Is an electronic passport usable as an authentication token?

Jean-Pierre de Rooij

Radboud University Nijmegen
Educational Institute for Computing and Information Sciences

Is an electronic passport usable as an authentication token?

Bachelor thesis

Jean-Pierre de Rooij
0249475
jrooij@science.ru.nl
Educational Institute for Computing and Information Sciences
Radboud University Nijmegen
July 11, 2007

Supervisor: Ronny Wichers Schreur

Preface

This document is the final requirement in completing my Bachelor's degree in Computing Science at the Radboud University of Nijmegen. This thesis describes my research on the usability of the electronic passport as an authentication token.

This thesis consists of seven chapters. Chapter 1 is the research introduction. Chapter 2 will explore the electronic passport and the protocols used to communicate with it. Chapter 3 presents the authentication method design when we will use the electronic passport as an authentication token. To analyze if there are any security risks involved with that authentication method, we describe several possible attacks in chapter 4. Chapter 5 will give a framework for the implementation of the authentication method for the Microsoft Windows 2000/XP operating system. Finally, chapter 6 summarizes the achievements and chapter 7 presents directions for future work.

Jean-Pierre de Rooij

Contents

List of Figures

Chapter 1

Introduction

Security is a very hot issue nowadays. More and more data is stored in a digital way. Obviously, we want this data to be stored securely. But we, as owners of the data, must have access to it. Therefore we have our own account, traditionally secured with a username and password. The problem here is the amount of different accounts we have.

For almost every automated system, we have our own account. We often use the same username and password combination for all of these systems. Otherwise we have to remember tons of different usernames and passwords. And if we instead do use different usernames and passwords, the password strength is most of the times the weakest link. When our passwords are very strong, it is most of the times written down on the well known yellow post-it notes next to our monitor. And if we do not find such post-it notes, the password is probably of the kind *1234* or *letmein*. The problem here is that this way of authentication is only based on what we know. And sadly enough, people have limited memory.

To reduce the risk that a hacker has access to all of your accounts when he has knowledge of just one single username and password combination, we can extend our security method. Instead of only using something we know, we can secure our accounts with something we have. And that *something we have* could be our electronic passport. The challenge is to see if the electronic passport can be used as a so called *authentication token*.

1.1 Problem proposition

When we want to extend our traditional authentication method with something we have, we first have to decide what we want to use as *something we have*. In this thesis we explore the possibilities of using the electronic passport as *something we have*. To see if we can use our electronic passport as an authentication token, we will answer the following research question:

> *Is an electronic passport usable as an authentication token?*

1.2 Justification

The use of an extra token for providing more secure systems, is very popular. Take for example the financial world, where customers can only do online transactions with the use of their mobile phone. The use of the mobile phone is seen as a secure token next to the traditional username and password. This way of authentication is called *two-factor authentication* because we not only need the factor *something we know*, but also the second factor *something we have*. But why build only secure systems for the financial world. Why not when we log on to our computer at work? If it is possible to use an electronic passport as an authentication token, we can build more secure systems with something almost everybody has: the electronic passport. I think it is very interesting to find out if our electronic passport can be used in a two-factor authentication environment.

1.3 Research structure

To give an answer to the research question, we have to define the word *usable* in that question. In this thesis we consider the electronic passport *usable* as an authentication token if:

- We can interact with the electronic passport;
- We can design an authentication method that uses the electronic passport;
- The designed method is secure enough;
- We can find an implementation for that method.

Therefore we will introduce four sub-questions to give a *"yes"* or *"no"* answer to the research question:

1. *Can we interact with the electronic passport?*
2. *Can we design an authentication method that uses the electronic passport?*
3. *Is the designed method secure enough?*
4. *Can we implement the authentication method?*

If all the four sub-questions result in a *"yes"*, the research question can be answered with *"yes"*. But if one or more of the four sub-questions result in a *"no"* as answer, we consider the research question as *"no"*.

There can be a discussion about the the phrase *secure enough* in the sub-question *"Is the designed method secure enough"*. In chapter 4 we describe six possible attacks on the authentication method. At the end of that chapter we have a good view on the possible security risks that are involved with the chosen authentication method. In our final conclusions we describe what we consider as *"secure enough"*.

Chapter 2

Electronic passport protocols study

To communicate with the electronic passport (ePassport), we first need the rules governing the syntax, semantics, and synchronization of communication specified for the electronic passport. These rules or protocols can than be used for communication between the electronic passport and a terminal. A terminal in this context, is nothing more than an electronic device which can read the data page and the contactless chip of the electronic passport.

There are four high level protocols, some of which are optional, to trust and retrieve the data stored on the electronic passport. The description about these protocols is adapted from the ICAO Technical Report[ICA04b] and the Radboud University paper [HHJ+06]. Protocol diagrams are adapted from the Radboud University sheets [JS05].

2.1 Basic Access Control

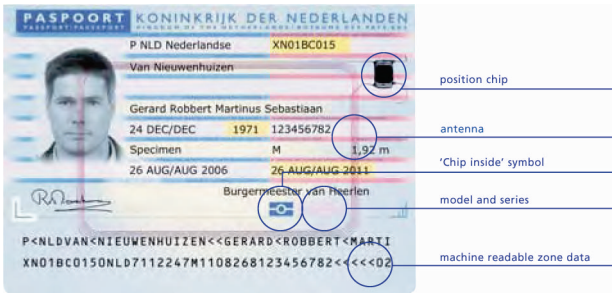

Figure 2.1: The ePassport characteristics with the MRZ at the bottom

The **optional** *Basic Access Control* (BAC), is used as a consent and confidentiality mechanism and so prevents skimming and eavesdropping of the ePassport's chip. And furthermore, the BAC sets up a secure channel for communication between the ePassport's chip and the terminal. To protect the data on that

chip, this challenge-response protocol denies access to it, unless the terminal can prove that it is authorized to do so.

To prove this, the terminal first has to read the two lower printed lines on the data page inside the ePassport. These lines are called the *machine-readable zone* (MRZ), which contains the passport number, birth date and expiry date. The characteristics of the ePassport, with the above mentioned MRZ, are shown in figure 2.1. When the terminal successfully reads the contents of the MRZ, the terminal derives the access keys K_{ENC} and K_{MAC} from the data it reads as shown in figure 2.2. Since the MRZ cannot be read from a closed ePassport, it is accepted that the ePassport was handed over for inspection. This prevents skimming of the data.

