneier in [Schneier, 1999a], they are a simple tool for assessing the security of a system. An attack tree represents attacks in a tree structure. The goal of the attacker is in the root of the tree, and each node is a task an attacker may take towards this goal. Leaves represent atomic tasks while nodes represent more complex ones. There are two kind of nodes in the tree: 'and' nodes and 'or' nodes. In order to perform the task in an 'and' node, an attacker needs to perform the tasks in each subnodes. In order to perform the task in an 'or' node, an attacker needs to perform a task in one of the subnodes. Schneier also proposed that various attributes (e.g. cost of a certain task) can be assigned to each leaf, and thus the cost of performing the whole attack can be estimated. Thus, they can be used to evaluate, how cost-effective the security measures in a security system are.

Attack trees are scaleable, so the root (result) of one tree can be used instead of a leaf in another one. Perhaps, the main advantage of attack trees is that they are easy to use and understand, so they provide a plausible way to illustrate the amount of costs required to attack a system.

A general attack tree for smart cards was proposed in [Kincses, 2004]. I do not try to provide a general attack tree for all aspects of smart cards security. Figure 1 shows an attack tree where the goal of the attacker is to obtain a digitally signed message in the name of the user in a public key infrastructure.

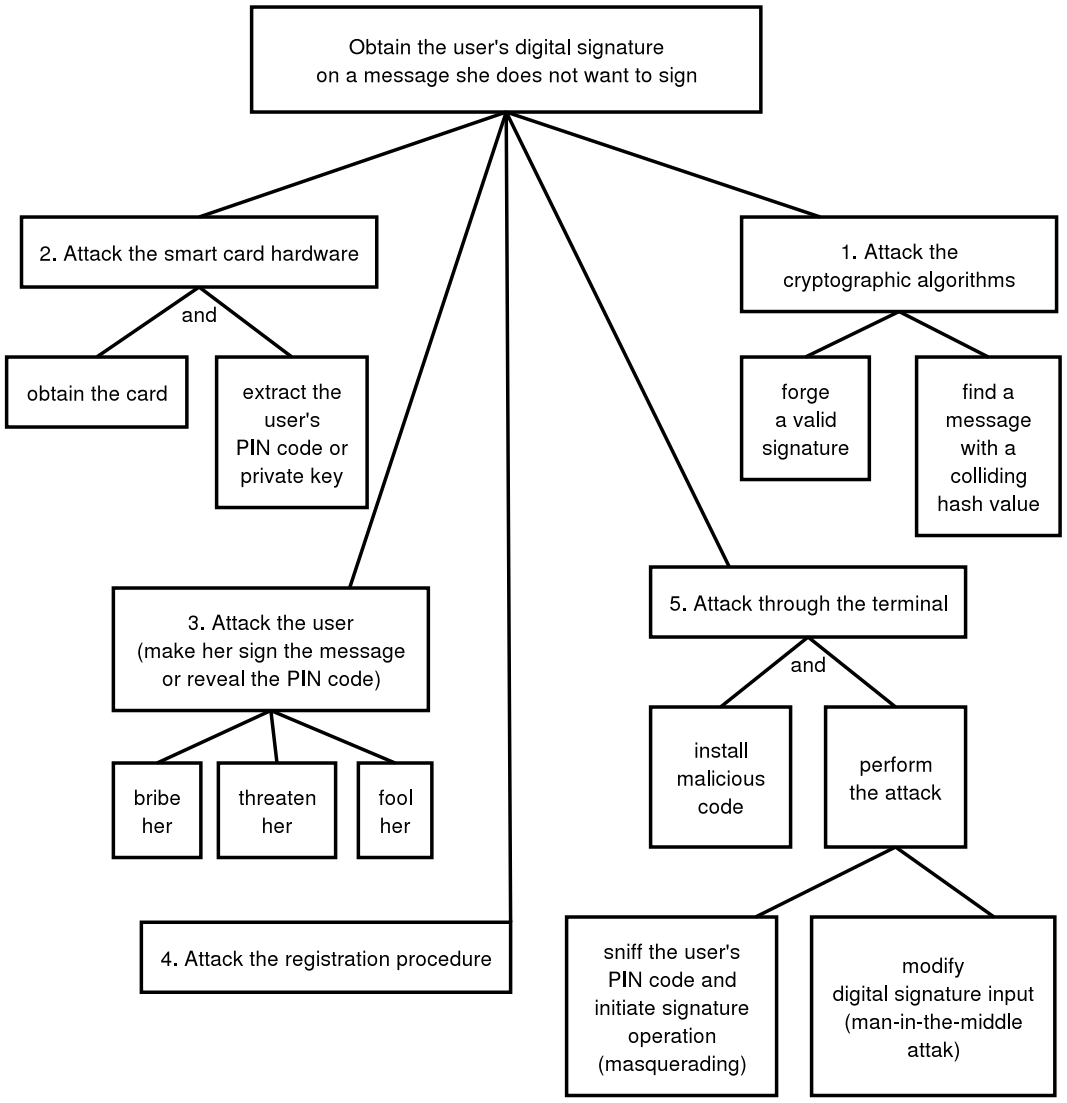

Figure 1: Attack tree – Obtaining a digitally signed message in the name of the user

The tree illustrates that various parts of the system can be attacked to fulfill this goal:

1. The attacker may choose to attack the cryptographic algorithms. One possible target is

the public key algorithm that is used to compute the digital signature (e.g. RSA with an 1024-bit-long key) or the hash function (e.g. SHA-1).

2. The attacker may also choose to attack the smart card hardware or software. First, the attacker needs to obtain the card, then the attacker needs to extract the user's private key or make the card perform the digital signature operation.

3. The attacker may choose to attack the user. There are a wide range of possibilities ranging from fooling her into signing m', to forcing or bribing her. Another possibility is to make her reveal her PIN code and then steal her card.

4. Another possibility is to attack the registration procedure of the authority that issues the card (or the certificate of the user). If the attacker can masquerade the user towards this authority or can bribe or blackmail registration officers, the attacker can sign messages in the user's name.

5. Finally, the attacker may attack the user's terminal, so that she cannot control what messages she signs. The attacker may mount the attacks described in Section 1.1

Within this dissertation I do not try to approximate the values of the cost of various attacks. However, I would like to highlight that the cost of the attack is in a different magnitude in the five main branches of the attack tree.

1. Perhaps, attacking the cryptographic algorithms has the highest cost among all branches. Very strong algorithms exist that are thoroughly evaluated by the academic community, attacking them is infeasible for probably all attackers.

2. Smart cards are designed to protect the information they store. They are constructed in a way that it should be very hard to extract information from them. Moreover, if the card detects that it is being decomposed, it may destroy the information it stores. [Rankl and Effing, 1997], [Zoreda and Oton, 1994], [Smith et al., 1999], [Yee and Tygar, 1995]

   Time-to-time, experts find various tricks to attack the cards and sometimes even find vulnerabilities that could be exploited in a feasible way. (e.g.: [Kelsey et al., 2000], [Skorobogatov and Anderson, 2002]) However, an attacker still needs considerable expertise to exploit the – often merely theoretical – vulnerability of a certain device. Just as Shaw described the evolution of security mechanisms [Shaw, 1905], experts at card manufacturers develop countermeasures against these vulnerabilities. (see e.g. [Izu et al., 2004], [Karpovsky et al., 2004], [Anderson and Kuhn, 1997], [Fung et al., 2001]). I reckon, even if there are some feasible attacks against certain specific smart card platforms, challenging the constantly developing technology of card manufacturers is a difficult task.

   Although, not as infeasible as attacking the cryptographic algorithms, attacking the hardware or software of the smart card is still beyond the capabilities of most attackers.

3. Compared to 1 and 2, attacking the user is very easy. Generally, this part of the system is not protected by IT countermeasures, but by making users more security conscious. [Berta et al., 2005a] An organization may form policies and rules on how to react e.g. to various attempts to fool the user. The organization should motivate its users well-enough (by both financial and non-financial methods [Torrington and Hall, 1987]) so that it is harder to bribe the users. Still, it is very expensive to make countermeasures in this branch ultimately strong, because e.g. if a user's family is held kidnapped, most users would sign whatever the kidnapper demands.

4. Generally, it is significantly harder to fool or bribe trained registration officers than non-professional users. However, I reckon this is still a feasible option for most attackers.

5. Attacking the terminal is a very easy and straightforward option, and is very difficult to protect against. Computers run operating systems that are far from perfect from the security point of view. In many systems, an attacker can gain control of the PC remotely by exploiting a known vulnerability. In case of certain attacks, no interaction from the user is needed (i.e. the user does not need to click on anything). Attackers developed malicious software to perform these attacks at large scale, at an automated way. Such viruses and worms attack computers at random IP addresses and infect the computer, install themselves on it, and continue to propagate. The state-of-the-art malicious software can infect hundred thousands of computers in a couple of hours. [Zou et al., 2003], [White et al., 2003], [Dantu et al., 2004] Many viruses open backdoors on computers they infect, so the author of the virus can take control of the computer easily. (Perhaps, modern viruses are merely tools to prepare large scale denial-of-service attacks. [Bencsáth, 2004], [Bencsáth and Vajda, 2004]) As details of viruses become public, any attacker can use these backdoors. Thus, infected computers are relatively easy to take control of.

   According to [Loney and Lemos, 2004], computers connected to the Internet are attacked about every 20 minutes. Although most operating system vendors release patches regularly for newly discovered vulnerabilities, often there is not enough time to download patches; the computer gets infected before the path could be installed. [SANS, 2004]

   An attacker can reach the same result this way (i.e. the attacker can obtain the users signature for an arbitrary message) as if it broke the cryptographic algorithm.

Clearly, the two weakest branches of the tree are branches 3 and 5. This dissertation focuses on countermeasures against branch 5.

As long as we cannot guarantee that our own terminals do not attack us, it has little sense to invest a vast amount of money in developing stronger cryptosystems, more sophisticated smart cards, or in training and motivating users, as the attacker will choose the least expensive routes from the leaves to the root of the tree in order to attack the weakest link in the system.

This led to the statements of [Schneier, 2003]: 'SSL doesn't solve an important security problem' and 'Nobody bothers eavesdropping on the communications while it is in transit.' and 'By now it should be obvious that hackers don't steal credit card numbers one by one across the network;

they steal them in bulk – by the thousands or even millions – by breaking into poorly protected networks.'

1.3 Which terminals should a user trust?

Theoretically, a user should trust a terminal only if she verified that neither the hardware nor the software of the terminal has any hidden features. Unfortunately, not even a completely open source can fulfil the above criteria, because most systems are too complex, and not even a professional user can verify them without outside help. In practice, the user has to make a compromise.

Various users possess different skills and different background knowledge and a different degree of paranoia that they rely on when judging a terminal. Whether a terminal can be trusted or not also depends on the purpose for which the user intends to use it. (For example, assume that a user perceives that there is a small chance for her home computer to be infected by a virus. She may decide to accept this risk when she makes micropayments with her smart card, but may decide to compute digital signatures at a more secure device.)

Deciding whether a user trusts a terminal or not depends solely on the user. Thus, *the same terminal may fall into different categories for different users and for different situations*.

In order to decide whether to trust a terminal or not, the user should answer two questions:

1. Does the user trust the intentions of the entity operating the terminal?
   The user answers 'yes' if she assumes that *the operator of the terminal will not mount an attack* on her. Naturally, this implies that the user recognizes the entity (person or organization) responsible for that particular terminal.

   She may answer 'yes' in case of her own home computer or in case of an ATM of the bank next door. She may answer 'no' in case of an ATM of an unknown bank, or in case of a mobile phone. Still, some users will answer 'yes' in these latter cases too. Some users may claim that although they recognize the logo of a known bank on an ATM, they believe that the logo could have placed on the ATM by an attacker too (i.e. they cannot authenticate the terminal).

2. Does the user assume that the entity operating the terminal is able to protect the integrity of the terminal?
   The user answers 'yes' if she assumes that *the integrity of the terminal has not been violated*.

   Sometimes an attacker leaves obvious traces when violating the integrity of the terminal. If the user finds such traces, she should always answer 'no'. However, the signs are often not obvious, especially when it is only the software of the terminal the attacker tampers with.

   Another way of deciding this question is assuming that certain terminal types are tamper-resistant. For example, a user may assume that an ATM or a 'trusted reader' is tamper-resistant, while in case of PCs the user may assume that there is a virus running on it.

Some users may consider mobile phones to be secure, but some may suppose that even a phone can be tampered with.

Some users may assume that their home PC has not been tampered with, while some others are too cautious to make this assumption. In order to prevent malicious code from running on her computer, the user at least needs to the following. [Mossberg, 2004]

- install and configure an antivirus software
- use anti-spyware software
- install and configure a personal firewall
- use a safe browser and email client, and configure them not to run any active content (e.g. ActiveX, Flash, Javascript, etc.)
- update and patch her system regularly
- use the system with some common sense. (e.g. not to download software from untrusted source; though it is not always obvious to identify trusted software [Albert et al., 2002])

Clearly, most users do not do all of the above, most of them are not even conscious of what they should do. However, for certain purposes, certain users may consider their home computer to be trusted.

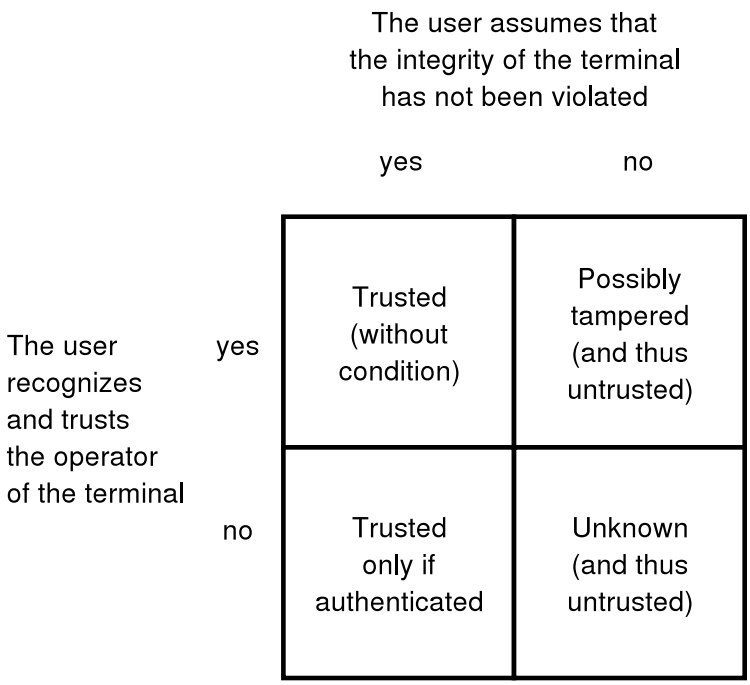

Figure 2: Trusting the intentions vs trusting the expertise of the terminal operator

Thus, whether the user trusts the intentions or the expertise of the terminal operator, are two distinct questions. Based on possible answers to these two questions, four categories of terminals can be identified that require different treatment (see Figure 2):

- Terminals that fall into Category *'trusted'* can be trusted and used for any sensitive purpose. Unfortunately, a cautious user can consider very few terminals to belong into this category. (Some users may consider their home PC or a known ATM next door to fall into this category.)

- A terminal in Category *'possibly tampered'* should not be trusted, countermeasures in this dissertation are applicable. (Note that whether a user's home PC may either fall into this category or into Category *'trusted'*, depends solely on the expertise and self-confidence of the user.)

- A user needs assistance to authenticate a terminal in Category *'trusted only if authenticated'*. Once authenticated, the user may decide to trust the intentions of the operator of the terminal and consider it as if it was belonging to Category *'trusted'*. However, if the user does not decide to trust the terminal operator after authenticating it, or the user cannot authenticate the terminal, she should consider the terminal untrusted, and use countermeasures for Category *'unknown'*.

  (An unknown ATM may belong to Category *'trusted only if authenticated'*, if a user would like to avoid fake ATMs that steal the card and PIN of users. [Anderson, 1996] Another example could be a PC that may belong to a trusted person. Once this person assures us that his or her machine is 'safe' we may use it as if it was our own, because we trust his or her expertise and we also trust him or her for not mounting an attack against us.

  Note that while some most paranoid users trust nothing but their own 'fortified' computers, a user's home PC is the least likely to have a certificate to authenticate itself according to [TeleTrusT WG2, 2003]. According to [CEN CWA 14890-1, 2003], there is no need for authenticating the terminal in a 'trusted' environment, and it is up to the user to decide whether an environment is trusted or not.)

- A terminal in Category *'unknown'* is untrusted by nature. Countermeasures in this dissertation are applicable. (The typical example for this is a PC in an Internet café.)

Unfortunately, when closely examined, few terminals can be considered trusted. If there are any trusted secure terminals, there can be very few of them only, because operating such a terminal is very expensive. Few entities posses the resources and the expertise to operate such a device.

Naturally, a user has to find a reasonable tradeoff between security and costs in practice. The user (or the entity operating the terminal) should evaluate how sensitive transactions are done on a particular terminal, and how much resources can be sacrificed to keep the terminal secure.

Probably, a computer used only for gaming does not need very expensive protection. However, if a user is willing to use her smart card to compute a qualified digital signature at the terminal untrusted, she is risking as much as she would risk when signing a document she has not read.

The degree of protection the terminal needs (i.e. the amount of confidence the user would like to gain in the terminal) depends on for what purpose she is using the terminal.

1.4 Why do people use untrusted terminals?

Using untrusted terminals for sensitive purposes is dangerous. Still, we are all using untrusted terminals all the time.

Probably, the main reason for this is that *we often do not have any other option*. Certain services can be accessed only via the terminal of the service provider. (Naturally, the service provider does not advertise his or her terminal as 'untrusted'.) For example, if a user would like to ride a bus, she needs to use the bus to access this service regardless of how secure notebook or PDA she has. Similarly, if a user would like to drink coffee, she needs to approach the coffee machine. In these cases, the PDA or notebook of the user does not have the peripherals necessary to provide the appropriate service. Theoretically, it could be possible to pay via a trusted terminal and access the service of the service provider, but there is no low-cost standardized portable terminal that is widespread enough that all users can have it.

Smart cards mean one important step into this direction. Although they are low-cost and standardized and portable, they do not have a user-interface on their own, so the user still has to rely on the service providers terminal. Unfortunately, this implies that the smart card is of little help to the user.

Web-based payments are also a step towards this direction. The user can make the payment via her own computer – that she assumes to be trusted – and access the service via the service providers distribution channels. For instance, ordering a DVD via the web instead of buying it from the shop next door is an example for this. We do not rely on the merchant's terminal to handle our credit card, but we make the transfer via the web. Unfortunately, a magnetic stripe card cannot give too much protection either way. Any party who knows the credit card number (and user name and validation date) can make payments in the cardholder's name.

The other main reason for so many people using untrusted terminals is *that attacks in the past caused relatively little losses*. Credit card companies suffer losses from credit card fraud and phising ([Kay, 2004]), but – naturally – they will not make any significant countermeasures (like switching from magnetic stripe cards to more secure smart cards) until the cost of the countermeasure is lower than the cost of the fraud. When I interviewed a marketing expert of the banking sector, she told me that they had trouble penetrating the market for PKI services in the home banking sector. She argued that both users and banks were satisfied with the current username and password based solutions because there was little or no example of fraud in such solutions. [Berta, 2004]

Although past attacks may have caused relatively little losses, this may not be a sound argument for making decisions about the future. On the one hand, IT systems are being used for more and more sensitive purposes by the man-of-the-street too. One good example for this is the area of home banking or internet banking systems that are getting widespread, another example is the field of qualified electronic signatures. Within both areas, a single attack can be devastating for

the user (compared to e.g. smart phone cards where the user risks only the amount of money stored in the card).

On the other hand, as systems are becoming more and more standardized, attackers can launch large-scale attacks against the global community of users. For example, in case of a standard interface to an internet bank, an attacker can write a worm that infects millions of computers and launches bank transfers in the name of users of the infected systems.

In spite of the above future aspects, it is common today, that all but few users are using untrusted terminals day by day.

1.5 Summary of the problem

- When used for security-sensitive tasks, untrusted terminals may pose a dangerous threat.
- When closely examined, surprisingly many terminals fall into the ’untrusted’ category.
- Operating a trusted and secure terminal is very costly.
- In practice, the user needs to make a compromise between costs and security. She needs to answer estimate, what degree of security she needs for a certain task.
- Often, we do not have any other option than using an untrusted terminal.

When using an untrusted terminal for a security-critical task, the human user needs additional protection.

2 State-of-the-art

In this section, I review various solutions from literature and from practice that aim to provide countermeasures against attacks from malicious terminals.

2.1 Literature review

2.1.1 Identification of untrusted terminals

Terminal identification is perhaps the most basic problem addressed in the literature. It aims at distinguishing between terminals of various trust levels. In the most simple model, terminals are categorized into two main groups: trusted terminals and untrusted terminals. Users trust terminals in the former group completely, while they want to avoid terminals in the latter group. This branch of literature gives guidance for users for terminals that fall into *’trusted only if authenticated’*of Section 1.3.

In order to help users to distinguish between trusted and untrusted terminals, terminals are required to authenticate themselves before they are used. It is assumed that only trusted terminals

are able to authenticate themselves correctly. It must be noted that trusted terminals have to be tamper resistant, otherwise tampered terminals could serve an attacker while still being able to authenticate themselves.

Asokan et al. [Asokan et al., 1999] and Rankl and Effing [Rankl and Effing, 1997] show a simple protocol that – using smart cards and one-time passwords – enables the identification of fake terminals. In their solution, a secret password is shared between the user and the smart card. The card only presents the password to the terminal if the terminal has identified itself correctly by a challenge and response method. The user accepts those terminals as trusted ones that are able to present the password. However, this approach is still vulnerable to tampered terminals and terminal-in-the-middle attacks. While the latter can be prevented by distance bounding protocols [Brands and Chaum, 1993], the detection of tampered terminals is practically impossible for users, as it needs external access to the hardware of the terminal [Thompson, 1984].

The work of Asokan has been incorporated into modern smart-card standards. For example, [TeleTrusT WG2, 2003], [CEN CWA 14890-1, 2003] and [IBM Magyarország Rt. et al., 2004] all include a so-called 'display message' that the terminal can access only after it was authenticated by the card via a certificate based challenge and response method.

