An Observe-and-Detect Methodology for the Security and Functional Testing of Smart Card Applications

Germain Jolly, Sylvain Vernois and Christophe Rosenberger
Universite de Caen Basse Normandie, ENSICAEN, UMR 6072 GREYC, Caen, France

Keywords: Security, Analysis, Smart Card application, Observation, Detection, Evaluation, WSCT Framework.

Abstract: Smart cards are tamper resistant devices but vulnerabilities are sometimes discovered. We address in this paper the security and the functional testing of embedded applications in smart cards. We propose an original methodology for the evaluation of applications and we show its benefit by comparing it to a classical certification process. The proposed method is based on the observation of the APDU (Application Protocol Data unit) communication with the smart card. Some specific properties are verified as a complementary method in the evaluation process and allows the on-the-fly detection of an anomaly and the reasons that triggered this anomaly during the test. Here are presented two uses of this method: a simple use to illustrate the use of properties to verify an implementation of an application and a more complex illustration by applying the fuzzing method to show what we can obtain with the proposed approach, i.e. an analysis of an anomaly.

1 INTRODUCTION

While smart cards have invaded our daily lives, the question of the quality of the card application arises. EMVCo reports that 1.62 billion payment cards and 23.8 million terminals in circulation globally are based on EMV technology (EMVCo, 2013). The development of an application on a smart card is not an easy task (Rankl, 2007). Not clear enough specifications or having a too large number of important variables make the development of smart card implementation difficult. Testing aims at discovering anomalies in actual behavior of a system compared to what is expected. Moreover, it is not possible to test an application in a smart card over the full range of input variables. The full testing would cost too much in time and significant resources.

For functional and security testing, it is necessary to validate each step and each element of a smart card. It is unrealistic to use a secure and certified component with a poorly implemented program. The program must also be validated and certified. The definition of the criteria for evaluating security allowed product certification (Wallace et al., 1996). Recognition of these criteria allows mutual recognition between different structures and different countries. A certified product is tested not only by certification laboratories but also by the manufacturer during its development. The certification allows recognizing the level of security of a product by an external laboratory. EMVCo is providing the EMVCo Security Evaluation Process to its members (Alliance, 2011). It ensures a robust security foundation for smart card and related products. The Card Type Approval process allows testing compliance with the EMV specifications. Another document called Card Type Approval process provides an increased level of confidence. To summarize, EMVCo provides documents to test, evaluate and approve a smart card but also a list of Approved Security Evaluation Laboratories (EMVCo, 2012). For validation purposes, meaningful I/O sequences are generated.

While in the industrial field, evaluation is especially enabled by testing tools, academic research has proposed many different methods. There are several kinds of verification and validation (V&V) methods: control (technical notice, reviews), analysis (mathematical verification) or test (focus on input/output). Most of the academic research works consider white-box systems (known source code and secrets).

- Static analysis is an analysis method ((Distefano and Parkinson J, 2008), (Philippaerts et al., 2014) or (Ahrendt et al., 2005)) allowing to ensure that defined coding practices are being followed.
- The model checking is a method consisting of verifying if a model of an abstraction or a system follows the specifications ((Sabatier and Lartigue,

Jolly, G., Vernois, S. and Rosenberger, C.
An Observe-and-Detect Methodology for the Security and Functional Testing of Smart Card Applications.
DOI: 10.5220/0005682202820289
In *Proceedings of the 2nd International Conference on Information Systems Security and Privacy (ICISSP 2016)*, pages 282-289
ISBN: 978-989-758-167-0
Copyright © 2016 by SCITEPRESS – Science and Technology Publications, Lda. All rights reserved

1999)). The model checking is used to check all possible paths of the state machine defining an application on smart cards.

- The test method allows to discover errors at all levels of systems ((Philipps et al., 2003), (van Weelden et al., 2005)). We can also discern the structural test from the functional test. This method is the most used method in the industrial field. The main problem of the test is to build data input that will provide the higher test coverage.
- Fuzzing is a automatic generation of a large number of test cases ((Lancia, 2011), (Bekrar et al., 2012) or (Alimi et al., 2014).). The test is used to verify one specific path in the state machine of the application. This approach allows an automatic generation of test cases. The problem is that this method costs smart cards (as they are not scheduled to accept so many transactions). The main drawback of this approach is also the difficulty to fully understand the scenario that permits to generate an attack.

