Designed to Fail: A USB-Connected Reader for Online Banking

Arjan Blom[1], Gerhard de Koning Gans[2], Erik Poll[2], Joeri de Ruiter[2], and Roel Verdult[2]

[1] Flatstones, The Netherlands.
arjan@flatstones.nl

[2] Institute for Computing and Information Sciences, Digital Security Group, Radboud University Nijmegen, The Netherlands.
{gkoningg,erikpoll,joeri,rverdult}@cs.ru.nl

Abstract

We present a security analysis of an internet banking system used by one of the bigger banks in the Netherlands, in which customers use a USB-connected device – a smartcard reader with a display and numeric keyboard – to authorise transactions with their bank card and PIN code. Such a set-up could provide a very strong defence against online attackers, notably Man-in-the-Browser attacks, where an attacker controls the browser and host PC. However, we show that the system we studied is seriously flawed: an attacker who controls an infected host PC can get the smartcard to sign transactions that the user does *not* explicitly approve, which is precisely what the device is meant to prevent.

The flaw is not due to a simple implementation bug in one of the components (e.g. the device or the software components on the PC). It is a more fundamental design flaw, introduced in assigning responsibilities to the different components and designing the protocols between them.

The system we studied, used by the Dutch bank ABN-AMRO, was developed by the Swedish company Todos AB. This company has since been acquired by Gemalto. ABN-AMRO is one of the three biggest banks in the Netherlands, with 6.8 million customers. Given the popularity of internet banking in the Netherlands, this means that millions of these devices are in the field. The manufacturer claims this device is "the most secure sign-what-you-see end-user device ever seen" [3]; this paper demonstrates this claim to be false.

1 Introduction

For internet banking, many banks let their clients use a hand-held smartcard reader with a small display and keypad. In combination with a smartcard and PIN code, this reader then signs challenges reported on the bank's webpage, to log-on or to confirm bank transfers. Many of these systems use EMV-CAP

[3] http://www.gemalto.com/financial/ebanking/case_studies/ABN_AMRO.html

(EMV-Chip Authentication Program), a proprietary standard of MasterCard built on top of the EMV (Europay-Mastercard-Visa) standard, as does the system discussed in this paper.

Such a two-factor authentication, which requires access to a smartcard and a PIN code, is of course much stronger than a traditional password. Still, a serious and fundamental limitation of these hand-held readers is that the very short challenges, of only eight digits, do not offer much scope for a message that is meaningful to the user. Often the eight digits are just a random number, in which case the user has no real idea what he is authorising. This means that a Man-in-the-Browser attack, where the attackers controls the browser on the client's PC, can still let someone unwittingly approve unwanted transactions.

This risk can be mitigated by letting the user sign additional challenges, for instance the amount or say the last eight digits of the bank account, which are meaningful to the user. Some online banking sites use such additional challenges for transfers with high amounts or transfers to bank accounts not used before by a customer. The downside of this is more hassle for the user, typing the extra challenges and responses. Also, a compromised browser could *still* trick users into signing these additional challenges, as a user cannot tell if an apparently random challenge is not in fact an amount or the last 8 digits of the attacker's bank account. Users could be asked to type in more than eight digits, possibly using devices with a bigger display, but clearly there will be a limit on how much hassle users are willing to accept.

Connecting the reader to the PC with a USB cable can solve the problem, or at least drastically reduce the risk and impact of Man-in-the-Browser attacks. The user no longer has to retype challenges and responses, making longer challenges acceptable. Moreover, the display can be alphanumeric (even if the keyboard is numeric only), allowing more meaningful challenges for the user. Not only security is improved, but also user-friendliness, as users do not have to retype challenges and responses.

The system we studied works in this way. Prior to confirming a bank transfer, the user sees details of the transfer he is about to approve on the display of the reader, as shown in Fig. 1. Gemalto calls this system SWYS, "Sign What You See". There is a patent application describing this solution [11].

Of course, connecting the reader to a PC introduces new risks: the PC could be infected and controlled by an online attacker. Such an attacker can then interfere with communication over the USB line, passively eavesdropping or actively changing communication. Still, the functionality exposed over the USB line can be very restricted, and given the simplicity of the device – and the obvious security requirements – controlling this risk seems easy. However, as this paper shows, the system we studied fails to properly ensure one of the core security objectives.

Fig. 1. The USB-connected reader, with the bank card sticking out the top, showing transaction details for the user to approve a login (left) or a bank transfer (right).

2 Background: EMV-CAP

The internet banking system we studied uses a variant of EMV-CAP, a proprietary standard of Mastercard that is widely used for internet banking. EMV-CAP is based on the EMV standard [9]. EMV, which stands for Europay-Mastercard-Visa, is used in most (if not all) banking smartcards.

In EMV-CAP, a regular EMV transaction is started but cancelled in the last step. In the course of an EMV-CAP transaction a smartcard generates two so-called *Application Cryptograms* (ACs) as proof of authorisation. The first cryptogram, an *ARQC* (Authorization Request Cryptogram), is used as authorisation of some online banking transaction. The second cryptogram, an *AAC* (Application Authentication Cryptogram), just serves to cleanly terminate the transaction.

