Practical Security Aspects of Digital Signature Systems

Florian Nentwich, Engin Kirda, and Christopher Kruegel
Secure Systems Lab, Technical University Vienna
{fnentwich,ek,chris}@seclab.tuwien.ac.at

Technical Report - June 2006

Abstract

A digital signature is an electronic token that creates a binding between an entity (e.g., a person or a company) and a data record. Typically, the signing process is implemented with the help of public key cryptography; the signatory uses her private key to create a digital signature for a document. In the last few years, there has been a rapidly growing demand for a working digital signature framework, both from the public and private sector. The goal is to facilitate both e-government and e-commerce applications over the Internet. In many countries, legislative considers digital signatures equivalent to handwritten signatures. However, international regulations such as EU Directive 1999/93/EC and national signature laws provide only rough guidelines that define necessary properties of secure signature-generation systems. Unfortunately, it is not clear that systems, even when obeying to the stated guidelines, are actually secure.

Most signature-creation systems are currently based on smart cards, which store all necessary cryptographic information of a user (e.g., keys and certificates). Thus, this paper looks at the practical security aspects and possible weaknesses of smart-card-based solutions. More precisely, we perform a security analysis of the path between the application that creates a document to be signed and the chip on the smart card where the signature is actually generated. After a general security analysis, we present a case study of three concrete attacks against applications that make use of the Austrian citizen card framework (which provides support for digital signatures). The first attack shows how an attacker can hijack an authenticated session that is established with an e-government web application. The second attack demonstrates how the content of a digitally signed mail can be manipulated. The third attack shows how a secure viewer application can be tricked into signing a different document than what is displayed to the user. Finally, we discuss possibilities to improve the practical security of current digital signature generation systems.

1 Introduction

In 1999, the European parliament and the council issued Directive 1999/93/EC, introducing the legal framework for the electronic signature in the European community. The goal of this directive was to support electronic commerce on the Internet and to facilitate the adoption of e-government in the member states. To this end, the concept of an advanced digital signature was conceived that should be treated legally equivalent to a handwritten signature. More precisely, Article 5 of the directive states that a digital signature "shall satisfy the legal requirements of a signature in relation to data in electronic form in the same manner as a handwritten signature satisfies those requirements in relation to paper-based data; and be admissible as evidence in legal proceedings." Using digital signatures, citizens of EU member states can carry out their public administration duties over the Internet. Of course, the digital signature infrastructure can also be used to carry out transactions (e.g., signing of contracts) between business partners without the need to meet in person.

To qualify as equivalent to a handwritten signature, a digital signature must be based on a qualified certificate [1] and must be created by a secure signature-creation device. The qualified certificate is necessary to ensure that the key used to sign the data is genuine and linked uniquely to the person carrying out the signature process. It is issued by a trusted certificate authority (CA) certified by a national or European body. While not mandated by the EU directive, secure signature-generation devices are typically realized as a combination of a smart card, a card reader, and a software component running on the user's computer. The smart card holds the user's public-private key pair and the qualified certificate. When signing a document (e.g., a contract or a tax report), the software component shows all relevant information on the user's display. The user can then initiate the signature process in which the data (or a hash of the

[1] A *certificate* is a binding to a person, digitally signed by a trustworthy authority and containing data and a public key.

data) is sent to the smart card. After entering a secret pass code (or PIN), the smart card first uses the private key that it stores to sign the received input value and then returns the signature output to the program for further processing.

One of the tasks of the secure signature-creation device (as specified in the appendix of the EU directive) is that it must not alter the data to be signed, or prevent such altered data from being presented to the user prior to the signature process. In other words, the secure signature-creation device *must guarantee* that the data that is actually signed is the data that the user wishes to sign (after all, it is a legally binding signature that can be used as evidence in court). While this requirement is reasonable and easy to decree on paper, it is much more difficult to enforce in practice.

Although the EU directive was presented more than seven years ago, the adoption of digital signatures was a relatively slow process. Recently, however, this process gained momentum as several EU member countries created national certification infrastructures. In addition, some countries now provide smart cards with digital signature capabilities for free and subsidize card reader equipment. Together with the fact that todays ATM cards also have the necessary chip to support digital signatures, the number of commercial and government applications that make use of digital signatures is rapidly increasing.

The rapid growth of digital signature systems, their broad availability for the public, and their legal implications naturally raise questions about their security. Interestingly, there is a general belief among many business organizations that digital-signature-based systems are the perfect security solution. In this paper, we look at the practical security aspects of digital signatures and discuss possible attacks against the secure signature-generation process. In particular, we analyze the difficulty for an attacker to bypass the security mechanisms of such systems, for example, by installing a Trojan horse on a user's computer. The main contributions of the paper are as follows:

- We analyze the process of digitally signing data (or documents) by following the complete path, from the application requesting the signature to the user providing her signature. More precisely, we assess the vulnerabilities of digital signature solutions that use smart cards. This allows us to reason about possible attack vectors that target different parts of the process.

- We describe the findings of our security assessment of the Austrian citizen card solution and explain how we launched three successful attacks that subverted the security of various applications making use of digital signature functionality. This underlines that theoretical weaknesses can be turned into practical attacks (e.g., by installing Trojan programs on users' computers).

- We discuss a number of possible countermeasures that can thwart our attacks and improve the security of digital signature systems in general.

