IADIS International Journal on Computer Science and Information Systems
Vol. 10, No. 2, pp. 1-16
ISSN: 1646-3692

A FORMALLY VERIFIED DIGITAL SIGNATURE DEVICE FOR SMARTPHONES

Peter Trommler. *Laboratory for Safe and Secure Systems (LaS^3), Technische Hochschule Nürnberg-Georg Simon Ohm, Nuremberg, Germany.*

ABSTRACT

Attacks on Internet banking using a malware called "High Roller" triggered the EU cyber security agency to issue a warning and call for action. With the proliferation of smartphones, customers want to do online banking on their phone, too. But a smartphone could also be compromised and so the customer cannot trust what is shown in the display and PINs could be sent to the attacker. We assume an attacker motivated by financial gain through diverting manipulated bank transactions to an account under his control.
Based on that assumption, we propose signed transaction summaries where the signature is created in a separate security device after the summary has been shown to the user and the user has approved it. Keeping the requirements to the absolute minimum, we derive a hardware implementation for the Jolla smartphone and, based on that hardware, a secure software implementation. We use commercial off-the-shelf components and, by keeping the protocols simple, reduce the trusted computing base as much as possible. We then demonstrate how the program in the micro controller avoids common software flaws and show fragments of a formal verification of the correctness of the microcontroller program.

KEYWORDS

Secure digital signing unit, Internet banking, transaction summaries, chip cards, formal verification.

1. INTRODUCTION

Attacks on Internet banking called "High Roller" caused losses in the range of "tens of millions of dollars" (ENISA, 2012, citing McAfee and Guardian Analytics) and that triggered the EU cyber security agency to issue a warning and call-for-action in a press release (ENISA, 2012). The attack was carried out using malware to automatically attack the victim's bank transactions.

To counter attacks against the customer's computer doing online banking, banks demand an inexpensive yet secure solution. The solution must be easy to use by the average customer. The weakest link in the security chain is the customer's Internet enabled device, be it a conventional desktop PC, a smartphone or a tablet computer. Malicious software could be installed on the customer's device. Malicious software includes key loggers that send passwords and transaction numbers to an attacker who in turn uses this information to break into the customer's account.

To cope with the issue of potentially compromised customer devices, banks have introduced software and hardware solutions. (ENISA, 2013) surveys the solutions used by European banks. Software solutions against key loggers try to make it more difficult for the attacker to gain authentication information, for example, by simulating keyboards in the browser and having the user type in their information using a pointing device. Hardware based solutions employ external devices. A detailed review of hardware based solutions will be given in Section 5.

In Internet banking mobile transaction numbers (mTAN) are widely used: a short transaction summary and a transaction number are sent to a mobile phone by short message service (SMS) and the user types in that transaction number to confirm the authenticity of the transaction. The security of this method relies on the separation of the two communication channels: the Internet connection of the browser on the one hand and the mobile phone network on the other hand. With Internet banking on a smartphone this separation is questionable, especially when we assume a compromised smartphone where the attacker also has access to text messages on the phone. For that reason, banks started to require that "the device used for receiving the TAN (e.g. mobile telephone) is not to be used simultaneously for online banking." BHF Bank Aktiengesellschaft (2015). Having to use a second mobile phone to receive the mTAN and then type it into the smartphone is not an attractive proposition.

This paper proposes an external security device that connects to a smartphone. Starting with an attack model we develop an architecture for the security device and present how that architecture is integrated into existing industry standards (Section 2). The architecture leads to a prototypical implementation based on the Jolla smartphone (Section 3). We focus on an implementation with minimal functionality and explain how this reduces the attack surface against both hardware and software of the device, and show how selected functions can be formally verified for correctness (Section 4). Finally, we compare our approach with other approaches to hardware security devices for online banking (Section 5) and conclude with pointers to future applications and future work (Section 6).

2. ARCHITECTURE

2.1 Attack Model

We assume that the customer's smartphone could be compromised and that an attacker has complete control over the smartphone. In particular, the customer cannot trust what is shown in the display and keystrokes will be logged and sent to the attacker. The fact that a compromised smartphone includes both a potential key-logger and a compromised display has been discussed in academia for quite a while (Langweg and Schwenk, 2007) and has been acknowledged by the European Union Agency for Network and Information Security (ENISA, 2012).

We assume an attacker who is motivated by financial gain by diverting bank transactions to an account under his control or creating transactions without the customer knowing about it. We also assume that the attacker controls all Internet traffic to and from the smartphone.

The attacker in our model, however, does not have physical access to either the smartphone or the security device. We only consider Internet-based attacks.

2.2 Protect Financially Relevant Data

To profit from security flaws in Internet banking an attacker manipulates transactions to reroute money transfers into an account directly or indirectly (through the use of so-called money mules) controlled by the attacker. Protecting against an attacker who is motivated by financial gain then requires protecting the financially relevant information in every bank transaction, e.g. ensure that the amount transferred and the beneficiary of a transfer have not been manipulated on the user's device or in transit on the Internet before a transaction is executed by the bank.