These cryptograms are not digital signatures using asymmetric cryptography, but MACs (Message Authentication Codes) generated using a symmetric key shared between the bank and the smartcard (typically a 3DES key). We will still talk about the card 'signing' challenges, even though technically this terminology is incorrect. The reason for using symmetric crypto is historic: most modern bank cards are now capable of doing asymmetric crypto, but EMV was designed to be used with cheaper cards that did not.

EMV-CAP readers typically use an 8 digit numeric challenge, and respond with a 8 digit numeric response which is constructed from the first cryptogram, an ARQC, reported by the smartcard. In most modes of EMV-CAP the response is constructed by applying a bit filter to the ARQC, which selects some bits of the MAC and the least significant bits of the smartcard's transaction counter[4], which is reported by the smartcard together with the ARQC.

[4] Called the ATC, for Application Transaction Counter, in EMV terminology. The ATC is included in the computation of the ARQC, so the bank needs to know it to verify the authenticity of the response.

EMV-CAP has been largely reverse-engineered [8, 15] and an informative description is leaked (apparently accidentally) in [5, Appendix 1]. This has revealed some potential for ambiguities [8] (e.g. between transactions to log-in and to transfer a zero amount) and, more worryingly, bad design decisions in some variants of EMV-CAP [15] (notably, *not* using unpredictable input, such as a nonce or transaction data, as input for the ARQC).

The system we studied also uses EMV-CAP, albeit with an additional twist, namely that there is an additional and unknown step to 'mangle' the challenge and the response, as explained in more detail later.

3 The e.dentifier2

The device we investigated is called the e.dentifier2. It is a smartcard reader with a display and keyboard, making it a class 3 device in the FinTS/HBCI classification[5]. The display can contain up to 68 characters. The keyboard provides numeric keys, an OK, and a Cancel button. The e.dentifier2 can be used for online banking with or without USB connection. Our attack only concerns the connected mode, but for sake of completeness we also describe the unconnected mode. As far as the interaction between the device with the smartcard is concerned, there is no difference, and the interaction is a regular EMV-CAP transaction.

3.1 Unconnected mode

Without USB connection, the device behaves like a standard EMV-CAP reader: the user types in an 8 digit challenge and the reader returns a 8 digit response. However, the device does not quite conform to known variants of EMV-CAP, and the device performs some additional mangling of the data on top of EMV-CAP. The challenge sent to the smartcard for computing the ARQC is not identical to the challenge typed in by the user, but there is some additional hashing and/or encryption. This additional mangling means that there are unknown functions hard-coded in the device. The secrecy of these functions is not crucial for the security, but it does prevent interoperability with EMV-CAP readers issued by other banks, and it prevents the construction of a software emulation of the device, as is available for other EMV-CAP readers[6].

As already mentioned, the attack reported in this paper does not concern the system in unconnected mode.

3.2 Connected mode

To use the connected mode, customers have to install a special driver (only available for Windows and MacOS). The web-browser then interacts with this

[5] Available from `http://www.hbci-zka.de`
[6] Available from `http://sites.uclouvain.be/EMV-CAP`

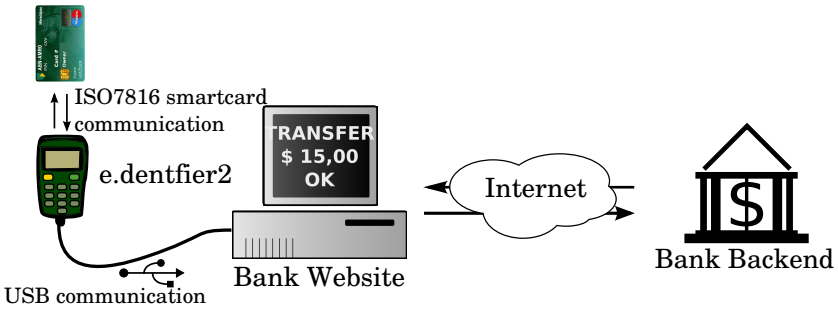

Fig. 2. Setup for internet banking with the e.dentifier; the web browser communicates over USB with the e.dentifier, into which the bank card is inserted.

driver via JavaScript and a browser plugin. The browser plugin checks whether it is connected to either the domain `abnamro.nl` or `abnamro.com` using SSL. If this is not the case, the plugin will not function. The Firefox plugin also allows local files to use the plugin. This might introduce security risks as, for example, this means that attachments in emails could also use the plugin.

In connected mode, an internet banking session starts with the reader reading the bank account number and the card number from the smartcard and supplying it to the web-browser. So the user does not have to type this in, making the system more user-friendly.

To log in, the reader first prompts the user for his PIN code. It then displays a message saying that the user is about to log in and asks the user to confirm this by pressing OK (see Fig. 1).