My work relies on the work of Asokan et al.; e.g. in Section 7, I assume that there is a group of trusted terminals that the user is able to identify. However, in contrast to the work of Asokan et al., the goal of the user in my model is to use untrusted terminals (even those of Category *'unknown'*) and execute certain sensitive operations (like digital signatures) on them.

2.1.2 Sending messages from untrusted terminals

This section lists literature that propose solutions for terminals of Categories *'possibly tampered'*and *'unknown'*in Section 1.3. In these cases the user assumes that the terminal may mount an attack, but would still like to use the terminal for sending sensitive messages.

Solutions based on futuristic devices

The problem of man-in-the-middle attacks of untrusted terminals was addressed by Abadi et al. [Abadi et al., 1992] first, by analyzing the dangers of delegation of rights to a terminal. They show that this problem could be solved with a smart card that has peripherals to communicate directly with the user, and also show secure protocols for such a device. Later on, they strip as much of these peripherals from the card, as possible. They prove that with the resulting card that has no clock and no keyboard but has a display only, the same degree of security can be implemented without placing too much load on the user. However, after more than 10 years of development, the smart card with a display is still not a feasible assumption.

Similarly, Gobioff et al. [Gobioff et al., 1996] analyze various hypothetic smart cards having secure input or output channels, and identify various classes of equivalence between them. For example, they show that a smart card with a private input channel (keyboard) is equivalent with one with a private output channel (display). Their contribution adds rather little to that of Abadi et al.

Balfanz and Felten [Balfanz and Felten, 1999] show that a trusted PDA with a trusted user interface could be more secure for generating digital signatures. Their work is an implementation that supports the principles of Abadi et al. by evidence, but does not extend them by any means. However, they also raise the question of whether a PDA could be considered a trusted device. Moreover, a PDA is very expensive compared to a smart card, so organizations (like banks) are unlikely to equip there users with PDAs.

The solution of Clarke et al. [Clarke et al., 2002] uses a super-smart card, a device equipped with a digital camera, which is connected to the network while continuously monitoring the screen of the terminal. This camera-based device analyzes the contents of the screen, and compares it with the data received on an authentic channel. The device warns the user via a LED display in case of any difference. Although this device is currently technically infeasible, this solution would enable authentic communication without requiring the user to perform any calculations. However, as pointed out by Rivest in [Rivest, 2001], there remains a fundamental conflict between having a secure device and having a reasonable customizable user interface. He argues that complex interfaces make a device hard to evaluate and more vulnerable to various attacks.

In contrast to the above solutions, which are based on super-smart cards, in my model, the card does not need to have a user interface or any special peripheral. I propose solutions that can be implemented using realistic smart cards that exist today.

Solutions based on realistic devices

Stabell-Kulo et al. [Stabell-Kulo et al., 1999] proposed a protocol for sending authentic messages from untrusted terminals. Their solution gains authenticity by encryption using a one-time-pad together with a monoalphabetic substitution table. They use a smart card to sign the message and a trusted third party to certify the card's signature. Unfortunately, in case of long messages the user is not able to memorize one-time keys, so the solution of Stabell-Kulo et al. only works with short messages. In contrast to the work of Stabell-Kulo et al., my solutions do not require the user to perform cryptographic operations or to memorize long cryptographic keys.

[Gruschka et al., 2004] and [Girard et al., 2004] more or less simultaneously proposed solutions where the smart card verifies a document in an XML format. Both work suppose that a trusted reader exists in the system with trusted keyboard (and in case of Gruschka et al. a small trusted display too). They assume that the user prepares the message on the convenient environment of the untrusted PC, and 'the most important part' of the message is verified by the user via the trusted peripherals of the reader. The main difference between the solutions is the methodology of how 'the most important part' of the message is selected.

In the solution of Gruschka et al., the card verifies that the document has a certain XML structure (i.e. it conforms to a predefined XML scheme; e.g. it is a bank transfer in an XML format), and refuses to sign arbitrary documents. The card selects the most important fields from the document (e.g. the amount to be transferred and recipient of the transfer), and asks the user to confirm these most important fields via the display and keyboard of the trusted reader. The main

drawback of this solution is that the user cannot use her digital signature as a handwritten one; she cannot sign arbitrary documents, she can only sign 'forms' predefined by the XML scheme.

The solution of Girard et al. allows the user to sign arbitrary XML documents, and lists a couple of methods for selecting that part of the document that is to be verified by the user. Girard et al. argue that if the words to be verified are selected by the card at random, then there it is possible that the important part of the document is signed without being protected. The authors also note that a malicious user may intentionally select unimportant words to confirm and later claim to be a victim of a troyan horse.

A major drawback of both solutions is that they assume that a trusted reader is present. Generally, when a user approaches a terminal, the user does not have the possibility to provide her own reader. I reckon, she is unlikely to be able to verify that a reader is trusted and it has not been tampered with.

In my solutions, I do not assume that a trusted device is present to aid the user.

Solutions based on humans only

Since smart cards did not solve the problem of untrusted terminals, another idea emerged. Pencil-and-paper cryptography (or human-computer cryptography) tries to give the user a method to protect the secrecy or authenticity of the message alone, without the help of a smart card. Among historical methods (like the book cipher [Kahn, 1967], [Singh, 2000]) the one-time-pad can be considered quick and easy enough for the limited computational power of the human. However, in case of long messages the user would need secure storage space for long one-time keys. In addition, there is a key exchange problem (keys must be set up manually before the communication) which makes this approach unsuitable for many applications.

The solitaire algorithm of Schneier [Schneier, 1999b] provides strong encryption, and uses a deck of card for keying. The key is the initial order of the deck. As the deck is shuffled, it is used as a pseudo-random number generator. Solitaire is a stream-cipher that modularly adds the output of this PRNG to the plaintext. Although it is optimized for use by humans, in case of long messages, encryption requires a significant amount of time, so this algorithm is more suitable for secret agents than every-day people.

Methods proposed by Naor and Pinkas [Naor and Pinkas, 1997] rely on visual cryptography ([Naor and Shamir, 1995]), which uses transparencies placed on the computer's screen. Their algorithm relies on a one-time-pad, where the XOR operation is accelerated by the fast visual processing of the human being. The key is composed by the transparent and non-transparent sectors on the transparencies. The required key-size is very large (especially for long messages), so users cannot memorize these keys. Methods of Naor and Pinkas enable a remote partner to send authentic messages to a user at an untrusted terminal, or to identify the user in a secure way, while the basic visual cryptography enables private communication towards the user.

Matsumoto [Matsumoto, 1996] developed a human identification scheme that enables challenge and response identification of humans at untrusted terminals. His solution relies on an assumption that humans can easily understand and 'decode' certain images, while computers have trouble with them. The remote partner transmits a one-time key via such 'questions', and

the user combines the answer with this one-time key. He suggests that such a scheme could be used for encryption too. However, such a scheme would require the remote computer to select 'questions' from a significantly large space, which can be problematic. Moreover, the scheme can be undermined if the attacker can use human interaction too.

In contrast to the above solutions, my solutions presented here do not require the user to perform any cryptographic operations herself. In Section 8 I do make use of abilities and properties that are unique for the human: I use biometry that none of the above solutions addressed.

Pering et al. [Pering et al., 2003] provided a biometry-based solution that allows the challenge and response authentication of users through untrusted terminals. A set of photographs are provided as a challenge, and the user has to select her own personal photographs from this set.

In contrast to the work of Pering et al., the solutions I propose allow sending authentic messages from untrusted terminals.

2.2 Countermeasures in practice

2.2.1 Fallacious countermeasures

Several vendors advertise their products in a way that the user believes it protects against untrusted terminals. In many cases the situation is not this, and the user falls victim of the vendor's marketing tricks. In this section, I have listed some fallacious countermeasures that some vendors can advertise in a dim and mystifying way to make the user think it provides a secure environment for digital signatures.

Note that some of these countermeasures may be useful against other kind of threats, but they are all useless against untrusted terminals.

- *Authenticating the user* means that the smart card that performs the digital signature operation gains confidence that the user initiated the digital signature operation. This can be an important security countermeasure against many threats (e.g. card theft), but – without combining it with other countermeasures – it *is useless against untrusted terminals*. Even if the smart card is sure that the message was originally coming from the user, it detect if a malicious terminal modifies it before it reaches the card.

  Such methods are for instance:

  - PIN code: PINs do not give any protection against untrusted terminals. The terminal may sniff the PIN code or perform a man-in-the-middle attack.
  - One-time-PIN: This is an expensive security measure; though, it does not give any protection against the terminal's man-in-the-middle attack.
  - '*Trusted readers*' – readers with PIN pads: The PIN reaches the card securely, but the terminal can easily modify the digital signature input. No protection against untrusted terminals. (Another common disbelief is that the 'trusted' reader is secure, so an attacker cannot modify it. Even if this is true, an attacker is unlikely to

modify the reader, the attacker will modify the terminal the user prepares the digital signature input at. Terminals are more standardized, they are much easier to modify.) [FINREAD, 2004]

- Biometrical user-authentication: Even if the card can detect if the terminal replays a previously recorded sample of the user, the terminal can easily mount a man in the middle attack by modifying the digital signature input coming from the user.

- *Requesting confirmation from the user* means that whenever the user initiates a digital signature operation, the user has to state: 'Yes, I want to sign this message.' For example, a window could appear on the screen of the terminal to ask the user's consent.

  As long as this task is performed by the terminal, *this countermeasure is useless*. A malicious terminal will not follow the protocol, it will not ask the confirmation from the user, or it it may simply replace the digital signature input. (If this task is performed by a trusted device, than we do not speak of a user in an untrusted terminal environment anymore.)

- *Requesting confirmation from a trusted third party* is not much different from the former scenario. Main question is: Does the user have any other way for communicating with the trusted third party than using the untrusted terminal? If not, than communicating with the trusted third party is the same problem as communicating with the smart card in the user's pocket. If the user has other means, then the user is not in an untrusted terminal environment anymore.

  A widespread solution following this paradigm is the one known as *secure messaging* or *secure channel*. This means, a smart card performs a certificate-based mutual authentication with a so-called IFD (interface device), they exchange secret keys, and use these keys to encrypt and authenticate their communication. (The concept is similar to that of SSL/TLS and SSH. [Blake-Wilson et al., 2003], [Ylönen et al., 2000])

  The IFD is a secure module inside the terminal or inside a remote server. Figure 3 shows the former scenario while Figure 4 shows the latter one. In both figures, solid lines mean protected (encrypted and authenticated) channels, while dashed ones mean unprotected channels.

  As it can be seen on both figures, communication between the IFD and the smart card is secure: it is encrypted and authenticated. However, secure messaging does not address the problem of secure communication between the cardholder and the IFD, so it cannot protect any information flowing from the cardholder towards the smart card and vice versa. Secure messaging cannot protect the PIN code (without a trusted reader), it cannot protect the digital signature input (coming from the user) and it cannot protect the confidentiality of decrypted messages (passed to the user) on the channel between the cardholder and the IFD.

  While secure messaging can prevent an attacker from eavesdropping between the card and the card reader, an attacker is unlikely to mount such an attack because it requires sophisticated hardware and it is significantly cheaper to attack via the terminal.

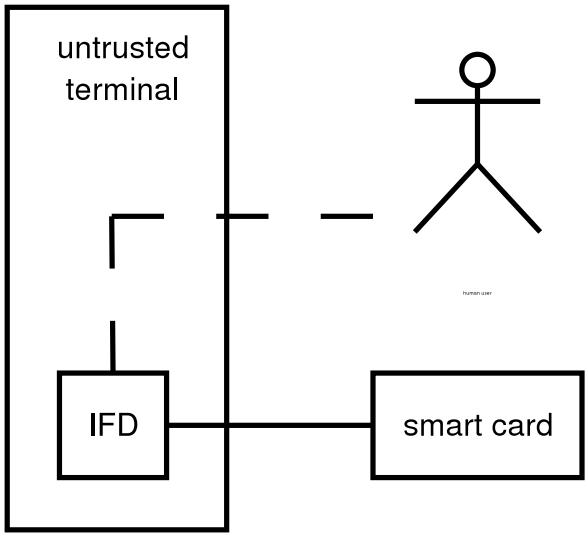

Figure 3: Using 'secure messaging' – the IFD is a secure module in the terminal

Today, secure messaging is being incorporated into modern smart card standards, e.g.: [TeleTrusT WG2, 2003], [DIN V66291-4, 2000], [CEN CWA 14890-1, 2003] [IBM Magyarország Rt. et al., 2004]. I reckon, it is fallaciously advertised as a countermeasure against the man-in-the-middle attacks of the terminal.

If we examine the benefits gained from secure messaging at terminal of various trust-categories (see Section 1.3), we come to the following conclusions:

'trusted':
: It is futile, as it is assumed that these terminals do not attack.

*'possibly tampered'*and *'unknown'*:
: It is useless, as these terminals can bypass secure messaging and attack the channel between the user and the IFD.

'trusted only if authenticated':
: According to the definition of *'trusted only if authenticated'*, if the user decides to trust the terminal operator after authenticating him or her then see *'trusted'*, otherwise see *'unknown'*.

  It must be noted that the mutual authentication mechanism (that is used for key exchange) and the 'display message' mechanism (see Section 2.1.1) can be used to authenticate the terminal. However, this mechanism can be bypassed easily if the attacker can have access to an IFD (e.g. by stealing a terminal). As the implementation of secure messaging in e.g. [TeleTrusT WG2, 2003] does not allow the card to check whether the IFD's certificate has been revoked[1], an

[1] In fact, the standards use an architecture different from PKI that does not allow a certificate revocation list for so-called 'card verifiable' certificates

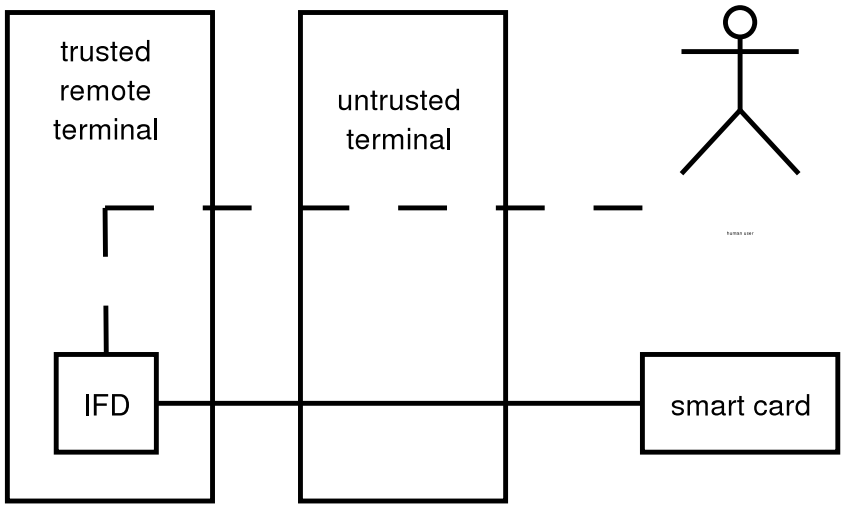

Figure 4: Using 'secure messaging' – the IFD is a secure module in a remote server

attacker can compromise the security of the entire system by stealing one terminal. An example of an attack that allows fake terminals to identify themselves as authentic ones is presented by e.g. [CEN CWA 14890-1, 2003] and is known as the 'grandmaster chess attack'.

While secure messaging gives the user no help for using the untrusted terminals, it can be used by a trusted remote server to read information from a smart card in an encrypted or authenticated way. While a human cannot communicate with the card directly, a computational device can make use of these advanced capabilities of the card.

- The policy that *the user should remove the card if not using it* is not a useful countermeasure at all. On the one hand, the user cannot control how many times the terminal turns to her card during the time the card is in the reader. On the other hand, it still does not prevent a man-in-the-middle attack.

  However, following such a policy a user can control when a terminal may attack. It can be relatively easy to explain any user: 'as long as the card is not connected to the reader, the terminal cannot abuse it'. Among countermeasures that give a similar degree of protection, this is one of the cheapest. [Berta and Mann, 2000b]

2.2.2 Workable countermeasures

In this section I summarize solutions that are sound and give protection against the threat of malicious terminals. Unfortunately, some of these countermeasures are quite awkward.

- *Preventing the attacker from taking full control of the terminal* aims that the terminal

should not be completely malicious.[2] Naturally, this solution works only if the terminal is operated by a trusted entity and its aim is to conserve the terminal's integrity, i.e. in the case of certain terminals of *'possibly tampered'*.

– *Read-only terminals* provide one possibility. In this case, the area on the disk of the terminal that turns to the card has to be made read-only (e.g. used from a CD ROM). In this case the attacker may create a tampered copy of this area and modify the references to the original area. To prevent this, the whole binary area of the terminal needs to be made read-only. Although, this is a workable solution, it is awkward because it contradicts customization and prevents the terminal manufacturer from correcting (or patching) its products.

  There are some practical solutions (even products) that follow this paradigm. For instance, there are firewall software that are sold as a bootable CD of Linux. Configuration files for such firewalls can be stored on floppy disks. (Floppy disks can be made physically read-only, so a network intruder cannot modify their contents.)

– *Trusted computing* is a paradigm that aims to keep computers secure. Making *all* computer secure may seem like fighting windmills, but this approach has the most support from industry. Naturally, this approach is useful against terminals where the user trusts the intentions of the terminal operator, but would like to gain confidence in the integrity of the terminal. (*'possibly tampered'*)

  Several projects follow this direction, including TCPA, which would restrict a terminal to executing only those applications that are digitally signed by a trusted party. This solution may be awkward, because at the cost of preventing the terminal from running the code of viruses, it would also prevent the user from executing programs that she wrote. Neither does this approach solve the problems of buggy software (the attacker can take control of) and software that can execute macros. [TCPA, 2004]

  There are other approaches that aim to make it harder for network attackers to tamper with the software of a computer. One option is protecting the program code in the memory with cryptographic algorithms. For example when a processor loads a program into the memory, it could encrypt it and calculate a cryptographic checksum on it. The keys for these algorithms could be stored in a register directly inaccessible to the program code. This way, an attacker mounting a buffer overflow attack cannot easily overwrite the program code in the memory, the attacker would need to overwrite it with a corresponding cryptographic checksum. Just like the previous solution, this is also a step away from the von Neumann architecture. Such solutions all require very strong hardware support from the processor running the program. They also require new and fast cryptographic

[2] The idea of having a trusted IFD in the terminal was already a step towards this direction. However, it does not even mitigate the problem of malicious terminals unless the trusted IFD includes the user interface (keyboard and screen) which is rather unlikely. [Rivest, 2001] suggests that it is costly to implement comfortable, customizable but trusted user interfaces.

algorithms can encrypt or authenticate every instruction on the fly. [Pyo, 2004], [Shao et al., 2004], [McGregor et al., 2003b], [McGregor et al., 2003a]

- *Providing the user with secure channels of communication* means supplying users with trusted computational devices with user interfaces. Today, this does not seem to be a cost-effective solution.

  However, many users consider their mobile phone a trusted device. This assumption might not be completely sound, as a mobile phone does not go under a thorough evaluation (like a smart card). Most mobile phones have closed platforms, so it is difficult to check if it has backdoors installed by the phone manufacturer. Actually, many phones do have 'backdoors' in the sense that the mobile phone operator can (and is allowed to) run various tests on the phone without the user's consent.

  On the other hand, most network attackers do not have the expertise to attack a mobile phone yet.

  Perhaps, combined with a PC terminal, a user may safely assume that it is unlikely that the same attacker is able to control the communication channels of both the PC and the mobile phone. There are more and more solutions today that rely on this assumption. For example, there are many web-based services where the user can pay by sending an SMS to a number given at the webpage. The service provider receives the money from the mobile phone operator, and sends the password to the website by SMS in return. [Berta, 2004]

- *Neglecting the problem of malicious terminals* is perhaps not a countermeasure in the technical sense but economically this may be a cost-effective solution. Actually, this solution is the most widely used today. Note that I consider neglecting the problem a *solution* only if the risks of using a malicious terminal are carefully evaluated.

  Credit card systems offer little protection for the user. Any party who knows the credit card number and expiration date may make payments in the name of the user, which may seem to be a ridiculous security measure. If a user uses his or her card at a malicious terminal, this device may simply sniff this information and abuse it. The only protection the user has is that she receives a list of her transactions regularly and may decide to revoke certain transactions.

  Surprisingly, this area of payment is blooming in many countries. It seems, only a small percentage of users and only a small percentage of merchants is cheating. At least, credit card companies choose neglect the cost of fraud and make honest, non-malicious users pay its costs. Although the number of attacks is growing rapidly, credit card companies seem to be satisfied with this solution.

  However, the rules of these systems are unclear for many users who often get exposed to banks and credit card companies. In particular, it is often the bank's or credit card company's decision whether the revocation of the user is accepted or not. Sometimes the bank decides to pay the loss of the user to prevent the scandal in the press, but sometimes the user has to prove that she is right. [Anderson, 1996] made a survey of credit card

frauds in many countries and found that the security of credit card systems largely depends on whether the law requires the bank to prove that the user was malicious or the user to prove that she is innocent. (Anderson also notes that the user often cannot prove this.)

The solution I propose in Section 7 is based on the experience learned from existing credit card systems. In contrast to those solutions, I also fix the rules of the game and give both the user both the merchant certain cryptographic proofs that could be used in court as evidence.

- *If a user could execute cryptographic operations* she could overcome the problem of untrusted terminals. However, it is questionable if there are cryptographic algorithms that are easy enough to be executed and give a the same degree of protection to state-of-the-art cryptographic algorithms (like AES or RSA with 1024-bit-long keys).

  I reckon it is unlikely that such an algorithm exists. Moreover, if there was such an algorithm, it has not been discovered yet. If there was a known cryptographic algorithm that gives the same degree of protection to the above while requiring significantly less resources, that particular algorithm would be used instead of the above ones.

Note that all workable solutions point outside the original problem of a user at an untrusted terminal. These solutions either make the terminal trusted, or give the user another trusted terminal, or make the user herself a trusted terminal.

In the next sections, I will develop a model (first an informal one, then a formal one) for the problem, and investigate what chances the user has to protect her messages without any computational aid.