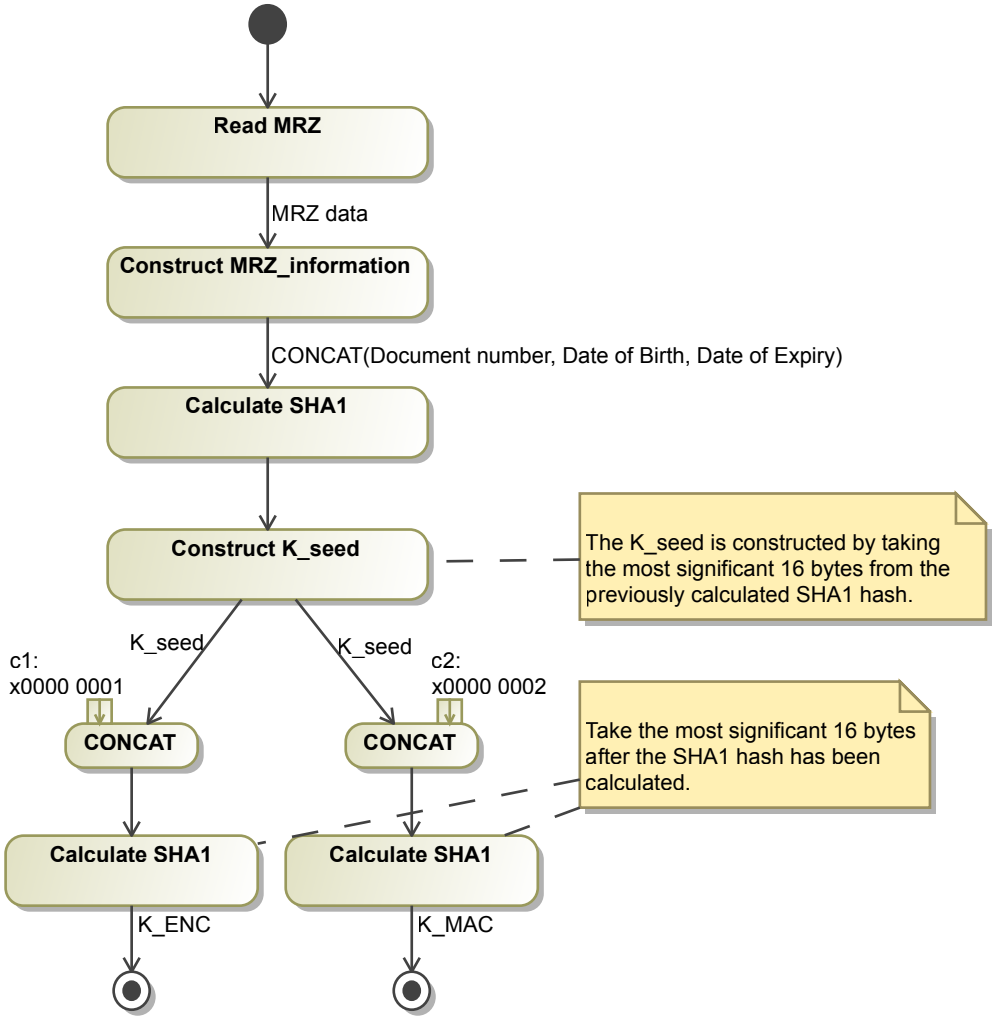

Figure 2.2: The computation of K_{ENC} and K_{MAC}

Now the ePassport and terminal mutually authenticate and derive session keys. The authentication and key establishment is provided by a three pass challenge-response protocol as shown in figure 2.3. The abbreviations used in figure 2.3 are: nonce N, key K, passport p and terminal t. All nonces are 8 byte long and the keys are 16 byte long. After a successful execution of the authentication protocol both the ePassport and terminal compute session keys KS_{ENC} and KS_{MAC} using the key derivation mechanism shown in figure 2.2 with $(K_P \ XOR \ K_T)$ as key seed. Due to the encrypted channel that has been set up, eavesdropping on the communication would require a considerable effort.

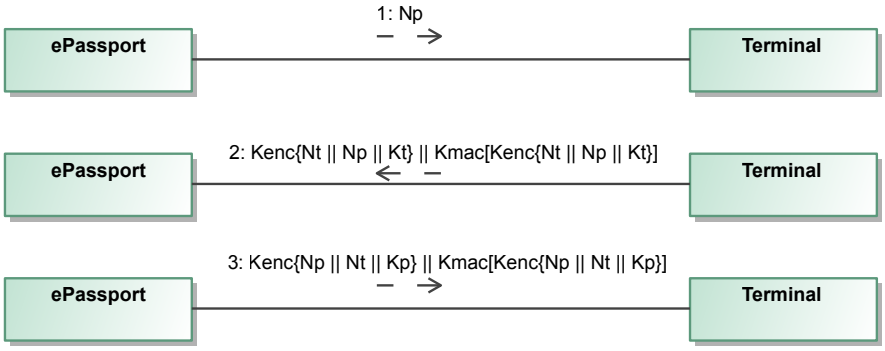

Figure 2.3: Authentication and key establishment

2.2 Passive Authentication

Passive Authentication (PA) is the second and **required** protocol which proves that the contents of the Document Security Object (SO_D) and Logical Data Structure (LDS) are authentic and not changed. The LDS contains all the ePassport's data stored in separate groups, such as the MRZ, name and photo. To prove that the LDS contents are authentic, each group of the LDS is first hashed and then the hashes of all groups are signed with the private key of the Document Signer (DS).

To prove that the LDS contents are authentic and unchanged, the terminal has to compare the LDS groups with the ones stored in the SO_D. To do so, the terminal needs to have knowledge of the key information of the DS. With the stored Document Signer Public Key (KPu_{DC}), the digital signature of the SO_D is verified. This ensures that the SO_D is authentic. Now the terminal reads the relevant LDS groups, hashes them and compare the results with the corresponding hash value stored in the SO_D. This ensures that the contents of the LDS are authentic and unchanged.

Notice that this protocol **does not** prevent cloning of the chip content. To prevent this, we need another protocol.

2.3 Active Authentication

To prevent the earlier mentioned cloning of the chip content, the **optional** integrity mechanism called *Active Authentication* (AA) comes in action. To ensure that the chip has not been substituted, a challenge-response protocol is used between the ePassport and the terminal. For this protocol the chip contains its own Active Authentication Key pair (KPr_{AA} and PKu_{AA}).The public key (PKu_{AA}) is stored in Data Group 15 of the LDS, the hash representation of the PKu_{AA} is stored in the SO_D and the private key (PKu_{AA}) is stored in the chip's secure memory.

To ensure that the chip content is genuine, the first thing we need is the MRZ. Because the BAC is optional, the MRZ has to be visually read from the ePassport's data page if not already done. The MRZ is then compared with the MRZ value in Data Group 1 of the LDS. Similarity ensures us that the visual MRZ is authentic en unchanged, because the required PA already checked the authenticity and integrity of Data Group 1. Because PA also checked Data Group 15, which holds the public key (PKu_{AA}), we also know that PKu_{AA} is authentic and unchanged. In the challenge-response protocol (figure 2.4) the terminal first sends a nonce N to the ePassport. This nonce N is then signed by the ePassport with $KPru_{AA}$ and send back to the terminal. The terminal finally uses the PKu_{AA} to compare the previously send nonce N with the signed one received form the ePassport. Similarity proves that the SO_D belongs to the data page, the chip is genuine and chip and data page belong to each other.

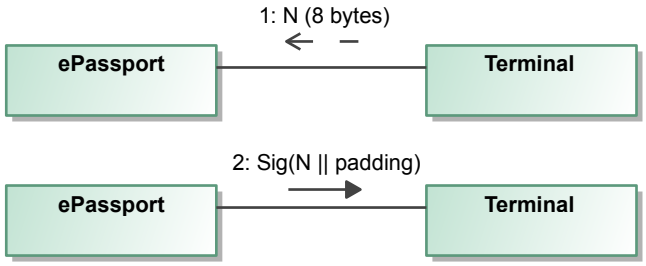

Figure 2.4: The Active Authentication challenge-response protocol

2.4 Extended Authentication

The *extended authentication* (EA) protocol implements the protection of other biometric data, such as fingerprints. Because these data is quite sensitive, access to it is limited and only granted to a limited number of organizations such as the government. Therefore the EA protocol can not be used in an authentication mechanism.

Chapter 3

Authentication method design

All of the data stored on the ePassport is considered as privacy sensitive. Giving away this data can be a serious risk. But without reading any data from the passport, it is impossible to use it as an authentication token. Therefore we have to examine which data we have to store and retrieve from the ePassport, to minimize the risk that a potential hacker has access to a fortune of privacy-sensitive information.

3.1 The authentication method

When using traditional authentication methods, the credentials used to log on to a system are somewhere stored on a (logon)server. This also holds when we use the ePassport to authenticate ourselves. When we have determined which credentials we need to store, we can split the authentication method into two sub methods: user enrollment and user logon.