To confirm a bank transfer, or a set of bank transfers, the reader will again prompt the user for his PIN code. It then displays a message giving *the number of transfers* the user is about to approve and *the total amount of money* involved, and asks the user to approve this by pressing OK (see Fig. 1).

The additional security of the connected mode over the unconnected mode here is that *you see what you sign*, even if the browser or the PC it runs on is controlled by malware. Section 4 below discusses the security objectives to guarantee 'What-You-Sign-Is-What-You-See' in more detail.

Connecting the e.dentifier2 to a possibly infected PC by USB does introduce a new attack vector: malicious code on the PC could try to interfere with the device. Still, given the device is so simple and offers so little functionality, it should be possible to design and implement it so that it is secure against such attacks.

4 Security Objectives of WYSIWYS

The crucial security objectives to ensure *WYSIWYS* (What-You-Sign-Is-What-You-See) are that

– the text on the display is always included in the data that is signed,

– and that signing is only done after the user has given his consent.

Apart from the text on the display, the signature should also include some session-unique information, such as a nonce or transaction counter, to prevent replays.

The device has to be *a trusted display in the narrow sense*, by which we mean that what is displayed on the screen is (part of) what is signed by the smartcard. The device could also be *a trusted display in the broader sense* in that the messages displayed are guaranteed to originate from the bank. This second property is not strictly necessary for ensuring the user's consent in signing what is shown. Still, it is clearly a nice property to have; if the device is not a trusted display in this broader sense, then an attacker could use the display as part of sophisticated phishing attacks.

The device has to be trusted for PIN entry, as it passes the PIN code the user provides to the smartcard, so it should meet the security requirements this brings. The PIN code entered should not leak in any way: it should not be shown on the display when the user types it, and it must obviously not be revealed over the USB line. Only PIN codes entered on the keypad should be passed on to the smartcard as PIN guesses, and never data coming over the USB line. The latter would enable malware on the PC to perform a Denial-of-Service (DoS) attack, by blocking the card with three PIN guesses, or to guess the PIN code right in 0.03% of the cases (or more, if users can choose their own PIN [4]).

More generally, possibilities for interaction over the USB line have to be kept to a minimum: the device should not offer functionality for the PC to read data entered on the keypad over the USB line, or for the PC to send commands to the smartcard over the USB line (and then receive the card's responses). Clearly the device will have to send some commands to the smartcard on instruction from the PC, and then send back some of the responses, but such functionality should be limited to what is strictly necessary.

Note that it is the job of the smartcard to check whether the PIN code is correct. It is also natural that the smartcard only generates a signature after it has been supplied with a PIN code, and preferable does so only once[7].

Handheld readers for home use are typically not designed to be tamper-resistant or tamper-evident to any serious degree. This would make the devices far too expensive, while the online attackers that the reader is meant to protect against do not have physical access. (The FINREAD initiative [10] did aim at tamper-resistant readers, but was not successful because of the costs.) Still, although online attackers are the main threat, one should not overlook creative possibilities for attackers to get physical access. In the Netherlands criminals successfully tampered with readers that were available *inside* bank offices for customers to use, in order to obtain PIN codes and enough data to reconstruct valid mag-stripes. Of course, allowing the mag-stripe to be constructed from data supplied by the smartcard chip was a very bad design choice in EMV, and has since been remedied.

[7] Tests we have done with bank cards show that this indeed the case [1].

5 Analysis tools

Reverse engineering the precise working of the system, and subsequently carrying out our attack, required eavesdropping and sometimes also actively altering traffic on the three channels shown in Fig. 2, i.e. communication (i) with the bank's website over HTTPS, (ii) between the PC and the reader over USB, and (iii) between the reader and the smartcard. For this we used various tools.

HTTP(S) communication To intercept and modify network traffic between the browser and the bank web application we used Fiddler2, a web debugging proxy[8].

ISO7816 smartcard communication To eavesdrop on communication between the smartcard and the reader we initially used the APDU Scanner by RebelSIMcard[9]. This software and the associated hardware to physically intercept the traffic have been developed to study SIM cards (typically, to study how SIM locking works), but it works for any ISO7816 smartcard.

Later on, we also used a tool for smartcard communication analysis that we developed ourselves [6], which also allowed active Man-in-the-Middle attacks and replaying of earlier sessions to the smartcard reader. This tool consist of a USB-connected FPGA (Field Programmable Gate Array) and can be used for passive eavesdropping, relay attacks (where smartcard communication can be relayed for instance over IP), and active Man-in-the-Middle attacks. The design of this device is open and the software is available as under the GNU GPL.

Eavesdropping USB communication To eavesdrop on communication over the USB line we used USBTrace[10] and Wireshark[11]. USBTrace has the advantage that in addition to the raw traffic of individual packets it also shows the aggregated communication of larger messages that in fact take several packets. However, this also caused some confusion, as initially it was unclear what really went over the line.