2.3 Summary of the state-of-the-art

- Compared to the importance of the problem of malicious terminals, relatively few works in literature discuss this problem.
- There are certain methods a user can rely on to *authenticate* terminals. These methods are included in modern standards on smart cards.
- The problem of *using* a malicious terminal is still a severe problem. No widespread solutions exist in this field.
- Many practical experts know of the problems of untrusted terminals, So advertising a product by claiming that it deals with this problem seems to mean competitive advantage. However, there are many fallacious solutions on the market that do not even mitigate this problem.
- The problem of untrusted terminals is an important problem that is unsolved yet.

3 An informal model

Having considered theoretical and practical aspects of the man-in-the-middle attacks of malicious terminals, we may now construct a model for a user at a malicious terminal.

Assume that user U would like to communicate with remote partner R. As user U would like to send sensitive information, she would like to prevent attackers to read or modify it.

- User U is a human being, she is an 'average person' (Section 3.1), she does not have any special computational abilities. As she has very strict limitations both in terms of memory and computational power, most cryptographic algorithms are beyond her capabilities.
- Remote partner R is a human being at a trusted computer. Being a human, R has all the resources U has. Having a computer, R is also able to perform state-of-the-art cryptographic operations.
- Terminal T is the only device user U may use in order to send her message to R, i.e. there are no other devices with a user interface. User U considers that terminal T belongs to the Category *'possibly tampered'*or Category *'unknown'*, so she assumes that terminal T may mount an attack.
- User U may have a smart card C, a trusted personal microcomputer at her disposal. Unfortunately, smart card C does not have a user-interface on its own, so the user has to rely on untrusted terminal T to communicate with the card.

  Due to their small size and secure architecture, smart cards have severe computational limitations in practice. However, as these limitations are true for general-purpose operations [Berta and Mann, 2002] frequent operations can be accelerated by their cryptographic coprocessor. Thus, without loss of generality we may assume that a smart card can run all the algorithms a computer can.

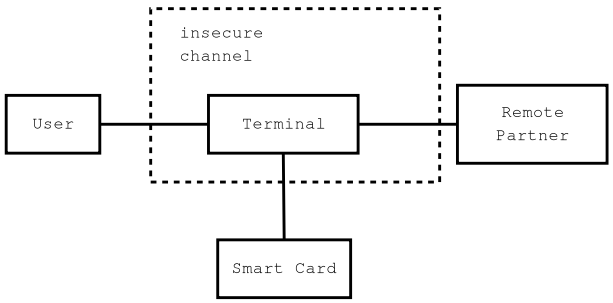

Figure 5: A widespread model for systems with insecure terminals

The interconnection of the above four parties is illustrated on Figure 5. Just as observed by [Stabell-Kulo et al., 1999], terminal T is trusted by neither U nor R, so the untrusted terminal is not a real participant in the model, it is rather a part of the insecure channel.

I reckon that there is also a serious problem regarding smart card C: Figure 5 highlights that user U has the same possibilities for communication with C and R. Thus, if the user has a way to communicate with smart card C, she could use the same method for communication with R.

This means, while the role of the smart card can be important in a practical implementation (e.g. for storing the private key of the user), there is no need for smart card C when searching for a theoretical solution against the man-in-the-middle attacks of the malicious terminal.

Let's assume the most simple scenario, where both U and R trust each-other. Can they exchange sensitive information through the untrusted terminal?

To answer the above question, there is another particularly important question that still remains open: Does the user have enough resources to perform cryptographic operations (like encryption or message authentication)?

3.1 Estimating the resources of the average user

First of all, I must note that the resources of humans may largely vary. While some people have trouble with simple additions, there are a few exceptional ones who can multiply 3-digit-long numbers without using even a pencil and a paper. Some users have trouble memorizing 4-digit-long PIN codes, while there are examples of people learning a book by heart. Some users send several hundred messages per day, and are unwilling to spend more than a few seconds for sending one, while others consider a single message so important that they decide to sacrifice a whole week for encrypting or authenticating it.

In this dissertation I focus on the field of commercial applications, i.e. I search solutions that can be applied for a large mass of users. This means, in my model user U does not have any extraordinary abilities and is willing to sacrifice a 'reasonable amount of time' for sending her message. What does this mean? Let's see an example!

User U can be considered a 'poor cryptographic device' because of the following limitations:

- *User U has very small memory.* We may approximate that she can memorize up to 10-12 random characters. Although this may seem little, this is an optimistic approach. These characters have to be random ones, so they should not be regular words. Moreover, has the user to replace it regularly (if it is used as a cryptographic key or password), and may not write it down but has to keep it in her memory.

  If we consider a character to be 8 bytes long, we find that the user can store a cryptographic key of 80-96 bytes.

- *User U can compute very slowly.* We may approximate that she can perform a few dozen operations a minute. Note that it is not obvious what we call an operation in case of a human. Probably, it would be quite awkward for a human to compute bitwise operations; they are more likely to feel comfortable with characterwise operations. Simple, two-operand characterwise operations (like modular addition) could be accelerated by using a public substitution table.

If the user is patient, she can execute any algorithm a computer can, but it is rather unlikely that a user would spend years with calculations in case of a single credit card transaction.

- *User U makes mistakes in her calculations.*

In certain fields the user also has some good capabilities too:

- She can memorize long passphrases.
- She can easily generate random numbers of very good quality (e.g. by flipping a coin or by rolling dice). In this field, a human user can outperform many computers.

Having considered the abilities of the average user, the next question is: Are these resources enough for protecting the user's messages?

3.2 Sending one single bit of information

Let's consider the most simple case: user U would like to transmit only 1 bit of information towards R.

3.2.1 Secrecy

In case of secrecy, there exists a perfect solution that provides unconditional secrecy (i.e. the attacker cannot have a better strategy for learning the message bit than flipping a coin to guess it): the *one-time-pad* [Shannon, 1949]. In this case, only one bit of symmetric key needs to be stored, and one XOR operation needs to be performed. This solution is feasible, probably any user could execute it in case of a one-bit-long message.

3.2.2 Message authenticity

A similar solution can be constructed for message authenticity too. Assume that U and R share two n-bit-long one-time passwords: p_0 and p_1. (Both passwords are random, the only relation between them is $p_0 \neq p_1$.) If U would like to send message bit 1, she sends p_1. If U would like to send message bit 0, she sends p_0.

If the attacker would like to forge a message '1' in the name of user U, the attacker needs to guess p_1, and has a $\frac{1}{2^n}$ chance of guessing it successfully. The attacker has the same probability for forging message '0'. If the attacker observes message '1' and wishes to modify it to message '0' (or vice versa), then the attacker needs to guess p_0 (or p_1), and has a $\frac{1}{2^{n-1}}$ chance of guessing it successfully.

The previously described 'average user' can share two 40-bit-long (5 ASCII character long) passwords with the remote partner, that can already give a considerable protection. As this solution does not require the user to perform any cryptographic operations, it can be called feasible.

3.3 Sending long messages

In the previous section I showed that there are solutions that a user – any user – can use to protect 1-bit-long messages against the man-in-the-middle attack of the malicious terminal. Probably, the user can use the same solutions for sending short messages too, but the user would need to sacrifice a fresh keybit (or a fresh password) for every messagebit too. However, most users would like to send longer messages too. I reckon, a typical message could be a business letter or a contract, that is a few pages – a few thousand characters (or bytes) – long.

In this case, it would not be feasible to send the message bit-by-bit using the solutions described in the previous section, because neither of the above solutions allow the user to reuse keybits. On the one hand, solutions requiring one-time-keys suffer from severe key management problems, on the other hand, the user would not have enough memory to store keys required for protecting even a small fragment of the message. Note that these solutions could be feasible if we allowed the user to write the keys down and store them in on a paper. Generally, it is not advised to write passwords and cryptographic keys on a paper, because an attacker may obtain this paper and thus learn the key or password. However, against a network attacker from a distant continent who does not know the user but attacks a random victim, a piece of paper is secure, simply because it is in the physical world and is inaccessible from cyberspace. If we do not allow the user to have any external storage, we must search solution other than using one-time keys.

The limited amount of memory does not prevent the user from calculating a strong encryption or a cryptographic checksum on a message. For example, the DES (that is already obsolete, but still gives a considerable amount of protection) utilizes 56-bit-long keys, and could be used for both encryption and authentication. [Berta and Mann, 1999] Memorizing such a key is not beyond the capabilities of the user. (However, the DES requires an amount of computational power that most users do not possess.)

Alone, the limitation of computational power does not prevent the user from protecting her messages. There are cryptographic operations (like the above mentioned one-time-pad) that requires a very limited amount of computational power. If we observe the requirements e.g. on block ciphers (completeness, avalanche effect, statistical independence [Buttyán and Vajda, 2004]) we also find that they all can be satisfied with very simple operations.

The problem that humans make errors during calculations could also be alleviated by introducing (like error correction codes) redundancy in during the calculations. (However, this would further increase the required computational power.)

Generally, it is possible to make trade-offs between computational power and memory. A typical example is when computation is accelerated by using precalculated tables. This way, some storage space of the computer is sacrificed, and some additional time is saved.

It is also possible to sacrifice computational power to gain storage space. Let us see the following example: Assume that A would like to send an encrypted message to B. A can memorize a key of at most K (e.g. 50) bits. Assume that a computer can perform an exhaustive search over a keyspace of 2^K. Assume that no computer can perform an exhaustive search over a keyspace of 2^{2K}.

Under these circumstances, A can send an encrypted message to B if they share a K-bit-long

symmetric key k. A can create a $2K$-bit-long key $k||r$, where r is K bits of one-time random. This way, the attacker, who does not know k has to perform an exhaustive search over a keyspace of 2^{2K}. The recipient B who knows k can decrypt it by performing an exhaustive search for r over a keyspace of 2^K.

Unfortunately, user U has severe limitations both in terms of memory and computational power, so such solutions are out of the question.

3.4 Problem with the asymptotic approach

Even if an algorithm could be constructed that is 'efficient' within the limitations described in Section 3.1, I reckon, many users would find it awkward because of the following problem.

Algorithmic complexity theory usually expresses resource constraints as function of the input length. [Rónyai et al., 1999] Generally, an algorithm is considered to be efficient if it requires a number of operations polynomial in the length of the input. This approach is practical, because the speed of computers increases rapidly every year, and we would like to compare the performance of algorithms that run on computers on different architectures and process different magnitudes of input length.

Probably, this approach is not practical for humans, especially for humans in commercial applications. On the one hand, we cannot speak of such a rapid increase in the processing speed of humans. On the other hand, the length of the text of the messages humans would like to protect does not increase dramatically. Few users would like to send a text as long as 'War and Peace' in the future; the length of contracts and business letters is unlikely to increase.

Thus, when designing algorithms for 'running on humans' we cannot convince a user to use algorithm for message authentication by saying: 'If you wish to send n bytes, you need to perform $O(n)$ operations.' Users would be more interested in how much time (or operations they need) to authenticate a typical message. For example: 'You can authenticate a message in one minute.' Thus, instead of an asymptotic boundary, the user is interested in a *constant upper bound for typical messages*. I reckon, this constant upper bound should be very low for every-day people.

If we consider any message authentication algorithm, it is an obvious requirement that at least n operations are necessary for processing n-character-long messages. This means, that the cryptographic algorithm needs to read its input; otherwise the attacker would have an $\frac{1}{n}$ chance of modifying the message successfully. (The same statement is true for encryption; otherwise the attacker would be able to learn at least one character of the message.)

If the message is 1000-character-long, n operations mean 1000 operations, and I reckon it is already unacceptable for many users.

3.5 Summary of the user's cryptographic abilities

'Can the user perform cryptographic operations?' The first, obvious reply to this question is *'Of course, not!'*. However, the situation is not so simple, because in certain situations, e.g. when the

message is very short or when the user may have secure external storage, feasible and *practically applicable solutions exist*.

In case of long messages, we cannot speak of an existing solution. In fact, the above question could be rephrased as: 'Is there a cryptographic algorithm that gives a similar degree of protection to AES and RSA (with e.g. 1024-bit-long keys) but require significantly less resources?' Clearly, no such algorithm is known today.

There are two more problems: On the one hand, users would not judge a pencil-and-paper algorithm as it is done by experts of algorithmic complexity theory. Users would require a constant upper bound to the algorithm they use to encrypt or authenticate their typical messages. On the other hand, the resources users vary largely among users. Some users are going to reject algorithms even with the smallest possible sound upper bounds.

In the next section, I introduce a different approach. Instead of trying to formalize the abilities of the user by defining what they *can do*, I will state what they *cannot do*, i.e. that they cannot compute cryptographic operations on long messages. Using this – already formal – model of the user, in Section 4 I am going to prove that such a user cannot take part in any cryptographic protocols that allow sending encrypted or authenticated messages.

4 Formal limitations of the user

4.1 A formal model

Let U denote the user who wishes to communicate with the remote partner R using the untrusted terminal T. While U is a human being, R and T are computers[3]. User U would like to send the message m to R, and tries to 'protect' (encrypt or authenticate) it by combining it with the secret key k, which is a shared secret between U and R. Both m and k are strings of characters from a binary alphabet $I = \{0, 1\}$. Parties U, R and T are able to execute various efficient algorithms (randomized algorithms of polynomial complexity in their input parameter [Goldreich, 1997b]) that perform an $I^* \mapsto I^*$ mapping.

The key k is n-bit-long, where n is a security parameter. I assume the message length $length(k) < length(m) \leq p(n)$, where $p(n)$ is a polynomial.

For input x algorithm h produces $h(x)$ as output. Notation $h(x)$ also implies that algorithm h can process x as its input. Henceforth, notation $x|y$ stands for the concatenation of strings x and y. Furthermore, notation $[\alpha]$ stands for the value of the expression α.

Using the above notations, I give a definition for the notion of computational easy and hard. Naturally, both easy and hard are relative to the amount of resources a certain party (U, R or the attacker) has. According to literature (e.g. [Goldreich, 1997a]), I express such resource constraints as functions of the security parameter n.

[3] Actually, R can also be viewed as a human using a trusted computer. Since in this section humans are considered slow devices that cannot outperform computers, R can be considered a computer.

Definition 1. *Let a and b be two efficient algorithms. We say that it is $t(n)$-hard to compute $b(x)$ from $a(x)$, if for all efficient algorithms h with $t(n)$ resources and input $a(x)$, for all polynomials $p(n)$, and for large enough n:*

$$\Pr_x \left(b(x) = h_{t(n)}(a(x)) \right) < \frac{1}{p(n)}$$

where x is uniformly chosen from its space.

The above definition follows the usual formalism of the asymptotic approach of algorithm complexity theory. Note that the above problem can only be addressed probabilistically, because the attacker can always guess $b(x)$ with a non-zero probability. A problem is considered hard, if it is hard to solve it on the average, so the probability of the attacker's success is negligible. Negligible means being bounded above by all functions of the form $\frac{1}{p(n)}$. [Goldreich, 1997a] Thus, the higher the security parameter is, the harder the problem is, so the lower chance of success is allowed for the attacker.

Based on the above definition, let's introduce predicate $hard(a(x), b(x), t(n))$ that takes value $true$ if computing $b(x)$ based on $a(x)$ is $t(n)$-hard. Otherwise, predicate $hard$ takes value $false$.

Definition 2. *Let a and b be two efficient algorithms. We say, it is $t(n)$-easy to compute $b(x)$ from $a(x)$, if (for algorithms a, b) there exists an efficient algorithm h with $t(n)$ resources and input $a(x)$ such that for any x:*

$$b(x) = h_{t(n)}(a(x))$$

Based on the above definition, let's introduce predicate $easy(a(x), b(x), t(n))$ that takes value $true$ if computing $b(x)$ based on $a(x)$ is $t(n)$-easy. Otherwise, predicate $easy$ takes value $false$.

In the following paragraphs I define ciphers and message authentication codes (MAC) using predicates $easy$ and $hard$. I simplify the sophisticated probabilistic approach of literature to the binary logic of predicates. I do not discuss to what extent a certain algorithm can protect the secrecy or authenticity of messages, but I divide algorithms into two groups: one group qualifies for being a cipher or MAC under the given resource constraints, while the other group does not.

I chose to use binary logic because my goal was to elaborate a formal model for an average user and give a yes-or-no answer on whether she can communicate securely in an untrusted terminal environment.

Note that my proofs (in Sections 4.2 and 4.3) depend on the definitions of $cipher$ and mac only, and do not depend on the exact definitions of $easy$ and $hard$. (For example, the proofs would hold even if the definition of $easy$ was generalized by allowing an ε error rate for the user. The definition of $hard$ could also be made more rigorous by making it difficult to compute 'any deterministic function of $b(x)$ based on $a(x)$'. Such refinements would not affect the validity of my proofs.)

Definition 3. *Algorithm f is a $t_{attacker}(n)$-strong cipher, if it is $t_{attacker}(n)$-hard to obtain the whole input M of algorithm f from $f(k|M)$ without knowing key k. If the key is known,*

$t_{encrypt}(n)$ resources are required to perform the encryption and $t_{decrypt}(n)$ resources are required for decryption. Formally:

$$\begin{aligned} &cipher\,(f, t_{encrypt}(n), t_{decrypt}(n), t_{attacker}(n)) \Leftrightarrow \\ &\quad \Leftrightarrow hard(f(k|M), M, t_{attacker}(n)) \wedge \\ &\quad \wedge easy(k|M, f(k|M), t_{encrypt}(n)) \wedge easy(f(k|M)|k, M, t_{decrypt}(n)) \end{aligned}$$

Definition 3 of an encryption resembles the well-known security definition of 'plaintext recovery' for symmetric key encryption transformation. [Bellare and Rogoway, 2002] The main difference is, that not access to an encryption oracle is allowed. The reason is that we assume one-time keying (as described below).

Definition 4. *Algorithm f can compute $t_{attacker}(n)$-strong message authentication code, if – based on one pair of observed input M and output MAC – it is $t_{attacker}(n)$-hard to present a different input M' and corresponding MAC without knowing the key. If the key is known, it is $t_{calc}(n)$-easy to compute the MAC, and $t_{check}(n)$-easy to check it. Formally:*

$$\begin{aligned} &mac(f,\ t_{calc}(n),\ t_{check}(n),\ t_{attacker}(n)) \Leftrightarrow \\ &\quad \Leftrightarrow hard(M|f(k|M), M'|f(k|M'), t_{attacker}(n)) \wedge \\ &\quad \wedge\ easy(k|M, f(k|M), t_{calc}(n)) \ \wedge\ easy(k|M''|x, [f(k|M'') == x], t_{check}(n)), \end{aligned}$$

where $M \neq M'$ (but both $M = M''$ and $M \neq M''$ are possible).

Later on, I use these definitions to prove negative statements on the abilities of the user. My definitions of $cipher$ and mac could be refined to be closer to practical ciphers and MACs. For instance, they do not make any statement on the case when the attacker learned a part of the key. (A practical cipher should not collapse if the attacker learns one keybit.) However, there is no use to refine these definitions in the direction of putting additional handicaps on the user – in Sections 4.2 and 4.3 I prove that the user does not have a chance even without these handicaps.

The key k can either by considered a symmetric key shared between the user and a remote partner, or it can be considered the concatenation of a 'public' key and a corresponding private key, where the 'public' part of the keypair is also kept secret form the attacker. This way, the definitions of $cipher$ and mac cover both the symmetric and the asymmetric key algorithms, but in case of asymmetric key algorihtms is in a more beneficial position, because the attacker does not know the public key.

We also have to define the amount of resources U, R and T have. Since R and T are computers, they have $t_{computer}(n)$ resources. Both of them are able to execute algorithms of polynomial complexity in n. Definition of the $t_{human}(n)$ resources of the human being is more difficult. Heuristically, $t_{human}(n) \ll t_{computer}(n)$. Our implicit definition of $t_{human}(n)$ is given by the following natural way:

$$\begin{aligned} \models &\neg\exists f\,\{cipher\,(f,\ t_{human}(n),\ t_{computer}(n),\ t_{computer}(n))\} \wedge \\ &\neg\exists f\,\{cipher\,(f,\ t_{computer}(n),\ t_{human}(n),\ t_{computer}(n))\} \wedge \\ &\neg\exists f\,\{mac\,(f,\ t_{human}(n),\ t_{computer}(n),\ t_{computer}(n))\} \wedge \\ &\neg\exists f\,\{mac\,(f,\ t_{computer}(n),\ t_{human}(n),\ t_{computer}(n))\} \end{aligned} \quad (1)$$

This way, I defined $t_{human}(n)$ by claiming that no $t_{computer}(n)$-strong cipher and no $t_{computer}(n)$-strong mac exists, that can be executed with $t_{human}(n)$ resources at the coding or decoding side. Thus, U is able to perform weak encryption or weak authentication only, that can be successfully attacked by the terminal with high probability.

Assume, we are in a world where the above limitations hold. We also suppose that the Kerckhoff principle is valid ([Kerckhoff, 1883]), so the attacker T, knows every algorithm U and R uses, but does not know the secret key k. In order to make the proposition more general, let's make the environment as advantageous for the user as possible. Thus, let's assume that:

- One-time keying is used, so key k is replaced after each message m is sent. (Note that $length(k) < length(m)$, so a one-time-pad cannot be used.)
- In case of secrecy, the attacker is able to eavesdrop only, and cannot modify the messages on the channel.
- In case of message authenticity, the attacker is active.

After these preparations we are able to examine the following problem: Is it possible for U and R to solve the problems of encryption and message authentication by interaction, i.e. not in one step, but in several interactive protocol steps?

4.2 Secrecy

In this section we examine if U with t_{human} resources can transmit a secret message to R using a finite two-party protocol.

Proposition 1. *If user U cannot encrypt message m with $t_{human}(n)$ resources with a security that cannot be broken with $t_{computer}(n)$ resources with significant probability, then no N-step-long protocol P exists between U and R that has the following properties:*

(S1) *P enables R to decrypt m with $t_{computer}(n)$ resources,*

(S2) *P prevents the attacker (who also has $t_{computer}(n)$ resources but does not know k) from decrypting m.*

Proof. Let's consider the following general protocol for interactive encryption:

Protocol 3. – General protocol for interactive encryption

Initially: $\sigma_0 = \emptyset$

1. $U \rightarrow R$: $f_1(k|M_1)$, where $M_1 = m|\sigma_0$
   $\sigma_1 = f_1(k|M_1)$

2. $R \rightarrow U$: $g_2(k|M_2)$, where $M_2 = \sigma_1$
$\sigma_2 = \sigma_1|g_2(k|M_2)$

... ...