The paper is structured as follows. In Section 2, we discuss attack vectors against various parts of the trusted path between the application that issues data to be signed and the user that is signing this data. In Section 3, we describe our case study concerning the security of the Austrian citizen card and present practical attacks that subvert its security. Then, in Section 4, we present mechanisms that could counter our attacks and increase the security of digital signature generation. Finally, we discuss related work in Section 5, and we briefly conclude in Section 6.

2 Security Analysis

For our security analysis, we only consider secure signature-generation systems that use smart cards. Other solutions for digital signatures exist, however, they are out of the scope of this paper. For example, the cryptographic information can be stored directly on the user's computer. These systems can be easily compromised by an attacker that obtains administrative access to the machine and hence, can steal the private key. As a result, the security of these systems is not considered adequate for generating legally-binding signatures: a national certification authority would not sign (or certify) such weakly-protected key pairs, effectively preventing them from participating in the digital signature process. Another approach to allow a user to generate a digital signature is based on mobile phones. Here, the phone acts as a replacement for the smart card. When the user presents her name and password to a server run by the mobile phone provider, she receives a message with a one-time code on her phone that can then be used to digitally sign data online. While mobile-phone-based solutions are available on the market, their share is negligible compared to smart-card-based systems.

In a secure signature-generation system based on smart cards, the card holds the cryptographic information (user identification, public-private key pair, certificate) for a particular user. To access the smart card, the user's computer has to be equipped with (or attached to) a smart card reader, a device that implements the ISO 7816 smart card standard to communicate with the card. The cryptographic functions of the smart card are made available to applications through a card reader-specific device driver that is installed in the host's operating system. There are a number of interfaces that a smart card driver can implement to offer the card's functionality to applications. The most common interface is the cryptographic token interface standard, PKCS #11, which is specified by RSA Security. This standard defines a general application programming interface (API) to cryp-

tographic tokens such as hardware cryptographic accelerators and smart cards. Unfortunately, Microsoft decided to not support this standard and defined an alternative, general API for cryptographic functions. This API, called CryptoAPI or CAPI, contains a cryptographic service provider (CSP) layer that allows third party software and smart card solutions to extend it with the necessary functionality. In addition, there are applications that neither use PKCS #11 nor CSP/CAPI, adding undesirable complexity to the deployment of smart card solutions. Note that the device driver (and even the card reader) cannot immediately trigger the generation of a digital signature. Instead, it is necessary that the user provides a secret code (or PIN) to unlock this functionality on the card. This code can be entered either through the card reader, when it is equipped with a keypad, or directly on the user's computer.

When an application (either local or remote) requires the user to digitally sign data, this data has to go through a number of steps before it is eventually processed by the smart card. These steps, shown in Figure 1, are parts of what is often referred to as a *trusted*, or *secure* path. In the figure, the citizen card environment (CCE) represents a special set of local applications and libraries that provide support for commonly used smart card and digital signature use cases. For example, the CCE typically contains a "secure viewer", which can be used to securely display and sign documents, and a component that provides support for user authentication to remote services.

The connection between the application that requires the signature and the smart card is called trusted path, because data is typically not allowed to be modified in transit. For example, a person who would like to sign a document has to be sure that the document she is signing is indeed the document that she is viewing on her computer screen. In this case, the data must not be modified on the secure path between the document viewer application and the chip on the smart card (where the signature is eventually created). In the following subsections, we discuss weaknesses and possible attacks against the various components of this trusted path.

2.1 Card and Card Reader

We begin our analysis of the secure path with the leftmost part in Figure 1, the chip on the smart card and the card reader. Although the chip and its operating system are based on a closed specification, it may not be impossible to analyze the chip's layout or functionality. There are several techniques, both invasive and non-invasive, that can be used to gather information about the implementation. The most common and effective ones are different forms of side channel analysis, where not the actual cryptographic algorithms are attacked, but the side effects of this operation are analyzed. Typically, the goal is to extract information about the secret key stored inside the card. When the secret key is extracted, obviously, arbitrary documents can be signed in the name of the victim.

The side channel attack based on time measurement [1], for example, makes use of the fact that one can record different execution times for different values of a key. With the aid of statistical analysis, it is possible to extract the hidden private key. Power analysis [2], which depends on different power consumption for different operations, is another approach that can lead to more information about the chip and stored secrets. Of course, research is not only looking at possible attacks but also at feasible countermeasures, such as equal execution times for different branches when looking at time-based analysis. For a more comprehensive overview of side channel analysis and countermeasures, the reader is referred to [3].

In addition to the chip's hardware structure, one has to consider the implemented public key algorithms that are used to encrypt and decrypt content, as well as creating and verifying signatures. The most common used ones are DSA, RSA, and those based on elliptic curves. All these algorithms share the property that their security depends on the length of the key that is used for signing or encryption. Although keys might be strong enough today (e.g., RSA with a length of at least 1024 bit), they might be broken in the future (e.g., because of the availability of faster hardware or novel cryptographic attacks). In this case, some problematic legal issues may arise with "legacy" signatures, even when weak keys are replaced with stronger ones. For example, an attacker could take an old and invalid key and sign a document predated to the time when this key was still valid. As a result, even though a strong key would be active at present, it might be difficult to argue that the document was not signed in the past.