An analysis by Hehn (2009) shows that transactions of the FinTS standard at the time of that writing can be classified as follows:

- Protect beneficiary: Transfers
- Protect customer: Direct debit, term deposit
- No protection possible: Batch processing
- No protection necessary: Cancel credit card, order pre-printed forms

The classification still applies to the release of the FinTS standard (Die Deutsche Kreditwirtschaft, 2014a and 2014b), which is the latest release at the time of this writing.

Batch processing is not something that is typically done by a user of an Internet banking service on a smartphone where individual transactions are entered into Web forms. Hence, we focus on the transaction classes where protection is possible.

We propose to digitally sign a transaction summary on a secure signature device that the user owns and controls. That digital signature and the transaction summary now form a proof that the financially relevant information of the original transaction has not been tampered with by an attacker. The transaction summary contains the financially relevant data of the transaction and a serial number to counter replay attacks. The example of transaction numbers also demonstrates that a plain text transaction summary is acceptable to users. This observation will enable us to create a simple signature device for a smartphone.

A bank transaction consists of the following steps:

1. User enters transaction data into a form in her browser or banking app on the smartphone and sends it to the bank's server.
2. Bank server returns a text (transaction summary) to the smartphone that contains the financially relevant data of the transaction.
3. Show complete transaction summary to user on the security device.
4. On the security device the user verifies and approves the transaction summary or aborts the transaction.
5. User enters PIN on the security device to enable a single public key signature on the chip card.
6. Chip card receives the transaction summary from the security device and produces a digital signature.
7. The security device sends the digital signature to the smartphone.

8. Digital signature and the transaction summary and the original transaction data are sent to the bank by the smartphone.
9. Bank server checks that the transaction summary matches the transaction and verifies that the signature matches the transaction summary.

2.3 FinTS/HBCI Protocol Extension

In this section we discuss how the scheme to protect financially relevant data described above is integrated into the FinTS standard (Die Deutsche Kreditwirtschaft, 2014d) for Internet banking. The FinTS messages can be used in a banking app or can also be generated by a Web application running in a browser.

FinTS supports various methods to authenticate a transaction. For digital signatures several digital signature formats are offered, including a user-defined signature. FinTS uses XML to transmit data structures. A user-defined signature has the following components:

- FinTSProperty: Protocol-specific parameters such as signer role and time stamps
- BankID: Bank-specific identifier
- UserID: User name, customer number or other identifier
- Method: Identifier for signature method
- Option: Additional parameters for signature method
- UDSData: Actual signature data.

The field UDSData of a user defined signature has type "any", i.e. that field can contain further XML elements. For our signed transaction summary we define a new XML-Element `TransactionSummarySignature` with two nested elements called `TransactionSummary` and `SignatureData`.

```
<UserDefinedSignature>
     <UserID>Customer 1</UserID>
     <Method>TransactionSummary</Method>
     <UDSData>
            <TransactionSummarySignature
                   xmlns="Transaction summary signature"
                   xsi:schemaLocation="TransSumSig.xsd">
                   <TransactionSummary>
                          Transfer Account: DE…007 BIC: XXXXXXX
                          Amount: EUR 7.37 Serial: 5
                   </TransactionSummary>
                   <SignatureData>GAcAC…YoDBm</SignatureData>
            </TransactionSummarySignature>
     </UDSData>
</UserDefinedSignature>
```

Figure 1. Sample XML structure of a signed transaction summary

2.4 Verification of a Transaction Summary Signature

The bank receives the transaction signed with a transaction summary signature and must verify that the transaction is authentic. The first check is for freshness of the presented transaction summary by checking that the serial number has not been used in a previous transaction. This is done to prevent replay attacks. The next step is to check that the

transaction summary matches the transaction data. To verify authenticity of the transaction, the transaction summary needs to be parsed. Then the transaction type and its parameters can be extracted from the transaction summary and compared with the respective values found in the transaction. If those values match the transaction summary then the signature of the transaction summary is verified. The public key of the customer is retrieved and the hash value is recovered form the signature. Then the hash function is computed over the transaction summary. If both of the hash values match, then the transaction is approved. Otherwise it is rejected. This workflow is depicted in Figure 2.

The order of the checks ensures that computationally cheap operations like the freshness check are performed before expensive operations involving cryptographic functions. The freshness check is also cheaper than parsing the transaction summary and the transaction data.