$(2L-1)$. $U \rightarrow R$: $f_{2L-1}(k|M_{2L-1})$, where $M_{2L-1} = m|\sigma_{2L-2}$
$\sigma_{2L-1} = \sigma_{2L-2}|f_{2L-1}(k|M_{2L-1})$

$(2L)$. $R \rightarrow U$: $g_{2L}(k|M_{2L})$, where $M_{2L} = \sigma_{2L-1}$
$\sigma_{2L} = \sigma_{2L-1}|g_{2L}(k|M_{2L})$

... ...

N. $U \rightarrow R$: $f_N(k|M_N)$, where $M_N = m|\sigma_{N-1}$
$\sigma_N = \sigma_{N-1}|f_N(k|M_N)$

where in each step j, σ_j denotes all the data that was interchanged by U and R via the public channel by U and R, thus σ_j denotes the database of the attacker too. We consider N-step-long protocols, so R is able to acquire m after step N only.

The proposition (S1 and S2) can be formalized as follows:

$$(1) \rightarrow \neg\exists\sigma_N \left\{hard\left(\sigma_N, m, t_{computer}(n)\right) \wedge easy\left(k|\sigma_N, m, t_{computer}(n)\right)\right\} \tag{2}$$

In contrary, assume that:

$$\exists\sigma_N \left\{hard\left(\sigma_N, m, t_{computer}(n)\right) \wedge easy\left(k|\sigma_N, m, t_{computer}(n)\right)\right\} \tag{3}$$

If algorithm f_j can be executed by U, then user U has enough resources to run it:

$$\models \forall f_j \left\{easy(k|M_j,\ f_j(k|M_j),\ t_{human}(n))\right\} \tag{4}$$

According to the assumption about the abilities of the human (1), no algorithm that U can run, can be a $cipher$. So, according to (4), none of the algorithms f_j can be a $cipher$. This has the following implication:

$$\begin{aligned}\models &\forall f_j \left\{easy(k|M_j, f_j(k|M_j), t_{human}(n)) \rightarrow\right. \\ &\left.\rightarrow \neg hard(f_j(k|M_j), M_j, t_{computer}) \vee \neg easy(f_j(k|M_j)|k, M_j, t_{computer}(n))\right\}\end{aligned} \tag{5}$$

Note that $M_j = m|\sigma_{j-1}$ if j is odd.

User U cannot compute a $cipher$, but can choose between two lesser alternatives. One of them is to choose an algorithm, where $\neg easy(f_j(k|M_j)|k, M_j, t_{computer}(n))$. This means, remote partner R is unable to obtain m. The other alternative is to choose an algorithm, where $\neg hard(f_j(k|M_j), M_j, t_{computer})$. Such an algorithm is a weak encryption, where the attacker

might be able to obtain m. This latter would violate the S2 property, so user U should choose an f_j where:

$$\models \forall f_j \{\neg easy(f_j(k|M_j)|k, M_j, t_{computer}(n))\} \tag{6}$$

According to (3), R should be able to obtain m after step N using an algorithm $m = g_{N+1}(k|\sigma_N)$ from k and σ_N, while the attacker can be successful with a negligible probability only.

$$\models hard\,(\sigma_N, m, t_{computer}(n)) \wedge easy\,(k|\sigma_N, m, t_{computer}(n)) \tag{7}$$

Let's substitute σ_N with $\sigma_{N-1}|f_N(k|m|\sigma_{N-1})$ into (7).

$$\begin{aligned}\models & hard\,(\sigma_{N-1}|f_N(k|m|\sigma_{N-1}), m, t_{computer}(n)) \wedge \\ & \wedge easy\,(k|\sigma_{N-1}|f_N(k|m|\sigma_{N-1}), m, t_{computer}(n))\end{aligned}$$

The above formula can be simplified if we suppose that f_N includes σ_N in its output. This does not spoil the security of the system, since σ_N is already public. Then we obtain:

$$\begin{aligned}\models & hard\,(f_N(k|m|\sigma_{N-1}), m|\sigma_{N-1}, t_{computer}(n)) \wedge \\ & \wedge easy\,(k|f_N(k|m|\sigma_{N-1}), m|\sigma_{N-1}, t_{computer}(n))\end{aligned}$$

Finally, we substitute $m|\sigma_{N-1}$ with M_N:

$$\models hard\,(f_N(k|M_N), M_N, t_{computer}(n)) \wedge easy\,(k|f_N(k|M_N), M_N, t_{computer}(n)) \tag{8}$$

Note that (8) contradicts (6) for f_N. We have come to a contradiction, so the above protocol does not exist. □

4.3 Message authenticity

In this section the question of message authenticity shall be considered. Assuming that U is unable to provide strong message authenticity in one step, I prove that U and R cannot solve the problem with several interactive protocol-steps either.

Since single steps cannot be authenticated in a 'secure way', so neither U, nor R will be able to decide if the messages have been tampered with before the protocol is finished. I shall use the following notation:

$A \Rightarrow B : \alpha$ means, party A sends the message α towards party B via an insecure channel, where the attacker can modify α on the channel to α', so B receives α'.

Proposition 2. *If U is unable to perform strong authentication with $t_{human}(n)$ resources, then no N-step-long protocol P exists between U and R that has the following three properties:*

(A1) R learns message m when protocol P terminates (after step N).

(A2) *R is able to verify that m is authentic.*

(A3) *Without the key k, the attacker is unable to produce a valid pair of datablocks σ'_N and m' ($m \neq m'$) with significant probability.*

Proof. Let's consider the following general protocol for interactive authentication:

Protocol 4. – General protocol for interactive authentication

---

Initially: $\sigma_0 = \emptyset, \omega_0 = \emptyset$

1. $U \Rightarrow R$: $f_1(k|m)$
   $\sigma_1 = \sigma_0|f_1(k|m), \omega_1 = \omega_0$

2. $R \Rightarrow U$: $g_2(k|\sigma'_1)$
   $\sigma_2 = \sigma_1, \omega_2 = \omega_1|g_2(k|\sigma'_1)$

3. $U \Rightarrow R$: $f_3(k|m|\omega'_2)$
   $\sigma_3 = \sigma_2|f_3(k|m|\omega'_2), \omega_3 = \omega_2$

... ...

$(2K)$. $R \Rightarrow U$: $g_{2K}(k|\sigma'_{2K-1})$
   $\sigma_{2K} = \sigma_{2K-1}, \omega_{2K} = \omega_{2K-1}|g_{2K}(k|\sigma'_{2K-1})$

$(2K+1)$. $U \Rightarrow R$: $f_{2K+1}(k|m|\omega'_{2K})$
   $\sigma_{2K+1} = \sigma_{2K}|f_{2K+1}(k|m|\sigma'_{2K}), \omega_{2K+1} = \omega_{2K}$

... ...

N. $U \Rightarrow R$: $f_N(k|m|\omega'_{N-1})$
   $\sigma_N = \sigma_{N-1}|f_N(k|m|\omega'_{N-1}), \omega_N = \omega_{N-1}$

R becomes more and more confident in the authenticity of m' with every received datablock σ_j, because the probability of a successful attack decreases continuously. The protocol terminates at the Nth step, when this probability is considered small enough, so that R can verify the authenticity of m'. R can check the authenticity of m' by recalculating the σ_j values. Note that both U and R are capable of executing probabilistic algorithms too, so R has to determine if σ'_N is a possible outcome of a valid protocol run. R accepts m' as authentic if:

$$F(k|m'|\omega_N) \equiv f_1(k|m'|\omega_0) \mid f_3(k|m'|\omega_2) \mid ... \mid f_N(k|m'|\omega_{N-1}) = \sigma'_N \tag{9}$$

The attacker is successful in the above protocol, if there is a non-negligible probability of R accepting m' as authentic, where $m' \neq m$.

The proposition (A1 and A2 and A3) can be formalized as follows:

$$(1) \rightarrow \neg\exists\sigma_N\{hard(m|\omega_N|\sigma_N, m'|\omega_N|\sigma'_N, t_{computer}(n))\wedge \\ easy(k|m'|\sigma'_N|\omega_N, [F(k|m'|\omega_N) == \sigma'_N], t_{computer}(n))\} \quad (10)$$

where $m \neq m'$.

In contrary, assume that the above protocol provides secure authentication, i.e:

$$\exists\sigma_N\{hard(m|\omega_N|\sigma_N, m'|\omega_N|\sigma'_N, t_{computer}(n)) \wedge m \neq m' \wedge \\ easy(k|m'|\sigma'_N|\omega_N, [F(k|m'|\omega_N) == \sigma'_N], t_{computer}(n))\} \quad (11)$$

where $m \neq m'$.

Algorithm F has the following properties:

- U is able to run F. According to (9), the execution of F requires as much resources as executing all of the algorithms f_i sequentially. Formally:

$$\models easy(k|m|\omega_N, \sigma_N, t_{human}(n)) \quad (12)$$

- If U is able to run an algorithm, R is able to do it too, because $t_{human} < t_{computer}$. Thus, R is able to check F by simply recalculating it. Formally:

$$\models easy(k|m'|\omega_N, \sigma'_N, t_{computer}(n)) \rightarrow \\ \rightarrow easy(k|m'|\omega_N|\sigma'_N, [F(k|m'|\omega_N) == \sigma'_N], t_{computer}(n)) \quad (13)$$

If we substitute $m|\omega_N$ with M and $m'|\omega_N$ with M', and we also substitute σ_N with $F(k|M)$, then from (12) and (13) we obtain:

$$\models easy(k|M, F(k|M), t_{human}(n))\wedge \\ \wedge easy(k|M'|\sigma'_N, [F(k|M') == \sigma'_N], t_{computer}(n)) \quad (14)$$

Because of the assumptions about the user's abilities (1), (14) implies:

$$\neg hard(M|F(k|M), M'|F(k|M'), t_{computer}(n)),$$

where $M \neq M'$.

If we revert the above substitutions, we obtain:

$$\neg hard(m|\omega_N|\sigma_N, m'|\omega_N|F(k|m'|\omega_N), t_{computer}(n)),$$

where $m|\omega_N \neq m'|\omega_N$, which is equivalent with $m \neq m'$.

The attacker can compute $F(k|m'|\omega_N)$, and is able to use it as σ'_N while attacking. Thus, we can substitute $F(k|m'|\omega_N)$ with σ'_N and obtain:

$$\neg hard(m|\omega_N|\sigma_N, m'|\omega_N|\sigma'_N, t_{computer}(n)), \quad (15)$$

where $m \neq m'$.

Note that (15) contradicts the indirect assumption (11). We have come to a contradiction, so the above protocol does not provide secure authentication. □

5 Conclusions

In Section 4 we have shown that a user who would like to send messages from an untrusted terminal, can guarantee neither the secrecy nor the authenticity of long messages. This means, no possible solution exists within the boundaries of the model introduced in Section 4.1.

Thus, in order to protect the user's messages, we should *extend our model* and search a solution in a world where one or more of the assumptions of the model is not true. Possible solutions may exist if:

1. The terminal is not under the total control of the attacker (i.e. if she is not using an untrusted terminal).
2. The user is only sending short messages (i.e. if $length(k) < length(m)$ is not true).
3. The user has an extraordinary amount of resources to perform strong cryptographic operations (i.e. if formula (1) is false.)
4. A completely different model is used, or the signature has to meet different requirements.

Part II
Solutions

6 Conditional signatures against untrusted terminals

Conditional signatures were introduced by Lee and Kim [Lee and Kim, 2002], who used this concept for solving fair exchange problems without expensive cryptographic primitives like verifiable escrow. A conditional signature of U on a message m is U's ordinary signature $sig_U(m, c)$ on m and a description of a condition c. If $sig_U(m, c)$ is correct and condition c is true, then $sig_U(m, c)$ is considered to be equivalent with $sig_U(m)$, U's ordinary digital signature on m. However, if c is false, then U is not responsible for m. Intuitively, U's conditional signature is U's commitment: *'I signed m, but if c is not true, then my signature on m is not valid.'*

I propose that conditional signatures can be used to protect the authenticity of messages sent from untrusted terminals. In my solutions, the user has a smart card at her disposal that enforces that a message is signed together with a condition. All of my solutions have the following properties:

1. The user sends a message to the card using the untrusted terminal. Naturally, the terminal may read the message, so its secrecy cannot be ensured. The user cannot prevent the terminal from altering the message at this point.
2. The smart card signs the message together with condition c. Note that the user cannot prevent the smart card from signing any message chosen by the terminal. However, this arbitrary message is signed together with condition c.
3. The condition becomes false if the user did not intend to sign the message.
4. The condition reaches the remote partner (the verifier of the signature) on a channel the untrusted terminal cannot attack.
5. The remote partner accepts the conditional signature as the signature of the user only if the condition appended to the message is true.

While properties 1, 2 and 5 are quite straightforward to implement, it is challenging to design systems with properties described as 3 and 4. In the next two sections I propose two solutions – two implementations for the above. They both utilize conditional signatures, but in a completely different way.

In the solution proposed in Section 7, I assume that while the user needs the untrusted terminal to access certain services, time-to-time she is able to access trusted terminals too. The user performs signatures at untrusted terminals, and reviews the signatures from trusted ones. In this case, the value of condition c depends on whether she revokes or confirms the signature from a trusted terminal. (Property 3) Condition c reaches the remote partner on a secure channel from a trusted terminal. (Property 4)

In the solution proposed in Section 8, I make use of the fact that the human user is not merely a poor computer. In this scenario she prepares the message in a biometric format (e.g. as a video message). I assume that such messages are more difficult to attack. In this case the value of c depends on the biometric verification of the message and on the distance of certain timestamps

the smart card and the user appended to the message. (Property 3) One part of condition c the remote partner receives is protected by biometry (that I assume the terminal cannot easily attack). The other part of condition c reaches the remote partner under the protection of the signature of the smart card. (Property 4)

7 A Solution based on Revocable Signatures

In this section I propose a framework that allows the user at an untrusted terminal to produce digitally signed messages (e.g. checks) and to send them to an untrusted party (e.g. a merchant who may collude with the terminal operator); my framework allows the user to review messages at untrusted terminals and revoke unintended signatures.

7.1 Extensions to the basic model

One of my key objectives is to propose a solution for commercial use, so I assume that the user is a human being without any exceptional computational abilities. User U has limited memory and computational power. By this I mean that U is able to memorize some passwords or PIN codes, but she cannot memorize cryptographic keys, neither can she perform cryptographic computations. For this reason, the private key of U is stored on and the signatures are generated by a smart card C in possession of user U.

Essentially, C is a trusted personal microcomputer without direct interfaces towards U. Card C is connected to the terminal in front of U, and all messages between C and U, must pass through the untrusted terminal. I assume the following about the smart card:

C1 Smart card C is manufactured by a trusted manufacturer and hence, it is assumed to function correctly. In particular, C does not try to leak the private key of U or to use the private key without authorization.

C2 Smart card C is able to perform cryptographic operations, like encryption or digital signature, to generate good quality pseudo-random numbers, and to store a few thousand bytes of data. Many smart cards on the market satisfy this assumption[4].

I assume that *untrusted terminal* T (that either belongs to Category *'possibly tampered'* or to Category *'unknown'*) in front of U is fully under the control of an attacker, who may have installed all kinds of malicious software on the terminal before U started to use it. This means that the attacker is able to steal and abuse any information typed in by U on the keyboard of the terminal, to send fake messages to U through the display of the terminal, and to modify messages that U sends to C for signing before passing them on to C. Thus, the attacker can obtain signature

[4]In [Berta and Bencsáth, 2002] I surveyed the pseudo random number generator of several commercial cards from different manufacturers, and found that black box testing could not reveal any weakness on cards that had a cryptographic coprocessor. Today, some more sophisticated smart cards claim to be capable of generating real randomness based on monitoring certain physical processes.

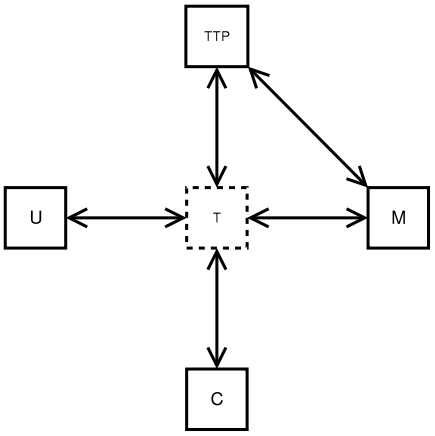

Figure 6: Physical connections

from the smart card for an arbitrary message. However, we assume that from time to time, U has access to C from a *trusted terminal* too (Category *'trusted'*). Such a trusted terminal could be the home PC of U, but it can also be a terminal operated by a trusted organization and believed to be tamper resistant (e.g., an ATM machine). Of course, in order to use a terminal for this purpose, it must be properly authenticated first.

Thus, two different scenarios exist for U: one where the terminal is untrusted, and another one where the terminal is trusted. In the first case, all logical channels between U and the other actors C, M, and TTP are insecure, as they pass through the untrusted terminal, and thus they are controlled by the attacker. In the second case, secure logical channels between U and the other actors C, M, and TTP can be established, since U and the trusted terminal together can be viewed as a computer that can set up cryptographically secured connections with C, M, and TTP. Since C, M, and TTP are able to perform cryptographic computations, they can establish encrypted and authenticated channels between each other, regardless of the terminal in front of user U.

By M I denote the intended recipient of the digital signature generated by C. M could be a service provider, a merchant, another user, etc. In contrast to R (the trusted remote partner in Section 4.1), M is not trusted by user U (and neither does M trust the user). Since in many applications, T is operated by M, I also assume that T and M may collude.

In certain protocols I am going to assume that there is a trusted third party TTP in the system that both U and M trust. (In Section 7.4 I explain why a trusted third party is necessary in practical protocols.) Depending on the exact protocol used, TTP may have different functions. E.g. in the protocol presented in Section 7.3.1, TTP needs to countersign each message signed by the user's card.

While in Sections 7.2 and 7.3 user U considers TTP to be completely trusted, in Section 7.4, user U would like to retain her privacy with respect to TTP. This means that U trusts TTP only for the revocation of unintended signatures, but U would like to prevent TTP from knowing where,

when and what messages she signed. Thus, she would like to prevent TTP from knowing, which merchants or service providers she does business with. TTP follows the given protocols, and does not try to cheat by breaking into the terminal or intercepting messages for other parties. Neither does TTP collaborate with T or M to reveal the identity of the user. (This concept is similar to that of a semi-trusted TTP in [Franklin and Reiter, 1996].)

The entities of the model and their interconnections are illustrated in Figure 6.

Summary of extensions

- In this solution, the recipient of the signature does not have to be trusted, M may cooperate with the untrusted terminal T to alter the message the user is willing to sign.
- The user is carrying a smart card C. The smart card is tamper-resistant and is trusted by all parties in the system.
- Apart from untrusted terminal T, there are trusted terminals in the system. User U regularly visits trusted terminals, but has to use untrusted terminals too to access certain services.
- In certain protocols I assume that a trusted third party is present.
- In this solution, there are different requirements for the digital signature. The digital signature is non-repudiable proof of the user's consent only if the user did not revoke the signature under certain circumstances.

7.2 Generic protocol

In order to detect attacks mounted by the attacker, I propose a framework that allows users to sign messages on untrusted terminals with the help of their smart cards, review the signatures later in a trusted environment, and revoke the fake ones (or authorize only the valid ones). In this section I propose a generic protocol that uses conditional signatures. Later, in Sections 7.3 and 7.4 I show specific protocols that derive from this generic one.

7.2.1 Core protocol

As it is impossible to prevent the terminal from obtaining signature from the card on an arbitrarily chosen document, instead of generating an ordinary signature, I propose that C generates a conditional signature such that it is guaranteed that the condition cannot become true before a certain amount of time has passed. This should leave time for the user to move to a trusted terminal for checking the signatures generated by the card, and to enforce that the conditions of the fake signatures can never become true.

Within my framework, a conditionally signed document is sent to the recipient. The user cannot change it anymore, but she can claim that she did not sign that particular document. If she

revokes an intended signature, the recipient can use the revoked signature as plausible evidence for proving that the user was present at the terminal and initiated a transaction.

These thoughts lead to the following generic protocol: (Note that while steps 1-4 happen at an untrusted terminal, steps 5 and 6 are performed using a secure terminal and via secure channels.)

Protocol 5. – Generic protocol for signature revocation

Step 1: $U \rightarrow T$: m

When U wants to sign message m at an untrusted terminal, she first provides the terminal with m, then she inserts her card C into the terminal's smart card reader.

Step 2: $T \rightarrow C$: m

The terminal forwards the message.

Step 3: $C \rightarrow T$: $c, sig_U(m, c)$

The card logs m in its internal memory, computes the conditional signature $sig_U(m, c)$ of U on m, where c is a condition that includes (among other things) deadline t, and outputs $(c, sig_U(m, c))$ to the terminal. The intention is that the signature $sig_U(m, c)$ will not be valid before t; in addition, it will become valid after t if and only if the other conditions in c hold.

Step 4: $T \rightarrow M$: $(m, c, sig_U(m, c))$

The message is sent to the intended recipient M along with the conditional signature.

Step 5: $C \rightarrow U$: M, m, c

Later, but before the deadline t, U reviews the list of messages logged by C at a trusted terminal. This can be done, for instance, by U returning to her home and inserting C into the smart card reader of her home PC. Before outputting its log, C authenticates the terminal to be sure that it is a trusted one.

Step 6: For each message m that U intended to sign, U ensures that the condition c becomes true; for the rest of the messages, U ensures that the condition becomes false. This might involve additional steps and further communication with M or TTP. More details on how the condition can be made true or false can be found in Section 7.3.

A third party needs to check if the digital signature $sig_U(m, c)$ of the card is correct and condition c is true in order to verify a conditional signature.