Note that not only the chip can be analyzed. Surely, the card reader might also be tampered with. This could allow to sniff the secret PIN code when it is entered on the reader's keypad and to arbitrarily record and tamper data exchanged between the card and the user's computer. Although the card reader usually has a seal to prove that it was not opened and manipulated, there is still a problem with a user who does not know how the seal should look like. Hence, it could be possible to replace the seal after the manipulation without anyone noticing the difference.

The aforementioned attacks have the potential to reveal secret information, such as the user's private key and PIN code. However, they require physical access to the card or the card reader device. Therefore, a wide-scale attack that targets many users is not really practical and difficult to carry out from the attacker's point of view.

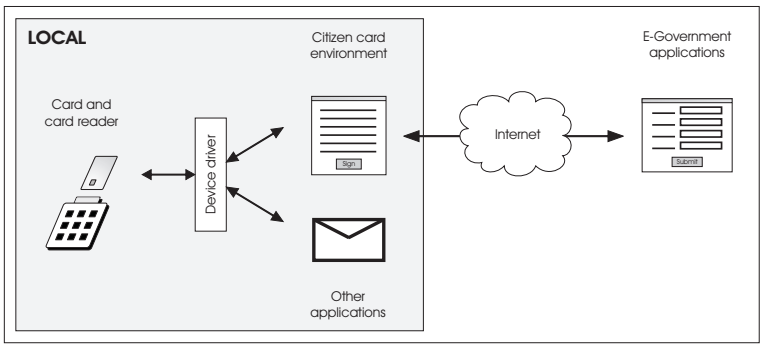

Figure 1. Secure path of citizen card applications.

2.2 Operating System and Device Drivers

The operating system that hosts the device driver of the card reader, as well as the citizen card environment, is an important part of the overall security. When an attacker is capable of gaining control of the operating system, the code and memory of all local processes can be arbitrarily altered. In this case, a significant portion of the trusted path has to execute on top of an untrusted platform. This is a problem, as it allows the attacker to alter the data to be signed anywhere between local applications and the card reader's device driver. The following paragraphs describe in more detail the straightforward mechanisms to manipulate a victim process and the code that it executes under the Microsoft Windows operating system. In this description, we focus particularly on Windows because it is the prevalent platform on the Internet and is the most likely target of attackers or malicious software that aim to exploit the digital signature process. Similar mechanisms also exist for other platforms such as Unix, Linux, or Mac OS.

One possibility to interfere with the secure path is to replace the device driver for the smart card reader with a malicious version. In this fashion, the attacker can observe the complete traffic (data and invocation of cryptographic operations) that are exchanged between applications on the host and the smart card. Alternatively, one could modify the code of certain local applications of interest, either by changing their static program image on disk or by altering the process image while the program is running. Extending the code of a running process can be easily achieved with the help of `detours` [4], a software package provided by Microsoft that allows to intercept and replace program functions. To this end, detours creates a DLL that is loaded together with the target process. This DLL has access to the process' address space, a property that can be used to inject malicious code into a program. More precisely, when the DLL initialization routine is called, it can determine the starting address of a function to be manipulated (by calling *GetProcAddress*). In a second step, the first bytes of the old function are overwritten by an unconditional jump to the code to be executed instead. To be able to still use the original function, a *trampoline function* can be created, in which the bytes overwritten by the jump command are put at the beginning, followed by an unconditional jump to the first retained instruction of the original function. In this fashion, the attacker can add functionality, or replace the complete function. Changing the running process image directly, instead of the static program image on disk, could be preferable for an attacker when an application contains self-protection routines that check the integrity of its code before loading and executing it. When intercepting functions in a program that make use of digital signatures, it is easily possible to replace content to be signed before it is sent to the smart card (but after it has been checked by the user for correctness).

In addition to being able to change the code that processes execute, the attacker also has complete control over the program's input and output. Windows applications usually have a main window that can receive so-called *window messages* (e.g., commands such as a mouse click or repainting of the window) from other application windows or processes. This circumstance could be used by a malicious process to modify and even craft input (e.g., closing a window or emulating a mouse click on a specific button). Moreover, when given the window title or class, the window handle can easily be retrieved with the `FindWindowEx` function provided by the Windows API. With this window handle, the window itself can be manipulated. That is, it is possible to close child windows, create new elements, or even paint over the window content. To be able to achieve this, one has to first retrieve a handle to the device context (DC) with `GetDC`. Retrieving this second handle is not difficult and can easily be obtained from the window handle. Also, when the PIN code (i.e., the secret code to trigger the generation of the signature on the smart card) is not entered directly on the card reader's keypad, but on the computer keyboard, a compromised operating system can capture this input by logging the key strokes. With the PIN code, arbi-

trary data can be signed without requiring any user cooperation as long as the smart card is accessible via the card reader.

Interestingly, vendors of smart card solutions are aware of possible attacks against applications and have created the concept of the so-called "secure viewer". The secure viewer is a crucial part of the citizen card environment whose main task is to securely display a document that shall be signed. Of course, in the process of certifying a secure viewer, the operating system is always assumed to be unmodified and trusted. It is evident that a malicious operating system could send undesired data to the smart card for signing, while the user reviews a benign document using the secure viewer.

Being able to subvert the operating system requires the attacker to hold administrator (i.e., root) privileges. Unfortunately, such a privilege is often not too difficult to obtain. For example, a large fraction of Windows users still run their computers with administrator rights because many programs exist that cannot be started under a non-administrator account. Furthermore, a comprehensive analysis performed by Webroot (an anti-spyware software producer) and Earthlink (a major Internet Service Provider) showed that a large portion of computers on the Internet is infected with spyware, and that, on average, each scanned host has 25 different spyware programs installed [5]. Thus, for our practical attacks, we assume that the attacker has already obtained administrator rights.