The presented method here allows both verification of the complete application on a smart card, or at least provides an accurate documentation of a detected anomaly. The method detects an anomaly relative to the expected behavior and provides accurate documentation of the reasons that triggered this error. The proposed method detects vulnerabilities through the observation of the APDU communications. It improves the documentation comparing to the results of known evaluation methods as the test method used to verify, validate and evaluate a smart card. We would like to mix the efficiency of the whitebox methods with the simplicity of the blackbox ones.

In the second section, we provide some necessary background on smart cards and their certifications. The methodology is defined in the third section. Section 4 deals with the results obtained by creating a proof of concept using the WSCT framework for the payment application and the second use of the method mixed with a fuzzing method.

2 BACKGROUND

We focus on the smart card application. For the payment, EMVCo provides EMV (Europay MasterCard Visa) specifications and the CPA (Common Payment Application) specification (Watanabe et al., 2006). Moreover, proprietary specification is added to develop a smart card application.

In this work, the MasterCard M/CHIP specification, an EMV-compliant specification for Master-

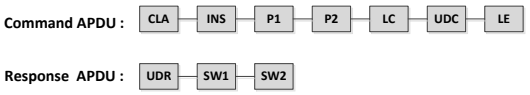

Figure 1: Communication between the terminal and the smart card.

card smart cards, is studied. The application contained on the chip can take several states (selected performed, GPO performed, ...). The evolution of the state of the application is allowed by sending commands APDU (select, get data, GPO, ...) and receiving responses APDU (9000, 6283, ...). A machine state is given by Mastercard to illustrate the M/CHIP application in its proprietary specification.

The payment application can pass from a state to another only by receiving and responding to a series of command/response pairs, this is the concept of application state. We cannot only consider the acceptance of a command and its response without considering its current state.

A terminal communicates with the smart card by sending commands and receiving responses. In the figure 1, we can see the structure of APDU commands and APDU responses according to ISO/IEC 7816. CLA indicates the type of a command, INS a specific command, P1 and P2 are parameters for a command, LC indicates the length of the UDC which is optional data and LE indicates the length of the expected data. Expected data, only contained in the response if LE is in the command, are the UDR. Finally, SW1 and SW2 are mandatory in the response and are the command processing status.

During the life of a product, we have to ask some questions (Radatz et al., 1990) :

- **Verification:** Are we building the product right ?
- **Validation:** Are we building the right product ?
- **Evaluation:** How can we interpret the verification and validation ?
- **Certification:** Are the evaluation results approved by a specific firm?

To evaluate and validate the product, several tests are made during the different steps of the life of the smart card (Rankl and Effing, 2010). All aspects of the card (application, chip and personalization data) must be validated to ensure a secure product. Once designed and validated, the card is certified. The certification allows recognizing the level of security of a product by an external laboratory.

According to ISO 8402 (for Standardization, 1994), quality is defined as the totality of features and characteristics of a product or service that bear on its ability to satisfy the stated or implied needs. The

two most important attributes of quality are robustness and freedom from defects and errors. Concerning this field, the test is mostly used for the evaluation of a smart card and the evaluation of a software is called software evaluation. The purpose of testing is always to find defects and errors, and the entire testing process must be oriented towards this purpose. Tests can never be complete, since it is never possible to fully work through all possible permutations. We obtain a desired level of quality. As tests can never fully cover everything, it is difficult to define a completion criterion for test development. The rule of thumb is that the same amount of effort should be spent on test development as on product development (i.e. try to achieve full requirements coverage with the tests). The minimum level of instruction coverage should be in the 95% range, and the branch coverage should be in the 80% range (Rankl and Effing, 2010).