Replaying USB traffic To replay USB communication we wrote our own software. The USB interface of the e.dentifier2 requires a vendor specific driver for communication. The workings of this driver is proprietary and not publicly available. We used the open-source libusb library[12] to build a simplified driver ourselves. Libusb takes care of all the low-level details of the communication between the computer and USB device. The driver we developed uses bulk data transfers, which are unspecified raw messages that allows a user to send and receive binary data with an arbitrary length. The eavesdropped communication showed that all communicated messages were a multiple of 8 bytes long. Although the details of the communication protocol were not available, it was rather simple to replay an

[8] `http://fiddler2.com`
[9] `http://rebelsimcard.com/network-sim-apdu-scanner.html`
[10] `http://www.sysnucleus.com`
[11] `http://www.wireshark.org/`
[12] `http://www.libusb.org/`

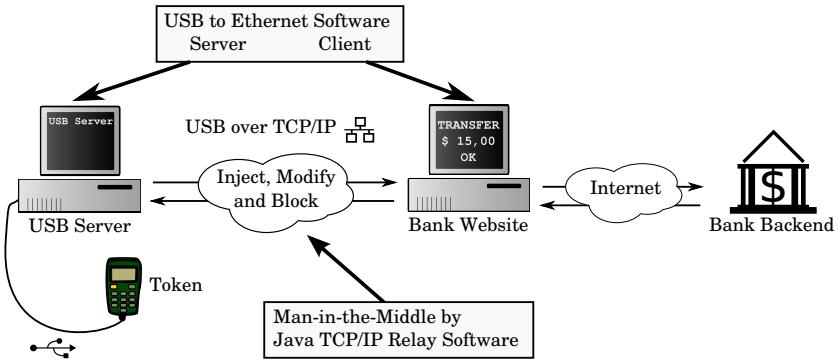

Fig. 3. Setup for Man-in-the-Middle attack on USB traffic; the communication between the browser and the e.dentifier2 is tunnelled over TCP/IP to another PC (labelled USB Server), which forwards it to the e.dentifier2.

earlier recorded transaction of USB messages. Since there is no authentication between the device and the bank, all features of the e.dentifier2 become available directly after plugging the device into a computer.

Man-in-the-Middle attack on USB traffic Executing and validating the attack in a real online banking session required a Man-in-the-Middle attack on the USB traffic in which we actively altered some data. We could not find any existing software to carry out a Man-in-the-Middle attack on USB. We therefore did it in a somewhat roundabout way, by tunnelling the USB traffic over TCP/IP, and then modifying this tunnelled traffic, as depicted in Fig. 3. We also considered reverse engineering and modifying the driver code, and tried to develop a custom device driver using the LibUsbDotNet library[13], but these alternatives seemed more work. Tunnelling the USB traffic over TCP/IP could be done with existing software, namely the *USB to Ethernet Connector*[14]. The advantage of this method is that there is no need to fully understand the USB protocol. Furthermore, the software of the banking application, notably the custom device driver, could be used without any modifications.

6 The SWYS Protocol

The tools described in the previous section allowed the reverse engineering of online banking with the e.dentifier2 in connected mode.

Fig. 4 outlines the abstract SWYS protocol for logging in or confirming a transaction (i.e. a bank transfer). In either case the web-browser sends so-called *signdata* to the reader. This signdata consists of two parts, namely some text (that is shown on the display) and some number (which is not):

[13] `http://sourceforge.net/projects/libusbdotnet`
[14] `http://www.eltima.com/products/usb-over-ethernet/`

– In the case of a login, the signdata text specifies the account number and bank card number; in case of a transaction, it specifies the number of transactions and the total amount (see Fig. 1).
– In the case of a login, the signdata number is a 4 byte value giving the current UNIX time; in case of a transaction, it is a 20 byte apparently random value.

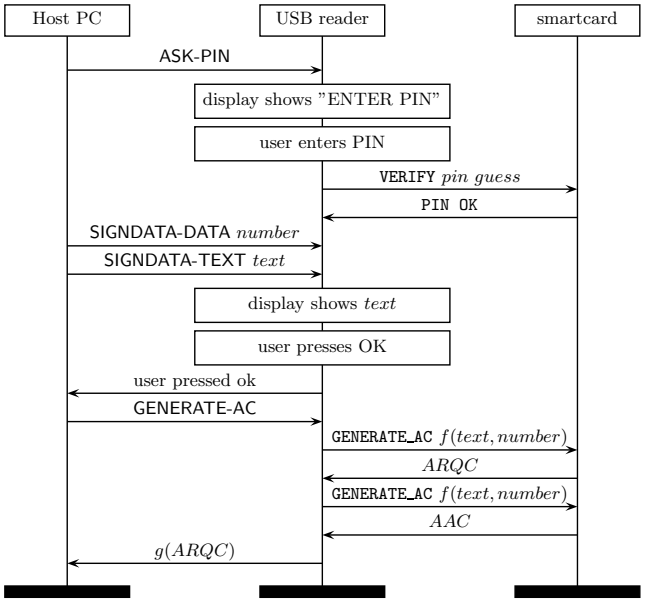

Fig. 4. SWYS protocol for log-in and for transactions

In the protocol, the reader asks the smartcard to produce two cryptograms, using the `GENERATE_AC` command. Included in these commands is a 4 byte payload[15]. (As is usual in EMV-CAP transactions, the first cryptogram is an ARQC, the second an AAC, but these details do not matter here.)

