# Generation of local and expected behaviors of a smart card application to detect software anomaly

Germain Jolly, Baptiste Hemery, Christophe Rosenberger

# Generation of local and expected behaviors of a smart card application to detect software anomaly

Germain Jolly*, Baptiste Hemery* and Christophe Rosenberger*
*Normandie Université, Caen, France;
UNICAEN, GREYC, F-14032 Caen, France;
ENSICAEN, GREYC, F-14032 Caen, France;
CNRS, GREYC, F-14032 Caen, France

***Abstract*—The electronic payment transaction involves the use of a smart card. A card application is a software, corresponding to standards and non-proprietary and proprietary specifications, and is stored in the smart card. Despite increased security with Europay Mastercard Visa (EMV) specifications, attacks still exist due to anomalies in the card application. The validation of the card application enables the detection of any anomaly, improving the overall security of electronic payment transactions. Among the different ways of validating a card application, we can use the verification of required behaviors. These behavior can be materialized as properties of commands sent by the terminal and responses from the smart card, using the Application Protocol Data Unit (APDU) from the ISO/IEC 7816 standard [1]. However, the creation of these behaviors is complicated. We propose in this article a way to automatically create such behaviors by using a genetic algorithm technique.**

## I. Introduction

According to Eurosmart [2], 7.23 billions smart cards were used for payment, transport, access and other uses worldwide in 2013. Theses smart cards are based on standard, such as ISO/IEC 7816, and specifications, such as the defined by Europay Mastercard Visa (EMV), which define security and interoperability for payment. In 2012, the number of EMV smart cards was 1.62 billions and the adoption rate was 44.9% [3].

In addition of these specifications, each smart card application has its own proprietary specification, e.g. VIS specification for Visa application and M/CHIP specification for MasterCard application. It allows the creation of theoretical secured smart card application.

However, before being used in everyday life, a smart card must be validated and certified. Several specific firms can verify and validate a product by evaluating the conformance of the product to the standards and specifications it follows [4].

We believe that this conformance evaluation might be realized by a software observer, listening the communication between the terminal and the smart card. If the communication violates some properties, the observer might declare the smart card non-certified. This requires being able to produce the said properties.

We present in this article an automated approach to produce properties. The generation uses genetic algorithm technique to generate random properties, and select the correct ones.

This remaining of the paper is organized as follow : section 2 presents the context of the study, i.e. the studied standards and specifications. Section 3 presents how smart cards are produced and evaluated throughout their life. Section 4 presents our approach concerning the automated generation of properties. Finally, we give some results in section 5, and conclude in section 6.

## II. Context of the study

### A. Electronic payment transactions

*1) Payment transaction specifications:* An electronic payment transaction [5], i.e. the transaction between a terminal and a smart card, allows a payment with the smart card. A payment transaction is composed of several transaction units. A smart card, which is a passive device, is powered by the terminal and automatically responds to it after receiving a command, according to the card application.

The communication, between a terminal and a smart card, is defined by the ISO/IEC 7816 standard [1], complemented by ISO/IEC 14443 for contactless payment transactions [6]. Additionally, Europay Mastercard Visa (EMV) specifications aim to facilitate the interoperability and acceptance of secure payment transactions [7]. These specifications are managed by the EMVco agency, which includes MasterCard, Visa, JCB international and American Express (Europay was absorbed by MasterCard). Moreover, each manufacturer of smart cards has its own specifications. For example, M/CHIP specification for MasterCard application or VIS specification for Visa application.