3 PROPOSED EVALUATION METHODOLOGY

3.1 Objectives

From a certified card, we get a collection of properties, local and required behaviors (first, this collection is created manually but we are working on this topic to automatize the generation). These properties are used by the Observer to determine if a card application is correct. In order to reach this goal, a modified smart card, containing a modified smart card application of the certified one, is used with a terminal. In our case, it is a terminal doing a nominal EMV transaction. The Observer therefore takes as input the communication between the tested card and terminal and the collection of properties to check. Note that the verification of the state machine is possible. During the communication, the Observer recovers transmitted data and detects an anomaly on the fly. Finally, a report is created.

3.2 Property Language

To detect abnormal behaviors and give some documentation about the reasons that triggered those behaviors, we define the properties that correspond to required and local behaviors of an application. The section defines the property language.

3.2.1 Basis

Unlike formal methods such as model checking or proof of theorem approaches, we are checking properties while observing the transmitted communication during the transaction. In fact, in studies like (Posegga and Vogt, 1998) (Sabatier and Lartigue, 1999) (Lanet and Requet, 2000) (Jacobs et al., 2004) or (Haneberg et al., 2007), the verification needs a formal model or the access to the source code. With our language, we can define the required behavior (local and global) using only the transmitted data.

3.2.2 Definitions

The language allows us to define both a state machine, allowing to follow the actual behavior of the application and also to define the properties as defined in (Jolly et al., 2014).We can follow the global behavior of the smart card but also detect local changes in the application behavior.

To define this behavior, we have two types of elements. First, we have the element representing the APDU communication. The APDU data is defined by :

- **Instruction:** represents the CLA and INS bytes of a command APDU
- **Parameters:** represents the P1and P2 bytes of a command APDU
- **Status:** represents the SW1 and SW2 bytes of a response APDU

The second type is the logical relation between two or more elements. The logical relation between APDU are defined by:

- **And:** elements are all true
- **Or:** one element is true
- **Nor:** elements are all false

In addition to the previous elements, defining a machine state, i.e. a global behavior of a smart card application, requires these tags :

- **Cardstate:** the name of the state of the application used to follow the actual behavior of the application.
  - name: name of the cardstate
  - default: the initial state of the machine state
- **Transition:** allows modifying the actual state of the smart card application
  - name: name of the transition
  - from: previous cardstate
  - to: next cardstate
- **From / To:** to construct the transitions between cardstates
  - nom: name of the begining cardstate

For the definition of property, we need the following elements to define properties :

- **Property:** required and local behavior of the smart card
  - name: name of the property
  - explanation: description of the property
- **Step:** indicates the order of elements

For each of these elements, we can work on transmitted data (i.e. UDC or UDR data). We added tags to recover the plain transmitted data. If we recover TLV data, we should be able to work with only one or several specific values.

- **Cdata:** allows to recover the plain udc data
- **Rdata:** allows to recover the plain udr data
- **Tag:** allows to verify the value of a particular TLV data, contained in a response APDU
  - name: name of the tag (example : 0x9F27)
  - value: value of the TLV data (example : 0x40)
- **Mask:** allows to apply a mask on the values
- **Call:** allows to call a function to manage the data for a particular property (example : checking of the validity of a specific data). This last element permits to improve the verification by adding some functionnality to the properties and particularly on the studied data that can be encrypted and can be added to every step of a property.

3.3 Development Tools

Many tools exist to help verifying and validating smart cards. Here are several independant or academic tools able to work with smart cards:

- SmartCard Framework, an independant tool to easily develop a smart card application (Rouit, 2011)
- Open SCDP, a collection of tools for the development, test and deployment of smart card (CardContact, 2012)
- Javaemvreader, a tool to communicate with, and read data from an EMV smart card (Sasc, 2014)
- PCSC_sharp, an independant tool in order to work with smart cards (Mueller, 2012)
- CardPeek, an independant tool to read the contents of smart cards (Pannetrat, 2010)
- OPAL, a library for developing tools for Smart Card research and education (Bkakria et al., 2011)
- WSCT, a framework able to easily explore and work with smart cards (Vernois and Alimi, 2010)

	Funct.*	Doc.	Maint.
SmartCard Framework	⋆	⋆⋆	⋆
OpenSCDP	⋆⋆	⋆⋆⋆	⋆⋆
Javaemvreader	⋆⋆	⋆⋆	⋆
PCSC_Sharp	⋆	⋆	⋆
OPAL	⋆⋆	⋆⋆	⋆⋆
CardPeek	⋆	⋆⋆	⋆⋆⋆
WSCT	⋆⋆⋆	⋆	⋆⋆⋆

* Functionalities contain observation, extension and replay mechanisms. The more there are stars like ⋆, the more the tool is interesting for this feature for the Functionnalities, Documentation and Maintenance criteria.