By replaying earlier transactions, and varying the text and number parts of the signdata, we could confirm that this 4 byte payload does depend on both the text and the number. Hence this payload is written as $f(text, number)$ in Fig. 4. It has to depend on the text to make sure that 'we sign what we see'; including the number is useful to diversify the input and prevent replays, though the computation of the cryptograms will also involve the card's transaction counter. We do not know what the function f is.

To finish a transaction, the reader sends a response back to the web-browser, which is based on the cryptograms generated by the smartcard. Again, we do not precisely know how this response is computed, only that it depends on the

[15] called the `Unpredictable Number` in EMV terminology

ARQC reported by the smartcard. Hence it is written as $g(ARQC)$ in Fig. 4. So g, like f, is an unknown function that is implemented in the device.

ASK-PIN	01 03 04 00 00 00 00 00
SIGNDATA-DATA-LOGIN	01 03 05 06 00 00 00 00
SIGNDATA-DATA-TRANSACTION	01 03 05 16 00 00 00 00
SIGNDATA-TEXT	01 03 05 46 00 00 00 00
GENERATE-AC	01 03 06 00 00 00 00 00

Fig. 5. commands sent over the USB line

Protocol details The webpage interacts with the driver through JavaScript. The data that is passed to the driver, such as the signdata, is TLV encoded.

Communication between the reader and the smartcard uses the ISO7816 protocol, using the instructions as defined in the EMV standard. The protocol excerpt shown in Fig. 4 involves two instructions, `VERIFY` and `GENERATE_AC`. On start-up there is some additional communication with the smartcard, to read out some of the data from the smartcard, such as the account number, which is used in communication over the internet.

The communication between the PC and the reader over USB was harder to understand. It involves more data than between the reader and the smartcard, and it does not follow any standards, such as ISO7816 and EMV, that we knew. For most of the traffic from the PC to the reader we could eventually determine the meaning, see Fig. 5. though we see some additional data for which the meaning is unclear. The reader responds with `01 03 02 00 00 00 00 00` to indicate OK. Fig. 6 contains a communication trace between computer and e.dentifier2. It shows the USB frames for a successful transaction of €123.00.

```
Tx: 01 03 01 02 00 00 00 00 ........
    00 02 65 6e 00 00 00 00 ..en....
Rx: 01 03 01 01 00 00 00 00 ........
    00 01 01 01 00 00 00 00 ........

Tx: 01 03 04 00 00 00 00 00 ........
Rx: 01 03 02 00 00 00 00 00 ........

Tx: 01 03 05 16 00 00 00 00 ........
    00 06 00 14 9a 2c 83 9a .....,..
    00 06 4f 4f 3c 8e 3e b8 ..OO<.>.
    00 06 ef 39 0b 60 b9 46 ...9.‘.F
    00 04 30 12 01 00 b9 46 ..0....F
Rx: 01 03 02 00 00 00 00 00 ........

Tx: 01 03 05 46 00 00 00 00 ...F....
    00 06 01 44 44 6f 6d 65 ...DDome
    00 06 73 74 69 63 20 20 ..stic
    00 06 20 20 20 20 20 20 ..
    00 06 20 31 20 74 72 61 .. 1 tra
    00 06 6e 73 61 63 74 69 ..nsacti
    00 06 6f 6e 28 73 29 20 ..on(s)
    00 06 45 55 52 20 31 32 ..EUR 12
    00 06 33 2c 30 30 20 20 ..3,00
    00 06 20 20 20 20 20 43 .. C
    00 06 6f 6e 66 69 72 6d ..onfirm
    00 06 20 77 69 74 68 20 .. with
    00 04 4f 4b 20 20 68 20 ..OK h
Rx: 01 03 02 00 00 00 00 00 ........

Tx: 01 03 06 00 00 00 00 00 ........
Rx: 01 03 03 19 00 00 00 00 ........
    00 06 80 00 6d b8 c7 cc ....m...
    00 06 ab 00 12 a5 00 03 ........
    00 06 02 00 00 00 00 00 ........
    00 06 00 00 00 00 00 00 ........
    00 01 ff 00 00 00 00 00 ........
```

Fig. 6. Communication between computer and e.dentifier2 over the USB line

7 The attack

As can be seen in Figure 4, the reader sends a message to the host PC indicating the user pressed OK. After this the host PC sends a command to generate the