7.2.2 Practical extensions to the core protocol

There are two problems with the above core protocol: First, C is required to log every message that it signed. However, C is a smart card with limited storage capacity. In some applications (where large messages have to be signed), it may be infeasible for C to store every message.

Second, C is required to input the whole message to be signed. Again, if messages are large, then this may be impossible or impractically slow.

The general protocol can be easily extended in order to overcome these potential problems. The first problem can be solved by outsourcing the logging function to an external log server. This log server needs to be trusted by U only, so it may even be the user's home PC if it is online. The second problem can be solved, by letting the terminal compute a hash of the message to be signed; only this hash is passed to the smart card, and the conditional signature is generated on this hash.

In order to include these extensions in the generic protocol only steps 2, 3 and 5 need to be changed. In the description below, LS denotes the log server and it is assumed that C and LS shares a symmetric key $K_{C,LS}$.

Protocol 6. – Practical extension of Protocol 5

Step 1: $U \rightarrow T$: m
Same as before.

Step 2: $T \rightarrow C$: $h(m)$
The terminal first computes the hash $h(m)$ of m. Then, it passes only $h(m)$ to C for signing.

Step 3.1: $C \rightarrow T$: $LS, C, \{C, n, h(m)\}_{K_{C,LS}}$
C encrypts $(C, n, h(m))$ with the key $K_{C,LS}$, where n is a sequence number maintained by C, and is increased by each log request. C outputs $LS, C, \{C, n, h(m)\}_{K_{C,LS}}$ to the terminal.

Step 3.2: $T \rightarrow LS$: $(C, m, \{C, n, h(m)\}_{K_{C,LS}})$ to LS.
The terminal sends message m to LS along with its encrypted hash value.

Step 3.3: LS looks up the key that it shares with C, and decrypts $\{C, n, h(m)\}_{K_{C,LS}}$. LS then does the following:

- it verifies if the decryption was successful by checking that the first field of the decrypted message is C;
- it verifies that n is greater than any sequence number previously received from C;
- finally, it computes the hash of m and compares the result with $h(m)$ received in the encrypted part of the message.

If any of the verifications above is not successful, then LS aborts the protocol. Otherwise, it logs (C, n, m), and sends an acknowledgement to the terminal:
$LS \rightarrow M$: $mac_{K_{C,LS}}(LS, C, n, h(m))$

Step 3.4: $T \rightarrow C$: $mac_{K_{C,LS}}(LS, C, n, h(m))$
The terminal forwards the acknowledgement to C.

Step 3.5: $C \rightarrow T$: $(c, sig_U(h(m), c))$
C verifies the acknowledgement, and if it is correct, then it continues the core protocol by generating a conditional signature $sig_U(h(m), c)$ on $h(m)$.

Step 4: $T \rightarrow M$: $(m, c, sig_U(h(m), c))$
Same as before, the message is sent to the intended recipient M along with the conditional signature.

Step 5: Later, but before t, U downloads the logged messages from LS and reviews them at a trusted terminal. Before this operation takes place, the terminal is authenticated in order to be sure that it is a trusted one.

Step 6: Same as before: U ensures that only those conditions become true where she intended to sign the corresponding messages.

Henceforth, I assume that smart card C is able to log the entire message m.

7.3 Protocols for revocable digital signatures

As we have seen in the previous section, it is always required that condition c is not true before a given deadline t. This is an inherent requirement in our scheme, which gives the user some time to move to a trusted terminal and review the signatures generated so far by her smart card. However, various approaches are possible to define what happens after t, or more precisely, how the user can enforce the truth value of c after t. In this section, I discuss some of these approaches.

In most of the applications, it is desirable that the status of a digital signature does not vary in time. In this scheme, this is not fully supported, since every signature is invalid until t, and then it may become valid. There is a good reason to allow this, namely to mitigate the untrusted terminal problem. Note, however that the schemes that I propose below guarantee that once the user reviewed and accepted a signature, it cannot be revoked anymore.

It seems to be a good idea to define a default truth value for c after t that cannot be changed later, because this ensures that the status of each signature will indeed become stable after t independently of the negligence of the involved parties. In other words, if the user does not take any steps until t to confirm or to revoke a signature, then the status of the signature will take the default value at t, and the user can no longer do anything about it. Depending on the default truth value, we can distinguish between two classes of protocols.

- Protocols in the first class support the *default accept* approach, where a signature automatically becomes valid after t (and remains valid forever) unless it is explicitly revoked by the user before t. Unfortunately, all of these protocols require a TTP.

- Protocols in the second class support the *default deny* approach, where a signature remains invalid after t (and forever), unless it is explicitly confirmed by the user before t. Protocols of this class might not be suitable in certain applications, because users may tend to forget to confirm conditional signatures, which means that they revoke them.

In the following two subsections, I present two protocols and corresponding conditions in order to illustrate the finalization of the status of signatures. [Berta et al., 2004a]

7.3.1 A protocol based a 'simple deadline' condition

In this scheme, condition c is the following: *"My signature on the above message is valid if and only if deadline t has passed and TTP countersigned it."* This scheme follows the default accept approach, but a similar scheme can be constructed for the default deny approach too.

Protocol 7. – A protocol based on a 'simple deadline' condition

---

U signs message m at an untrusted terminal:

Step 1: $U \rightarrow T$: m

Step 2: $T \rightarrow C$: m

Step 3: $C \rightarrow T$: $t, TTP, sig_U(m, t, TTP)$

Step 4: $T \rightarrow M$: $m, t, TTP, sig_U(m, t, TTP)$

U reviews signed messages from a trusted terminal:

Step 5: $C \rightarrow U$: M, m, t, TTP

Step 6: If U did not intend to sign message m and deadline t has not passed, then:
$U \rightarrow TTP$:
'I revoke my signature $sig_U(m, t, TTP)$*'.*
Otherwise, U does not need to act.

Step 7: $M \rightarrow TTP$: $t, TTP, sig_U(m, t, TTP)$

Step 8: If U did not revoke the signature at TTP before t, then:
$TTP \rightarrow M$: $sig_{TTP}(sig_U(m, t, TTP))$

A third party needs to check if the digital signatures $sig_U(m, t, TTP)$ and $sig_{TTP}(sig_U(m, t, TTP))$ are correct in order to verify the conditional signature.

7.3.2 A protocol based on a 'trapdoor function' condition

The previous protocol had two very expensive operations: On one hand, it required TTP to compute a digital signature for every signature of every user. On the other hand, two signatures need to be verified in order to verify the conditional signature.

My next protocol follows the default deny approach. In this scheme, condition c is *"My signature on the above message is valid if and only if the preimage of $h(r)$ is presented."* ; where $h(r)$ is the hash of an unpredictable random number r.

Protocol 8. – A protocol based on a 'trapdoor function' condition

---

U signs message m at an untrusted terminal:

Step 1: $U \rightarrow T$: m

Step 2: $T \rightarrow C$: m

Step 3: $C \rightarrow T$: $h(r), sig_U(m, h(r))$

Step 4: $T \rightarrow M$: $m, h(r), sig_U(m, h(r))$

U reviews signed messages from a trusted terminal:

Step 5: $C \rightarrow U$: $M, m, h(r), r$

Step 6: If U intended to sign message m then:
$U \rightarrow M$: r
Otherwise, U does not need to act.

A third party needs to check if the digital signature $sig_U(m, h(r))$ is correct and r is presented in order to verify the conditional signature.

This protocol is efficient, it does not require any help from TTP. However, it does not fully support the requirement, that the value of condition c cannot be changed after a certain deadline (in fact, in this case c does not contain any deadline). Note that while U is able to change the value of c from false to true any time, she cannot do it vice versa. Help from TTP is required if we liked to freeze the value of c after a deadline.

7.4 Protocols protecting the user's privacy

7.4.1 Practical protocols require a trusted third party

In Section 7.3, I identified two classes of protocols: default accept or default deny. Protocols following the default deny approach can be more simple, but they require user U to explicitly

confirm each signature; within the default deny approach, merchants and service providers cannot do business with users who forget to confirm their signatures. Protocols supporting the default accept approach seem more practical, but all of them require a TTP. In these protocols, after Step 3, the conditional signature is in the hands of the untrusted terminal (or the untrusted merchant). Condition c (and the signature of the user) will become valid after a certain deadline automatically, unless U revokes it. If U has to revoke it at an untrusted party (like T or M), such a party could simply repudiate the receipt of such a revocation. While smart card C is also a trusted party, C is not online continuously. Thus, it is impossible for C to maintain a signature revocation list that is trusted and can be checked by all other parties.

A trusted third party is needed to enforce a default value for the signature and to handle revocation. It seems that all of the practical protocols require the help of a TTP.

(Note that while protocols following the default accept approach are more practical, they are vulnerable to attackers that are able to block the channel between the trusted terminal and the TTP. Such an attack is not possible in may practical situations, e.g. a terminal at a foreign airport might not be able to block the channel between the user's home computer and a TTP in her home country. If we assume that the adversary is omnipresent on the network, then either out-of-band signature revocation channels – like a direct telephone line, non-electronic mail post or personal contact, etc. – or default deny signature revocation protocols should be used.)

If TTP is able to log all the messages a user signs, it is in a very critical position. Few organizations would be trusted enough to be a TTP in such protocols. I reckon that if the protocol prevented the TTP from linking the user with the merchant or service provider, more organizations would qualify to be a TTP.

7.4.2 Privacy at untrusted terminals

Naturally, the most straightforward way a user can protect her anonymity is refusing to provide information that can be linked with her. However, a non-repudiable, digitally signed message is – by definition – linkable with the user. In this work, I address the problem of sending such a message from an untrusted terminal while allowing the user to retain a degree of privacy.

Papers that discuss the possibilities of users with limited resources in a malicious environment rarely address the privacy of the user simultaneously. However, I found that anonymous payment systems address a very similar privacy-problem to ours. There is generally a bank (or trustee) in such systems who issues e.g. digital cash, and users would like to make their transactions to be untraceable by this trustee [Claessens et al., 1999]. Chaum introduced an anonymous payment system (the late DigiCash), based on blind signatures [Chaum, 1982]. The foundations of some other famous anonymous payment systems are introduced in [Brands, 1994] and [Franklin and Yung, 1992]. Jakobsson et al. [Jakobsson and Raïhi, 1998] also propose an electronic payment system in which anonymity is based on a mix-network ([Chaum, 1981]).

The trusted third party in this paper is in a position very similar to that of a bank or trustee in the above papers. However, in their approach, the user and her trusted computer are one single entity, while in this case, an insecure channel may separate them.

I also rely on the existence of anonymous communication channels. Fortunately, both literature and practice provides solutions for anonymous communication (see e.g., [Chaum, 1981], [Syverson et al., 1997], [Anonymizer Inc., 1999], [Reiter and Rubin, 1998]).

7.4.3 Objectives

My goal is to develop a protocol for signature revocation that allows U *to retain her privacy with respect to* TTP. Thus, if there are many users in a system, I would like to minimize the probability of TTP successfully linking a signature with a particular user. While user U may trust TTP for signature revocation, she does not want TTP to know, where, when and what messages she wanted to sign.

During a protocol run, user U would like to prevent TTP from obtaining any information that can differentiate her from other users of the system. In particular, she would like to hide $id(U)$ (her user name or identifier), and message m. Moreover, she would also like to prevent TTP from obtaining any information that can be linked with these too. It is clear that user U does not want to protect this information against M, because she intends to send message m to service provider M. Moreover, she cannot protect m against T, because she types the message using the keyboard of the terminal in Step 1.

Note that in a system with n users, TTP has at least a $\frac{1}{n}$ chance of selecting the particular user who took part in the protocol. My aim is to develop a protocol, where this chance is minimal, i.e. TTP's probability distribution of users sending the message is uniform [Serjantov and Danezis, 2002].

In this section I propose three protocols (and three corresponding conditions) that allow the user to retain her privacy with respect to TTP. [Berta et al., 2004b].

7.4.4 Common principles

All proposed protocols follow the generic concepts of Section 7.2. The first deviation from the generic protocol appears in Step 3, when the smart card sends a cryptogram encrypted by the public key of TTP that contains condition c along with revocation token r. Unlike in the protocol described in Section 7.2, terminal T is unable to verify the signature in this step. Thus, we need to refine assumption 1 about the smart card:

C1+ Smart card C is assumed to be trustworthy and tamper-resistant. *Tamper-resistance* means, it is impossible to alter its behavior, reverse engineer it or extract information from it. *Trustworthiness* means that the device is manufactured by a trusted manufacturer. Since smart cards undergo extremely rigorous evaluation and certification, and not even the manufacturer can alter their behavior after issuance, I consider this assumption to be justified.

Since card C is trustworthy and tamper-resistant, all other parties (U and M and TTP) consider C a trusted party. Thus, terminal T assume in Step 3 that card C follows the protocol, and is not sending garbage.

This cryptogram that the smart card outputs in Step 3 is forwarded to the merchant in Step 4 and later to TTP in Step 7. The user receives revocation token r from the card via a trusted terminal, and may repudiate her signature by submitting r to TTP in Step 6 via an *anonymous channel*. I assume that such an anonymous channel exists. (Naturally, anybody may repudiate the signature using r, so it is advisable not to let the untrusted terminal compute it. Otherwise, T would be able to repudiate messages in the name of U, and could spoil the reputation of the user.)

TTP decrypts the cryptogram that was sent by the merchant in Step 7, and enforces condition c to become true (in Step 8) if the revocation token r inside the cryptogram was not submitted before. Revocation token r is a random number, statistically independent from the identity of U, the contents of message m or the value of conditional signature $sig_U(m, c)$. Based on r, TTP is unable to link U with M. (Note that the identity of M is not hidden from TTP.)

While TTP needs to store revocation token r, it may not be necessary to store it forever. This problem could be solved e.g. by introducing a lapse time, so TTP could refuse to validate very ancient conditional signatures.

7.4.5 A protocol based on 'bit commitment'

My first protocol follows the spirit of bit commitment protocols [Schneier, 1996]. User U commits herself to her signature to M. However, U does not reveal her signature to M immediately, only after deadline t contained in c. In this case, condition c is the following string: *"My signature on the above message is not valid before deadline t."*

In contrast to the classical bit commitment, the "reveal" phase is not performed by user U (because of reasons described in 7.4.1), but by trusted third party TTP in Step 8. Moreover, in this case not even TTP is allowed to read the bits U committed herself to. Thus, in this case not all known bit commitment methods can be used (e.g., the solution proposed in [Naor, 1991] cannot be used in this case), only those that do not require sending bits (the user committed herself to) in cleartext when they are revealed.

I propose the following protocol to protect the privacy of U with respect to TTP:

Protocol 9. – Privacy protecting protocol based on a bit commitment scheme

---

Step 1: $U \rightarrow T$: m

Step 2: $T \rightarrow C$: m

C generates random symmetric key k and revocation token r.

Step 3: $C \rightarrow T$: $c,\ E_k[sig_U(m)],\ E_{TTP}(r, k, c)$

Step 4: $T \rightarrow M$: $m,\ c, E_k[sig_U(m)],\ E_{TTP}(r, k, c)$

Later, at a trusted terminal:

Step 5: $C \rightarrow U$: M, m, c, r

If user U would like to repudiate the signature on message m then

Step 6: $U \rightarrow TTP$: r (via an anonymous channel)

After deadline t:

Step 7: $M \rightarrow TTP$: $E_{TTP}(r, k, c)$

If deadline t has passed, and r was not submitted to TTP, then:

Step 8: $TTP \rightarrow M$: k

Step 9: M decrypts $E_k[sig_U(m)]$ using k and obtains $sig_U(m)$.

It is an important merit of this protocol that a third party needs to have m and $sig_U(m)$ only in order to verify the conditional signature of U. Since, this conditional signature is not different from a regular one, its verification requires the same procedure too. Note that in this protocol TTP does not have to perform a digital signature operation.

7.4.6 A provable degree of privacy

In this protocol, U is able to retain a *provable degree of privacy* with respect to TTP.
The following datablocks appear during a run of Protocol 9:

m**:** The message may contain information that can be linked with the identity of U.

c**:** The condition consists of a fixed string constant and a deadline t. I assume, deadline t cannot be linked with the identity of U.

r, k**:** One-time random numbers of uniform distribution.

$sig_U(m)$**:** Using the public key of a certain user, it is possible to check if a signature was calculated by that particular user. Thus, $sig_U(m)$ can be linked with the identity of U.

$E_k[sig_U(m)]$**:** I assume that it is not possible to link this block with the identity of U without having the corresponding secret key k.

$E_{TTP}(r, k, c)$**:** Datablocks r, k and c cannot be linked with U, and the public key of TTP cannot be linked with U either, so datablock $E_{TTP}(r, k, c)$ cannot be linked with U.

By ω I denote the set of datablocks user U needs to conceal with respect to TTP.

$$\omega = (id(U), m, sig_U(m))$$

By σ I denote the set of datablocks TTP receives during a protocol run. Note that apart from the messages TTP receives in Steps 6 and 7, TTP receives a message *directly from the user*.

I suggest that a system that provides channels for anonymous communication should be used for sending this particular message. I model a possibly imperfect anonymous channel with a perfect one, where TTP obtains additional information $anonch(U, r)$ in Step 6. In case of the perfect anonymous channel, TTP perceives that each user is equally likely to be the sender of the message, so the *size of the anonymity set* (the number of users) can be used to describe anonymity in the system [Serjantov and Danezis, 2002]. I assume that $anonch(U, r)$ may be linked with the user; based on $anonch(U, r)$ TTP may perceive that user U is more likely to be the sender of message m than some other users. (Formally: $I(anonch(U, r); \omega) \geq 0$.)

Proposition 3. *Protocol 9 allows user U to retain the following degree of privacy with respect to TTP:*

(a) If all parties behave honestly, user U has unconditional privacy. Formally: $I(\sigma_a; \omega) = 0$

(b) If user U decides to revoke the signature, she has the degree of privacy provided by the anonymous channel. Formally: $I(\sigma_b; \omega) = I(anonch(U, r); \omega)$

Proof.

(a) If all parties behave honestly, Step 6 of Protocol 9 is not executed, so the only message TTP receives is $E_{TTP}(r, k, c)$ in Step 7. Based on this message, TTP can compute (r, k, c). Thus, $\sigma_a = (r, k, c, E_{TTP}(r, k, c))$.

$I(\sigma_a, \omega) = 0$ is obvious.

(b) If Step 6 is executed, TTP receives r via the anonymous channel. Thus, $\sigma_b = (\sigma_a, anonch(U, r))$.

$$I(\sigma_b; \omega) = I(\sigma_a, anonch(U, r);\ \omega) = H(\omega) - H(\omega|\sigma_a,\ anonch(U, r))$$

Since ω and σ_a are independent if the value of $anonch(U, r)$ is fixed,

$$H(\omega) - H(\omega|\sigma_a,\ anonch(U, r)) = H(\omega) - H(\omega|anonch(U, r)) = I(anonch(U, r); \omega)$$

□

7.4.7 A protocol based on 'blind signatures'

My next protocol relies on the concept of blind signatures introduced in [Chaum, 1982]. In this scheme, the cryptogram the card outputs in Step 3 contains the conditional signature of the user. TTP has to countersign the conditional signature in order to validate it, without being able to read it. To prevent TTP from reading the conditional signature, the signing process of TTP is blinded by the card. Smart card C also releases a token that M will be able to use to unblind the signature.

In this case, condition c is the following: *"My signature on the above message is valid if and only if deadline t has passed and TTP countersigned it."*

When presenting a solution based on blind signatures, we cannot treat cryptographic primitives as black boxes (like sig or E), but we need some details on the algorithms used. Thus, I need to introduce a slightly different notation that gives deeper insight into certain operations and their parameters.

I use the following notation to present our solution based on blind signatures:

- The signature of the user on message x is denoted as $sig_U(x)$.
- TTP has two key pairs. One of them is an RSA key pair for digital signatures, the other keypair is for an arbitrary algorithm for encryption/decryption.
  - The RSA keypair of TTP for digital signature is the following: the public key is e, the public modulus is m, and the private exponent d. According to the RSA algorithm, the signature of TTP on message x is $x^d \bmod m$.

    Note that an attacker may obtain signature (or decryption) from this keypair on an arbitrary message, so TTP should not use this keypair for any other purpose.
  - Encryption of message x with the public encryption key of TTP is denoted as $E_{TTP}(x)$. Decryption of message y with the private decryption key of TTP is denoted as $D_{TTP}(y)$.
- Symbol "$*$" stands for multiplication modulo m. Modulus m should be larger than the largest possible value of $sig_U(x)$.

The proposed protocol is as follows:

Protocol 10. – Privacy protecting protocol based on blind signatures

---

Step 1: $U \rightarrow T$: m

Step 2: $T \rightarrow C$: m

Step 3: $C \rightarrow T$: $c, b, E_{TTP}(c, r, sig_U(m, c) * b^e)$
C generates random numbers r and b, where r is a repudiation token and b is going to be used to blind the signature of TTP.

Step 4: $T \rightarrow M$: $m, c, b, E_{TTP}(c, r, sig_U(m, c) * b^e)$

Later, at a trusted terminal:

Step 5: $C \rightarrow U$: M, m, c, r

If user U would like to repudiate the signature, then:

Step 6: $U \rightarrow TTP$: r (via an anonymous channel)

After the t deadline:

Step 7: $M \rightarrow TTP$: $E_{TTP}(c, r, sig_U(m, c) * b^e)$

If deadline t has passed and r was not submitted to TTP, then:

Step 8 : $TTP \rightarrow M$: $[sig_U(m, c)) * b^e]^d = [sig_U(m, c)]^d * b$

Step 9: M acquires $[sig_U(m, c)]^d$ using b.

A third party needs to have m, c and $(sig_U(m, c))^d$ in order to verify the conditional signature of U.

Again, this protocol provides a *provable degree of privacy for U with respect to TTP*, since TTP receives only r (which is just a random number) and $E_{TTP}(c, r, sig_U(m, c) * b^e)$. In this latter cryptogram only $sig_U(m, c) * b^e$ carries information that could be connected to user U. However, according to [Chaum, 1982], the blind signature is unconditionally secure, so no algorithm exists that can compute $sig_U(m, c)$ based on $sig_U(m, c) * b^e$ (with a probability better than $1/2^{|sig_U(m,c)|}$) without knowing parameter b. Thus, the degree of privacy user U has is the same as the one provided by the channel that is used in Step 6 for signature revocation. Moreover, if everyone behaves honestly, U obtains unconditional privacy.