*2) ISO/IEC 7816: APDU communications:* Communications between terminals and the smart cards are based on APDU command-response pair, respectively called command APDU and response APDU, as shown in the figure 1

The terminal application, which is a software, sends command APDU and the smart card application, a program, automatically sends response APDU. A command APDU contains four compulsory bytes: the class (CLA), the instruction (INS) and two parameters P1 and P2. It then optionally contains data sent to the card, the LC and UDC fields, as well as maximal length of the expected result LE. The payment application on the smart card detect the type of the command (INS byte) and associated data (P1, P2 and UDC); and delivers an optional response data UDR, followed

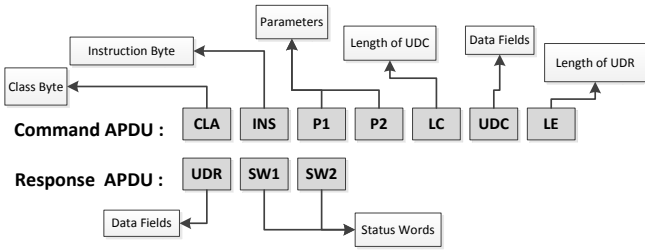



Fig. 1: Structure of APDU communication

by compulsory status word (SW1, SW2), that indicates a success or a failure of the command.

*3) Terminal view (EMV specification):* A payment is performed by several command-response pairs between a terminal and a card. Users, also known as cardholders, see three steps on the terminal screen: first, the amount of the transaction is showed; then, the user is verifying his pin code; finally, a success message or an error message appears on the terminal. Actually, there are more steps to do an EMV transaction. Figure 2 shows these steps.

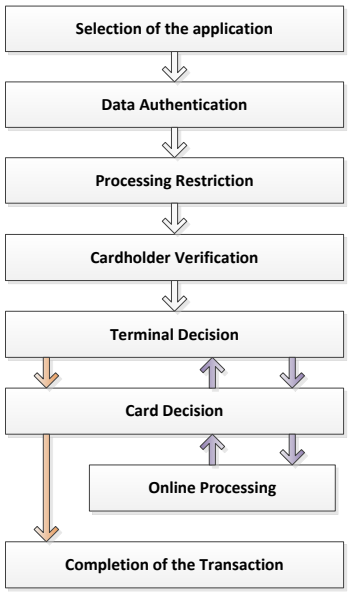



Fig. 2: Simplified EMV transaction

Each step of EMV processing, composed of several APDU commands/responses pairs, has a specific purpose:

- Application selection: selection of the desired application on the card;
- Data authentication: initiation of the transaction process and computation of data authentication to ensure the authenticity and the uniqueness of the card;
- Processing restrictions: checking the compatibility between the card and the terminal;
- Cardholder verification : verification of the carholder by asking for a signature or a pincode;
- Terminal decision : safety measure performed by the terminal to protect from fraud and decide for an online or offline transaction;
- Card decision : safety measure performed by the card to protect from fraud and decide for an online or offline transaction;
- Online processing : ask for an authorization of the user's bank's;
- Completion of the transaction : the card emits the payment if authorized.

*4) Smart Card view (proprietary specification):* To allow a payment transaction, a smart card may contain an application in conformance with EMV specifications. However, the smart card application design is completed by proprietary specifications. The smart card application can be seen as a finite-state machine. This theoretical finite-state machine is used by constructors to develop a payment application from the specifications (e.g. Mastercard provides the M/CHIP state machine[8] and Visa the VIS specification [9]).

### B. Transactions in danger

Despite extensive security reached by several phases of testing and certification, we can see that some attacks are known [10]. There are attacks either for contact or contactless smart cards. We can also discern invasive attacks (involving partial or total destruction of the card) and non-invasive attacks. Some attacks are exposed in [11], [12].

The more interesting attacks for our study aims the smart card application. The logical attacks use the weaknesses of the application, due to imprecise specification or inadequate development [13], to realize unauthorized payments.

- Skimming: a very thin element is added on a smart card reader to capture all transmitted data, e.g. card number. With the use of a camera, we can have enough data to pay on internet. [14].
- Yes card: is a partial copy of a smart card. However, when there is the PIN verification, a false cryptogram is sent to force the transaction without the right PIN code.
- Relay attack: we can relay the transaction and modify it, e.g. change the amount [15].
- Cambridge attack: contrary to the Yes Card, we don't check the PIN code. On one hand, the terminal thinks the card can only do signature verification and on the other, the smart card thinks the PIN code verification is performed [16].