3.2 The used protocols in short

To determine which data we need as our credentials, let us summarize what we know after the protocols study:

- Basic Access Control: Used for skimming protection and setting up a secure channel. BAC is not required, but ePassports which have the BAC implemented must be able to use it as an authentication token. For the BAC we only need the printed MRZ on the ePassport's data page.
- Passive Authentication: Ensures that the Document Security Object and Logical Data Structure are authentic and not changed. PA is required and will be implemented in the authentication mechanism.
- Active Authentication: Ensures that the ePassport's chip is not a clone. Because the authentication method is not only based on something we know (a username and password), but also on something we have (the

ePassport self), we only accept ePassports which have AA implemented although it is an optional protocol.

3.3 User enrollment

Before you can log on to a secure system, it will be required to provide some information for enrollment. This information is stored in a database to grant or deny access based on the given credentials when logging on to the secured system. After enrollment we know that the presented ePassport belongs to the user and that the chip inside the ePassport belongs to the ePassport. The enrollment process is similar to the one a border officer would go trough. The enrollment is composed of four distinct phases as shown in figure 3.1.

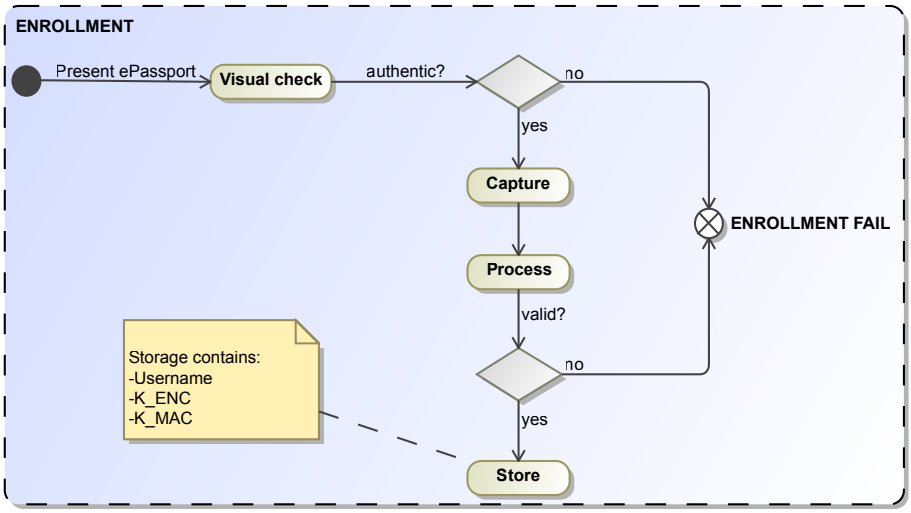

Figure 3.1: The user enrollment diagram

3.3.1 Visual check

First the system administrator performs the basic security checks like examining the ePassport's visual security characteristics, comparing the photo on the data page with the user in front of him, etc. When the system administrator has verified that the ePassport belongs to the user, other checks can be made at this stage. Take for example the computers in an university. When a new user wants access to the campus' network, he first has to go to the system administrator who takes care of the enrollment. But in this scenario it could be necessary to know is the new user is a student or staff member at that university. At this stage of the enrollment process, the system administrator has to verify this.

3.3.2 Capture

Now the system administrator opens the enrollment tool and enters the first two data fields: the username and MRZ. To capture the MRZ, he opens the ePassport to unveil the data page and then types in or scans the printed MRZ which is then used as input for the *process* phase. After a click on the *next button*, a new *user object* is created which holds the unique username and MRZ. The MRZ is only used during the enrollment process. After enrollment the MRZ will not be stored in the credential database.

3.3.3 Process

Now that the user object has been created, the tool performs the required steps to go through the ePassport's protocols. With the MRZ in hand, we first go into the *Basic Access Control* (BAC). As explained in section 2.1, the K_{ENC} and K_{MAC} are calculated and an encrypted channel for further communication with the ePassport is set up. Now we have been granted access to the public data stored on the ePassport's chip and secure communications between the ePassport and our system.

Now the tool goes into *Passive Authentication* (PA) to compare the data in the *Logical Data Structure* with the data stored in the *Security Object*. Therefore the *Document Security Object* (SO_D) is read from the chip. First the Document Signer (DS) is read from the chip. The digital signature of the SO_D is verified, using the *Document Signer Public Key* (KPu_{DS}). The KPu_{DS}, together with the Document Signer Certificate (C_{DS}), can be obtained in two ways:

- Retrieve the C_{DS} from the ePassport optionally stored in the SO_D;
- If not stored in the SO_D, download the C_{DS} from the issuing state.

This ensures that the SO_D is authentic and unchanged and that all of its contents can be trusted. To complete the PA the LDS groups 1 and 15 (which respectively hold the MRZ and the Active Authentication Public Key) are read, hashed and compared with the corresponding hash values stored in the SO_D. After PA we know that the contents of the SO_D and the LDS are authentic and not changed.

Finally, to verify that the ePassport is not a clone, the tool goes into *Active Authentication* (AA). The previous PA process ensures that the Active Authentication Public Key (KPu_{AA}) in data group 15 used for AA, is authentic and unchanged. This public key KPu_{AA} is now used together with the private key KPr_{AA}, stored in the chip's secure memory, in a challenge-response protocol. After a successful challenge-response protocol it is proven that the SO_D belongs to the data page, this chip is genuine and that the chip and data page belong to each other. The ePassport is thus authentic, valid and not a clone.

3.3.4 Store

Finally the tool stores the username and K_{ENC} and K_{MAC} read from the *user object*. Because the MRZ yields privacy-sensitive information, we store the derived K_{ENC} and K_{MAC} from the MRZ rather than the MRZ itself for further authentication with the ePassport. After storage, the *user object* is deconstructed and the enrollment is finished.

3.4 User logon

When the enrollment of a new user has finished, the user can now log on to the system. This logon process is composed of three distinct phases as show in figure 3.2.

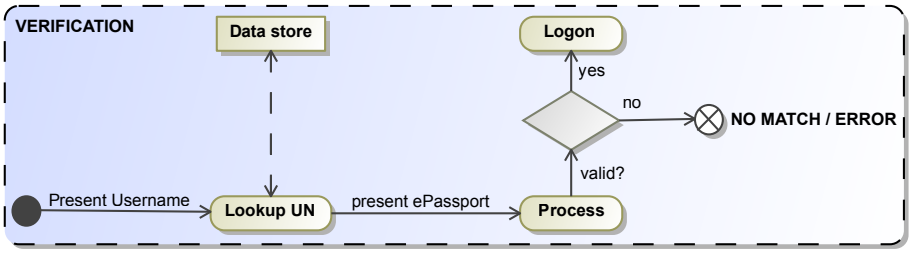

Figure 3.2: The user logon diagram

3.4.1 Lookup UN

First the user has to present his username. The authentication method tries to find the given username in the data store. This data store is the credential database used by the enrollment process. When the username does not exist, we do not panic and keep this information secret. The compare method issues the appropriate error messages.

3.4.2 Process

When the username is looked up, the user presents his ePassport. The K_{ENC} and K_{MAC} stored with the username are then used to set up the encrypted session channel, as part of the BAC. Now we take the same steps as mention in section 3.3.3, to go through PA and AA.

3.4.3 Validation

Finally we have the validation check. When the previous process of BAC, PA and AA succeeds, we know that the K_{ENC} and K_{MAC} belong to the presented ePassport. The user is logged on to the system and has access to the secured environment. When the previous process of BAC, PA and AA fails, the following three scenarios are possible:

- The given username does not have a matching K_{ENC} and K_{MAC} for the presented ePassport or vice versa;
- Or the given username does not exist in the data store and therefore does not have a matching K_{ENC} and K_{MAC} for the presented ePassport;
- Or the presented ePassport is not authentic.

If one or more of the above mentioned scenarios occurs, the user is not logged on to the system and presented with the logon dialog box again.