3.4 Discussion

WSCT (Vernois and Alimi, 2010) is a framework developed by Sylvain Vernois et al. at Greyc for several years. It is written in C# and allows to work with smart cards, e.g. for exploration and finding fault on smart cards. The two main purposes of this tool are to provide :

- an **object-oriented API** to access a smart card reader.
- an **evolutive GUI** (with creation of plugins) to manipulate different types of smart cards.

Using this framework, we have easily created two plugins. Indeed, WSCT framework has an easy extension mechanism. We choose to create seperate plugins to emphasize that the Observer platform is not linked to the terminal. The Observer can be used on every kind of terminal (nominal or test transaction).

We used this tool to develop the proposed property language in order to observe communication, thanks to the possibilities of the framework, and detect anomalies, thank our language.

4 ILLUSTRATION

Here, we are showing two studies done with this observation tool. The first deals with EMV transaction, i.e. we are verifying an existing implementation of the EMV transaction. The second is about the use of the properties to analyze Fuzzing'results on a PME implementation, an e-wallet application. We are focusing on the smart card application using a correct and non-malicious smart card reader. For each study, the reference to see the divergence with the correct

behavior is the collection of properties (a set of assertions based on the transmitted data between the certified smart card and the terminal) that can be partial or complete. In our study, we use specific, local and required behaviors in order to check a specific part of the application.

4.1 Observation of the EMV Transaction

4.1.1 View of the Platform

Two plugins are needed for our method: one to communicate with the smart card and one to observe the communication and do the properties verification. We add an automat to follow the behavior of this smart card. We decided to add the machine state of the EMV application based on (Aarts et al., 2013). The inputs of the Observer are communication APDU and the selected properties to verify.

4.1.2 The Used Protocol

The protocol we used is based on three main points:

- the smart card application and its documentation. Using the documentation, we manually model the local behaviour. Also, it should be mentioned sooner that the properties are encoded with XML. We used our own EMV smart card application certified by a laboratory.
- the application to do the transaction because this method uses APDU communication between a terminal application and the smart card application. We used the WSCT framework to create an EMV explorer able to do the EMV transaction with the smart card application. (It can be simplified by the use of an extern terminal observed by the WSCT functionalities).
- the Observer, an application able to recover APDU communication and detect a bad behavior of the smart card application. It simply compares the transmitted APDU communication with the theorical behaviors defined in the xml file.

4.1.3 Observation of Smart Card Application

To define the machine state, we need two elements: ”cardstate” and ”transition”. The listing 1 shows a partial machine state of the payment application. We call ”selected” the state when the application is select (and so select performed) and ”initiated” when the GPO command (initiation part of the transaction) is performed.

Listing 1: Partial machine state of the payment application.

```
<machine>
 <cardstate name="Selected"/>
 <cardstate name="Initiated"/>
 <transition text="GPO">
  <from name="selected" dir="Down" />
  <to name="initiated" dir="Up" />
  <and>
   <instruction="0x80A8"/>
   <status="0x9000"/>
  </and>
 </transition>
 <transition text="else">
  <from name="initiated" dir="Right"/
      >
  <to name="selected" dir="Right"/>
 </transition>
</machine>
```

Figure 2 presents an illustration of the machine state observation of the payment application, which is implemented in the Observer plugin.

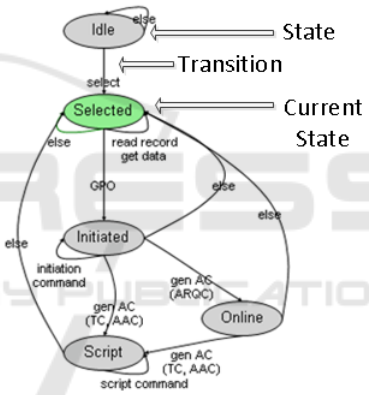

Figure 2: Observation of the machine state evolution.