## III. Evaluation of the smart card application

### A. Cycle of life of a smart card

In figure 3, we can see a simplified smart card cycle of life, based on the smart card handbook [17]. To evaluate and validate the product, several tests are made during the different steps of the life of a smart card [18]. All aspects of the card (application, chip, personalization data) must be validated

which ensures a secure product. Once designed and validated, the card is said certified.

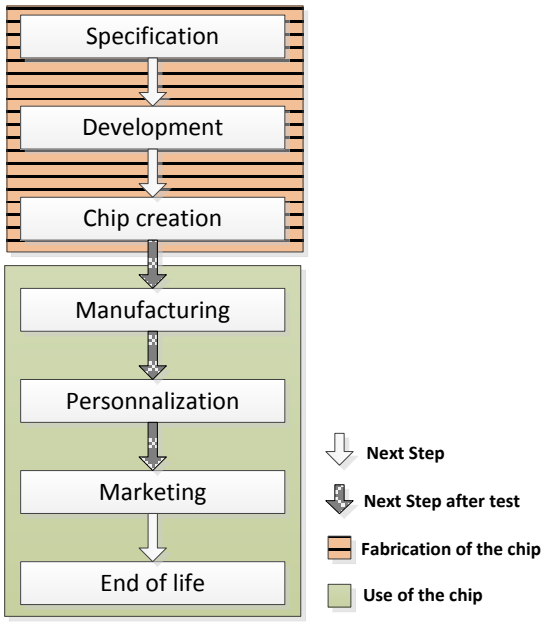



Fig. 3: Simplified cycle of life of a smart card

During the life of a product, we have to ask some questions [19]:

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

Indeed, 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 [20]. The program must be validated and certified. The creation of the criteria for evaluating safety and security allowed product certification [13]. Recognition of these criteria allows mutual recognition between different structures and different countries. We can talk about software evaluation when we are evaluating software systems [21]. In fact, in this paper, we focus on software evaluation but a real certification is not only about the evaluation of software.

### B. Certification

Certified products are tested by the certification laboratories, and generally by the manufacturer during their creation. The certification process gives the level of safety of a product, and is realized by an external laboratory. EMVCo is providing the EMVCo Security Evaluation Process to its member [22]. It ensures a robust security foundation for smart card and related products. The Card Type Approval process allows to test compliance with the EMV specifications. To summarize, EMVCo provides documents to test, evaluate and approve a smart card but also a list of Approved Security Evaluation Laboratories.

### C. Smart card application assurance

The test mainly used to verify smart card applications is a set of activities which try to highlight the differences between the actual and the theoretical behaviors of the application under test. To test an application, two elements are needed:

- The input vector: it's the generation of tests to be applied to the smart card;
- The verdict: it's an oracle that hold the truth.

We particularly deal with the second problem. The idea is to use a model of the system, it can be created in different ways.. Indeed, this problem is the more important problem of system test automation [23]. Specifically, in our field, a model can be generated by several ways. Previously, work have been done on generation using the documentation of a program [24] but this work is limited due to inaccessible data values. Another solution can be to send commands to the application in order to create the needed model [25]. We can know the full behavior of the application. A model can also be a second version of the same program (e.g. a validated implementation of the same application) as in [26]. In black-box, we can use the generation of statements in [27]. With this study, we see that the generation of the data, used by the oracle, is a main topic [28] but not necessary mandatory to do good test [29]. The table 1 shows a comparison of these methods.

## IV. Generation of local and required behaviors

We present a method of oracle's generation for a smart card application. As we want to work with a black box approach, the oracle cannot be another implementation or a formal model which need more information than only documentation or specification. Indeed, we place ourselves in the context where we don't know the inside of the card, only accessible by specific certification firm. We are building the oracle as a collection of local and expected behavior. It consists in the generation of assertions on the APDU communication, and more specifically on the command-response pair.