2.3 Applications

The last component of the secure path is the collection of applications that make use of the signature-creation functionality. That is, for a complete analysis, the origin of the data to be signed has to be taken into account as well. Depending on the type of service (e.g., web application, local browser, mail client), different security aspects have to be considered. Therefore, we distinguish between remote and local applications in the following discussion.

2.3.1 Remote Applications

When the data to be signed is produced by a remote application, the execution of this program is not affected by a compromise of the user's computer. However, many web applications suffer from input validation errors that lead to buffer overflow or cross-site scripting vulnerabilities. These vulnerabilities can be exploited by an attacker to seize control over one endpoint of the secure communication channel. Hence, the adversary can inject malicious content into the data stream that is sent to the client for signing. From the perspective of the attacker, the advantage of taking control of a remote server is that she can typically attack a large number of clients. On the other hand, the client could check the data before generating the signature, thus possibly detecting malicious requests.

Another interesting aspect of digital signatures is the necessity of a certificate or key infrastructure that is publicly available (e.g., through LDAP directories). This is because the recipient of a signed document needs to easily obtain the public key of the person who provided the signature. Of particular interest are the email entries of the public certificates. Apart from being a large source of high-quality email addresses that can be used as recipient addresses for unsolicited mails, spammers can also use the public keys to send encrypted spam messages. Since encrypted mails cannot be filtered automatically by a spam filter, the user has to first decrypt each mail by using her smart card and providing the secret PIN code. Encrypted spam mails are particularly problematic, since not all smart-card-based systems allow bulk decryption or encryption. In this case, each email would have to be decrypted *individually*, thus creating a denial of service situation for the user.

2.3.2 Local Applications

As previously discussed in Section 2.2, there are a number of ways an attacker can manipulate memory and code of operating system processes. In particular, we considered how device drivers or parts of the citizen card environment (CCE) can be manipulated by an attacker that has obtained administrator privileges. Of course, similar attacks can be carried out against local processes, even when the attacker only has the privileges of the user running these applications. For example, when a user wishes to sign an email (e.g., with a client such as Mozilla Thunderbird), the mail content being signed can be tampered with by intercepting and changing the appropriate functions of the mail client. Of course, when the signature is being replaced, the mail body being sent has to be modified as well so that the signed text and the signature match. Using this technique, the attacker can send a valid, signed email from the victim with a content of his choice even without obtaining administrative access to the machine.

2.4 Communication Channels

Between the different components of the secure path, there are communication channels that can be attacked as well. An exposed channel is between the local machine with the smart card and remote applications. While these channels are typically encrypted and cannot be easily eavesdropped on, man-in-the-middle attacks are possible. There are several ways of launching such an attack:

- A machine on the route (e.g., the gateway) can directly intercept the packets.

- With DNS spoofing (responding faster to a DNS query than the DNS server), the client resolves the requested domain to a wrong IP, therefore sending data to the attacker's server.

- Before sending a DNS request, Windows checks the file `C:/WINDOWS/system32/drivers/etc/hosts`. If there is an IP entry for the target domain, Windows will immediately use it instead of sending a DNS query. Unix and Linux machines use a similar mechanism based on the `/etc/hosts` file.

When the attacker can act as a man-in-the-middle, it is possible to directly answer client requests, to deny complete access, or to act as a relay, thus forwarding and sniffing the traffic in both directions. If a digital signature, for example, is used to authenticate a user to a remote application, the relaying server can let the packets through until the login process is complete. Then, it can interrupt the communication to the victim, thereby hijacking the user's session.

Obviously, man-in-the-middle and session hijacking attacks of communication with a remote server can also be performed locally. Consider the typical case where the smart card (and a digital signature) is used to log into a remote service that runs over HTTP. Due to the fact that HTTP is a stateless protocol, a mechanism is needed to handle multiple requests from an authenticated client. This is usually done with a *session cookie*, a random (i.e., difficult to guess) value that is stored in the browser. This value is sent with each request to the given site so that the web application can associate the request with a session. A malicious program or browser plug-in can steal this session cookie, allowing the attacker (often from another remote machine) to send requests on behalf of the authenticated user.

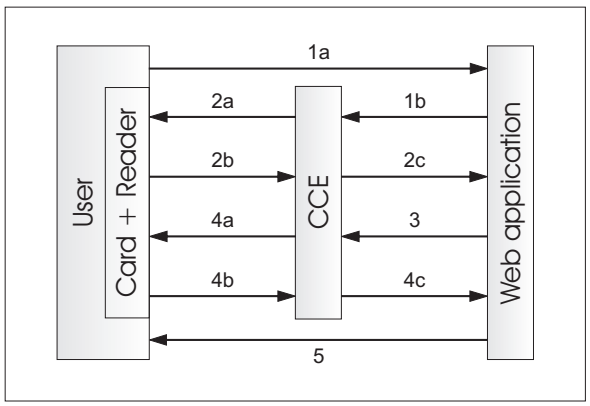

Figure 2. A typical citizen card application scenario.