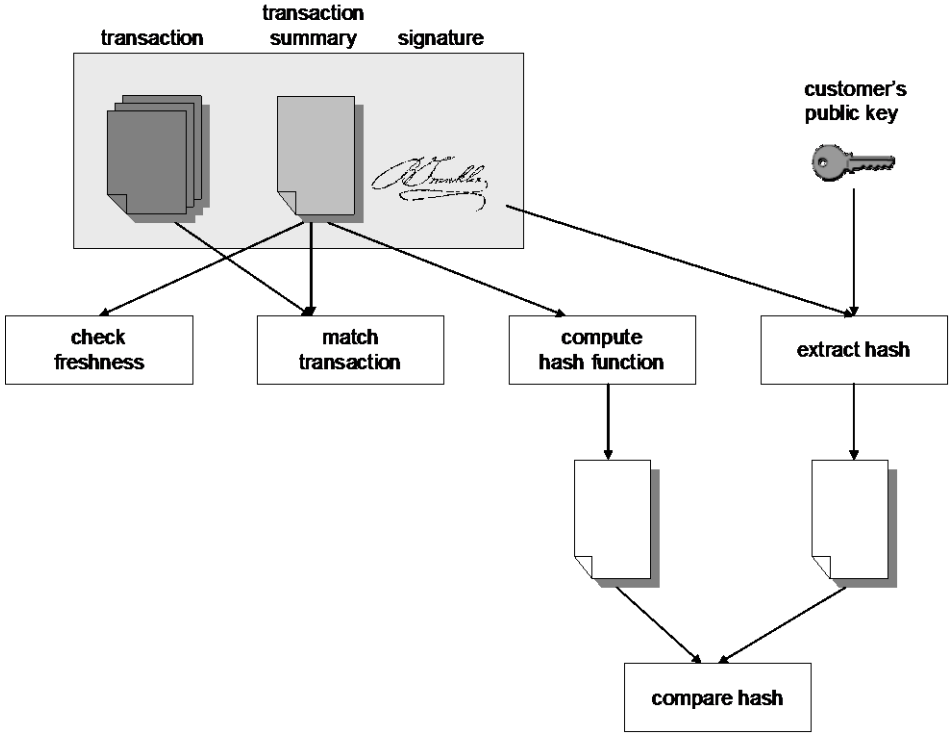

Figure 2. Workflow of the verification of a signed transaction summary on the bank server

3. SIMPLE SIGNATURE DEVICE

Signed transaction summaries must be implemented in a separate security device because in our attack model we assume the smartphone is compromised. So there is no trusted input channel and no trusted display on the smartphone. Starting with requirements, we derive a hardware implementation and, based on that hardware, a secure software implementation.

3.1 Requirements

The security device accepts a transaction summary from the smartphone and presents the transaction summary to the user on a display of the security device. The user then decides whether to reject and hence abort the transaction or to accept the transaction and move on to the next step. The user's choice is entered on a key pad of the security device. If the user rejects the transaction an error message is reported back to the smartphone. In the case of an acceptable transaction the user enables the signing function of a chip card for one signature by entering a signing PIN. The PIN is sent to a chip card following the established protocols of ISO 7816. Upon positive validation of the PIN, the transaction summary is sent to the card to sign and the digital signature is received form the card. The security device sends that signature to the smartphone where it is then added to the user defined signature as defined in Section 2.3.

3.2 Prototype based on Jolla "The Other Half"

In this section we describe a prototype of a security device for the Jolla smartphone and a back cover for Jolla phones called "The Other Half." We will specify which components we used in our prototype for display, chip card attachment, keypad, and microcontroller, and will describe how those components are connected.

With "The Other Half" a Jolla smartphone offers a replaceable back cover that can be equipped with a "smart device." The smartphone detects when the back cover is removed and reattached to the phone body. To tell different covers apart each cover can have an NFC tag that can also be used to personalize the cover to an individual smartphone. The phone offers 3.3 V power to a smart device in the cover and communicates through I^2C with a slave smart device. An interrupt line can be used to signal events to the smartphone.

We chose a SparkFun Pro Micro 3.3 V/8 MHz Arduino-compatible micro controller development board with I^2C attachment to the Jolla smartphone, a Nokia 5110/3310 monochrome LCD with 84 by 84 pixels, and a standard key pad with a resistor network which is read by one analogue pin of the micro controller. The serial port chip card communication is controlled directly by the micro controller. Finally, the chip card is a Java Card IBM JCOP41 by NXP with our minimal signing applet installed.