4.1.4 Example of Property Detection

To illustrate our study, we take the example from (Jolly et al., 2014). The property P3 represents the behavior between the ”selected” and ”initiated” states using only one command. The property P3 is defined on three clock cycles, i.e. three pairs APDU command/response. If one wants to check the value of a TLV data, one could have added a TAG verification as previously defined. Here, we checked only the success or the failure of the GPO (Get Processing Options) command between two particular states of the application. The xml file in the listing 2 is the P3 property usable by the Observer.

Good behavior with three GPO commands:

$$P_3 : ((SW1(1) = 90) \; and \; (SW2(1) = 00) \; and \; (SW1(3) \neq 90) \;\; and \;\; (SW2(3) \neq 00) \; and \; (SW1(5) = 90) \; and \; (SW2(5) = 00)) \; or \; \overline{(INS(0) = A8) \; and \; (INS(2) = A8) \; and \; (INS(4) = A8)}$$

Listing 2: P3 property.

```
<property name="Third GPO">
 <step>
  <instruction="0x80A8" />
  <status="0x9000" />
 </step>
 <step>
  <instruction="0x80A8" />
  <nor>
   <status="9000" />
  </nor>
</step>
<step>
 <instruction="0x80A8" />
  <status="0x9000" />
</step>
</property>
```

To understand the scope of the detection properties method,here is a comparison of three possible behaviors of the application:

- **a)** The first case is the result of a successful and ordered test from the selection of the application to the use of Generate AC command. The tester knows the expected behavior of the application comparing to his test. The application responds appropriately to the known configuration of the test. The property is not violated.
- **b)** The second case presents an unusual refusal Generate AC command but also detection of the violation of the property P3, which explains the reason for the anomaly detected by the test. The property violation detection gives an improved documentation about the result of the test.
- **c)** The third case is correct behavior relative to the test result. But the property has been violated. In fact, the application contains an error but it's not critical comparing to the behavior of the application with the computed test.

4.2 Fuzzing Analysis of a PME Implementation

The same observation method can be used during a fuzzing session to complete the analysis of an implementation. The PME application is an electronic purse. It's a much more simple application than the EMV one.

4.2.1 Basis of the Study

A reference implementation, an own implementation and a set of scripts (one script is a possible transaction) are inputs of the evaluation system. (Items such an XML file representing the application and an XML file containing properties are generated from the reference implementation or associated documentation). The system sends the scripts to both applications and detects a difference between the implementation, i.e. an anomaly is detected. To analyze the reasons for this error, the system uses the violated properties observed by the tool.

The terminal sends the N transaction scripts to the two smart cards. if it detects a difference between our implementation and the reference one, an anomaly is detected. The Analysis tool is composed of three main modules:

- **A)** The Observation Module recovers APDU communication (we used a representation of the evolution of the application state as in figure 3 but it's informative information).
- **B)** The Detection Module detects a difference between two implementations of an application (the reference implementation and the tested one).
- **C)** The Analysis Module can give more information about a detected anomaly.

4.2.2 Experimental Results

To study the addition of the method on an evaluation method such as fuzzing, we have used electronic wallet application called PME application. In order to compute our tests, we used :

- 4 implementations: one is the reference implementation and the others contains errors (defective verification of balance during a debit operation, defective check of pin code and unmanaged blocking of the card when using false pin code)
- 2 scripts of transaction: these scripts allow to compute several different transactions in order to detect the presence of an anomaly (by checking the difference between the responses of the reference implementation and the responses of the tested implementation)
- 5 properties: we use a limited number of properties because we are showing the feasibility of the method on our own smart cards. Each property is a local and required behavior of the smart card application (given by the reference implementation which must be validated).

We may also use the documentation of an application to create these elements. Having a reference application is simpler for our study.