### A. Context

This approach is based on Assertion Based Design [30], [31]. We can apply the principles of this method on the field of smart card. Indeed, the APDU command-response pairs implicitly define a clock on which we can define sequences. The properties are defined on several clock cycles and there are sequences of fields of several APDU command-response pairs involved in the property. Figure 4 presents the verification of a property that lasts for three clock cycles. If a property is not verified, we can alert the user that the smart card application might be defective.

We propose to generate properties thanks to a certified smart card. Once properties are generated, we can use them to automatically compute the level of confidence we can have for an other smart card, as we can see in figure 5. The observer application listen the APDU communication between a card

TABLE I: Oracle generation methods

| Reference | BlackBox | Method | Use | Date |
|---|---|---|---|---|
| Peters and Pranas[24] | YES | Generation from documentation | Easy but limited | 1998 |
| Aarts *et al.* [25] | YES | Model Inference | Easy | 2013 |
| Alimi *et al.* [26] | NO | Reference Implementation | Need to develop | 2014 |
| Pacheco and Ernst [27] | YES | Set of assertions | Need a model | 2013 |
| Loyoal *et al.* [28] | NO | Assertion-based Oracle | Use of DODONA | 2014 |

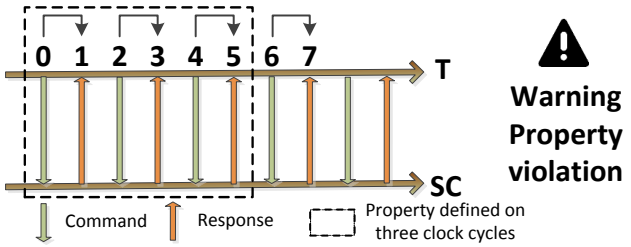



Fig. 4: Detection using APDU communication

and a terminal. The more properties are satisfied, the higher is the conformance of the card being tested.

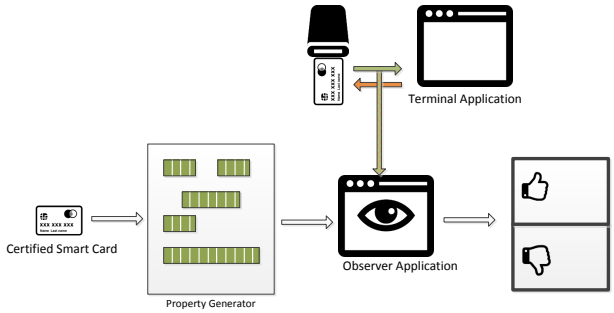



Fig. 5: Link between the generator and the observer

### B. Properties: local and required behaviors

Properties correspond to local and required behavior. They are local as they verified for few command-response pairs among all those involved to realize a payment, with a sliding windows strategy. It is also required as if the property is not satisfied, it indicates that the application card is faulty.

A property can be seen as a first order logic predicate. For example, the property $P_1$:

$$((SW1(1) = 90)\ and\ (SW2(1) = 00))\ or\ \overline{(INS(0) = A8)} \quad (1)$$

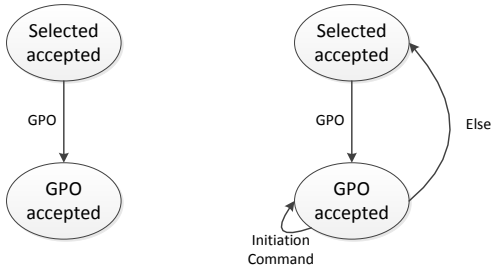
defines the required transition between two states of the finite-state machine that card application must follow: (1) the application is selected and (2) the application is initiated. The fact that SW1 and SW2 equals respectively "90' and '00" means that the command has been accepted by the card application. The INS set to 'A8' corresponds to the Get Processing Option (GPO) command in the M/CHIP specification. Thus, this property means that after that the application is selected, only the GPO command can alter the application's state. If the command is not accepted, it's not the GPO command.