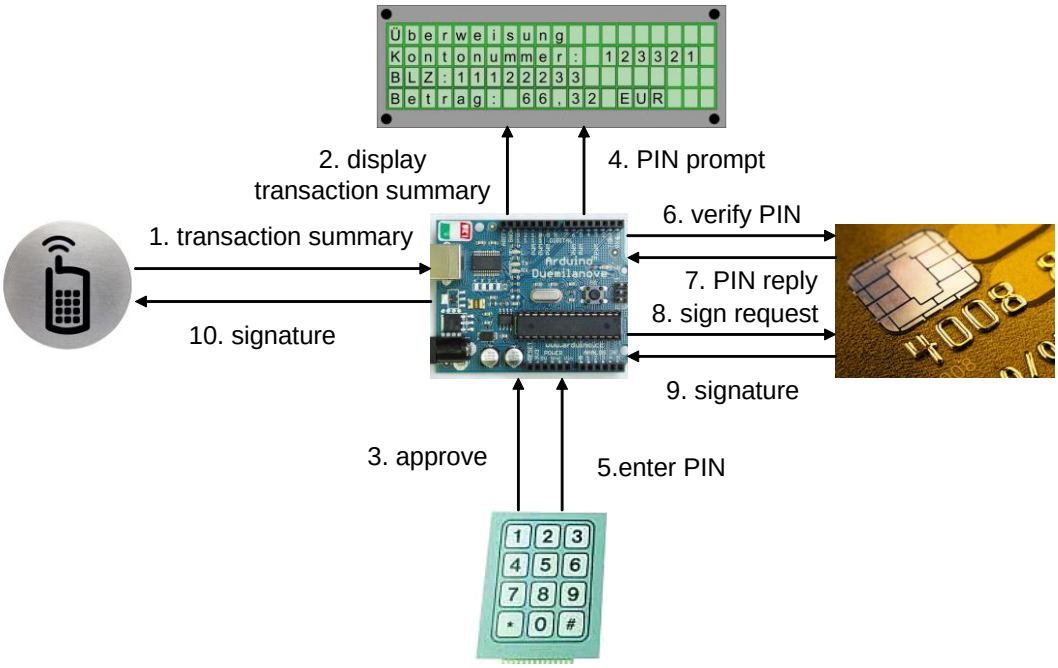

Figure 3. Interaction of the components to produce a signature for a transaction summary

4. VERIFICATION

4.1 Security Funnel: Reducing the Attack Surface

The workflow described in the previous section can be implemented in a class 3 chip card terminal for connecting a chip card to a PC. A class 3 chip card terminal has a chip card slot, display, and key pad. Class 3 chip card terminals are programmable, so you can implement a given workflow and chip card protocol. An attacker can compromise the security of the chip card terminal if he can attack either the program or an update mechanism of the card terminal's firmware.

Our goal is to use commercial off-the-shelf components, such as the ones described above, and reduce the trusted computing base as much as possible, to reduce the attack surface for a potential compromise. The code in the security device uses a minimal subset of the features of each component, which further reduces the attack surface.

The size of the program code to implement the workflow can be kept small if functionality is limited to the absolute minimum. So, in addition to reducing the hardware components' functionality, we reduce the functionality of the program as well, which reduces the attack surface even further. We call this the *Security Funnel*.

The Security Funnel depends on a correct implementation of the program, so that no undesired functionality is accessible which could be abused to subvert the integrity of the security device. Pfleeger and Lawrence Pfleeger (2006) classify the root cause of program flaws that lead to security issues as follows:

1. Buffer overflows, where memory allocated for data read from the environment is limited but the program allows for larger or unlimited length input. Overflowing a buffer overwrites other data in memory and in extreme cases can lead to an attacker being able to execute arbitrary code, thus subverting the security mechanisms of the program.
2. Incomplete mediation, where data read by the program is subject to interpretation but the interpreter fails to detect malformed data and continues processing it with undefined results.
3. Serialization flaw, where the result of a computation depends on the correct ordering of operations. Race conditions and time-of-check-time-of-use flaws are subclasses of that flaw that result in security issues, such as references to objects that can be modified by the attacker after access to the object has been checked by the program.

The following functions need to be implemented in the security device:

1. One protocol message to communicate the transaction summary from the smartphone to the security device, and two protocol messages to communicate back success plus the digital signature, or an error code for failure.
2. A maximum length for transaction summary and digital signature.
3. A predefined character encoding (e.g. a subset of ISO 8859-15 without control codes) of the transaction summary and a check that the transaction summary does not contain illegal characters.
4. One method to write a string to the display, encoded in the character encoding of #3 above, and a clear display method.
5. One method to read a twelve-key PIN pad.
6. Two protocol messages: one according to ISO 7816-4 for PIN verification and one according to ISO 7816-8 for digital signatures.

The protocol between the phone and the security device follows ISO 7816-8 (ISO, 2004) for signature messages, and the status messages are taken from ISO 7816-4 (ISO, 2005). Both standards are well established in the chip card environment. The ISO 7816-8 APDUs are mapped onto a sequence of I^2C messages and reassembled on the receiving end. The code must verify that an overly long APDU cannot overrun the buffer reserved for the APDU. The APDU header must be checked to confirm that the length byte matches the actual length of the transaction summary and whether the APDU header actually represents a sign command. If the APDU fails any one of these tests then a failure code is sent back to the smartphone.

The next step is to verify that the transaction summary does not contain any illegal characters. This is a simple loop that iterates over the length of the message in the buffer. The length has already been verified to be within the limits of a proper APDU so no buffer overflows can occur in this step.