4.3 Explanation of an Example

This example explains an anomaly detection: the de-

fective blocking of the card if the user is testing too many pin code. Without blocking the application, the so-called brute force attack would be possible. The script therefore sends over n times a wrong pin code (n is the maximum number of false pin code defined by the application documentation). When the pin code is accepted by the tested smart card but not by the reference smart card, an anomaly is detected. We just need to look the observer to see the violated property to understand the reason of the anomaly. We have, in this example, three properties in the listing 3.

- The first checks that the blocking displayed by the error code is effective, i.e. it is not just a superficial blockage.
- The second ensures that the application sends an error message, equal to 9170, once the counter is zero, i.e. the smart card has processed the information correctly.
- The latest involves a control method. We observe two failed uses of the verify command. In this case, the difference between the two error codes is 1, i.e. the counter is decremented correctly.

Listing 3: Example of explanation by properties.

```
<properties>
  <property name="blocageOK">
  <step>
   <instruction instruction="0x9020"
       />
   <status status="0x9170" />
  </step>
  <step>
   <instruction instruction="0x9020"
       />
  <require>
    <status status="0x9170" />
    </require>
  </step>
</property>

 <property name="blockOneLeft">
  <step>
   <instruction instruction="0x9020"
       />
    <status status="0x9111" />
  </step>
  <step>
   <instruction instruction="0x9020"
       />
    <status status="0x9170"/>
    <nor>
     <cdata cdata="01020304"/>
    </nor>
  </step>
</property>

 <property name="block">
   <step>
     <instruction="0x9020" />
  <nor>
     <status="0x9000" />
  <nor>
   </step>
   <step>
     <instruction="0x9020" />
  <nor>
     <status="0x9000" />
  <status="0x9170" />
  <nor>
  <require>
   <call method="control" />
  </require>
 </step>
</property>
</properties>
```

4.4 Advantages and Improvements

The use of properties allows verifying an implementation as an efficiant and stand alone evaluation method. The efficiency of this method is due to the efficiency of the description of the application, i.e. a full collection of properties must be defined. We need also have a full documentation of the application or a certified implementation of the application. The method can be used with another evaluation method as fuzzing in order to give more documentation about an error detecter by the fuzzing report.

5 CONCLUSIONS

This paper has defined a language property related to the verification approach exposed in this paper. The two main principles of this method are: the communication observation and the property violation detection. Using this language, we have defined the machine state corresponding to our studied smart card application to observe the global behavior. And more interesting, we have defined properties to detect anomalies and the reasons that triggered those anomalies. We have created the observe-to-detect tool using the WSCT framework. Finally, the violation detection allows us to give more documentations about a detected error or an undetected anomaly by a test plan. We have been able to validate our method with the creation of our own smart card application and our own intern terminal. Moreover, the use of this method mixed with a known evaluation method as the fuzzing shows complementary information about an anomaly. It is easier to explain fuzzing results, i.e. find the reasons of a detected error on a smart card application as the PME application.

SCITEPRESS SCIENCE AND TECHNOLOGY PUBLICATIONS

The perspectives of this work is to automatize the creation of properties for a specific application as an EMV payment application. Indeed, creating properties manually allows us to validate our method with suitable properties. With an automatic generation, we will be able to have a complete and user configurable collection.

REFERENCES

Aarts, F., De Ruiter, J., and Poll, E. (2013). Formal models of bank cards for free. In *Software Testing, Verification and Validation Workshops (ICSTW), 2013 IEEE Sixth International Conference on*, pages 461–468. IEEE.

Ahrendt, W., Baar, T., Beckert, B., Bubel, R., Giese, M., Hähnle, R., Menzel, W., Mostowski, W., Roth, A., Schlager, S., et al. (2005). The key tool. *Software & Systems Modeling*, 4(1):32–54.

Alimi, V., Vernois, S., and Rosenberger, C. (2014). Analysis of embedded applications by evolutionary fuzzing. In *High Performance Computing & Simulation (HPCS), 2014 International Conference on*, pages 551–557. IEEE.

Alliance, S. C. (2011). Card payment roadmap in the united states: How will emv impact the future payments infrastructure? *White Paper. February*.