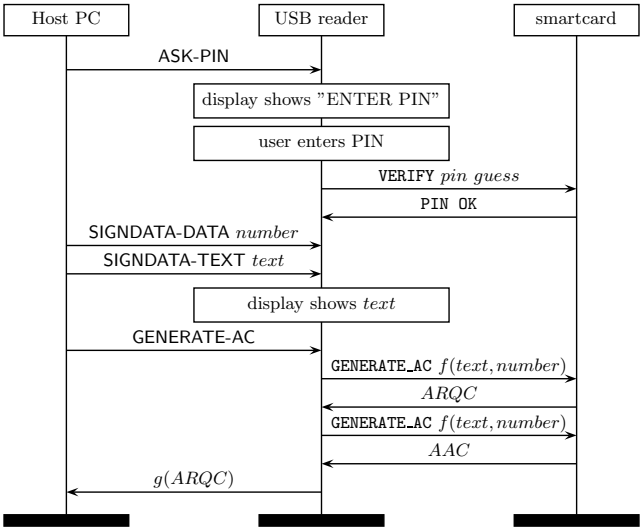

Fig. 7. Attack on SWYS protocol; the difference with Fig. 4 is that the driver directly gives the instruction to generate the ACs, without waiting for the user to press OK

cryptograms to the reader. This seems strange, as the reader would be expected to generate the cryptograms automatically after OK has been pressed. The driver on the host PC should not play a role in this.

This weakness can be exploited: by sending the request over the USB line to generate the cryptograms *without* waiting for the user to press OK, the cryptograms are generated and the reader returns the response over the USB line, without the user getting a chance to approve or cancel the transaction. To make matters worse, a side-effect of giving this command is that the display is cleared, so the transaction details only appear on the display for less than a second. We demonstrated this attack in an actual internet banking session.

This means that an attacker controlling an infected PC can let the user sign messages that the user did *not* approve, thus defeating one of the key objectives of *WYSIWYS*. The user still has to enter his PIN, but this is entered at the start of a transaction, and after this no more interaction is needed from the user to sign malicious transactions.

8 Other attack scenarios

In a different approach we tried to manipulate the text displayed on the screen. For this we investigated whether the fact that not only the text on the display but also some random data is signed, introduced a flaw. If the text and random data would simply be concatenated, part of the text might be shifted to the random data. This would result in less text being displayed (for example only 'Press

OK to continue'), while still producing the same signature. In our experiments this however did not work, suggesting that either some separator is used in the concatenation or a part of the USB packets is also included.

9 Extended length challenges

Initially it was mind-boggling to us how this vulnerability could ever have been introduced. Subsequently it became clear that the vulnerability may have been introduced as a side-effect of having additional functionality in the device, where the device shows *several* messages for the user to approve. Presumably this functionality is included for more complex transactions where more than 68 characters are needed to display transaction data.

We never observed such transactions in our use of the online banking system, but using our own driver we could see that the device is capable of doing this. Several messages can be sent to the reader in turn, with the next message being sent after the user presses OK, and then after the final message the driver can give the command to generate the response. We could observe that the 4 byte payload sent as challenge to the smartcard depended on all the texts that were sent to the reader and displayed there. Presumably the unknown function f implemented in the device hashes these texts together to compute this challenge.

For this variant of the protocol the reader needs to communicate with the driver after the user presses OK, namely to request the next part of the message. This might explain – but not excuse! – why the weakness has been introduced in the first place, and why it was missed in security reviews.

Note that this variant of the protocol results in an overloaded semantics for the OK button: it can mean an instruction (to the driver) to 'send more data to the display' or an instruction (to the smartcard) to 'go ahead and generate a cryptogram'. Since the reader is not aware of the meaning of the button, the host PC has to determine this, providing a possible origin of the vulnerability.

10 How could – and should – this have been prevented?

The attack that we found is something that should have been detected, and – as we argue below – could have been detected. After all, the attack does not involve some detailed cryptanalysis, rely on a cleverly crafted man-in-the-middle attack, or exploit some subtle low level coding mistake. A patent application describing an e.dentifier2-like solution has been filed by the company that produces it [11]; the higher level description of the protocol given there does not include the vulnerability that we found.

We have no insight in the procedures followed in design, implementation, or testing, or indeed any of the associated documentation, so we can only speculate how the vulnerability could possibly have been missed. Still, we can consider how it could – and should – have been spotted, or, better still, prevented in the first place.

Firstly, the attack breaks one of the central security objectives of WYSIWYS. This security objective should be used as a basis for considering abuse cases. (If the EMV specs [9] are anything to go by, the security objectives may well be completely undocumented and implicit in the specs.) An obvious abuse case that could already have been identified in the early design phase is: (i) malicious code on the PC that tries to get the reader to sign without the user's approval.[16] It seems unlikely that the vulnerability could have gone unnoticed in design, implementation and testing if this abuse case was made explicit, as it would then have been considered by e.g. a reviewer of the specifications or tester of the implementation. In fact, one would then expect someone developing the spec or implementing it to notice the problem long before any post-hoc security evaluation.

As discussed in Section 9, it seems that the semantics of pressing the OK button is overloaded. This overloading could be spotted in a high level spec that only considers the interaction with the user, if in such a spec one tries to make the semantics of pressing OK explicit. Overloading the semantics of actions or cryptograms is also noted as a potential problem in EMV-CAP [8].