The Nokia 5110 LCD display offers several libraries (Arduino Playground, 2015) that all need to define their own bitmaps for each character. For the prototype a version of 7-bit US ASCII was used. After the character set check in the previous step, all characters are within the range supported, and so an access to the table of bitmaps always returns a defined bitmap

pattern. The bit map patterns need to be visually verified for the correct shape and readability on the LCD.

Reading the PIN pad involves reading the voltage off a pin of the micro controller and converting it to the corresponding character. An Internet attacker has no way to modify the characters entered or the number of characters. Still, the program code should be robust and must be able to cope with a user typing in too many characters.

Attaching the chip card to the micro controller requires a library to control several pins, to control the card and communicate with the card, and to send APDUs to the card and receive the answer. To control the card a third party library available on Sourceforge (2015) is used. That library transmits a sequence of bytes to the card and reads the response. The sequences of bytes are an APDU for sending the PIN to the card for verification and an APDU that contains the sign command and the transaction summary to be signed. The PIN in the former APDU has been read from the user and checked by the micro controller. The latter APDU could have been manipulated by an attacker but previous checks ensure that the transaction summary meets length restrictions and does not contain illegal characters. The sending code does not perform any interpretation on the byte sequence sent, so there is no risk of incomplete mediation flaws in the program.

Summing up, we see that the only place in the micro controller program where data is interpreted is the the APDU header check of the sign message sent by the smartphone and hence this is the only place where an incomplete mediation flaw could be introduced. Reducing the protocol to the bare minimum and not allowing any options, the code can be reduced to a simple comparison of the header bytes. Serialization flaws cannot be introduced because after checking the header and checking for illegal characters, the contents of the transaction summary do not change the execution path of the program in the micro controller anymore. The risk of buffer overflow has been discussed with presentation of the individual steps of the micro controller program. The design of the program has reduced the attack surface significantly, and display, key pad, and chip card and their supporting libraries do not have to be part of the trusted computing base; moderate requirements that they work properly when fed properly-formatted data is sufficient.

4.2 Verification of Arduino Sketches with Frama-C

Frama-C is a tool-suite that integrates various proof techniques for program correctness in a plugin-based framework (Correnson et. al., 2015). Frama-C targets the C programming language and a behavioral interface specification language called ACSL is used to annotate source files. Those annotations live inside C comments so annotated files can be processed with a standard C compiler (Baudin et. al., 2015a).

Arduino sketches, the source files produced by the Arduino IDE, are C++ code where some boilerplate code is added by the Arduino IDE before the sketch is compiled by a C++ compiler. On the other hand, our Arduino code does not use object oriented features except for calling methods on objects provided by Arduino libraries. C++ method calls can be converted into C functions that take an additional parameter, a pointer to the object's state. A simple preprocessor macro is sufficient to reconstruct the original C++ code. For example, `wire.read()` is transformed into `wire_read(wire)`, where `Wire` is the class of `wire`. The idea to prove C code even though Arduino sketches are C++ code was inspired by Burghardt et. al. (2015) who show proofs of C++ STL algorithms implemented in C.

An Arduino sketch contains at least two functions, `setup` and `loop`. The former allocates resources and defines the I/O ports where peripheral such as a display are attached. The latter is run in an infinite loop that carries out the actual task of the security device. The precondition has been established by the setup function, and it must be proved that, at the end of the loop function, the device is in a state ready for the next round, i.e. tan invariant is preserved by the loop function. For the security device the invariant is the assignment of the I/O ports to peripheral devices and that the input buffer is allocated.

In the following two subsections the most critical parts of the code are shown with their specification and issues proving those functions correct are discussed:

1. Reading the message from the smartphone and making sure that the message does not overflow the buffer.
2. Check the message read from the smartphone for illegal characters and send it to the display.

4.3 Verify Input Buffer does not overflow

When the smartphone sends a message to Arduino, we must ensure that the message meets the size restriction and thus cannot overflow the receive buffer. The receive buffer is a globally allocated character array of fixed size that is written by the I^2C handler defined in function `receiveEvent`. Figure 4 shows function `receiveEvent` with the annotations that are necessary to verify the code. Two annotations are required:

1. A function contract that specifies preconditions and assignments (side effects) and
2. A loop variant to verify termination of the loop and assignments done by the loop.

```
/*@ requires \valid(MESSAGE_TO_SIGN+(0..(MESSAGE_TO_SIGN_SIZE-1)));
  @ requires howMany > 0;
  @ behavior good:
  @ assumes howMany < MESSAGE_TO_SIGN_SIZE;
  @ assigns *(MESSAGE_TO_SIGN+(0..howMany-1));
  @ assigns NEXT_MESSAGE_ID_TO_BE_SENT;
  @ behavior bad:
  @ assumes howMany >= MESSAGE_TO_SIGN_SIZE;
  @ assigns \nothing;
 */
void receiveEvent(int howMany) {
  if (howMany >= MESSAGE_TO_SIGN_SIZE)
    error();
  else {
    /*@ loop invariant 0 <= i <= howMany;
      @ loop assigns MESSAGE_TO_SIGN[0..howMany-1];
      @ loop assigns i;
      @ loop variant MESSAGE_TO_SIGN_SIZE-i-1;
     */
    for (int i = 0;
         (i < howMany) && (0 < wire_available(wire));
         i++)
           MESSAGE_TO_SIGN[i] = wire_read(wire);
    NEXT_MESSAGE_ID_TO_BE_SENT = MESSAGE_TO_SIGN[0];
  }
}
```

Figure 4. Read message from smartphone: Code and annotations.