Note that in the above protocol TTP signs an incoming message without being able to see what it is. Although TTP may authenticate M to prevent denial of service attacks, M may still obtain the signature of TTP on an arbitrary message. Since TTP does not use this private key for anything else, such a signature is useful only if a conditional signature of the user was signed. Although M can compute the cryptogram and repeat Step 7 at will, M can only gain countersignatures on signatures U did not revoke.

7.4.8 A protocol based on 'halving the digital signature'

In this I make use of the fact that even if TTP obtains one half of the bits of the conditional signature of U, TTP is still unable to compute the other half. Meanwhile, if TTP countersigns the half of the bits of the conditional signature, it is authenticated not much less securely than if TTP countersigned the whole signature.

In this case, condition c looks as follows: *"My signature on the above message is valid if and only if TTP countersigned its right half and deadline t has passed."*

According to our notation, $left(x)$ means the left half of the bits of bitstring x, and $right(x)$ means its right half. Naturally, $left(x)||right(x) = x$ where operation $||$ is concatenation. The proposed protocol is as follows:

Protocol 11. – Privacy protecting protocol based on halving the digital signature

---

Step 1: $U \rightarrow T$: m

Step 2: $T \rightarrow C$: m

Step 3: $C \rightarrow T$: $c,\ \mathit{left}[sig_U(c,m)],\ E_{TTP}(r, c, \mathit{right}[sig_U(c,m)])$
C generates random number r. Again, the terminal cannot verify the conditional signature, but since C is a trusted device, T may believe that the conditional signature on m was encrypted by the public key of TTP.

Step 4: $T \rightarrow M$: $m, c,\ \mathit{left}[sig_U(c,m)],\ E_{TTP}(r, c, \mathit{right}[sig_U(c,m)])$

Later, at a trusted terminal:

Step 5: $C \rightarrow U$: M, m, c, r

If user U would like to repudiate the signature, then

Step 6: $U \rightarrow TTP$: r (via an anonymous channel)

After the deadline t:

Step 7: $M \rightarrow TTP$: $E_{TTP}(r, c, \mathit{right}[sig_U(c,m)])$

If deadline t has passed and r was not submitted to TTP, then:

Step 8: $TTP \rightarrow M$: $\mathit{right}[sig_U(c,m)],\ sig_{TTP}(\mathit{right}[sig_U(c,m)])$

Step 9: M computes: $\mathit{left}[sig_U(c,m)] \,||\, \mathit{right}[sig_U(c,m)] = sig_U(c,m)$

A third party needs to have m, c, $sig_U(c,m)$, $sig_{TTP}(\mathit{right}[sig_U(c,m)])$ in order to verify the conditional signature of U.

Again, M may obtain multiple countersignatures on signatures not revoked by U. However, this is not a problem. See the note at the end of Section 7.4.7. In contrast to the other solutions I proposed, the smart card does not need to perform any complex computation (like blinding or encryption) in this one, apart from generating the digital signature.

7.4.9 Comparison of protocols for revocable digital signatures

In this section I proposed five protocols for revocable digital signatures. If the user does not forget to confirm her signature, Protocol 8 (based on the 'trapdoor function' condition) is the most efficient. On one hand, it requires the smart card to perform very simple operations (a hash function) apart from the digital signature. On the other hand, Protocol 8 does not require a TTP. (This implies that the user does not need to protect her privacy against a TTP.)

However, it is realistic that users tend to forget to confirm signatures, so I reckon that protocols following the default accept approach might be more suitable for practical use. Probably, the simplest protocol of this class is Protocol 7 (the one based on the 'simple deadline' condition).

Yet, in this protocol the user is defenseless against a TTP who may trace what transactions she was engaged in.

Among the protocols that require a TTP but also protect the privacy of the user, Protocol 11 (the one based on halving the digital signature) requires the least complex operations. Though, in case of Protocol 9 (based on 'bit commitment') the user retains a provable degree of privacy. If all parties behave honestly, the user has unconditional privacy, otherwise the user has a degree of privacy equivalent with the one provided by the anonymous channel. If the channel provides unconditional privacy (that is possible in case of Onion routing or a Crowds-based network – see [Syverson et al., 1997] or [Reiter and Rubin, 1998]), the user retains unconditional privacy with respect to TTP.

8 A Solution based on Biometry

In this section, I propose a solution that allows the user to send digitally signed messages from untrusted terminals. My solution combines the protection provided by the biometric features of the user and the cryptographic algorithms running on the trusted smart card.

8.1 Extensions to the model

In Section 3, I modeled user U, remote partner R and smart card C as three independent computational devices wishing to communicate with each-other, where U has limited resources that prevent the application of advanced cryptography. In this section I extend my model, and do not consider user U merely a slow computer, but *I consider the user a human being* that has additional abilities:

- Humans have *biometric features*. On the one hand, they can be identified by means of biometry; on the other hand, a human can also identify other humans using biometry.

  In particular, I make use of the fact that humans can produce and verify so-called 'biometric signatures'. I provide a discussion on the nature and the strength of such signatures in Section 8.2

- I also assume that user U has a timer independent from untrusted terminal T (e.g. a watch), so U is able to measure time.

- By removing the card user U is able to block the channel between smart card C and terminal T.

8.2 Biometric signatures

Most biometric features only enable the identification of humans, but some can also be used to transmit information. For example, fingerprint verification can only identify people, but the

recorded voice combines the identity of the individual with the content of the speech. Out of the known biometric methods speaker verification, audiovisual verification and handwriting verification can fulfill such criteria.

I use the concept of time (with the assumption that manipulating a biometric message requires a certain amount of time or even user interaction) to reinforce biometric authenticity. Since handwriting is less connected to time, I am going to refer to the two others as 'biometric methods'.

Henceforth, I will use notation $bio(m)$ to denote the bitstream that user U produces when transforming plaintext message m into a biometric format. To produce $bio(m)$, user U does not need to perform any mathematical operations; a typical example of $bio(m)$ can be an .avi video file that contains the video message of user U reading plaintext message m. The exact encoding of $bio(m)$ depends on the video camera connected to terminal T. I model the video camera to be a part of untrusted terminal T, so it does not need to be trusted by the user.

8.2.1 Required degree of security

The user U transforms the message m to a biometric format to ensure its authenticity. The $bio(m)$ message is in a format that carries the biometric features of the sender too. I suppose, it is 'safer' than the plaintext message m. In this section the extent of this 'safety' is analyzed.

1. *Unconditional security* would mean that $bio(m)$ messages cannot be counterfeited or manipulated.
2. *Practical safety*: This means that the attacker needs considerably more time to counterfeit $bio(m)$, than to counterfeit m itself.
3. *No significant security*: The other extreme case would be to consider $bio(m)$ messages as secure as m (plaintext messages), so attackers do not need considerably more time to manipulate them.

Unconditional security is not a realistic assumption. I assume that if the attacker has enough time and resources, any $bio(m)$ can be manipulated, so considering them perfectly secure, is out of the question. I also assume that it is possible to calibrate a biometric method (bio) to give practical safety.

8.2.2 'Key space'

A biometrically signed message – just like a cryptographically signed one – is a result of a transformation based on some secret k. While in case of cryptography, k is a set of bits, in case of biometric signatures $k_{U_{bio}}$ is more complex: it is a combination of the biometrical characteristics of U. However, these can be affected by the following factors:

- Biometric identity: time-invariant biometric characteristics of U

- Current condition: Ever-changing factors that cannot be pre-planned in any way. E.g.: mood, illnesses, fatigue, aging, etc.
- Conscious alterations: Various attempts of U to influence the current sample.

Although the above represent an infinitely large key space, only factors known to the R can be used as secret key. Moreover, the above factors have to be measured, and the precision of measurement limits the size of the key too. Biometric systems are constructed on the assumption that the measured *biometric identity of each individual is different*. This means, that the key space of biometric identities is considered extremely large, even if their value cannot be measured exactly.

8.2.3 Possible attacks

If the attacker has observed the previous $bio(m_1)$, $bio(m_2)$, ..., $bio(m_n)$, and wishes to prepare a counterfeited $bio(m')$ message, two kind of attacks are possible:

1. The attacker may observe the biometrical characteristics of U and try to extract k_{bio} from the recorded messages in order to synthesize a complete $bio(m')$. In this case the attacker tries to masquerade itself as U, and trick the *partner identification* mechanism of R. Fortunately, this attack is far beyond the capabilities of most attackers.
2. Not being able to synthesize a $bio(m')$ message, the attacker may choose to create it using cut-and-paste from one or more previous $bio(m_i)$ messages sent by U. Thus, the attacker violates the *integrity* of the message, while every part of $bio(m')$ originates from U.

8.2.4 Protecting the integrity

The main weakness of $bio(m)$ messages is that the bio transformation does not have the *completeness* property. [Buttyán and Vajda, 2004] This means that modification of one part of $bio(m)$ does not require modifications in any other parts. What is the smallest part an attacker may try to modify? Since $bio(m)$ does not consist of blocks, the attacker may try to manipulate segments of any size. However, since the aim of the attacker is more than successfully modifying $bio(m)$, but to make R accept X where $bio^{-1}(X) = m' \neq m$. If m consists of characters, no attack on $bio(m)$ can be called successful, that does not modify at least one character in m. Although, the bio transformation does not satisfy the criteria of completeness, biometric messages have an internal structure that chains segments of $bio(m)$ together. On an audio message this can be the changes in amplitude of the voice or the tone of the speaker. On a video message this can be the changes in the facial expressions or the position, behavior or even clothes of the person.

While checking these properties using algorithms could be very awkward, the human brain triggers for any sudden or unpredicted changes in the biometrical characteristics of the speaker. The task of the attacker can be made especially hard if the biometric method is calibrated using

other artificial methods to chain blocks of the $bio(m)$ together. Good examples for the audio channel could be a different systematic background noise or music under the message. On the video channel a clock or television in the background can be used to harden cut-and-paste attacks. The method to preserve the structure of the message need not be secret: U can announce the method at the beginning of $bio(m)$, since the method only has to be one-time and consistent in the whole $bio(m)$. If the attacker wishes to paste together segments from $bio(m_i)$ and $bio(m_j)$, the manipulation of the biometric structure of the messages is needed. Although, this is not impossible, it requires a significant amount of resources and human interaction too.

I assume that the attacker needs significantly more time, to prepare such an attack, than U needs to create $bio(m)$.

Protecting the sender's identity In case of plaintext messages m contains no algorithmic protection, an attacker is able to send a valid m' message in the name of any user U without any information on the person. Viruses, that spread by email in the name of the user use this simple principle. However, every part of $bio(m)$ carries the identity of user U. Since the key space k_{bio} is large, no attacker has realistic chances of creating a valid $bio(m')$ message without having any a priori information on user U. Thus, an attacker has to *plan the attack against each specific user U*. Messages of U has to be observed and recorded in order to synthesize a valid $bio(m')$ message.

Biometric messages are very hard to counterfeit. However, if we suppose, that the attacker is able to extract k_{bio} from $bio(m_1)$, $bio(m_2)$, ..., $bio(m_n)$, and is able to synthesize a $bio(m')$ message using k_{bio}, $bio(m)$ does not provide proper partner identification. On one hand, neither of the above is true for most attackers, on the other hand, since the integrity of messages can be guaranteed, even a simple algorithmic protection is able to prevent attacks. In (Section 8.3.3) I show an example for strengthening biometric messages with simple algorithmic mechanisms.

8.3 Proposed solution

8.3.1 The protocol

I propose, that instead of digitally signing plaintext messages, *the user should sign biometric messages*. These messages already carry the biometric signature of the user. However, since I assume that some attackers are able to forge or manipulate biometric messages, I propose that the biometric message should also be signed by the user's smart card to provide additional cryptographic protection.

In this section, I propose a protocol that provides a combination of biometric and cryptographic protection. The smart card of the user acts as a 'secure time gate' to ensure that the attacker has very little time to manipulate the message on the fly. (See Section 8.4 for a more detailed analysis.) To prevent the attacker from preparing counterfeited messages beforehand, I also introduced a simple algorithmic protection based on one-time-passwords. (See Section 8.3.3 for an example.)

In fact, the smart card computes a *conditional signature* on the user's message m. This conditional signature is equivalent with the users handwritten signature if the digital signature is correct, the biometric structure of the message has not been violated, timestamps originating from the user and from the card are at the appropriate locations, and the message contains the appropriate algorithmic authenticators.

Protocol 12. – Protocol for sending authentic biometric messages

---

1. $U \rightarrow C$: U inserts the card, C obtains the t_{start} timestamp, U removes the card.
2. U creates the $M = bio(t_{beginning}, f_k(m), t_{end})$ message in one of the above biometric formats.
3. $U \rightarrow C$: U inserts the card and sends a hash of M to C, C obtains the t_{sign} timestamp.
4. C calculates the $sign_{k_C}\{I_U, V_U, t_{start}, t_{sign}, k, M\}$ signature.
5. $C \rightarrow U$: $I_U, V_U, t_{start}, t_{sign}, k, sign_{k_C}\{I_U, V_U, t_{start}, t_{sign}, k, M\}$
6. $U \rightarrow R$: $I_U, V_U, t_{start}, t_{sign}, k, M, sign\{I_U, V_U, t_{start}, t_{sign}, k, M\}$
7. R verifies the message using biometric verification and by checking the digital signature and the timestamps. The verification is detailed in the next section.

An implementation of this protocol has been prepared by Örs Sánta. See Section 8.6.

8.3.2 Details of each protocolstep

In this section I give a detailed explanation of each step of Protocol 12.

1. $U \rightarrow C$: Before the creation of the biometric message, user U notifies card C by inserting the card. The user must remove the card after it has acquired a secure timestamp (t_{start}). Card C stores the timestamp t_{start}. The recording of the biometric message begins in the moment, when user U removes the card from the terminal.

   The notification, and the fixed timestamp t_{start} ensure, that terminal T cannot prepare a complete message m' before starting the recording, because biometric timestamp $t_{beginning}$ should match timestamp t_{start}, which is not known before the recording.

   The removal of the card is necessary, because otherwise terminal T may delay the user's notification to the card, so card C would obtain a t_{start} value chosen by terminal T. This would give terminal T more time to prepare a manipulated message. Thus, if the card does not respond to the user, it should be removed after a short time (e.g. 30 seconds), to prevent this kind of attack.

   I emphasize the significance of card removal in general too, since it is a robust method for the user to prevent the terminal from accessing the card. However, the user still cannot know how many T-C transactions were performed while the card was in the reader.

2. Since the integrity of a plaintext message cannot be guaranteed, user U has to prepare message m in a biometric format. Thus, the user prepares message M, where $M = (bio(t_{beginning}, f_k(m), t_{end})$. Biometry ensures integrity, and f is a simple transformation (Section 8.3.3) that protects against attackers who can synthesize a counterfeit $bio(m')$. (Section 8.2) The transformation f has a one-time secret parameter k that is a secret key between U and C. (see 8.3.3)

   The timestamps $t_{beginning}$ and t_{end} should be produced by U's own timer (e.g. watch) at the beginning and at the end of the recording of $bio(f_k(m))$. Since the message is being created on T, the terminal may try to manipulate at this point. Note that timestamps $t_{beginning}$ and t_{end} are 'biometric timestamps', they are contained in the biometric message (e.g. they are pronounced by the user on the video recording).

3. $U \rightarrow C$: User U inserts the card, and sends the M message to it. (More precisely, it sends the hash value only, where the hash is calculated by the terminal.) This is the last point, where T may tamper with the message. After this, the digital signature will be computed behind the hardware firewall of C, and T will not be able to manipulate the signed M. U inserts C into T only for a short time again. If the signature (step 5) does not arrive in e.g. 30 seconds, there is a high chance of an attack similar to the one described at step 1, so the card should be removed. T must forward the message to C now, since it cannot do it when U inserts C next time, because then the t_{end} biometric timestamp in M would be considerably different from the t_{sign} timestamp of the signing. After the hash code of M has arrived, C obtains another secure timestamp t_{sign}.

4. C adds a header including t_{start} and t_{sign} to the message and signs both the message and the header with it's own private key k_c. From this point further manipulation of the message without k_c impossible.
   The header C adds should contain the following:

I_U	The name of the user U and the public key of C (with a certificate).
V_U	Information required for the biometrical identification of U. I_U and V_U make this protocol 'public key', so that it can work without U and R having to agree on U's biometric features using a secure channel.
t_{start}	The time, when the recording of the biometric message was started. This is acquired in step 1.
t_{sign}	The time, when the recording of the biometric message was finished, and the message was signed. This timestamp is acquired in this step.
k	The secret parameter of f in step 2. (Section 8.3.3)

   Thus the signature computed by C becomes: $sign_{k_c}\{I_U, V_U, t_{start}, t_{sign}, k, M\}$

5. $C \rightarrow U$: $I_U, V_U, t_{start}, t_{sign}, k$, $sign_{k_c}\{I_U, V_U, t_{start}, t_{sign}, k, M\}$

6. $U \rightarrow R$: U removes the card from the terminal and forwards the above message to R together with M.

7. R has to verify the following:

   (a) Is $m = bio^{-1}(f_k^{-1}(m))$ a valid message? – Otherwise the message has been tampered with.

   (b) If $m_{bio} = (t_{beginning}, m, t_{end})$, is it true that $t_{end} - t_{beginning} = length(bio(m))$? This forces the attacker, to prepare a m' manipulated message where $length(bio(m)) = length(bio(m'))$.

   (c) Is it true that $t_{start} < t_{beginning} < t_{end} < t_{sign}$? This checks for causality. Any naturally and correctly created message must have this feature. If not, than the message has obviously been tampered with.

   (d) Is $t_{beginning} - t_{start} < t_{safety1}$ and is $t_{sign} - t_{end} < t_{safety2}$, where $t_{safety1}$ and $t_{safety2}$ are system parameters that should be scaled properly. Both of them are relatively small, approximately a few seconds. Their impact is critical on the security of the system, their role is detailed in Section 8.4.
   While t_{start} and t_{sign} are securely acquired by the card, $t_{beginning}$ and t_{end} are acquired by the user and transferred to C with no other protection than the biometric features of U. So, while the attacker cannot modify t_{start} and t_{sign}, there is a chance that $t_{beginning}$ and t_{end} have been tampered with. In this point, R checks if $t_{beginning}$ and t_{end} are close enough to t_{start} and t_{sign}.

   (e) If the message came from U, was it correctly signed by U's card, C? (This is the normal PKI problem that has to be solved in the traditional approach.)
   The digital signature certifies that C was present, and the message has not been modified from the time it was signed. The I_U parameter also serves as a secure way to identify U. However, the digital signature of C does not certify integrity.

   (f) Was the message created by the person, whose characteristics are described by V_U? Since the V_U parameter originates from C, it can be assumed to be authentic. This enables that U and R do not have to agree on U's biometric features before the transaction using a secure channel.

   (g) Was the m message signed by the f_k transformation correctly? (see 8.3.3)

   (h) Is the structure of the biometric message consistent or has it been tampered with? (E.g. Do the lips of the person on the video move, when the speech is heard? Is the background music consistent?)

   (i) Was the theft of C not reported by U?

If all the above have been ensured, R knows that:

- The message was created by U.

- The message was signed by U's card.
- The message was created between t_{start} and t_{sign}.
- The attacker had extremely little time to alter the message.

8.3.3 Algorithmic protection

In Section 8.2.4 I have shown that if a suitable biometric method is selected, it may be troublesome for the attacker to breach the integrity of a biometric message. In Section 8.2.4 I assumed it is very difficult and time consuming for the attacker to synthesize a biometric message that carries the characteristics of user U (without replaying a previously recorded sample). The concept of Protocol 12 is to force the attacker into producing a biometric message in a very short time, where this message has a well defined structure and it also contains fresh information that the attacker cannot guess beforehand.

In some situations, the attacker may be able to guess $t_{beginning}$ and t_{end}, so it seems beneficial to extend the space fo the above fresh information that needs to be included in counterfeit biometric message M'. The following, most simple mechanism of algorithmic protection is sufficient for achieving the above goal:

$$f_k(m) = m|k$$

where $|$ denotes concatenation. This means, one-time password k is appended to the end of message m.

I wish to avoid that U and R has to agree on k, so in my protocol the key is known to U and C (k can be issued to U together with C just like C's PIN code). C is able to send k to R on a secure channel if needed.

8.3.4 How can the card obtain a secure timestamp?

Today's smart cards do not have a timer. However, certain secure computational devices, such as iButtons ([Dallas Semiconductor, 2001]), do have a built-in secure clock. These devices can safely use their own timer to obtain a secure timestamp. Other cards need to get a secure time from a 'secure time server' (STS) with the following protocol:

Protocol 13. – Protocol for obtaining secure timestamps

---

1. $C \rightarrow STS$: n, where n is a nonce value.
2. $STS \rightarrow C$: $t, sign_{k_{STS}}\{n, t\}$

The purpose of this protocol is to obtain ($t_{request}$), the time of step 1. However, communication with STS takes time, so STS will return t instead of $t_{request}$. It also takes time for the message in step 2 to return to C, so C will receive t only at t_{answer}. Note that C can only communicate with the STS through T. While T is not able to modify the timestamp, it may delay any message. Thus, C can only be sure that $t_{answer} \geq t \geq t_{request}$. Alas, C – not having a timer – cannot know the exact $t_{request}$. If T delays step 1, $t_{request} \ll t$ occurs, and the terminal has more time to forge a $bio(m')$ message. If T delays step 2, $t \ll t_{answer}$ would occur, but T would gain no advantage of this.

However, in the above protocol, not only steps 1 and 2 can be delayed. Since C is always a slave in the T-C communication [Berta and Mann, 2000b], step 1 is also initiated by the terminal. Let's shall call this initiation step 0 that takes place after the user inserted the card at t_U. A delay in step 0 causes $t_U \ll t_{request}$, which also gives T more time to forge a $bio(m')$ message. It is trivial that no protocol is totally invulnerable to delays of step 0 and step 1. The fact that C has to *request* the secure time, induces that this request can be delayed. The practical solution for this is timeout. However, C cannot timeout this operation, because C has no timer, so U has to. U must remove the card from the terminal after a given time unconditionally (as it is described in our protocol in Section 8.3.1), and reset the above protocol. Naturally, connecting to a STS takes time, and requires higher $t_{safety1}$ and $t_{safety2}$parameters. (According to the implementation of [Sánta, 2004], 3-5 seconds were more than enough for the entire process for the entire process. I assume that 30 seconds are enough even in the worst-case scenario.) Still, in the protocol described in Section 8.3.1, devices with their own timer are more secure.