Chapter 4

Security analysis

Now we have a credential data store, an enrollment tool and a log on mechanism but how secure is this setup? What if a hacker gains root access to the system for instance? To ensure the whole setup used for authentication is secure, we examine what a hacker can do and if that would be a risk. In this chapter we will introduce Trudy as the hacker [Sch94].

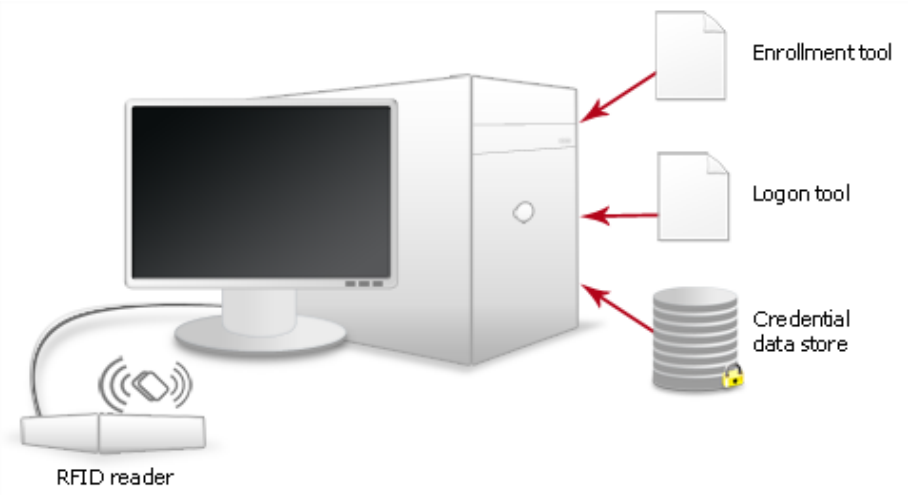

Figure 4.1: The authentication system setup

4.1 The authentication system setup

We assume that the whole authentication mechanism is set up on one single computer as shown in figure 4.1. This means that we have one computer with an RFID reader, the enrollment tool, the logon tool and the credential data store installed. Each of these four components could be vulnerable and therefore ideal for a hacker to try to gain access to or break into. A second computer with the same authentication method setup is used as a second victim. Is it possible

to use the information stolen from the first computer to log on to the second computer?

4.2 Some possible attacks

To analyze the risk that a hacker uses a backdoor in one or more of these components to access privacy-sensitive information or gain access to the secured environment even on another system without logging on, we consider the six upcoming possible attacks.

4.2.1 Session eavesdropping

During user enrollment and user log on, the ePassport has to communicate with the reader. Because of the use of a Radio Frequency Identification (RFID) chip in the ePassport, the data transmitted between the two travels through the air. To intercept the radio waves with this data, Trudy uses a large antenna. With the intercepted data, Trudy tries to log on to the computer system using a replay-attack. To see if this is possible, let us first summarize which data is send between the ePassport and reader during enrollment or log on.

The first protocol we need to go through is BAC. The challenge-response protocol that BAC uses is described in figure 2.3 of section 2.1. When Trudy intercepts the data send between the ePassport and the reader during that challenge-response session, Trudy has the following data in hand:

- The 8 byte long nonce N_P from the ePassport;
- The generated $K_{ENC}\{N_R||N_P||K_R\}||K_{MAC}[K_{ENC}\{N_R||N_P||K_R\}]$ from the reader;
- The generated $K_{ENC}\{N_P||N_R||K_P\}||K_{MAC}[K_{ENC}\{N_P||N_R||K_P\}]$ from the ePassport.

Because the reader generates a random nonce N_R and key K_R, Trudy cannot use an earlier intercepted stream unless the N_R and K_R are exactly the same as the one she intercepted. With an 8 byte long nonce N_R, we can generate 2^{64} different nonces. With the 16 byte long key N_R, we can generate 2^{128} different keys. The chance that Trudy can use the earlier intercepted stream with the same N_R and K_R will be 1 in 2^{192}. We see that Trudy cannot use an intercepted stream to perform a replay attack unless she is very lucky. Therefore it is very important to have an highly random random generator.

This rather trivial conclusion can be extended when we introduce another component Trudy can eavesdrop: the MRZ. Trudy can use a camera or simply sneak into the victim's room when he or she is away and quickly take a look inside the ePassport where the MRZ is located. Now going through the BAC is not a problem anymore. Trudy simply generates her own nonce N_P and sends it to he reader. The reader will respond to that challenge by sending back

$$K_{ENC}\{N_R||N_P||K_R\}||K_{MAC}[K_{ENC}\{N_R||N_P||K_R\}].$$

Because Trudy now has the MRZ, she can derive the K_{ENC} and K_{MAC} as described in figure 2.2, decrypt the response from the reader and generate and send

$$K_{ENC}\{N_P||N_R||K_P\}||K_{MAC}[K_{ENC}\{N_P||N_R||K_P\}].$$

After this successful execution of the authentication protocol, a secure channel is setup and Trudy must go through the next protocol: PA. First the SO_D is read from the ePassport's chip, but from which ePassport? The problem here is that Trudy does not have the ePassport of the victim. Again no luck for Trudy with this approach. Trudy can not gain access to the secured environment using this technique, but she does have the MRZ with all the privacy-sensitive information. But that would be the same as losing your ePassport in a crowded bar or lunchroom.

The above described passive and active replay attacks, where based on partial eavesdropping. Trudy intercepted the first signals and immediately concluded she could not use the information to perform a replay attack. But Trudy can still eavesdrop all the data send during BAC, PA and AA. Could that be a risk? Again, Trudy has to accept a *no* as an answer to that question. Because after a successful authentication during BAC, all further communication is protected by Secure Messaging. The data groups of the LDS read during PA for instance, are encrypted thanks to Secure Messaging. If Trudy is in possession of the MRZ, she can decrypt the secure message and read the transmitted data groups 1 and 15 of the LDS. Group 1 holds the MRZ, which Trudy already has, and group 15 holds the Active Authentication Public Key. With that public key we go trough the challenge-response protocol of the AA. For a successful AA we need the ePassport itself with the matching private key. The private key is stored in the protected area of the ePassport's chip and not send to the reader. Again we see that Trudy needs the ePassport to authenticate herself.

4.2.2 Obtaining the credential data store

In this scenario Trudy becomes root on the computer and has free access to our credential database. Great news for Trudy, but is this a serious risk? With the credential database in hand, Trudy can see all the stored usernames with their K_{ENC} and K_{MAC}. The only thing Trudy now knows is who are enrolled to the computer system and has access to the secured environment. Because the username does not contain privacy-sensitive information, this is no big deal. The same holds for the K_{ENC} and K_{MAC}. These two keys are indeed derived from the privacy-sensitive MRZ, but are both hashed SHA-1 values and can therefore not be reversed to their original value. No need to worry either. But can Trudy use the username and K_{ENC} and K_{MAC} to gain access to another computer system where the same authentication mechanism is installed? Because Trudy would still need the matching ePassport to go successfully through PA and AA for the selected account during log on, she still will not be able to log on to another computer system.

When there is more user information stored in the credential database, like an e-mail address or telephone number, it should be obvious that in that case Trudy does have some valuable information. She still can not log on with the extra information, but she can use them for other purposes like spamming.

4.2.3 Compromising during enrollment

After the attempt in the previous section, Trudy is still root and tries to compromise the computer system during user enrollment. Further more, because of her root privileges, Trudy already has access to the whole computer system including the secured environment. But access as root to the computer system is location and time based. Trudy can only take control over the system where she has broken into and only for the time her backdoor has not been fixed by the system administrator. It would be far more effective when Trudy can use the stolen credentials to use on another computer system to log on. In the previous section we saw that obtaining the credential data store or eavesdropping the communication between ePassport and reader would not lead to this situation. Maybe a modified enrollment tool can do. Because Trudy provided herself with the right privileges to change the enrollment tool, she can manipulate the enrollment process.