The wire library contract for `receiveEvent` specified that `howMany` contains the number of bytes that are available on the I^2C for read. If the number of bytes would overflow the buffer, the security device needs to be reset, which is done by the `error` function. Otherwise all characters are read sequentially and stored in the global buffer (`MESSAGE_TO_SIGN`).

The full specification of the function contract contains two behaviors: one for the good case where the message meets the size restriction and one for the error case where the device will be reset. The contract as shown in Figure 4 can be verified by the weakest precondition plugin of Frama-C (Baudin et. al., 2015b) and all proof obligations are discharged by the built-in automated decision procedure.

For a loop, a termination proof must be provided by specifying a term (variant) that will strictly decrease each loop iteration and the loop will terminate once zero is reached. Further, assignments of variables, global (`MESSAGE_TO_SIGN`) and local (`i`), and a loop invariant are specified.

All proof obligations can be discharged by the built-in automatic decision procedure or the alt-ergo prover.

4.4 Verify Display Code

The message coming from the smartphone must be checked for illegal characters, i.e. ASCII characters that are not printable or characters out of range. Only strings with characters within the range of the ASCII characters SPACE and DEL must be passed to the `display_string` function. The display function does not check the validity of the characters, so the calling function must ensure that all characters are within range. The set of legal characters is a subset of the range supported by the `display_string` function, in fact the DEL character is the only character not permitted in strings received from the smartphone.

The input buffer `MESSAGE_TO_SIGN` contains the message sent by the smartphone between start and end and a check at the application protocol level guarantees that the next byte is zero. In other words the input buffer contains a C string that starts at position `start`. If the protocol is changed later and the string is no longer terminated by a zero byte then the respective proof obligation cannot be discharged anymore. Developers can then fix the issue.

Before the message can be displayed it must be checked for illegal characters. If `validate_message` finds an illegal character then `error` is called and the security device is reset.

```
/*@ requires start <= end;
  @ requires \valid(MESSAGE_TO_SIGN+(start..end+1));
  @ requires MESSAGE_TO_SIGN[end+1] == '\0';
 */
void show_message(int start, int end) {
  if (validate_message(MESSAGE_TO_SIGN, start, end)) {
    display_clear(display);
    display_string(display, MESSAGE_TO_SIGN+start);
  } else {
    error();
  }
}
```

Figure 5. Display message: Code and annotations

A contract for function `show_message` and an implementation is shown in Figure 5. The function contract specifies that the range of `start` and `end` must be within the bounds of the input buffer and the string terminator character must be present. In a valid range `end` must be at least `start` or greater. The rest of the contract is elided for clarity.

Function `validate_message` iterates over the characters in the array and returns `0` (false) immediately when an illegal character is found and `1` (true) if all characters are legal. This is implemented as a simple loop.

```
/*@ requires start <= end;
  @ requires \valid(msg+(start..end));
  @ assigns \nothing;
  @ behavior good:
  @ assumes \forall integer k; start <= k <= end ==> ' ' <= msg[k] <=
'~';
  @ ensures \result == 1;
  @ behavior bad:
  @ assumes \exists integer k; start <= k <= end ==>
                    ' ' > msg[k] || msg[k] > '~';
  @ ensures \result == 0;
 */
int validate_message (char *msg, int start, int end) {
  /*@ loop invariant start <= i;
    @ loop assigns i;
    @ for good: loop invariant \forall integer k; start <= k < i ==>
                          ' ' <= msg[k] <= '~';
    @ loop variant end-i;
  */
  for (int i = start; i <= end; i++)
    if (msg[i] > '~' || msg[i] < ' ' ) {
      return 0;
    }

  return 1;
}
```

Figure 6. Validate message: Code and annotations.

The function contract for `validate_message` shown in Figure 6 specifies that the range is well-formed and that the input buffer is backed by allocated memory. Two behaviors have been specified, one for the case where all characters within the range are legal and one for the case where there is at least one illegal character in the range. The loop behavior specifies a general invariant involving the iteration variable `i` and also that `i` is assigned in the loop. The variant to prove termination is straightforward. The invariant holds in the good case and is required to establish the assumption of the good case.