Formal techniques for protocol verification have been shown to be capable of handling complex real-life protocols. For example, it has been shown that the core of the EMV protocol can be fully formalised and that tools can then reveal known weaknesses [7]. Such techniques could also be used to look for weaknesses as the one discussed in this paper.

Finally, even if the problem went unnoticed in the design, the problem could still have been detected by more systematic testing. Exhaustive testing is of course impossible, even for a system as simple as this: the messages over the USB line are small but too long for exhaustively testing all possible contents. Still, the number of different *types* of messages over the USB line is very small. The number of internal states of the reader, which correspond to particular states of the simple protocol it implements, is very small too. So exhaustively testing every type of message in each protocol state is feasible, and this would have inevitably revealed the problem. Such testing can be done in a systematic and largely automated way using model-based testing [16].

11 Related work and related online banking systems

We are not aware of any related work investigating USB-connected smartcard readers for internet banking. As discussed in Section 2, unconnected smartcard readers for internet banking has been thoroughly investigated [8, 15]. For contactless smartcard technology, such as Near Field Communication (NFC), there are several proposals [13, 2, 14] in the literature for secure internet banking.

[16] Other abuse case would be: (ii) malicious code on the PC tries to get the reader to sign something not shown to the user and (iii) malicious code on the PC tries to send a PIN code guess to the smartcard.

There are other banks that use USB-connected readers that act as smartcard readers. The Gemalto website gives an overview of banks that use their products, but not all these systems work in the same way.

The Swedish Handelsbanken also uses a Gemalto reader, but here things work differently from the system described above. For internet banking with Handelsbanken, the reader is used to obtain bank account details from the smartcard so the user does not have to enter this manually. However, the display of the reader is not used at all, except to show a progress bar when entering PIN codes! To log in or confirm a transaction shown on the webpage, the user has to enter his PIN code on the reader, but all further interaction between the PC, reader and smartcard is invisible to the user. Effectively, this means that the class 3 smartcard reader (with display and keyboard) is only used as a class 2 smartcard reader (with keyboard but without display). This suggests Handelsbanken does not consider Man-in-the-Browser attacks a serious threat, and only uses the USB connection to make internet banking more user friendly, as users do not have to type in their account numbers or re-type challenges and responses.

The only other USB-connected reader used for internet banking we are aware of is at another Swedish bank, SEB. SEB uses a reader from a different vendor, namely the DIGIPASS 920 by Vasco[17]. This reader is used similarly to the way Handelsbanken uses their reader, i.e. without using the display.

Unlike the systems above, which do not even try to achieve WYSIWYS, there are alternative systems for internet banking that do try to achieve this.

The German Volksbank uses the Sm@rtTAN optic where an optical signal (a flickering bar code) on the webpage is used to transmit information about the transfer to a handheld smartcard reader (which has an optical interface). The reader then shows transaction details and produces a confirmation code that the user has to type in. Advantage of this system is that communication is physically guaranteed to be one way – from the PC to the reader – unlike with a USB connection. The disadvantage is that it introduces a bit more hassle for the user, e.g. the user has to hold the reader in front of the bar code during the transmission of the data.

IBM's ZTIC (Zone Trusted Information Channel) is a USB-connected device with a small display [18, 17][18]. Unlike the devices mentioned above, the ZTIC has the capabilities to set up a secure channel to a remote site, meaning that it is a trusted display in the broad sense as discussed in Section 4. It can therefore provide stronger security guarantees than the devices above, at the expense of a more complex device, which now also is able to store key material (notably public keys to set up secure channels) and needs the cryptographic processing capabilities for this. The ZTIC does not have a keyboard, but does have a turn-wheel to allow user input, which can be used for entering a numeric PIN code.

[17] `http://www.vasco.com/company/case_studies/seb.aspx`
[18] `http://www.zurich.ibm.com/ztic`

12 Conclusions

The basic idea of having a simple USB connected device that provides a trusted display and keyboard for users to authorise transactions, by letting their smartcard provide digital signatures, is very sensible[19]. This makes it all the more disappointing that this particular implementation of the idea fails so miserably. It is ironic that a system that is marketed under the name SWYS, 'See What You Sign', should fail to meet the central security objective it is named after.

The weakness cannot be fixed by improvements in the device driver. The problem could be fixed in new versions of the card reader without requiring changes in the smartcard or the device driver. Of course, this leaves the practical problem that there are millions of card readers already deployed.

To make matters worse, the problem is not caused by a subtle flaw or low-level coding defect that is easily overlooked. The fundamental design decision of giving the PC such fine-grained control over the reader (in particular, letting the reader report to the PC that the user pressed OK, and then letting the PC instruct the reader to let the smartcard generate a cryptogram) goes against the whole idea of What-You-Sign-Is-What-You-See. This design decision does not inevitably lead to a security flaw, but it is clearly asking for trouble, and opens the door to a whole category of potential attacks.