As described in section 3.3.3, after the visual check the system administrator types or scans in the MRZ which is thus in hands of Trudy. Now the enrollment tool performs the basic steps to go trough BAC, PA and AA. As described earlier there will be no extra information unveiled for Trudy, but she modified the enrollment tool at the point where PA take place. Instead of only reading group 1 and 15 from the *LDS*, Trudy lets the tool read all the groups of the *LDS*. Now Trudy has all the privacy-sensitive information stored in the *LDS* as shown in figure A.1 of appendix A. When the process continues and the tool goes trough AA, there will be nothing more to intercept for Trudy. She will still not be able to log on to another computer system without the physical ePassport in hand.

In this scenario Trudy can capture a huge amount of privacy-sensitive information. The only downside for this approach is that it would take a lot more time to capture all the data groups of the *LDS* than the usually two data groups. The picture stored in data group 2 for example is saved in JPEG 2000 format and would take considerable more time to retrieve than the MRZ stored in data group 1. But when the system administrator notices the more time-consuming enrollment process, it is already to late and Trudy can go home with the information. But the downside for Trudy is that the system administrator knows that there was a hacker on the system. And like most hackers, Trudy rather did not wanted to expose herself.

4.2.4 Compromising during log on

Besides the fact that in this scenario the MRZ is not typed in by the user, the previous section can be applied to discuss this attack. The only difference here is that a user, most of the time with less knowledge than a system administrator, would not notice the more time consuming log on process when Trudy read the whole *LDS*. The user would simply blame the computer for not working hard enough or simply ignore it and take a sip of coffee. But when Trudy reads the content of the *LDS* when the logon process has finished, the user would not even notice any difference in time during log on. But therefore we have to assume that the user is not taking away his ePassport from the RFID reader after logon.

Again, Trudy can not log on herself without the ePassport, but with the ignorance of the user(s) she can collect privacy-sensitive information every time a (new) user logs on to the computer system with the modified logon tool installed.

4.2.5 Social engineering

From an interview of Kevin Mitnick, an infamous hacker in the 1980s and 1990s, with the BBC News Online 6:

> *"The biggest threat to the security of a company is not a computer virus, an unpatched hole in a key program or a badly installed firewall. In fact, the biggest threat could be you.*
>
> *What I found personally to be true was that it's easier to manipulate people rather than technology. Most of the time organizations overlook that human element."*

Social engineering is all about the manipulation of people. This can be done in a digital form or in a human face-to-face manner. The digital form, like phishing, can be a not authentic email form your creditcard company which is send to you with the request to reply and send your creditcard number so *"they can update their database"*. If you do so, which obviously would not be a very cleaver idea, your reply to that email would not go to you creditcard company, but to Trudy who is afterwards booking a vacation with your money.

Because our authentication mechanism is based on a two-factor mechanism, Trudy not only needs the username of her victim, but also the matching ePassport. We will outline that the ePassport as token in combination with a username enhances the security. Consider the following scenario where Trudy acts like being the system administrator at the University of Nijmegen. By choosing a public accessible place, nobody will notice that Trudy is a total stranger out there. When Trudy is inside the building, she can talk to any of the passing employees with the request to hand over their ePassport. Trudy just tells them that it would be for system maintenance only, and that they will get their ePassport back as soon as possible. If a trustful employee is willing to cooperate, Trudy is in! But is it really effective?

For the time she has the ePassport, it surely is effective. Trudy can for example install a backdoor under the user's account to visit the compromised system from another system. But Trudy has also access to all the data stored on the system and therefore can put the data on a removable disk for offline usage. But with the use of an ePassport as token instead of just a smartcard, it is not that likely that an employee is willing to give his ePassport away that easy. And if Trudy can find an employee who is willing to give it away, it is again time and location based. In fact Trudy can just walk away and never come back, but in that case the employee must notice that something is not right. Blocking the account and report the ePassport as stolen would be the next step for the fooled employee.

Social engineering is a serious risk and applied on a large company with a lot of employees, the chance on a "mission accomplished" for Trudy is present.

4.2.6 Grandmaster Chess Attack

Annex G.2.2 of [ICA04b] describes the possibility of a Grandmaster Chess Attack. In our scenario Trudy can present her ePassport, equipped with a special chip, to the computer system that uses our authentication method. Trudy's chip works as a proxy for a genuine chip located in a remote place like a public network or via the internet. See figure 4.2 for an illustrated view of the Grandmaster Chess Attack.

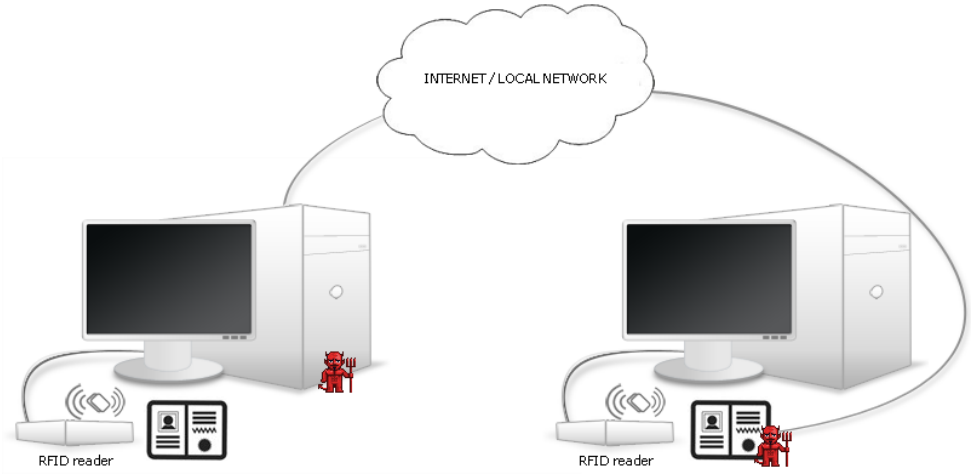

Figure 4.2: The Grandmaster Chess Attack system setup with the victim's system on the left and Trudy's system on the right.

Take for example two computer systems with our authentication method installed. The victim's system with the authentic ePassport used as a proxy, is connected to the internet or local network and is equipped with a backdoor so Trudy can access it over the internet or local network. The fake ePassport Trudy uses is equipped with software that can relay the requests from the system Trudy wants to gain access to, and sends them to the victim's system. The victim puts his genuine ePassport on his RFID reader for authentication and after a successful logon, he does not take away his ePassport. Trudy uses her fake ePassport with a special chip to start authentication on her local computer system. When the authentication system needs to communicate with Trudy's fake ePassport, the chip inside the ePassport communicates with the genuine ePassport of the victim. Trudy's local computer system is not able to notice that it has authenticated a remote chip instead of the fake presented chip. After a successful authentication, Trudy has access to her local system and uses the identity of the victim.

We see that a Grandmaster Chess Attack is possible and that Trudy can gain access to the secured environment using this scenario. The only requirement for this attack is that the victim has his genuine ePassport put on top of his RFID reader during the attack. We also assume that it is technical possible to build such a fake ePassport with the special chip.

Beth and Desmedt [BD90] propose solutions to the Grandmaster Chess Attack, using highly synchronized clocks, but these solutions are impractical for smart cards, because smart cards are not continuously powered nor do they include clock functions. A smart card can only learn the time from the external world, and the external world is not necessarily trustworthy [KK05]. This means roughly that there must be a measurable difference in time between sending a challenge and receiving a response in an authentic system setup and in a Grandmaster Chess Attack system setup. In this chapter we only described some possible attacks and not how we can solve them. For more in-depth information about solving the Grandmaster Chess Attack, see [BD90].