Bekrar, S., Bekrar, C., Groz, R., and Mounier, L. (2012). A taint based approach for smart fuzzing. In *Software Testing, Verification and Validation (ICST), 2012 IEEE Fifth International Conference on*, pages 818–825. IEEE.

Bkakria, A., Bouffard, G., Iguchi-Cartigny, J., and Lanet, J.-L. (2011). Opal: an open-source global platform java library which includes the remote application management over http. In *e-Smart 2011*.

CardContact (2012). http://www.openscdp.org/.

Distefano, D. and Parkinson J, M. J. (2008). jstar: Towards practical verification for java. In *ACM Sigplan Notices*, volume 43, pages 213–226. ACM.

EMVCo (2012). https://www.emvco.com/approvals.aspx.

EMVCo (2013).

for Standardization, I. O. (1994). *ISO 8402: 1994: Quality Management and Quality Assurance-Vocabulary*. International Organization for Standardization.

Haneberg, D., Grandy, H., Reif, W., and Schellhorn, G. (2007). Verifying smart card applications: an asm approach. In *Integrated Formal Methods*, pages 313–332. Springer.

Jacobs, B., Marché, C., and Rauch, N. (2004). Formal verification of a commercial smart card applet with multiple tools. In *Algebraic Methodology And Software Technology*, pages 241–257. Springer.

Jolly, G., Vernois, S., and Lambert, J.-L. (2014). Improving test conformance of smart cards versus emv-specification by using on the fly temporal property verification. In *Recent Trends in Computer Networks and Distributed Systems Security*, pages 192–201. Springer.

Lancia, J. (2011). Un framework de fuzzing pour cartes à puce: application aux protocoles emv. In *Symposium sur la Sécurité des Technologies de lInformation et des Communications (SSTIC)*, page 82.

Lanet, J.-L. and Requet, A. (2000). Formal proof of smart card applets correctness. In *Smart Card Research and Applications*, pages 85–97. Springer.

Mueller, D. (2012). https://code.google.com/p/pcsc-sharp/.

Pannetrat, A. (2010). https://code.google.com/p/cardpeek/.

Philippaerts, P., Mühlberg, J. T., Penninckx, W., Smans, J., Jacobs, B., and Piessens, F. (2014). Software verification with verifast: Industrial case studies. *Science of Computer Programming*, 82:77–97.

Philipps, J., Pretschner, A., Slotosch, O., Aiglstorfer, E., Kriebel, S., and Scholl, K. (2003). Model-based test case generation for smart cards. *Electronic Notes in Theoretical Computer Science*, 80:170–184.

Posegga, J. and Vogt, H. (1998). Byte code verification for java smart cards based on model checking. In *Computer SecurityESORICS 98*, pages 175–190. Springer.

Radatz, J., Geraci, A., and Katki, F. (1990). Ieee standard glossary of software engineering terminology. *IEEE Std*, 610121990:121990.

Rankl, W. (2007). *Smart Card Applications: Design Models for Using and Programming Smart Cards*. Wiley Online Library.

Rankl, W. and Effing, W. (2010). *Smart card handbook*. John Wiley & Sons.

Rouit, O. (2011). http://www.codeproject.com/articles/17013/smart-card-framework-for-net.

Sabatier, D. and Lartigue, P. (1999). The use of the b formal method for the design and the validation of the transaction mechanism for smart card applications. In *FM99Formal Methods*, pages 348–368. Springer.

Sasc (2014). https://github.com/sasc999/javaemvreader.

van Weelden, A., Oostdijk, M., Frantzen, L., Koopman, P., and Tretmans, J. (2005). On-the-fly formal testing of a smart card applet. In *Security and Privacy in the Age of Ubiquitous Computing*, pages 565–576. Springer.

Vernois, S. and Alimi, V. (2010). Winscard tools: a software for the development and security analysis of transactions with smartcards. *Norsk informasjonssikkerhetskonferanse (NISK)*.

Wallace, D. R., Ippolito, L. M., and Cuthill, B. B. (1996). *Reference information for the software verification and validation process*, volume 500. DIANE Publishing.

Watanabe, T., Howell, P., and Pugh, S. (2006). Easing emv: Emvco's new common payment application. *Card Technology Today*, 18(2):12–13.