Finally, local applications do not always communicate directly with the operating system and the device driver, but request services from the local citizen card environment (CCE). For example, a local application could invoke CCE services to retrieve a user's identity or sign documents using the secure viewer. An example of a more complex interaction between a remote service (web application), the CCE, and a digital signature is shown in Figure 2.

1. After some browser interaction by the user (a), the web application has to assert the user's identity and therefore sends the appropriate request to the citizen card environment (CCE) (b).

2. Then, the citizen card environment contacts the smart card (a) to retrieve the user identity (b) and sends the response back to the web application (c).

3. After that, the web application sends a document to be signed by the user. This is done so that the user can prove her identity.

4. Having signed the document (a+b), the environment sends the response (including the data to be signed) with the signature back to the application (c).

5. The application verifies the signature, and, if it is correct, takes the appropriate action (e.g., a successful login or the forwarding of the document to the appropriate authorities). Finally, the user is notified.

The communication between the CCE and applications can be realized in different ways. In our case study, the citizen card environment listens on TCP port 3495 for XML requests that conform to the so-called *security layer* specification. These requests can be submitted as plain TCP streams or embedded into a HTTP request (Step 1.b). Subsequent communication between the web application and the citizen card environment is then handled over an SSL connection (Step 2.c, Step 3, Step 4.c). Again, attackers that obtain access to the local machine can tamper with these connections and launch a man-in-the-middle attack to record and/or tamper with the traffic.

3 Case Study: Austrian Citizen Card

According to the official Austrian citizen card[2] home page [6], the Austrian citizen card concept serves two main functions:

1. *Electronic signature*: token that proves the authenticity and integrity of signed data.

[2]German: "Bürgerkarte"

2. *Identification*: data that connects the citizen card to a unique identifier (i.e., number in the Austrian Central Register of Residents)

When applying for a citizen card, a registration office of a certification service provider verifies the citizen's identity. During this registration process, the customer's smart card (e.g., the Austrian health insurance card or the ATM/bank card) is used to store the following information on its chip:

- A *simple certificate* for tasks that do not have to conform to the Austrian Signature Act, such as signing emails.
- A *qualified certificate*, which is a certificate conforming to the Austrian Signature Act §5, to produce *qualified electronic signatures* (also referred to as *secure digital signatures*).
- The corresponding *private keys*.
- Several *"infoboxes"* (virtual store-rooms) to save the unique identifier (*identity link*) and other personal data (e.g., for mandates, which allow signing on behalf of another person).

The chip has the ability to sign and encrypt/decrypt data with the private keys of the certificates. These functions are protected by two separate PIN codes, one for each certificate. A third PIN code can also be set to secure the infoboxes.

As mentioned previously, smart card solutions typically make use of a card reader with the corresponding device driver and a set of helper applications and libraries, commonly referred to as the citizen card environment. Currently, there are two different citizen card environments available in Austria. Both can be downloaded for free:

- `trustDesk basic` by IT Solution, which is available at `http://www.cio.gv.at/identity/bku/`. trustDesk is a native Windows application, linked to the Microsoft Foundation Classes (MFC). It uses an embedded HTML control to implement secure viewer functionality.
- `HotSign` by BDC, which is available at `http://www.bdc.at/30.html`. HotSign is a mixed Windows API/Java application that uses an executable file and the Java Native Interface (JNI) to create a Java virtual machine for the main program. HotSign is being shipped together with the Java Runtime Environment (JRE) 1.4.0. This is done so that a user does not have to download additional packages and to provide a stable and tested environment for the program.

In the following sections, we present our analysis of the security of three applications that make use of the digital signature capabilities of the Austrian citizen card. In particular, we analyzed the security of a user interacting with the internal revenue service online, a user signing an email, and a user signing a document with the help of the secure viewer. To carry out our analysis, the following environment was used:

- *Operating system:* Windows XP Professional SP2 (the currently available citizen card environments only support Windows 2000/XP, but versions for other operating systems are planned).
- *Citizen card:* Bank card.
- *Card reader:* REINER SCT cyberJack pinpad USB.
- *trustDesk basic*, version 2.4.3.
- *HotSign*, version 1.3.1.

3.1 Session Hijacking

The aim of the first attack was to hijack a session that an authenticated user had established with a remote service. For this attack, the target was chosen to be the e-government portal of the Austrian ministry of finance (FinanzOnline, at `https://finanzonline.bmf.gv.at/`). The web application that handles access to the financial information of a citizen requires the user to digitally sign a document to prove her identity. This process is very similar to what was discussed in Section 2.4 and shown in Figure 2. When the login is successful (i.e., the digital signature is successfully verified), the server issues a session cookie. This cookie is used in subsequent requests by the browser to tag these requests as being sent by the previously authenticated user.

The first attack uses Internet Explorer to steal the session cookie after a successful login procedure, thus allowing an attacker to duplicate the session. To steal the session cookie, the attacker requires local access to the victim's machine with the privileges of the user running the browser. The easiest way to perform the attack is a malicious browser plug-in, implemented in our case as a *browser helper object* for Internet Explorer [7].

With browser helper objects, one can write components (specifically, in-process Component Object Model (COM) components) that Internet Explorer will load each time it starts up. Such objects run in the same memory context as the browser and can perform any action on the available windows and modules. Through a special interface, the browser helper object can access the functions of Internet Explorer, thus being able to read and manipulate data. This way, the program can easily fetch the session cookie,

even though the connection with the remote service is encrypted. To recognize a session with a remote service that is of interest for the attacker, one can filter client requests based on known URLs (e.g., `http://finanzonline.bmf.gv.at/fon/login?skey=` for FinanzOnline).