Chapter 5

Authentication method software design

Now that we have an authentication method designed and analyzed if there will be any security risks involved with the design, we can work on the software design. In this chapter a software design for the Microsoft Windows XP / 2000 operating system is given. Therefore we have to know the architecture of Microsoft's authentication mechanism and how we can extend it to work with our ePassport as security token.

5.1 Welcome to the Microsoft jungle

Finding a method to implement an extra security token, such as our ePassport, to interact with the Windows logon mechanism, is not a straight forward and well documented process. The research for such a method leads to various approaches.

Because the ePassport has a lot of similarities with a smartcard, searching for a smartcard based logon mechanism was the first approach. This leads to the extensively documented Microsoft's *Smart Card Modules* architecture. Although this architecture is very well documented, the road to a fully functional smart card module is a very time intensive job. Because of the restricted time schedule for this thesis, research for another architecture went on.

The next architecture for creating custom logon experiences is Microsoft's *Credential Provider*. After a quick research, this architecture is multi-factor based but only Windows Vista compatible. The documentation also points to the fact that the Credential Provider is the replacement of the Microsoft GINA model. After researching the Microsoft GINA model, we discovered that this was the Windows XP / 2000 based model for modifying the way we log on to either of the two operating systems.

GINA stands for *Graphical Identification and Authentication*. By modifying GINA we can also change the look and feel of the logon dialog box presented to the user, hence the name. But the main advantage of working with the Microsoft GINA model, is that there already exists a plugin-based modified version of GINA developed by Nathan Yocom. This modified GINA, called pGina, extends the original GINA with a convenient plugin-based architecture.

The only thing we have to concentrate on, is the design of our plugin that has to communicate with and validate the ePassport.

In this chapter we describe how the Windows logon architecture works, where the Microsoft GINA model comes in action during logon and how we can use pGina to extend the current username and password based logon to work together with our ePassport.

5.2 The Windows authentication architecture

Windows is built upon four basic logon types: interactive local logon, noninteractive network logon, batch, and service logon. The difference between interactive logon and noninteractive logon is whether the user's credentials are verified against a local database or a database elsewhere in the network. Batch logon is used by task schedulers (e.g., the "at" or "WINat" service); service logon is used by services. All of them have slightly different architectures. In this chapter, we will focus on interactive local logon.

5.2.1 Interactive local logon

The interactive local logon as described in this subsection, including figure 5.1, is adapted from the Microsoft TechNet article [Mic03].

The interactive local logon process, illustrated in figure 5.1, is the first step in user authentication and authorization and provides a way to identify authorized users and determine whether they are allowed to log on and access the system. The interactive local logon architecture includes the following components:

- Winlogon executable program
- Graphical Identification and Authentication DLL (the GINA)
- Local Security Authority (LSA)
- Authentication packages (Kerberos or NTLM)

The interactive local logon begins with the user pressing CTRL+ALT+DEL to initiate the logon process. The CTRL+ALT+DEL keystroke is called a secure attention sequence (SAS); Winlogon registers this sequence during the boot process to keep other programs and processes from using it. The GINA generates the logon dialog box. A user who logs on to a computer using either a local or domain account must enter a user name and password, which form the user's credentials and are used to verify the user's identity.

The following steps are taken to accomplish a successful local logon. First Winlogon and the GINA collect the user's credentials and then calls the LSA to give it the user's credentials. The LSA verifies the user's identity and then returns a logon success and the user's access token to Winlogon and the GINA. The user's access token holds the security ID (SID) that Windows uses to allow access to the resource in question when logged on. Winlogon and the GINA then activate the user's shell by creating a new process, such as Explorer.exe.

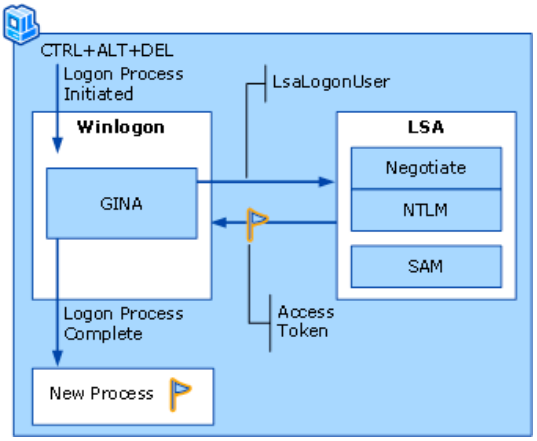

Figure 5.1: The Interactive local logon architecture

5.2.2 Extending Microsoft GINA

Because GINA is the only pluggable part of the WinLogon, we can replace or extend it to meet our demands. Besides a username and password, we would like to present and validate the ePassport for a successful logon. When replacing to original GINA with our custom designed GINA, we have to re-write all the original code and add some custom functions for the authentication with the ePassport. That would cost a lot of extra time and not exactly what we want. Therefore we choose to extend the original GINA with our custom GINA. We want to slightly change the look of the default logon dialog box, to let the user know that logging on to the system differs from the standard log on procedure. We also want to handle our custom ePassport authentication and pass the user's credentials to the original GINA for further authentication with the LSA. This extended GINA design is already implemented in the customizable package named pGina.

5.3 Introduction to pGina

pGina is an open source project of Nathan Yocom *[www.pgina.org]*. pGina works by inserting itself into the Windows operating system as a GINA module, hence the name. When pGina is installed, it inserts itself between the Winlogon process and Microsofts GINA and handles those things directly related to its own operation (logon, locking, etc.) and passes everything else transparently to the Microsoft module. When Winlogon loads pGina, pGina in turn loads a plugin chosen by the administrator. When a user attempts to log in, pGina uses the selected plugin to determine if and how they should be authenticated and/or authorized. The interactive logon using pGina and a custom RFID authentication plugin is illustrated in figure 5.2. This figure is a modified version of figure 5.1, obtained from the Microsoft TechNet article [Mic03].

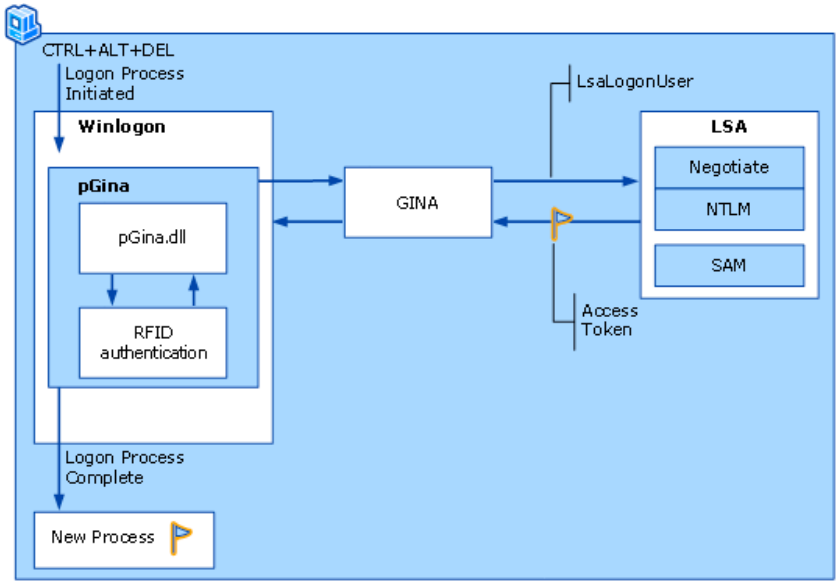

Figure 5.2: The Interactive local logon architecture with pGina

5.3.1 Configuration of pGina

The standard logon dialog box that comes with pGina, can be slightly personalized without touching a single line of source code with a custom bitmap and welcome text. A complete custom designed logon dialog box can be designed, but therefore we have to change the pGina source code and is not necessary in our situation. Interacting with the user during logon is done via (error)messages produced by our pGina plugins.