8.4 The importance of $t_{safety1}$ and $t_{safety2}$

After the message reached the card the attacker does not have the possibility to manipulate it without having to crack the digital signature. This means the message can only be modified before it is sent to the card. Since the card puts the t_{start} and t_{sign} timestamps onto the message right before signing it, these two cannot be modified by the attacker. The attacker does not have the chance to obtain additional – considerably different – timestamps, because the card is inserted into the terminal only at t_{start} and at t_{sign}. Moreover, the exact value of t_{start} is not known before U starts recording the message, and t_{sign} is not known before U finishes the message. If the attacker wishes to prepare $bio(m')$, it has to prepare the $bio(t'_{beginning})$ and $bio(t'_{end})$ manipulated biometric timestamps that carry the biometric characteristics of U and also satisfy the $t'_{beginning} - t_{start} < t_{safety1}$ and $t_{sign} - t'_{end} < t_{safety2}$ criteria.

The attacker may have the following strategies:

1. The attacker may try to guess t_{start} and t_{sign}, and create $bio(m')$ messages with proper $bio(t'_{beginning})$ and $bio(t'_{end})$ satisfying the above criteria. If $t_{safety1}$ and $t_{safety2}$ are small, there are but minor chances that both timestamps are correct.

2. The attacker may do the manipulations online. While U is recording the $bio(m)$ message, the attacker may manipulate it on T. This requires the attacker to load all the tools for

the precise manipulation of the biometric message onto T. Not all terminals are capable of this. Moreover, $bio(m')$ should have the same length as the $bio(m)$ message U is preparing, because the card's presence is needed for the signature, and the card is only present at t_{sign}. Since T has the time of the recording for manipulation, long messages are more vulnerable. However if the biometric message contains a checksum (even a very simple one) at the end, the attacker may only manipulate if the message is finished. This brings us to the next point.

3. The attacker may also try to manipulate the already finished message. That means that after U finishes $bio(m)$, the attacker may intercept it, and modify certain parts before forwarding it to C. The attacker has only $t_{safety2}$ time for this, and if $t_{safety2}$ is small – according to Section 8.2 – this is almost impossible.

As it can be seen, the exact values of the $t_{safety1}$ and $t_{safety2}$ parameters are critical for the security of the whole system.

8.5 Formal proof of the security of the protocol

I assume that the attacker has observed M_1, M_2, ... M_k biometric messages from user U and he also knows the corresponding m_1, m_2, ... m_k plaintext messages (where $M_i = bio(m_i)$). I also assume that the attacker was able to forge biometric messages M'_1, M'_2, ... M'_n that carry the biometric identity of user U (but he does not have a signature for any of them).

The aim of the attacker is to make R accept biometric message M', where $\nexists i$, $bio^{-1}(M') = m_i$ (i.e. where the meaning of message M' is different from the meaning of any previously observed message).

The relation of timestamps that appear in the protocol is illustrated on Figure 7. By $t_{timestamp}$ I denote the time required for the user to pronounce a timestamp and by $t_{password}$ I denote the time required for the user to pronounce a password.

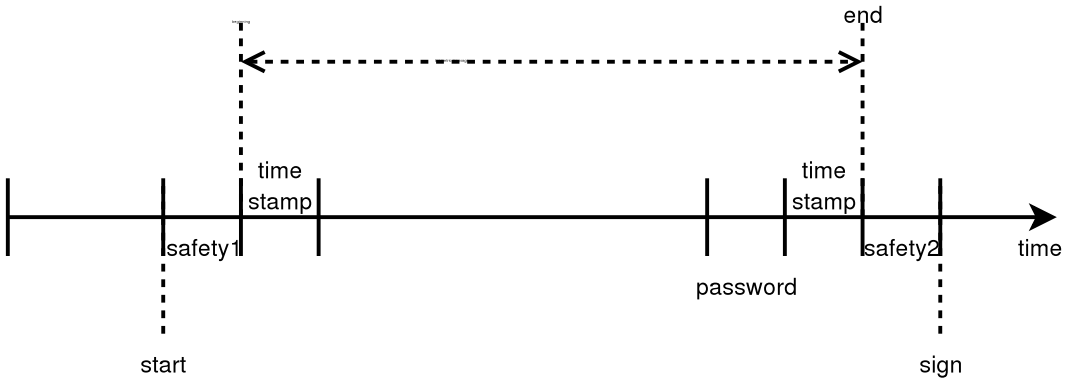

Figure 7: The relation of timestamps that appear in the protocol

We can make the following assumptions about the abilities of the attacker:

(B1) The attacker can produce the digital signature of user U on a message with a negligible probablitiy only, where the message is chosen randomly (with a uniform distribution) from the space of all possible messages.

(B2) The attacker can guess one-time password k with a negligible probability only.

(B3) The attacker has but a negligible chance to produce message M' in $(t_{safety2} + t_{timestamp} + t_{password})$ time for a specific one-time password k, where message M' has the following properties:

a) $\nexists i$ where $m_i = bio^{-1}(M')$.

b) R recognizes M as a biometric message originating from user U, i.e. R can *identify* user U.

c) R does not notice that the message has been tampered with, which means R does not notice that the biometric *integrity* (see Section 8.2.4) of the message has been violated.

d) R recognizes that the message has the structure defined by the protocol (i.e. it contains timestamps $t_{beginning}$ and a t_{end}, and it also contains a one-time password before timestamp t_{end}).

Assumption B3 can be summarized as follows: The attacker has but a negligible chance to produce message M' in $(t_{safety2} + t_{timestamp} + t_{password})$ time for a specific one-time password k, where *remote partner R percieves that message M' is a valid message originating from user U*. This means that the quality of the counterfeited message M' is good enough to trick R. This complex assumption limites the abilities of the attacker relatively to those of the remote parnter.

Proposition 4. *If assumptions B1, B2 and B3 hold, the attacker has a negligible chance to mount a successful attack against Protocol 12.*

Proof. There are three phases when the attacker may mount an attack (i.e. replace the user's message M_k with his own M'):

1. Before the user started to pronounce the one-time password, i.e. before $t_{end} - (t_{timestamp} + t_{password})$. In this phase the attacker does not know one-time password k, and according to Assumption B2, he has but a negligible chance to guess it. Thus, the attacker has a negligible chance to mount a successful attack in this phase.

2. After Phase 1, but before the hash of the biometric message is sent to the card, i.e. between $t_{end} - (t_{timestamp} + t_{password})$ and t_{sign}. Even if the attacker can guess the both the one-time password and timestamp t_{end} at the beginning of this phase, according to Assumption B3, he has too little time to produce a counterfeit message with a quality that R would accept. (Note that R will reject the message if $t_{sign} \geq t_{end} + t_{safety2}$.) Thus, the attacker has a negligible chance to mount a successful attack in this phase.

3. After the message is sent to the card, i.e. after t_{sign}. In this phase the message is protected by the digital signature of the card. According to Assumption 1, the attacker has a negligible chance to forge a valid signature for M'.

□

8.6 An implementation

Örs Sánta has prepared an implementation of this protocol [Sánta, 2004], his system can be used for sending authentic video messages from untrusted terminals. PC-side components of his system were implemented in Java, and Cyberflex Access Java Cards from Axalto were used for implementing the non-standardized smart card functions.

Figure 8: The recording of the message begins when the card is removed

His implementation consists of several components:

- The component that can be used for sending messages runs on the untrusted terminal. (Figure 8.6) As the user inserts and removes the smart card from the reader to obtain t_{start}, the recording of the message begins using a webcam connected to the computer. Naturally, this component was developed with the assumption that an attacker may replace it with a fake one. This component outputs the signed message to a file that can be sent to the recipient via email.

- The component that can be used for verifying messages is partly run by the computer, partly by the human. The computer is responsible for verifying the digital signature and the length of the message (in fact, a further safety parameter had to be introduced in order to overcome time synchronization problems [Sánta, 2004]) and the human is responsible for verifying the biometry of the message and the algorithmic protection.

- The secure time server is a very simple process that provides digitally signed timestamps. In this implementation, the card receives the public key of the secure time server at the time of personalization. The protocol for obtaining the secure timestamp was quicker than expected, 3-5 seconds were usually suitable for $t_{safety1}$ and $t_{safety2}$.

- There is a further component in the system, namely a web server that holds the webpage of the user. The webpage contains the information required for the biometric verification of the user V_U. The web server was introduced due to the severe storage limitations of the smart card. This way, the card passes an Internet address to the remote partner that can be used to authenticate the user. The web server needs to have a valid SSL certificate, so that the remote partner can access its contents in an authenticated way. Naturally, there can be a different web server for each user.

- The smart card is not only a component in the system, but also the tool that the user could rely on for starting and stopping the recording. This ensured that the t_{start} and t_{sign} timestamps are as precise as possible.

  Another critical question was the management of the one-time secret key k that is shared between the user and the card. In the current version, a set of keys can be installed onto the card from a trusted terminal, using a simple PIN code. The keys are generated by the user. If she runs out of keys, she needs to approach another trusted terminal for installing additional keys onto the card.

Part III

Theses

Thesis 1 – Formal model and proof of the limitations of the user

I have developed a formal model that – in commercial applications – better describes the algorithmic abilities of the user at an untrusted terminal than previous models. Within my model I have proven that a user at an untrusted terminal cannot establish a secure (encrypted or authenticated) channel with a third party.

Publications: [Berta and Vajda, 2003b], [Berta and Vajda, 2004] [Berta and Vajda, 2002] [Berta and Vajda, 2003a]

1.1 Model

I have developed a formal model that – in commercial applications – better describes the algorithmic abilities of the user at an untrusted terminal than previous models. In contrast to the usual approach of defining what abilities the user has, I defined what abilities she does not have.

While several models exist that describe the abilities of computers, they cannot be easily applied to humans. Previous models described the user as a poor computer, and estimated the amount of resources the user had. Most previous works assume that e.g. the user can perform a certain number of operations, and show a solution for this scenario.

Unfortunately, it seems to be very hard to describe the algorithmic abilities of a human. On the one hand, different humans have significantly different amount of resources. I assume, most humans are able to perform very few operations (e.g. bit or character operations) per second. However, some humans may possess an extraordinary amount of resources, or they may consider a single message important enough to sacrifice a long period of time for encrypting it or calculating a cryptographic checksum to protect it.

On the other hand, while algorithmic complexity theory usually expresses resource constraints as function of the input length, this approach cannot be easily applied to humans. While the speed of computers increases rapidly every year, we cannot speak of such an increase in case of humans. Neither does the length of the text of documents that need to be processed increase. Perhaps, instead of asymptotic boundaries of required resources, humans would prefer to have a constant boundary that is required for their typical messages.

Even if the message is a few kilobytes long, the time required for processing the message becomes unacceptable for average humans.

In contrast to the approach of defining the user's abilities by describing what she can perform, I have defined the abilities of the user at an untrusted terminal (i.e. she has no trusted computational devices at her disposal) by formalizing what she obviously *cannot* perform:

- The human user *cannot encrypt a message* with a level of security the terminal cannot easily breach (and neither can she decrypt ciphertexts with such a secure encryption).
- The human user *cannot compute a cryptographic checksum* with a level of security the terminal cannot easily breach (and neither can she verify such a strong checksum).

My model describes the scenario of commercial applications (where large masses of users need to send messages without having any extraordinary abilities), if the message to be sent is too long for using one-time-pads.

Within my model I have proven that a user with the above restrictions cannot take part in any protocol that could allow her to send ’long’ messages in an encrypted or authenticated way at a level of security the untrusted terminal cannot easily breach. Thus, a remote partner (who has a trusted computer too) cannot help the user at the untrusted terminal in establishing an encrypted or authenticated channel.

This means, no possible solution exists for the user for secure (encrypted or authenticated) communication within the severe but realistic boundaries of my model.

I do not know of any similar negative formal proof on the possibilities of the human.

1.2 Secrecy

Within my model I have proven that a user at an untrusted terminal cannot take part in any protocol that allows her to send messages with a level of secrecy the untrusted terminal cannot easily breach.

See proof in Section 4.2.

1.3 Message authenticity

Within my model I have proven that a user at an untrusted terminal cannot take part in any protocol that allows her to send messages with a level of authenticity the untrusted terminal cannot easily breach.

See proof in Section 4.3.

Thesis 2 – A solution based on revocable signatures

I have developed a framework that allows the user to perform digital signature operations at untrusted terminals, review the signatures from trusted terminals, and to revoke unintended signatures. I have showed that help from a trusted third party (TTP) is required in most practical scenarios. I have developed a solution where the user is able to retain a provable degree of privacy with respect to the TTP.
Publications: [Berta et al., 2005b], [Berta et al., 2003], [Berta et al., 2004a], [Berta et al., 2004b]

Current systems that use digital signatures do not have any protection against the attacks of untrusted terminals. Although there are solutions that provide means for the user for authenticating certain terminals, authenticating the terminal does not guarantee that the terminal has not been tampered with.

Credit card based payment systems already provide a way of mitigating the problem of untrusted terminals by allowing the user to repudiate certain transactions afterwards. I reckon, this is not a technical but a statistical method that relies on the experience that most users and most merchants are not malicious, so most transactions are not revoked. Credit card systems are still cost-effective, but the number of attacks increases rapidly, and new technologies allow villains to mount attacks at large scale. Existing solutions provide neither the user nor the merchant any proof of the transaction, usually both parties are at the mercy of the credit card company.

I have shown that – within the boundaries of my model – the user cannot send authentic messages from untrusted terminals. I extended my model by posing different, weaker requirements on digital signatures by following the paradigm of Rivest: the digital signature should not be non-repudiable proof, it should merely be plausible evidence.

I have developed a framework that – similarly to credit card systems – allows the user to review signatures and revoke unintended ones. In contrast to credit card based payment systems, my framework is not limited to payment, but the user can sign an arbitrary document at the untrusted terminal. Within my framework, a conditionally signed document is sent to the recipient. The user cannot change it anymore, but she can claim that she did not sign that particular document. If she revokes an intended signature, the recipient can use the revoked signature as plausible evidence for proving that the user was present at the terminal and initiated a trasaction.

I have identified two classes of signature revocation protocols. Those protocols where the user needs to confirm intended signatures (and unconfirmed signatures are rejected) are more simple, but a merchant cannot do business with users who forget to confirm their signatures. Those protocols where the user needs to revoke unintended signatures (and unrevoked signatures are accepted) are more suitable for practical use, but I showed that they all require help from a trusted third party (TTP). While the user may trust this TTP for signature revocation, she does not want the TTP to know where, when and what messages she signed. I have developed a protocol, where the user retains a provable degree of privacy: In case of this protocol I have proven that if all parties behave well, the user retains unconditional privacy (with respect to the TTP), and if the signature needs to be revoked, the user retains the degree of privacy provided by the anonymous channel that is used for submitting the revocation.

Thesis 3 – Biometric solution

I propose that a user should use her biometric abilities to protect her messages sent from untrusted terminals. I have developed a protocol where the user sends 'biometric messages' (easily produced, but very costly and time-consuming to attack) to a remote partner, and a smart card ensures that a malicious terminal has very little time to perform the attack.

Publications: [Berta and Vajda, 2003a], [Berta, 2002], [Berta and Vajda, 2002], [Bencsáth and Berta, 2004]

Many systems rely on biometric methods for authenticating users. For example, fingerprint recognition, voice pattern recognition or iris scan can be used for authenticating a user when she is accessing a certain room or device. Other systems recognize biometric patterns of a user. For example, a voice recognition system may recognize words a human says, so the human may issue orders to a computer in speech.

I have combined the above two systems, and proposed that their combination can be used against malicious terminals. A user should send her messages in a biometric format; e.g. as a video message. Biometric messages encapsulate the content of a plaintext message and the user's biometric identity. Any part of the message carries the biometric identity of the user, and these messages also have an implicit structure. (For example, there are regularities in how the rhythm or the tone of the user's voice may alter during a speech.) I found that voice, video and handwriting fulfil the above criteria.

Using biometric messages is safer (i.e. harder to counterfeit) than using plaintext ones. On the one hand, an attacker needs to *prepare a different attack against each user*. The attacker needs to observe many biometric samples of a targeted user to counterfeit a message. It is not possible to easily attack all users in a system simultaneously (where e.g. a computer virus can send fake messages in the name of many users). On the other hand, *the attack is costly and time-consuming*, it may even require human interaction. Performing such an attack requires resources that many adversaries do not possess. I assume that some attackers have enough resources to counterfeit biometric messages, but the *attacker requires significantly more time* for forging a biometric message than for forging a plaintext message.

I have proposed a protocol where the biometric protection of the message is strengthened by the digital signature of the smart card. I assume that the card can obtain the current time securely and also assume that the user has a timer independent from the terminal (e.g. a watch).

During the protocol the user has to read her watch and announce the current time, thus she produces a 'biometric timestamp'. Before the smart card signs the biometric message, it appends a securely obtained timestamp to the message. A party checking the signature on the biometric message should also check the biometric integrity of the message, and verify that the distance between the biometric timestamp originating from the user and the secure timestamp originating from the card is less than a safety-parameter.

I claim that with the proper calibration of the biometric method and the amount of the safety-parameter it can be assured that *a malicious terminal does not have enough time to perform the attack*. I have proven that if the biometric method is properly calibrated, my protocol is secure.

Part IV
Appendix

References

[Abadi et al., 1992] Abadi, M., Burrows, M., Kaufman, C., and Lampson, B. (1992). Authentication and Delegation with Smart-cards. Theoretical Aspects of Computer Software: Proc. of the International Conference TACS'91, Springer, Berlin, Heidelberg.

[Albert et al., 2002] Albert, I., Balássy, G., Rajacsics, T., Péteri, S., and Charaf, H. (2002). Kód-eredet alapú biztonság a .NET platformon. NetworkShop 2002, Eger.

[Anderson, 1996] Anderson, N. (1996). Why Cryptosystems Fail. Willaim Stallings, Practical Cryptography for Data Internetworks, IEEE Computer Security Press, 1996.

[Anderson and Kuhn, 1997] Anderson, R. and Kuhn, M. (1997). Low Cost Attacks on Tamper Resistant Devices. M Lomas et al. (ed.), Security Protocols, 5th International Workshop, Paris, Proceedings, Springer LNCS 1361, pp 125-136, ISBN 3-540-64040-1.

[Anonymizer Inc., 1999] Anonymizer Inc. (1999). . `http://www.anonymizer.com`.

[Asokan et al., 1999] Asokan, N., Debar, H., Steiner, M., and Waidner, M. (1999). Authenticating Public Terminals. Computer Networks, 1999.

[Balfanz and Felten, 1999] Balfanz, D. and Felten, E. (1999). Hand-Held Computers Can Be Better Smart Cards. Proceedings of USENIX Security '99 Washington, DC.

[Bellare and Rogoway, 2002] Bellare, M. and Rogoway, P. (2002). Introduction to modern cryptography. `http://www.cse.ucsd.edu/users/mihir/cse207/classnotes.html`.

[Bencsáth, 2004] Bencsáth, B. (2004). The problems and connections of network virus protection and the protection against denial of service attacks. Proceedings of the Networkshop 2004 Conference, NIIF, Hungary, 2004.

[Bencsáth and Berta, 2004] Bencsáth, B. and Berta, I. (2004). Hiteles üzenet küldése rosszindulatú terminálról. Networkshop 2004, NIIF, Györ.

[Bencsáth and Vajda, 2004] Bencsáth, B. and Vajda, I. (2004). Protection Against DDoS Attacks Based On Traffic Level Measurements. International Symposium on Collaborative Technologies and Systems, The Society for Modeling and Simulation International, 2004, Waleed W. Smari, William McQuay, pp. 22-28., The Society for Modeling and Simulation International, San Diego, CA, USA, January, Simulation series vol 36. no. 1., ISBN 1-56555-272-5.

[Berta, 2002] Berta, I. (2002). Hiteles üzenet küldésének problémája nem biztonságos termináról. HTE-BME 2002 Korszerű távközlő és informatikai rendszerek és hálózatok konferencia, Budapest, 2002.

[Berta, 2004] Berta, I. (2004). Why are not digital signatures spreading as quickly as it was expected? MBA dissertation, Buckinghamshire Chilterns University College, Buckinghamshire Business School, Számalk Open Business School, `http://www.crysys.hu/publications/files/Berta2004mba.pdf`.

[Berta and Bencsáth, 2002] Berta, I. and Bencsáth, B. (2002). Empiric examination of random number generators of smart cards. HTE-BME 2002 Korszerű távközlő és informatikai rendszerek és hálózatok konferencia, Budapest, 2002 (in Hungarian).

[Berta et al., 2004a] Berta, I., Buttyán, L., and Vajda, I. (2004a). Mitigating the Untrusted Terminal Problem Using Conditional Signatures. Proceedings of International Conference on Information Technology ITCC 2004, IEEE, Las Vegas, NV, USA, April.

[Berta et al., 2005a] Berta, I., Buttyán, L., and Vajda, I. (2005a). Standards for Product Security Assessment. Handbook of Information Security, Chapter 55, Edited by Hossein Bidgoli, John Wiley and Sons, accepted, to appear.

[Berta and Mann, 1999] Berta, I. and Mann, Z. (1999). A hitelesség biztosításának lehetőségei intelligens smart card segítségével. Scientific student circle conference, Budapest University of Technology and Economics.

[Berta and Mann, 2000a] Berta, I. and Mann, Z. (2000a). Programozható chipkártyák - elmélet és gyakorlati tapasztalatok. Magyar Távközlés, 2000, vol 4.

[Berta and Mann, 2000b] Berta, I. and Mann, Z. (2000b). Smart Cards – Present and Future. Híradástechnika, Journal on C^5, 2000., vol 12.