We can only wonder what possessed the developers to design the system in this way, or indeed how banks deciding to use this system could have missed it in even a superficial security analysis.

The bank was informed about this vulnerability immediately after we discovered it. When informing them we supplied a video demonstrating the attack; by checking the logs of the bank's back-end they could confirm that this video showed an actual attack. The bank responded promptly and contacted their supplier. This resulted in swift action to develop an improved version of the reader, which will be rolled out in the course of 2012. The supplier confirmed to us that they have no other clients that use the system with this weakness; it seems that ABN-AMRO has been an early adopter of this particular product.

Our paper is not the first to signal problems with security protocols used in the banking industry. Several problems have been noticed in the past [12, 15, 3]. Banks are now wise enough to use standard cryptographic algorithms, such as 3DES, RSA, and SHA-1, and not to rely on proprietary ones. However, they do still choose to design their own security protocols and try to keep them secret. Given that security protocols can be so tricky to get right, we believe it would be better to adopt the same approach for security protocols as for cryptographic algorithms, and only use published protocols that have been subjected to a rigorous public review.

[19] In fact, our initial motivation in reverse-engineering the system, in an MSc project by the first author, was to see if we could re-use the same reader, in combination with a different smartcard, for other applications where digital signatures are needed. Since the device itself does not need to contain any secrets, this should be possible; however, the unknown hashing mechanisms in the device, functions f and g in Fig. 4, currently still prevent this.

References

1. F. Aarts, E. Poll, and J. de Ruiter. Formal models of bank cards for free. Draft, 2012.
2. G. Alpár, L. Batina, and R. Verdult. Using NFC phones for proving credentials. In *16th Measurement, Modelling, and Evaluation of Computing Systems and Dependability and Fault Tolerance (MMB&DFT 2012)*, volume 7201 of *Lecture Notes in Computer Science*, pages 317–330. Springer-Verlag, 2012.
3. A. Barisani, D. Bianco, A. Laurie, and Z. Franken. Chip & PIN is definitely broken. Presentation at CanSecWest Applied Security Conference, Vancouver 2011. More info available at `http://dev.inversepath.com/download/emv`.
4. J. Bonneau, S. Preibusch, and R. Anderson. A birthday present every eleven wallets? The security of customer-chosen banking PINs. *FC'12: Proceedings of the the 16th International Conference on Financial Cryptography*, 2012.
5. Check-In-Phone – Technology and Security. Available from `http://upload.rb.ru/upload/users/files/3374/check-in-phone-technologie_security-english-_2010-08-12_20.05.11.pdf`.
6. G. de Koning Gans and J. de Ruiter. The smartlogic tool: Analysing and testing smart card protocols. *IEEE Fifth International Conference on Software Testing, Verification, and Validation*, pages 864–871, 2012.
7. J. de Ruiter and E. Poll. Formal analysis of the EMV protocol suite. In S. Mödersheim and C. Palamidessi, editors, *Theory of Security and Applications (TOSCA 2011)*, volume 6993 of *LNCS*, pages 113–129. Springer, 2012.
8. S. Drimer, S. Murdoch, and R. Anderson. Optimised to fail: Card readers for online banking. In *Financial Cryptography and Data Security*, volume 5628 of *LNCS*, pages 184–200. Springer, 2009.
9. EMVCo. EMV– Integrated Circuit Card Specifications for Payment Systems, Book 1-4, 2008. Available at `http://emvco.com`.
10. CEN Workshop Agreement (CWA) 14174: Financial transactional IC card reader (FINREAD), 2004.
11. P. Gullberg. Method and device for creating a digital signature, 2010. European Patent Application EP 2 166 483 A1, filed September 17, 2008, published March 24, 2010.
12. S. Murdoch, S. Drimer, R. Anderson, and M. Bond. Chip and PIN is Broken. In *Symposium on Security and Privacy*, pages 433–446. IEEE, 2010.
13. D. A. Ortiz-Yepes. Nfc-cap security assessment. Technical report, IBM Zurich Research Laboratory, 2009.
14. Z. Saleh and I. Alsmadi. Using RFID to enhance mobile banking security. *International Journal of Computer Science and Information Security (IJCSIS)*, 8(9):176–182, 2010.
15. J.-P. Szikora and P. Teuwen. Banques en ligne: à la découverte d'EMV-CAP. *MISC (Multi-System & Internet Security Cookbook)*, 56:50–62, 2011.
16. J. Tretmans. Model based testing with labelled transition systems. *Formal methods and testing*, pages 1–38, 2008.
17. T. Weigold and A. Hiltgen. Secure confirmation of sensitive transaction data in modern internet banking services. In *Internet Security (WorldCIS), 2011 World Congress on*, pages 125–132. IEEE, 2011.
18. T. Weigold, T. Kramp, R. Hermann, F. Höring, P. Buhler, and M. Baentsch. The Zurich Trusted Information Channel–an efficient defence against man-in-the-middle and malicious software attacks. *Trusted Computing-Challenges and Applications*, pages 75–91, 2008.