Because our authentication design is based on a two-factor authentication, pGina has to handle two authentication requests. The first one is the validation of the username and password combination against the local SAM and the second one is the validation of the ePassport against the local data store which holds the username, K_{ENC} and K_{MAC}. For a clean and robust implementation of this scenario, it would be ideal to have two separate pGina plugins that handle the two authentication requests. This can be accomplished by installing and configuring pGina's Chaining plugin. This plugin enables the use of multiple, chained, plugins. We have to configure the Chaining plugin so that each of our custom plugins must succeed during authentication to ensure we have a valid logon.

After we have installed and configured pGina and the Chaining plugin as described, we have to design the two plugins as discussed above. For detailed information about the installation and configuration of both pGina and the pGina Chaining plugin, please consult the manuals found at www.pgina.org.

5.3.2 Username and password validation plugin requirements

For the first authentication request, we have to validate the username and password combination against the local SAM. Therefore we have to capture the username and password the user provided in the pGina logon dialog box.

When the username and password combination does not exist in the local SAM, an error message is generated by the plugin and presented to the user. The authentication fails and the user is presented with the pGina logon dialog box to enter a valid and existing username and password combination.

When the username and password combination does exist in the local SAM, the authentication succeeds and the ePassport validation plugin is called.

The username and password are the only two requirements for this plugin. Because the username and password are provided by the user, no configuration for this plugin is required by the system administrator.

5.3.3 ePassport validation plugin requirements

For the second authentication request, we have to validate the ePassport and username combination against the local data store which holds the username, K_{ENC} and K_{MAC}. We assume that the system administrator has already set up such a data store. We assume that the data store is a simple local protected file, but can be for example replaced with a MySQL database. We also assume that the required certificates, as described in chapter 3, are already installed by the system administrator. For a successful authentication of this plugin, we have to capture the username the user provided in the pGina logon dialog box. This username is looked up in the local data store.

When the provided username does not exist in the local data store, an error message is generated by the plugin and presented to the user. The authentication fails and the user is presented with the pGina logon dialog box and has to go through the username and password validation plugin again.

When the username does exist, the K_{ENC} and K_{MAC} stored with the username are retrieved and used to go through the BAC, PA and AA protocols. When any of the BAC, PA or AA protocols fails, an error message is generated by the plugin and presented to the user. The authentication fails and the user is presented with the pGina logon dialog box and has to go through the username and password validation plugin again.

When the username exists and the BAC, PA and AA protocols succeed with the retrieved K_{ENC} and K_{MAC}, we know that the provided ePassport is valid, authentic and belongs to the provided username. The authentication succeeds and the user is logged on to the computer system.

The ePassport validation plugin requires a location to the file with the user credentials, a configuration string to tell the plugin which smart card reader is used for authentication and the path to the location where the certificates used for authentication are installed.

5.3.4 the pGina plugin framework

Now that we have the requirements for both the username and password validation plugin and the ePassport validation plugin, we have to examine the pGina plugin architecture to see how we can implement the two plugins.

The pGina plugin architecture is very straightforward. The plugins used by pGina must be compiled as a Dynamic Link Library (*DLL*). Every plugin needs a header file. This header file will export the plugin's functions defined in the *C++* source file and creates the pGinaInfo structure which hold all the information that will be provided by pGina. This structure also acts as the container for the information that we will pass back to pGina. The only member of the structure a plugin should change is the errorString. The errorString is, not very surprising, an errorstring that is displayed if logon fails. The framework for the plugin's header file is shown in listing 5.1.

Listing 5.1: The plugin's header file framework

```
/*
pGina header file framework
Copyright (C) 2007 Jean-Pierre de Rooij
Radboud University Nijmegen
Educational Institute for Computing and Information Sciences
Email: jrooij@science.ru.nl
*/

// Define macros for exporting methods
#ifdef  MYPLUGIN_EXPORTS
#define MYPLUGIN_API extern "C" __declspec(dllexport)
#else
#define MYPLUGIN_API extern "C" __declspec(dllimport)
#endif

typedef struct pGinaInfo {
                LPTSTR pathMSGina;
                LPTSTR pathPlugin;
                LPTSTR pathProfile;
                LPTSTR mapPaths;
                LPTSTR Username;
                LPTSTR Password;
                LPTSTR homeDir;
                BOOL isAdmin;
                BOOL disabled;
                int authType;
                HANDLE hUser;
                LPTSTR userGroups;
                LPTSTR userDescription;
                LPTSTR userFullName;
                BOOL allowPassChange;
                LPTSTR errorString;
                LPTSTR defaultDomain;
                BOOL Reserved3;
                BOOL Reserved4;
} pGinaInfo ;

// Export these methods... - REQUIRED
MYPLUGIN_API BOOL UserLogin(LPTSTR, LPTSTR,pGinaInfo *);
MYPLUGIN_API void ChangePluginSettings(void);
MYPLUGIN_API BOOL IsRequired(void);
```

Now that we have the header file framework, we create the *C++* source file framework which implements the exported functions. The functions we need to implement are **UserLogin()**, **ChangePluginSettings()** and **IsRequired()**. The framework for the *C++* source file is shown in listing 5.2.

UserLogin()

The UserLogin() method is called by the pGina.dll when a user requests to be authorized for login. It accepts both a username and password as parameters and returns a boolean value indicating the success or failure in authenticating the requested user.

```
BOOL UserLogin(
    LPTSTR Username,          // Username passed as a string
    LPTSTR Password,           // Password passed as a string
    pGinaInfo *settingsInfo  // Structure containing GID etc
);
```

ChangePluginSettings()

The ChangePluginSettings() method is called by a program when the system administrator wants to setup or change configurable options for the plugin. An interface for doing so should be provided with this method and solely within this method.

```
VOID ChangePluginSettings();
```

IsRequired()

The IsRequired() method returned value indicates whether the plugin is required. If the plugin is required, pGina will only check the local SAM if the login is a local administrator all other accounts(regardless of local availability) fail.

```
BOOL IsRequired(VOID);
```

Listing 5.2: The plugin's $C++$ source file framework

```
/*
pGina cpp file framework
Copyright (C) 2007 Jean-Pierre de Rooij
Radboud University Nijmegen
Educational Institute for Computing and Information Sciences
Email: jrooij@science.ru.nl
*/

#include "stdafx.h"
#include "pHeader.h"
#include <wchar.h>
#include <tchar.h>

/*
** This isn't strictly necessary, but can be handy if you are
** going to add Dialogs etc
*/
BOOL APIENTRY DllMain( HANDLE hModule,
                       DWORD  ul_reason_for_call,
                       LPVOID lpReserved)
{
    switch (ul_reason_for_call)
        {
                case DLL_PROCESS_ATTACH:
                case DLL_THREAD_ATTACH:
                case DLL_THREAD_DETACH:
                case DLL_PROCESS_DETACH:
                        break;
    }
    return TRUE;
}

/*
** This is the function that pGina calls to find out whether
** a user is authorized or not.
*/
MYPLUGIN_API BOOL UserLogin(LPTSTR Username,
                            LPTSTR Password,
                            pGinaInfo *settingsInfo)
{
}

/*
** This is the function that pGina calls when the plugin is
** being configured with the pGina configuration utility
*/
MYPLUGIN_API void ChangePluginSettings(void)
{
}

/*
** This function is called whether the plugin is required. If
** the plugin is required, pGina will only check the local SAM
** if the login is a local administrator. All other account
** (regardless of local availability) fail.
*/
MYPLUGIN_API BOOL IsRequired(void)
{
}
```

5.3.5 Username and password validation plugin framework

The rather simple username and password validation plugin requires no configuration. Therefore we only have to implement the **UserLogin()** and **IsRequired()** functions.

pGina has already build-in support for authenticating a username and password against a local SAM. There are no further requirements for this plugin. For the implementation of the local SAM authentication, see the pGina documentation.