4.5 Summary

Looking at the two sample proofs we see that the contract specification is larger than the actual code. But the specification also contains more information. In fact, in `validate_message` it looks like the specification would be sufficient to generate the

actual code from it. In 4.4 we pointed out that the specification documents the fact that the code relies on the protocol placing a zero byte after the message and a proof obligation will fail to be discharged once the implementation of an updated protocol violates that assumption.

Specifying the behavior in a different language forced us to think about the implementation from a different perspective. Frama-C helped uncover errors and omissions in both the implementation and the specification.

5. RELATED WORK

The Secoder (Zentraler Kreditausschuss, 2011) is a class 3 card terminal specified for use with FinTS. Secoder is the only solution presented in a survey (IT-Sicherheit.de, 2015) of security solutions offered by German banks. Secoder is attached to a personal computer via USB. The user is presented a summary of the transaction similar to our transaction summary and the chip card creates a signature only after approval. The transaction summary, however, is generated on the Secoder, i.e. the Secoder contains an interpreter for FinTS formatted XML-messages. The presence of an interpreter adds significant software complexity to the Secoder, and changes in FinTS require either a software update or even a replacement Secoder if the update function was not implemented. The Kobil KAAN TriB@nk implementation of a Secoder offers "offline signatures" where the Secoder does not need to be connected to the personal computer.

An offer similar to Secoder is the "DigiPass 875 Smart card Reader for mobile devices" (VASCO, 2014). The DigiPass 875 offers digital signatures, Sm@rtTAN, and other proprietary schemes. The device allows firmware updates and supports a variety of security protocols and communication protocols. Hence the firmware is more complex and thus it is more difficult to verify correctness of that firmware.

Fraunhofer developed a "Trusted Pocket Signer" (TruPoSign) that is a PDA-class device for digital signatures (Hartmann and Eckstein 2004). TruPoSign communication with a personal computer is wireless. The device shows the document on a trusted display following a "what you see is what you sign" paradigm. TruPoSign supports multiple document formats and all supported formats are interpreted in the software of the device. An update function for new document formats is offered. All these features must be implemented according to the specification of those document formats to guarantee that the user really sees all that she is about to sign. Especially in the presence of an update function it is important to know which software version was used when a document was signed and that software version must then be reconstructed if the user claims to not have signed the document.

The IBM Zone Trusted Information Channel (ZTIC) aims to secure Internet banking by intercepting the HTTP traffic to the bank (Weigold, 2008). ZTIC attaches to a personal computer by USB and then acts as a Web Proxy and terminates the TLS connection to the bank. The user enters bank transactions through the browser and ZTIC parses the HTTP messages exchanged with the bank and displays transaction summaries in a small display. The user can approve or reject the transaction. ZTIC must follow the entire conversation between the bank and the user's browser. When the protocol changes ZTIC can be updated online. To support multiple banks or applications other than online banking ZTIC must be updated with a protocol interpreter for the new applications. ZTIC relies on interpreters, which adds to the complexity of the software. Moreover, ZTIC does not support digital signatures.

chipTAN or cardTAN systems receive the transaction summary through flickering on the personal computer screen. A chip card computes a transaction number that the user then enters into the computer to authorize the transaction. The transaction number is much shorter than a digital signature because the user cannot be expected to copy a full-strength digital signature into the browser. We are not aware of any such device that has been formally verified for correctness. In some cases customers are told that the device is secure because it is not connected to the Internet (see for example Postbank, 2015). This is not true, because the device receives a message from the customer's PC and vulnerability against specially crafted messages could exists. To our knowledge, however, no such case has been reported yet.

6. CONCLUSION

With the security device presented above and digitally signed transaction summaries, a higher level of security in online banking is achieved. The digital signature protects the customer but also offers non-repudiation for the bank.

Our current prototype does not fit the form factor of "The Other Half" and to accommodate a chip card contact unit it will need to be larger than a standard "The Other Half". The security device can only be used with smartphones that offer I^2C connectivity, though other communication methods such as USB or Bluetooth can be implemented.

We showed the formal correctness of the most critical functions of the micro controller program. This analysis should be extended by a proof that the security device ensures the correct sequence of all actions: read a message from the smartphone, display, user approves, send to card for signature, and send the signature back to the smartphone. The Aoraï plugin for Frama-C could be used for that proof.

In the future we envision that signed transaction summaries are applied to other business areas where the security relevant information of a transaction can be summarized in a transaction summary.

ACKNOWLEDGEMENT

We wish to thank Würth Electronic of France for donating chip card connecting units to the project.

Part of this work was done during my sabbatical stay at Friedrich Alexander University Erlangen-Nuremberg. I'd like to thank Felix Freiling, Chair of IT-Security Infrastructures and his team for offering me an inspiring place to work and valuable input to the paper.