Interestingly, the stolen session cookie is not only valid for requests sent from the compromised machine itself, but also for requests from any host. That is, the cookie is not tied in any way to the IP address or the machine that initiated the login. We verified this fact by taking a session cookie of an already established connection and continued the session on another machine. To use a stolen cookie on another machine, requests can be specially crafted to contain the required cookie. Alternatively, a cookie manager plug-in on the attacker's browser can be used to load the session cookie information.

3.2 Mail Forgery

When digitally signing an email, it is important that the actual mail text as composed by the user is signed. The goal of this second attack is to show that an attacker can replace the content of an email with arbitrary data, retaining the validity of the signature. The implementation of the attack is split into the following two parts:

- The content of the mail that is sent by the mail client has to be replaced with some content chosen by the attacker. To this end, the communication between the mail client and the mail server is intercepted.
- The signature has to be changed so that it matches the content chosen by the attacker. To this end, `detours` is used to load a dynamic link library (DLL) with the email client (in our example, Mozilla Thunderbird). The goal is to intercept the function that initiates the digital signature process, replacing the content being signed with the malicious document.

When sending an email, the mail client needs to open a TCP connection to the outgoing mail server. The communication is handled by the Simple Mail Transfer Protocol (SMTP), which sends commands and data in plain text. Because of this, a proxy server that intercepts (i.e., buffers and then forwards) the traffic exchanged between the client and the server can selectively modify the content of mails. To perform the man-in-the-middle attack, the adversary first launches an SMTP proxy on the compromised machine. He then alters the preference settings of Thunderbird such that the connection to the mail server is redirected to this proxy. The preferences of the user are stored in the user's `prefs.js` file, and the setting for the outgoing mail server is saved as value for the key `mail.smtpserver.smtp1.hostname`. The proxy monitors mail traffic for the occurrence of messages that contain the string "`Content-Type: multipart/signed`." This indicates that a signed mail is being transmitted. In this case, the signed content is replaced with the data that the attacker wishes to be signed. Of course, unsigned mails are passed through unaltered.

To manipulate the signature appropriately (so that it corresponds to the text chosen by the adversary), the signature generation functions of the mail client have to be changed. To this end, our attack uses `detours` to change the behavior of two hashing functions `Hash_Begin` and `Hash_Update` that are used by Thunderbird in the signing process. More precisely, when Thunderbird is instructed to generate a signature for an email, it first calls `Hash_Begin` to allocate a data structure that is filled by subsequent calls to `Hash_Update` with the content of the mail body. In our attack, these functions are changed so that the call to `Hash_Begin` immediately fills the allocated structure with our chosen text. Then, all invocations of `Hash_Update` return without effect. When Thunderbird has finished assembling the hash data structure, it performs the actual hash calculation, packs the results into an object that follows the cryptographic message syntax (CMS), and sends it to the smart card to be signed. Eventually, the signed CMS object is included into the mail, which is then sent to the outgoing SMTP server.

When put together, the combination of the SMTP proxy and the modifications of the signature generation process result in a forged mail with a valid signature. The mail text is manipulated, as is the signature that corresponds to the changed text. Independently of the signature, the SMTP proxy can also freely change all headers of a signed mail, as the signature only covers the mail body.

3.3 Secure Viewer Manipulation

The third attack targets the secure viewer component of the citizen card environment. The goal is to trick the secure viewer into displaying a document that is different from what is actually being signed. Following EU Directive 1999/93/EC, the Austrian Signature Act §4 [8] declares that secure electronic signatures are legally equated with personal signatures (except for certain documents, e.g., a notarial act). Due to the fact that the secure viewer of a citizen card environment (Section 2.2) is used to create such secure electronic signatures, it is a security-sensitive element of the digital signature system.

For the following attack, we decided to target `trustview`, the secure viewer component that is bundled with `trustDesk basic`, one of the two free citizen card environments available in Austria. The attack is carried out in two steps. First, we replace the data that will be signed by `trustview`. In the second step, we then manipulate the

viewer to show the original file content to the user. This is necessary to make the user believe that a legitimate digital signature is generated.

To replace the original data to be signed with malicious content, we performed a detailed analysis of the signature generation process of the secure viewer application. During this analysis, we observed that whenever `trustDesk` received a request to create a signature, it created a temporary file with a copy of the request. Then, it opened an instance of `trustview`, which read the file, asked the user for the PIN code and, after having created the signature, put back the original request together with the signature into the temporary file. The idea of the attack now is to switch to a different file containing the attacker's data when `trustview` attempts to open and read data from the temporary location. To do so, we use `detours` to modify the Windows file access routines (in the Windows runtime library) so that these functions operate on the file of the attacker's choice instead of the secure viewer's temporary file.

The second part of the attack is to hide the fact that different data is loaded by the secure viewer than intended by the user. In particular, it is necessary to show the original document in the viewer window. To this end, the functions to display the content need to be altered. Otherwise, the user could notice the attacker's input file and cancel the signature generation. `trustview` uses a Windows COM component that implements an HTML control to render the content of the document to be signed. To manipulate the shown document, the attacker has to first obtain a window handle to the HTML control. Using this window handle, a reference to the `IHTMLDocument2` object encapsulated in the HTML control component can be requested. This object provides convenient functions that allow the attacker to edit or clear the HTML body of the control, effectively changing the information displayed by the secure viewer.