[Berta and Mann, 2002] Berta, I. and Mann, Z. (2002). Evaluating Elliptic Curve Cryptography on PC and Smart Card. Periodica Polytechnica, Electrical Engineering, 2002, vol. 46/1-2, pp. 47-75, Budapest University of Technology and Economics.

[Berta et al., 2003] Berta, I. Z., Buttyán, L., and Vajda, I. (2003). Mitigating the untrusted terminal problem using conditional signatures. CrySyS Lab Technical Report, `http://www.crysys.hu/publications/files/BertaBV2004condsig.pdf`.

[Berta et al., 2004b] Berta, I. Z., Buttyán, L., and Vajda, I. (2004b). Privacy protecting protocols for revokable signatures. Cardis2004, Toulouse, France.

[Berta et al., 2005b] Berta, I. Z., Buttyán, L., and Vajda, I. (2005b). A framework for the revocation of unintended digital signatures initiated by malicious terminals. IEEE Transactions on Secure and Dependable Computing, (accepted, to appear).

[Berta and Vajda, 2002] Berta, I. Z. and Vajda, I. (2002). Message Authentication using Smart Cards and Biometry. Second Central European Conference on Cryptography (HajduCrypt'02), Debrecen, Hungary.

[Berta and Vajda, 2003a] Berta, I. Z. and Vajda, I. (2003a). Documents from Malicious Terminals. SPIE Microtechnologies for the New Millenium 2003, Bioengineered and Bioinspired Systems, Spain.

[Berta and Vajda, 2003b] Berta, I. Z. and Vajda, I. (2003b). Limitations of humans at malicious terminals. TATRACRYPT 2003.

[Berta and Vajda, 2004] Berta, I. Z. and Vajda, I. (2004). Limitations of humans at malicious terminals. Tatra Mountains Mathematical Publications, to appear.

[Blake-Wilson et al., 2003] Blake-Wilson, S., Nystrom, M., Hopwood, D., Mikkelsen, J., and Wright, T. (2003). Transport Layer Security (TLS) Extensions. `http://www.ietf.org/rfc/rfc3546.txt`.

[Brands, 1994] Brands, S. (1994). Untraceable off-line cash in wallets with observers. In Crypto'93 Springer-Verlag, LNCS 773 pp. 302-318.

[Brands and Chaum, 1993] Brands, S. and Chaum, D. (1993). Distance-bounding protocols (extended abstract). In Theory and Application of Cryptographic Techniques, pages 344 359.

[Buttyán and Vajda, 2004] Buttyán, L. and Vajda, I. (2004). Kriptográfia és Alkalmazásai. Typotex, Budapest.

[CEN CWA 14890-1, 2003] CEN CWA 14890-1 (2003). CWA 14890-1 Application Interface for smart cards used as Secure Signature Creation Devices – Part 1 – Basic requirements.

[Chaum, 1981] Chaum, D. (1981). Untraceable electronic mail, return addresses and digital pseudonyms. Communications of the ACM, v24, n.2 pp.84-88.

[Chaum, 1982] Chaum, D. (1982). Blind signatures for untraceable payments. Advances in Proceedings of Crypto 82, D. Chaum, R.L. Rivest, & A.T. Sherman (Eds.), Plenum, pp. 199-203.

[Claessens et al., 1999] Claessens, J., Preneel, B., and Vandewalle, J. (1999). Anonymity controlled electronic payment systems. Proceedings of the 20th Symposium on Information Theory in the Benelux, Haasrode, Belgium, May 27-28.

[Clarke et al., 2002] Clarke, D., Gassend, B., Kotwal, T., Burnside, M., Dijk, M. v., Devadas, S., and Rivest, R. (2002). The Untrusted Computer Problem and Camera-Based Authentication.

[Dallas Semiconductor, 2001] Dallas Semiconductor (2001). iButton Home Page. http://www.ibutton.com/.

[Dantu et al., 2004] Dantu, R., Cangussu, J., and Yelimeli, A. (2004). Dynamic Control of Worm Propagation. Proceedings of International Conference on Information Technology ITCC 2004, IEEE, 2004, IEEE, Las Vegas, NV, USA, April.

[DIN V66291-4, 2000] DIN V66291-4 (2000). Chipcards with digital signature application/function according to SigG and SigV – Part 4: Basic Security Services.

[EU Directive, 1999] EU Directive (1999). Directive 1999/93/EC of the European Parliament and of the Council of 13 December 1999 on a Community framework for electronic signatures.

[FINREAD, 2004] FINREAD (2004). FINREAD Technical Specifications. `http://www.finread.com`.

[Franklin and Reiter, 1996] Franklin, M. and Reiter, M. (1996). Fair Exchange with a Semi-Trusted Third Party. `http://portal.acm.org/citation.cfm?id=266424`.

[Franklin and Yung, 1992] Franklin, M. and Yung, M. (1992). Towards provably secure efficient electronic cash. Columbia Univ. Dept. of CS TR CSUCS-018-92.

[Fung et al., 2001] Fung, W., Golin, M., and Gray, III., J. (2001). Protection of Keys against Modification Attack. HKUST Theoretical Computer Science Center Research Report 2001-04.

[Girard et al., 2004] Girard, P., Giraud, J., and Gauteron, L. (2004). Secure electronic signatures when Trojan Horses are lurking. e-smart 2004, Sophia Antipolis.

[Gobioff et al., 1996] Gobioff, H., Smith, S., and Tygar, J. D. (1996). Smart Cards in Hostile Environments. In Proceedings of the 2nd USENIX Workshop on Electronic Commerce, Nov 1996, 23-28.

[Goldreich, 1997a] Goldreich, O. (1997a). The Foundations of Modern Cryptography. In Proceedings of Crypto97, Springer's Lecture Notes in Computer Science, Vol. 1294, `http://theory.lcs.mit.edu/~oded/frag.html`.

[Goldreich, 1997b] Goldreich, O. (1997b). Introduction to Complexity Theory. `http://www.wisdom.weizmann.ac.il/~oded/cc02.html`.

[Gruschka et al., 2004] Gruschka, N., Reuter, F., and Luttenberger, N. (2004). Checking and Signing XML Using Java Smart Cards. CARDIS 2004, Tolouse, France.

[Hungarian Law, 2001] Hungarian Law (2001). Act XXXV of 2001 on electronic signatures. `http://www.hif.hu/english/menu4/m4_8/es.pdf`.

[IBM Magyarország Rt. et al., 2004] IBM Magyarország Rt., E-Group Magyarország Rt., Berta, I., Vajda, I., Buttyán, L., Bencsáth, B., and Veiland, T. (2004). HUNEID specifikáció – Az eID kártyán elhelyezkedő fájlszerkezet és kártyainterfész specifikációja. Információs

Társadalom Koordinációs Tárcaközi Bizottság, Intelligens Kártya Munkacsoport, http://www.itktb.hu.

[Izu et al., 2004] Izu, T., Itoh, K., and Tkenaka, M. (2004). Efficient Countermeasures against Power Analysis. CARDIS 2004, Tolouse, France.

[Jakobsson and Raïhi, 1998] Jakobsson, M. and Raïhi, D. (1998). Mix-based electronic payments. Fifth Annual Workshop on Selected Areas in Cryptography (SAC'98), Queen's University, Kingston, Ontario, Canada.

[Kahn, 1967] Kahn, D. (1967). The Codebreakers: The story of secret writing. Macmillan Publishing Company, New York, USA, 1967.

[Karpovsky et al., 2004] Karpovsky, M., Konrad, J., J., K., and Taubin, A. (2004). A Differential Fault Analysis Attack Resistant Architecture of the Advanced Encryption Standard. CARDIS 2004, Tolouse, France.

[Kay, 2004] Kay, R. (2004). Phising. Computerworld, January, 2004.

[Kelsey et al., 2000] Kelsey, J., Schneier, B., Wagner, D., and Hall, C. (2000). Side Channel Cryptanalysis of Product Ciphers. Journal of Computer Security, v. 8, n. 2-3, 2000, pp. 141-158.

[Kerckhoff, 1883] Kerckhoff, A. (1883). La Cryptographie Militaire. Journal des Sciences Militaires, 1883, Jan.

[Kincses, 2004] Kincses, Z. (2004). Attack Tree of Smart Card. e-smart 2004, Sophia Antipolis.

[Lee and Kim, 2002] Lee, B. and Kim, K. (2002). Fair Exchange of Digital Signatures using Conditional Signature. SCIS 2002, Symposium on Cryptography and Information Security.

[Loney and Lemos, 2004] Loney, M. and Lemos, R. (2004). Study: Unpatched PCs compromised in 20 minutes. http://news.com.com/Study:+Unpatched+PCs+compromised+in+20+minutes/2100-7349_3-5313402.html.

[Matsumoto, 1996] Matsumoto, T. (1996). Human-Computer cryptography: An attempt. In ACM Conference on Computer and Communications Security, pp 68-75.

[McGregor et al., 2003a] McGregor, J., Karig, D., Shi, Z., and Lee, R. (2003a). Enlisting hardware architecture to thwart malicious code injection. Proceedings of the International Conference on Security in Pervasive Computing (SPC-2003), LNCS 2802, pp. 237-252, Springer Verlag, March 2003.

[McGregor et al., 2003b] McGregor, J., Karig, D., Shi, Z., and Lee, R. (2003b). A processor architecture defense against buffer overflow attacks. Proceedings of the IEEE International Conference on Information Technology: Research and Education (ITRE 2003), pp. 243-250, August 2003.

[Mossberg, 2004] Mossberg, W. (2004). How to Protect Yourself From Vandals, Viruses If You Use Windows. Personal Technology from the Wall Street Journal, Sept 16, http://ptech.wsj.com/archive/ptech-20040916.html.

[Naor, 1991] Naor, M. (1991). Bit Commitment Using Pseudo-Randomness. Journal of Cryptology: the journal of the International Association for Cryptologic Research, vol 2, pp 151-158.

[Naor and Pinkas, 1997] Naor, M. and Pinkas, B. (1997). Visual Authentication and Identification. Lecture Notes in Computer Science, vol 1294.

[Naor and Shamir, 1995] Naor, M. and Shamir, A. (1995). Visual Cryptography. Lecture Notes in Computer Science, vol 950, pp 1–12, 1995, http://citeseer.nj.nec.com/naor95visual.html.

[Pering et al., 2003] Pering, T., Sundar, M., Light, J., and Want, R. (2003). Photographic Authentication through Untrusted Terminals. IEEE Pervasive Computing Issue 5, March, 2003.

[Pyo, 2004] Pyo, C. (2004). Run-time Detection of Buffer Overflow Attacks without Explicit Sensor Data Objects. Proceedings of International Conference on Information Technology ITCC 2004, IEEE, 2004, IEEE, Las Vegas, NV, USA, April.

[Rankl and Effing, 1997] Rankl, W. and Effing, W. (1997). Smart Card Handbook. John Wiley & Sons, 2nd edition, ISBN: 0471988758.

[Reiter and Rubin, 1998] Reiter, M. and Rubin, A. (1998). Crowds: anonymity for Web transactions. ACM Transactions on Information and System Security, vol 1, 1998, http://citeseer.ist.psu.edu/284739.html.

[Rivest, 2001] Rivest, R. (2001). Issues in Cryptography. Computers, Freedom, Privacy 2001 Conference http://theory.lcs.mit.edu/~rivest/Rivest-IssuesInCryptography.pdf.

[Rónyai et al., 1999] Rónyai, L., Ivanyos, G., and Szabó, R. (1999). Algoritmusok. Typotex, Budapest.

[SANS, 2004] SANS (2004). SANS Institute Internet Storm Center: Windows XP: Surviving the First Day. http://www.sans.org/rr/papers/index.php?id=1298.

[Schneier, 1996] Schneier, B. (1996). Applied Cryptography. John Wiley & Sons, ISBN: 0471117099.

[Schneier, 1999a] Schneier, B. (1999a). Attack Trees. Dr. Dobb's Journal December 1999, http://www.schneier.com/paper-attacktrees-ddj-ft.html.

[Schneier, 1999b] Schneier, B. (1999b). The Solitaire Encryption Algorithm. http://www.counterpane.com/ solitaire.htm.

[Schneier, 2003] Schneier, B. (2003). SSL Flaw. Crypto-Gram Newsletter, March 15, 2003, http://www.schneier.com/crypto-gram-0303.html.

[Schneier and Shostack, 1999] Schneier, B. and Shostack, A. (1999). Breaking up is Hard to do: Modelling security threats for smart cards. USENIX Workshop on Smart Card Technology, Chicago, Illinois, USA, http://www.counterpane.com/smart-card-threats.html.

[Serjantov and Danezis, 2002] Serjantov, A. and Danezis, G. (2002). Towards an Information Theoretic Metric for Anonymity. Privacy Enhancing Technologies, PET2002.

[Shannon, 1949] Shannon, C. E. (1949). Communication Theory of Secrecy Systems. Bell Systen Technicl Journal, vol 28, pp 656–715, 1949.

[Shao et al., 2004] Shao, Z., Xue, C., Zhuge, Q., Sha, E., and Xiao, B. (2004). Security Protection and Checking in Embedded System Integration Against Buffer Overflow Attacks. Proceedings of International Conference on Information Technology ITCC 2004, IEEE, 2004, IEEE, Las Vegas, NV, USA, April.

[Shaw, 1905] Shaw, G. (1905). Man and Superman.

[Singh, 2000] Singh, S. (2000). The Code Book : The Science of Secrecy from Ancient Egypt to Quantum Cryptography. Anchor Books; ISBN: 0385495323.

[Skorobogatov and Anderson, 2002] Skorobogatov, S. and Anderson, R. (2002). Optical fault induction attacks. Cryptographic Hardware and Embedded Systmes (CHES 2002.), citeseer.ist.psu.edu/skorobogatov02optical.html.

[Smith et al., 1999] Smith, S., Perez, R., Weingart, S., and Austel, V. (1999). Validating a High-Performance, Programmable Secure Coprocessor.

[Sánta, 2004] Sánta, r. (2004). Smart Card based Application for Message Authentication in a Malicious Terminal Environment. Master's Thesis, Fachhochschule Darmstadt.

[Stabell-Kulo et al., 1999] Stabell-Kulo, T., Arild, R., and Myrvang, P. (1999). Providing Authentication to Messages Signed with a Smart Card in Hostile Environments. Usenix Workshop on Smart Card Technology, Chicago, Illinois, USA, May 10-11, 1999.

[Syverson et al., 1997] Syverson, P., Goldschlag, D., and Reed, M. (1997). Anonymous Connections and Onion Routing. IEEE Symposium on Security and Privacy, Oakland, California.

[TCPA, 2004] TCPA (2004). Tcpa. http://www.trustedpc.org.

[TeleTrusT WG2, 2003] TeleTrusT WG2 (2003). TeleTrusT Working Group 2 "Security Architecture and Smart Cards" – German Office Identity Card.

[Thompson, 1984] Thompson, K. (1984). Reflections on Trusting Trust. Communication of the ACM, Vol 29. No. 8, August, 1984 pp 761-763.

[Torrington and Hall, 1987] Torrington, D. and Hall, L. (1987). Human Resource Management. Prentice Hall, Fourth Edition, ISBN: 0136265324.

[White et al., 2003] White, S., Swimmer, M., Pring, E., Arnold, W., Chess, D., and Morar, J. (2003). Anatomy of a Commercial Grade Immune System. IBM Thomas J. Watson Research Center, `http://www.research.ibm.com/antivirus/SciPapers/White/Anatomy/anatomy.html`.

[Yee and Tygar, 1995] Yee, B. and Tygar, J. D. (1995). Secure coprocessors in electronic commerce applications. First USENIX Workshop on Electronic Commerce,.

[Ylönen et al., 2000] Ylönen, T., Kivinen, T., Saarinen, M., Rinne, T., and Lehtinen, S. (2000). Ssh authentication protocol. Internet Engineering Task Force, Network Working Group, draft-ietf-secsh-userauth-09.txt. http://www.ietf.org/internet-drafts/draft-ietf-secsh-userauth-09.txt.

[Zoreda and Oton, 1994] Zoreda, J. L. and Oton, J. M. (1994). Smart cards. Artech House, ISBN: 0890066876.

[Zou et al., 2003] Zou, C., Gao, L., Gong, W., and Towsley, D. (2003). Monitoring and Early Warning for Internet Worms. `http://tennis.ecs.umass.edu/ czou/research/monitoringEarlyWarning.pdf`.

The author's publications in the subject of the dissertaion

Journal papers

1. I. Zs. Berta, I. Vajda, A framework for the revocation of unintended digital signatures initiated by malicious terminals, IEEE Transactions on Secure and Dependable Computing, (accepted, to appear), 2005

2. I. Zs. Berta, I. Vajda, Limitations of humans when using malicious terminals, Tatra Mountains Mathematical Publications, vo 29, pp: 1-16, 2004.

3. I. Zs. Berta, Z. Á. Mann, Evaluating Elliptic Curve Cryptography on PC and Smart Card, Periodica Polytechnica, Electrical Engineering, vol 46, pp 47-73, 2001.

4. I. Zs. Berta, I. Berta, Hardver és szoftver biztonság I, Elektrotechnika, 2003, vol. 2003/10, pp. 266-269.

5. I. Zs. Berta, I. Berta, Hardver és szoftver biztonság II, Elektrotechnika, 2003, vol. 2003/11, pp. 298-301.

6. I. Zs. Berta, Z. Á. Mann, Smart Cards - Present and Future, Híradástechnika, Journal on C5, 2000, 12, pp: 24-29.

7. I. Zs. Berta, Z. Á. Mann, Programozható chipkártyák és azok biztonsága, Magyar Távközlés, XI. Évfolyam, 4. szám, pp: 8-13 2000, 4.

Book chapter

8. I. Zs. Berta, L. Buttyán, I. Vajda, Standards for Product Security Assessment, Handbook of Information Security, John Wiley & Sons, 2005.

Conference papers

9. I. Zs. Berta, L. Buttyán, I. Vajda, Privacy Protecting Protocols for Revocable Digital Signatures, Proceedings of Cardis 2004, Kluwer, pp 67-82, Toulouse, France, 2004.

10. I. Zs. Berta, L. Buttyán, I. Vajda, Mitigating the Untrusted Terminal Problem Using Conditional Signatures, Proceedings of International Conference on Information Technology ITCC 2004, IEEE, 2004, IEEE, Las Vegas, NV, USA, April, 2004.

11. I. Zs. Berta, I. Vajda, Documents from Malicious Terminals, SPIE's Microtechnologies for the New Millenium 2003, Bioengineered and Bioinspired Systems, pp: 325-336, Maspalomas, Spain, 2003.

12. I. Vajda, I. Zs. Berta, Limitations of users when using malicious terminals, Third Central European Conference on Cryptography (Tatracrypt'03), 2003.

13. I. Berta, I. Zs. Berta, EMC and Information Security in Power Engineering, Proceedings of the 5th Power Systems Conference, Timisoara, Romania, 2003.

14. I. Zs. Berta, I. Vajda, Message Authentication using Smart Card and Biometry, Second Central European Conference on Cryptography (HajduCrypt) 2002, Debrecen, Hungary, 2002.

15. I. Zs. Berta, Z. Á. Mann, Programozható chipkártyák kriptográfiai alkalmazása, NetWorkShop'2001 Konferencia Anyag, CD Proceedings, Sopron, 2001.

16. I. Zs. Berta, B. Bencsáth, Hiteles üzenet küldése rosszindulatú terminálról, NetWorkShop2004, NIIF, CD Proceedings, Győr, 2004.

17. B. Bencsáth, I. Zs. Berta, Intelligens kártyák véletlenszám-generátorának empirikus vizsgálata, HTE-BME 2002 Korszerű távközlő és informatikai rendszerek és hálózatok konferencia, BME, 2002, BME.

18. I. Zs. Berta, Hiteles üzenet küldésének problémája nem biztonságos terminálról, HTE-BME 2002 Korszerű távközlő és informatikai rendszerek és hálózatok konferencia, BME, 2002, BME.

Theses

19. I. Zs. Berta, Why are digital signatures not spreading as quickly as it was expected? MBA dissertation, Buckinghamshire Chilterns University College, Számalk Open Business School, 2004

20. I. Zs. Berta, Programozható chipkártyák nyújtotta biztonság MSc. Thesis, BME, 2001

Others

21. I. Zs. Berta, I. Vajda, L. Buttán, B. Bencsáth, T. Veiland, E-Group Magyarország Specification of the Hungarian electronic ID card (HUNEID) Információs Társadalom Koordinációs Tárcaközi Bizottság, Intelligens Kártya Munkacsoport, `http://www.itktb.hu`, 2004

22. I. Zs. Berta, L. Buttyán, I. Vajda, Mitigating the Untrusted Terminal Problem Using Conditional Signatures, CrySyS Lab Technical Report, `http://www.crysys.hu/publications/files/BertaBV2004condsig.pdf`

23. I. Zs. Berta, I. Berta, Hardware and Software Security, Magyar Elektrotechnikai Egyesület XLIX. vándorgyülés, Sopron, 2002.

24. I. Zs. Berta, I. Vajda, Eliminating Man-in-the-Middle attacks of Malicious Terminals, Workshop organised by the IST Coordination Point of the Ministry of Education, Budapest, 2002.

25. I. Zs. Berta, Z. Á. Mann, Az elliptikus görbék elméletén alapuló nyilvános kulcsú kriptográfia elemzése chipkártyás és PC-s környezetben, Scientific Student Circles, 1st prize, BME, 2000.

26. I. Zs. Berta, Z. Á. Mann, A hitelesség biztosításának lehetőségei intelligens smartcard segítségével, Scientific Student Circles, 1st prize, BME, 1999.

List of Figures

List of Protocols

Abbreviations

ATM: Automatic teller machine

AES: Advanced encryption standard

DES: Data encryption standard

ICC: Integrated circuit card

IFD: Interface device

LS: Log server

MAC: Message authentication code

PIN: Personal identification number

SSL: Secure socket layer

SSH: Secure shell

TCPA: Trusted Computing Platform Alliance

TLS: Transport layer security

TTP: Trusted third party

Notations

|: Conditional probability
||: Concatenation
$\models$: Models
$\exists$: Exists
$\forall$: For all
==: Equals
$[x]$**:** The value of expression x