REFERENCES

Arduino Playground, 2015. Philips PCD8544 (Nokia 3310) driver. Retrieved on March 6th, 2015 http://playground.arduino.cc/Code/PCD8544

BHF Bank Aktiengesellschaft, 2015. Conditions for OnlineBanking, retrieved on March 6th, 2015: https://www.bhf-bank.com/w3/imperia/md/content/internet/financialmarketscorporates/conditions__online_banking.pdf

Baudin, P. et. al., 2015a. *ACSL: ANSI/ISO C Specification Language, Version 1.9 – Sodium-20150201*, CEA LIST and INRIA, Orsay, France.

Baudin, P. et. al., 2015b. *WP Plug-in Manual, Version 0.9 for Sodium-20150201*, CEA LIST, Software Safety Laboratory, Saclay, France.

Burghardt, J. et. al. 2015. *ACSL By Example, Towards a Verified C Standard Library*. Fraunhofer FOKUS research report.

Correnson, L. et. al., 2015. *Frama-C User Manual, Release Sodium-20150201*, CEA LIST, Software Safety Laboratory, Saclay, France.

Die Deutsche Kreditwirtschaft (editor), 2014a. *FinTS Financial Transaction Services – Schnittstellenspezifikation: Messages Multibankfähige Geschäftsvorfälle*, Version 4.1.

Die Deutsche Kreditwirtschaft, (editor) 2014b. *FinTS Financial Transaction Services – Schnittstellenspezifikation: Messages Multibankfähige Geschäftsvorfälle für den Zahlungsverkehr Inland*, Version 4.1.

Die Deutsche Kreditwirtschaft (editor), 2014c. *FinTS Financial Transaction Services – Schnittstellenspezifikation: Security, Sicherheitsverfahren HBCI (inklusive Secoder)*, Version 4.1.

Die Deutsche Kreditwirtschaft (editor), 2014d. *FinTS Financial Transaction Services – Schnittstellenspezifikation: XML-Syntax*, Version 4.1.

European Union Agency for Network and Information Security (ENISA), 2012. Flash note: EUcybersecurity agency ENISA; "High Roller" online bank robberies reveal security gaps. Press release, July 5th, 2012.

European Union Agency for Network and Information Security (ENISA), 2013. eID Authentication methods in e-Finance and e-Payment services. ENISA Report, Heraklion, Greece.

Hartmann, M. and Eckstein, L., 2004. TruPoSign – a trustworthy and mobile platform for electronic signatures. *Securing electronic business processes: Highlights of the Information Security Solutions Europe 2003 Conference*, Vieweg, Wiesbaden, 2004, pp. 97-107.

Hehn S., 2009. *Spezifikation einer Formalen Sprache zur benutzerfreundlichen Verifikation von FinTS Onlinebanking Transaktionen.* Thesis (Diplomarbeit), Technische Hochschule Nürnberg, Germany.

ISO, 2005. *ISO/IEC 7816-4, Identification cards – Integrated circuit cards – Part 4: Organization, security and commands for interchange*, Second edition 2005-01-15.

ISO, 2004. *ISO/IEC 7816-8, Identification cards – Integrated circuit cards – Part 8: Commands for security operations*, Second edition 2004-06-01.

IT-Sicherheit.de, 2015. Sicherheit beim Online-Banking. Retrieved on April 29th, 2015, https://www.it-sicherheit.de/ratgeber/it_sicherheitstipps/online_dienste_sicher_nutzen/online_banking/

Jolla Ltd, 2015. The Other Half – Technical Specification Version 1.1.

Kobil, 2008. *Erster Secoder mit Online-/Offline-Funktion erhält ZKA-Zertifizierung.* Press release. 2008.

Langweg, H. and Schwenk J., 2007, Schutz von FinTS/HBCI-Clients gegenüber Malware. Proceedings of DACH Security 2007, Klagenfurt, Austria, pp. 227-238.

Pfleeger, C. P. and Lawrence Pfleeger, S., 2006. *Security in Computing*. Prentice Hall, Upper Saddle River, USA.

Postbank, 2015. Postbank chipTAN comfort. Retrieved on November 8th, 2015, https://www.postbank.de/privatkunden/pk_chiptan.html

Sourceforge, 2015. Smartcard Lib for Arduino compat. Boards. Retrieved on March 6th, 2015, http://sourceforge.net/projects/arduinosclib/

Weigold, T. et. al. 2008. The Zurich Trusted Information Channel – An Efficient Defence against Man-in-the-Middle and Malicious Software Attacks. *Trusted Computing- Challenges and Applications, Proceedings of First International Conference on Trusted Computing and Trust in Information Technologies (Trust 2008)*, Villach, Austria, pp. 75-91, Springer Verlag, berlin Heidelberg.

VASCO Data Security, 2014. DIGIPASS® 875 Smart card Reader for Mobile Devices. Data Sheet.

Zentraler Kreditausschuss, 2011. Secoder – Connected Mode Reader Applications, Version 2.2 Final Version.