An important question is about the point in time when the content of the HTML control should be replaced. We address this issue by using `detours` to redirect the program flow of the Windows API function `DrawText`. This function is invoked every time some text has to be drawn on the window surface. Thus, when setting the content of the HTML control object with the benign document whenever `DrawText` is called, we can ensure that the secure viewer is refreshed with the benign document data regularly. In particular, a refresh is important when parts of the secure viewer window have to be redrawn (e.g., because they become visible on screen). In this way, the secure viewer will not accidentally display (parts of) the malicious document that is sent to the smart card to be signed.

The attack described in this section focuses on `trustview`. However, it would also be possible to manipulate the data that is displayed by the secure viewer component of `HotSign`, the second free citizen card environment. To this end, the attacker could first retrieve the handle to the secure viewer window and subsequently draw directly over the window context. This allows to change the appearance of the shown document even when no HTML control object is used. Hence, the possibilities of bypassing the security mechanisms of these systems are currently only limited by the creativity and motivation of an attacker.

4 Countermeasures

The previous sections show that it is important to create a secure path from the chip on the smart card to the application that makes use of it. Otherwise, the digital signature cannot be trusted. In this section, we concentrate on countermeasures against the gravest security problems that we discussed earlier. Usually, a trade-off between cost, user comfort, and security has to be made when designing computer systems. Nevertheless, although costs should be kept low and the system should be as user-friendly as possible, digital signature systems must ensure a high level of security.

4.1 Increased Security Awareness

Unfortunately, email-based attacks such as phishing or Trojan horse attachments are very common. The main reason is that many users have full control of their system, but often are not concerned about security risks and implications. Since smart cards on their own do not necessarily guarantee a secure system (e.g., an inexperienced user can unconsciously weaken the operating system or application security), it is important to educate computer users about hardening their systems. If the user does not execute untrusted software, keeps the system up-to-date, and uses a properly-configured personal firewall and antivirus software, the system is much less likely to be compromised. Clearly, another security improvement would be the strict separation of the use of administrator and user accounts on Windows platforms, similar to what is best practice on most Linux and Unix machines today. If a signature application is started with system privileges, a process with user privileges cannot tamper with its settings anymore.

4.2 Viewer on Card Reader

Card readers with integrated keypads are getting more and more common because they ensure that the PIN code never leaves the reader and the card. This is in contrast to older systems that required the PIN to be entered on the keyboard of the computer. As a result, key stroke loggers can no longer access the user's secret codes.

Since hardware that is part of the card reader cannot be tampered with as easily as the software on the host, the next

step would be to integrate the secure viewer into the reader. Surely, this would raise the production cost, but at the same time, it would also considerably enhance security. This is because the user could verify the data to be signed on the reader before triggering the signature creation process. Of course, malicious software can still alter the content to be signed, but during the verification process, the user has a chance to detect this change. Interestingly, there are already card reader solutions that use a limited form of display (e.g., for only one line of text), but usually it is not possible to verify more complex documents.

4.3 Secondary Channel

Another approach to address the problem of tampering with the trusted path between the smart card and applications is the introduction of a second and completely independent path. This secondary path is preferably such that an attacker cannot easily obtain access to it. Thus, it should not pass through the user's computer. One possibility, for instance, is to send a short message containing the data to be signed to the user's mobile phone so that the signature process can be confirmed and finished by replying to this message. One major drawback is the growing complexity for the user; there is not only the signature, but also an additional confirmation process.

4.4 Trusted Computing

The main goal of the trusted computing initiative (as launched by the Trusted Computing Group - TCG) is the creation of a secure execution environment for personal computers. To this end, hardware support in the form of trusted platform modules (TPM) is integrated with the computers' motherboards. Based on a TPM, a secure boot chain from the hardware over the operating system to the applications can be created. For example, a trusted boot loader allows only a trusted operating system to be loaded, or the integrity of the operating system is verified at boot-time. Then, this operating system makes sure that only certified applications are started. This ensures that only unmodified applications (as certified by the software vendor) are executed, making it much more difficult for an attacker to alter programs such as the secure viewer. Also, there were a number of recent proposals [9, 10] that claim to provide a trusted execution environment without any additional hardware. Some of these techniques, however, have already been broken [11].

5 Related Work

The security of smart cards and their cryptographic algorithms have been studied extensively in the past years (as discussed in Section 2.1). However, practical security aspects of smart-card-based solutions (and in particular, systems using digital signatures) have received less attention.

In [12], a number of citizen card environments (CCE) were surveyed. In this paper, the author looked at the ease of use and ease of installation of a number of CCE products, as well as some of their security aspects. In particular, the protection mechanisms that these systems employ against malicious modifications were analyzed. The author concluded that most systems are vulnerable to alterations by malicious code (a result that is confirmed by our own analysis), but no actual proof-of-concept attacks were described.

A previous study presented in [13] demonstrated the typical problem of a Trojan horse, capturing the secret PIN when it is entered directly on the computer and not via a card reader keypad. The authors also showed an attack against a secure viewer. While similar in spirit to our own work, we provide a more comprehensive discussion of possible attack vectors against the secure path. Furthermore, we show that after more than four years, some of the security problems outlined in the study are *still* valid. The same authors also proposed a solution to the problems they found by employing a trusted computing platform [14].