The property $P_2$ is more elaborate:

$$\begin{aligned}&((SW1(1) = 90)\ and\ (SW2(1) = 00)\ and\\ &\quad (SW1(3) \neq 90)\ and\ (SW2(3) \neq 00)\ and\\ &\quad (SW1(5) = 90)\ and\ (SW2(5) = 00))\ or\\ &\overline{(INS(0) = A8)\ and\ (INS(2) = A8)\ and\ (INS(4) = A8)}\end{aligned} \quad (2)$$

This property is an extension of $P_1$. Once the GPO has been accepted, a second GPO must be rejected, and thus the card returns to the state where the application is selected, where a third GPO can be accepted. This property permits to detect an error of implementation in the application, where the second GPO might be accepted instead of being rejected.

We can see these two properties as partial machine states, and are presented in figure 6.




(a) The $P_1$ statechart (b) The $P_2$ statechart

Fig. 6: Properties $P_1$ and $P_2$ as partial finite state machine

The fact is we cannot generate all possible behaviors. First, all behaviors are not relevant, and thus it is not necessary to generate all of them. Second, it is a combinatorial problem to generate all property. For example, if the application have four possible INS commands, and two responses (accepted or not), we have a minimum of 8 properties for one command-response pair, if we consider simple property like $P_1$. By choosing to study the application with behavior of up to three command-response pairs, we get a minimum of $8 + 8^2 + 8^3 = 584$ possible properties given by the specifications. Knowing that we have to manage more than four INS command, more than two possible responses, and that we have to take into account the CLA, P1 and P2 bytes, the number of properties quickly becomes too large to study.

### C. Generation of property: proposed approach

The approach we propose allows us to automate the generation of relevant properties. In order to do this, we randomly generate properties, and then we select the most relevant ones with respect to transaction logs between a terminal and the certified application.

The generation and selection of properties is made by a genetic algorithm [32]. A genetic algorithm is an optimization method, and works on the same principle as the natural evolution. Among the advantages of the genetic algorithm, we can cite that fact that it will randomly generate a high number of possible properties, and that it can generate several distinct correct properties at each run. In order to do this, we randomly create a population of properties, and we measure their performance with a fitness measure. Once all the population has been evaluated, we keep the best properties for the next generation, and we replace the missing one either by creating randomly new properties, mutate one of the kept properties, or fuse two of them. After several generation, we stop the evolution of properties, and the best one is added to the base of selected properties.

In Figure 7, we can see the genetic algorithm selecting properties and adding them to the database of selected properties. On the left, we can see a file that is used to generate properties, and evaluate them.