Because we want the log on fails when we do not have a matching username and password, we set this plugin to be required. We have the following framework for this plugin:

```
SAMPLUGIN_API BOOL UserLogin(LPTSTR Username, LPTSTR Password,
                             pGinaInfo *settingsInfo)
{
    if (username AND password combination in SAM)
        return TRUE;
    else
        return FALSE;
}

SAMPLUGIN_API BOOL IsRequired(void)
{
    return TRUE;
}
```

5.3.6 ePassport validation plugin framework

The ePassport validation plugin can be configured and therefore we need to implement all the tree functions as described earlier. Because we want the log on fails when we do not have the matching ePassport for the provided username, we set this plugin to be required.

For a successful authentication when using this plugin, we need to retrieve the MRZ stored in the data store matching the provided username and use that MRZ to authenticate the ePassport. We have the following framework for this plugin:

```
EPASPLUGIN_API BOOL UserLogin(LPTSTR Username, LPTSTR Password,
                              pGinaInfo *settingsInfo)
{
    string MRZ = GetMRZ(username);

    if (BAC(MRZ) AND PA(MRZ) AND AA(MRZ))
        return TRUE;
    else
        return FALSE;
}
```

```
EPASPLUGIN_API BOOL IsRequired(void)
{
    return TRUE;
}

EPASPLUGIN_API VOID ChangePluginSettings()
{
    SetRfidType();
    SetDataStore();
}
```

Chapter 6

Conclusions

At the beginning of this thesis, four sub-questions were stated. Looking at the results of this research, we can give answers to all of these four sub-questions:

1. *Can we interact with the ePassport?*
   As we saw at the end of chapter 2, we could communicate with the electronic passport without any restrictions: we could access the data we wanted to use in our authentication method. Therefore we can indeed interact with the electronic passport.

2. *Can we design an authentication method that uses the electronic passport?*
   In chapter 3 we designed two tools: the user enrollment tool and the user logon tool. Both tools formed the authentication method which used the electronic passport.

3. *Is the designed method is secure enough?*
   In chapter 4 we saw that two of the six possible attacks could be effective to let an unauthorized user log on to the secured environment. However, we discussed the fact that these two attacks are not very likely to occur, seen the fact that the chance of an successful attack is very small. However, when the authentication method is used to grant access to a system which launches nuclear missiles, we have to take that small chance serious. But in this thesis we assume that the authentication method is only used to grant access to a personal or business computer system. Therefore we consider the designed authentication method as secure enough.

4. *Can we implement the authentication method?*
   Finally chapter 5 showed us that we can implement the authentication method on a Microsoft Windows 2000/XP operating system. Although we did not build a working prototype, the described framework should give enough guidance for doing so.

We see that all the four sub-questions are answered with *"yes"* and therefore we can give our final answer on our research question:

> *Is an electronic passport usable as an authentication token?*
> Yes.

Chapter 7

Future work

This thesis has shown that it is possible to use the electronic passport as an authentication token. Because of the limited time available for this thesis, we could only give the framework for an implementation. That framework can be used to implement a working prototype.

7.1 Building the prototype

The given framework in chapter 5 can be used to build a working prototype, with the use of the described flexible and straight forward pGina API. What we did not described in detail is how to interact with the ePassport via a piece of source code. Further research on how to actually interact with the ePassport has to be done.

7.2 Code enhancements

We only described the basic graphical user interface changes to the login dialog box. The whole prototype can look a lot fancier and can be more user friendly when extending the login dialog box with a custom made design. Therefore the pGina core has to be modified. In this thesis we did not described how to accomplish this, so further research on modifying the pGina core has to be done.

7.3 Grandmaster Chess Attack timing issues

Finally we saw in section 4.2.6 of chapter 4 that the Grandmaster Chess Attack is possible. We only described in short how this can be solved using synchronized clocks. Further research on how to solve the Grandmaster Chess Attack like studying Beth's and Desmedt's article [BD90], can give more detail in solving the problem.

Appendix A

The Logical Data Structure

DATA GROUP	MANDATORY (M) / OPTIONAL (O)	DATA ITEM	
Detail(s) Recorded in MRZ of the MRTD			
1	M	Machine Readable Zone (MRZ) Data *[See 13.1]*	
Machine Assisted Identity Confirmation Detail(s) – Encoded Identification Feature(s)			
2	M	GLOBAL INTERCHANGE FEATURE	Encoded Face *[See 13.2]*
3	O	Additional Feature	Encoded Finger(s) *[See 13.2]*
4	O	Additional Feature	Encoded Iris(s) *[See 13.2]*
Machine Assisted Identity Confirmation Detail(s) – Displayed Identification Feature(s)			
5	O	Displayed Portrait *[See 13.3]*	
6	O	Reserved for future use	
7	O	Displayed Signature or Usual Mark *[See 13.3]*	
Machine Assisted Security Feature Verification – Encoded Security Feature(s)			
8	O	Data Feature(s) *[See 13.4]*	
9	O	Structure Feature(s) *[See 13.4]*	
10	O	Substance Feature(s) *[See 13.4]*	
Additional Personal Detail(s)			
11	O	Additional Personal Data Elements *[See 13.5]*	
Additional Document Detail(s)			
12	O	Additional Document Data Elements *[See 13.6]*	
Optional Detail(s)			
13	O	Discretionary Data Element(s) defined by issuing State or organization *[See 13.7]*	
Reserved for Future Use			
14	O	Reserved for future use	
15	O	Active Authentication Public Key Info	
Person(s) to Notify			
16	O	Person(s) to Notify Data Element(s) *[See 13.9]*	

Figure A.1: The Logical Data Structure. Adapted from page 19 of [ICA04a]

Bibliography

[BD90] Thomas Beth and Yvo Desmedt. Identification Tokens - or: Solving the Chess Grandmaster Problem. *Lecture Notes In Computer Science*, 537, 1990.

[HHJ+06] Jaap-Henk Hoepman, Engelbert Hubbers, Bart Jacobs, Martijn Oostdijk, and Ronny Wichers Schreur. Crossing borders: Security and privacy issues of the european e-Passport. In *Proc. IWSEC 2006: Advances in Information and Computer Security*, number 4266 in LNCS, pages 152–167. Springer, 2006.

[ICA04a] ICAO. DEVELOPMENT OF A LOGICAL DATA STRUCTURE - LDS For OPTIONAL CAPACITY EXPANSION TECHNOLOGIES. Technical report, ICAO, 2004.

[ICA04b] ICAO. PKI for Machine Readable Travel Documents offering ICC Read-Only Access. Technical report, ICAO-NTWG, PKI Task Force, Tom A.F. Kinneging, 2004.

[JS05] B. Jacobs and R. Wichers Schreur. A security review of the biometric passport, 2005. Laquso symposium at Eindhoven. Earlier versions of this talk were delivered at Safe-NL day at Nijmegen, 25 june 2005, and computer security colloquium at Standford University 15 june 2005.

[KK05] Gaurav S. Kc and Paul A. Karger. Preventing Security and Privacy Attacks on Machine Readable Travel Documents (MRTDs). Technical report, IBM Research Division, 2005.

[Mic03] Microsoft. How Interactive Logon Works. Technical report, Microsoft Corporation, 2003.

[Sch94] Bruce Schneier. *Applied Cryptography*. John Wiley & Sons, 1994.

Index

Colophon

This thesis was produced using TeXShop LaTeX and MagicDraw UML community edition.

Author – Jean-Pierre de Rooij

Supervisor – Ronny Wichers Schreur

Personal thanks to:

- My supervisor Ronny who helped me during my research;
- My friend and colleague David who designed the figures used in chapter 4;
- My friend and studymate Jordy who took the time for reading and commenting my thesis;
- Sanne for her love and support during my research.