Problems that arise when the process of displaying a document is separated from signing it are discussed in [15]. Also, operating systems of mobile devices were identified to be unsuitable for digital signature generation [16]. Finally, Germany has launched an initiative to create a system that will allow the secure generation of signatures despite the presence of malware on the user's computer. Unfortunately, except some announcements [17], no further information is currently available.

6 Conclusions

A large number of people can profit from the availability of a digital signature framework that allows to carry out public administration duties and business over the Internet. However, the fact that a digital signature is legally equivalent to a handwritten signature also raises security concerns. What happens when an attacker tricks a victim into signing a contract with unfavorable terms, an email that contains threats to the government, or a bank transaction with an unintended recipient? In all these cases, it might be very difficult for the victim to deny responsibility, especially when vendors claim their secure signature-systems to be certified and dismiss possible attack vectors as being "improbable" in practice.

In this paper, we analyzed the practical security of current smart-card-based solutions for secure signature-creation. In particular, we looked at all components of the secure path between applications creating the documents and the smart card signing the data. Along this path,

we identified a number of attack vectors that an adversary can exploit to subvert the security of the system. We also showed three real-life, proof-of-concept examples of how to subvert the security of applications that rely on the digital signature functionality of the Austrian citizen card.

One could counter that our attacks require root access to the victim's machine. Unfortunately, as the spread of worms and viruses demonstrates, a majority of users are using their administrator accounts for daily business and show very little security awareness. Thus, we believe that our attack scenarios are a realistic threat that must be considered. In addition, an anti-virus company that we are collaborating with has recently confirmed the existence of Trojan horses that explicitly target online banking applications. Even when banks switch to use digital signatures to authenticate transactions, the weaknesses presented in this paper can be used to maliciously alter these transactions. And as mentioned previously, because all transactions are personally signed by the victim, it is difficult to convincingly argue that the transaction was not carried out as intended.

With this paper, we hope to raise awareness that smart card solutions are not the remedy to all security-related problems, as marketing departments often want to make users believe. A security solution can only be as strong as its weakest link. In the case of the smart-card-based solutions, the weakest link is the untrusted operating system environment that the applications are running on.

References

[1] Werner Schindler. A Timing Attack against RSA with the Chinese Remainder Theorem. In *International Workshop on Cryptographic Hardware and Embedded Systems (CHES)*, 2000.

[2] Thomas Messerges, Ezzat Dabbish, and Robert Sloan. Examining Smart-Card Security under the Threat of Power Analysis Attacks. *IEEE Transactions on Computers*, 51(5), 2002.

[3] Yong-Bin Zhou and Deng-Guo Feng. Side-Channel Attacks: Ten Years After Its Publication and the Impacts on Cryptographic Module Security Testing, 2005.

[4] Galen Hunt and Doug Brubacher. Detours: Binary Interception of Win32 Functions. In *3rd Usenix Windows NT Symposium*, 1999.

[5] Earthlink and Webroot Release Second SpyAudit Report. `http://www.earthlink.net/about/press/pr_spyAuditReport/`, June 2004.

[6] Secure Information Technology Center Austria (A-SIT). The Austrian citizen card. `http://www.buergerkarte.at/index_en.html`, July 2005.

[7] Dino Esposito. Browser Helper Objects: The Browser the Way You Want It. `http://msdn.microsoft.com/library/en-us/dnwebgen/html/bho.asp`, 1999.

[8] Rundfunk und Telekom Regulierungs-GmbH. Austrian Signature Act. `http://www.signatur.rtr.at/en/legal/sigg.html`, December 2001.

[9] Rick Kennell and Leah Jamieson. Establishing the Genuinity of Remote Computer Systems. In *12th Usenix Security Symposium*, 2003.

[10] Arvind Seshadri, Mark Luk, Elaine Shi, Adrian Perrig, Leendert van Doorn, and Pradeep Khosla. Pioneer: Verifying code integrity and enforcing untampered code execution on legacy systems. In *ACM Symposium on Operating System Principles (SOSP)*, 2005.

[11] Umesh Shankar, Monica Chew, and Doug Tygar. Side effects are not sufficient to authenticate software. In *13th Usenix Security Symposium*, 2004.

[12] Hanno Langweg. Malware Attacks on Electronic Signatures Revisited. In *Sicherheit 2006. 3rd Jahrestagung Fachbereich Sicherheit der Gesellschaft fuer Informatik*, 2006.

[13] Adrian Spalka, Armin Cremers, and Hanno Langweg. Trojan Horse Attacks on Software for Electronic Signatures. *Informatica*, 26, 2002.

[14] Adrian Spalka, Armin Cremers, and Hanno Langweg. Protecting the Creation of Digital Signatures with Trusted Computing Platform Technology Against Attacks by Trojan Horse. In *IFIP Security Conference*, 2001.

[15] Auson Josang, Dean Povey, and Anthony Ho. What You See is Not Always What You Sign. In *Annual Technical Conference of the Australian UNIX and Open Systems User Group*, 2002.

[16] Tobias Murmann and Heiko Rossnagel. How Secure Are Current Mobile Operating Systems. In *IFIP Conference on Communications and Multimedia Security (CMS)*, 2004.

[17] German Federal Office for Information Security (BSI). Annual report 2004 (in german). `www.bsi.de/literat/jahresbericht/jahresbericht_2004/bsi_jahresbericht2004.pdf`, 2004.