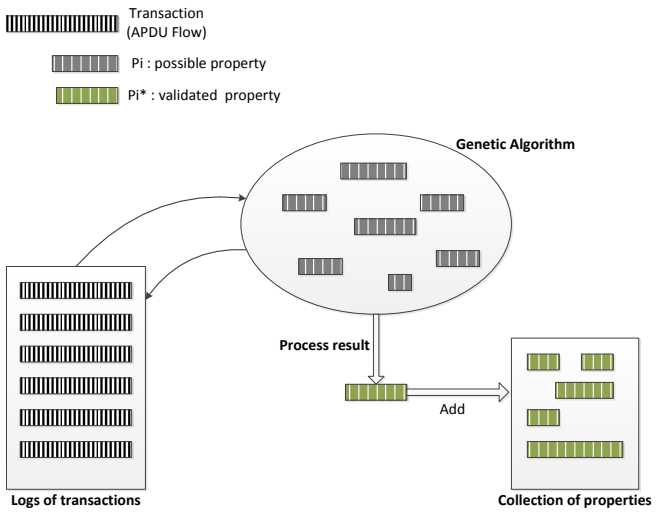



Fig. 7: Proposed approach

This input file for the genetic algorithm is a transaction log file, created from a communication between a terminal and certified smart card. It is a sequence of APDU command-response pairs, defined as follows:

```
-> APDU command
<- APDU response

-> :00|A4|04|00|A0 00 00 00 04 10 10|00
<- :90|00|6F 3F 84 07 A0 00 00 00 04 10 10
 A5 34 50 0A 4D 41 53 54 45 52 43 41 52 44
 87 01 02 5F 2D 04 66 72 65 6E 9F 11 01 01
 9F 12 0A 4D 41 53 54 45 52 43 41 52 44 BF
 0C 0A DF 60 02 0B 14 9F 4D 02 0B 14
-> :80|CA|9F|36||00
<- :6A|88|
-> :80|CA|9F|13||00
<- :6A|88|
-> :80|CA|9F|17||04
<- :90|00|9F 17 01 03
-> :80|CA|9F|4F||13
<- :90|00|9F 4F 10 9F 02 06 9F 27 01 9F 1A
 02 5F 2A 02 9A 03 9C 01
-> :80|A8|00|00|83 00|00
<- :90|00|77 0E 82 02 78 00 94 08 08 01 03
 00 10 01 04 01
-> :00|88|00|00||00
<- :67|00|
```

From this document, the generation algorithm will extract all possible values for the different fields such as CLA, INS, and will generate properties from this possibles values. The fitness function will then evaluate is the generated properties are relevant given this log file.

Once we have the generated properties, we produce an XML file, compatible with the observation tool, as we can see on Listing 1.

Listing 1: Template of properties as an XML file

```
<properties>
  <property>
    <step>
      <instruction instruction="XXXX"/>
      <parameters parameters="XXXX" />
      <status status="XXXX"/>
    </step>
    <step>
      <instruction instruction="XXXX"/>
      <require>
        <status status="XXXX"/>
      </require>
    </step>
  </property>
```

## V. Experimentation and results

### A. Optimization

In order to define our genetic algorithm, we have to define four functions: the generation, mutation and cross-over functions are used to manipulate the population of properties, and the fitness function enable the selection of properties among the population. In our case, we have defined these functions as follow:

- generation of initial population : randomly chosen from the log file.
- selection of individual : random selection biased by the score of individuals. We are removing the lowest individuals.
- mutation : a very small chance (0.1 to 1%) to randomly change the composition of a individual is used to avoid local convergence.
- cross-over : the last step consists of randomly chosen individuals to cross in order to create best individuals.
- fitness : evaluation of the population using the values of the fields constituting the commands and responses.

At each generation, all properties are evaluated with the fitness function, and only properties with best score are

kept for the next generation. The score of a property is the mean score of the command-response pairs it is composed of. The score of a command-response pair is computed by comparing the CLA, UDC, SW1 and SW2 fields to the one from command-response pairs in the learning database. The more correspondence are found between the command-response pair fields and the learning database, the higher the score.

The population counts 100 properties, and 10% are selected at each generation. After 10 generations, we keep the best property among the population. The genetic algorithm can be launched as many times as we want properties, one being generated at each launch.

### B. results

The generation of properties is effective. We can see in Listing 2 an example of two generated properties with the genetic algorithm.

Listing 2: Examples of generated properties

```
<properties>
  <property>
    <step>
      <instruction instruction="0xA4"/>
      <status status="0x6A82"/>
    </step>
    <step>
      <instruction instruction="0xA8"/>
      <status status="0x6985"/>
    </step>
    <step>
      <instruction instruction="0x20"/>
      <require>
        <status status="0x6985"/>
      </require>
    </step>
  </property>
  <property>
    <step>
      <instruction instruction="0xA4"/>
      <status status="0x9000"/>
    </step>
    <step>
      <instruction instruction="0xA4"/>
      <require>
        <status status="0x9000"/>
      </require>
    </step>
  </property>
</properties>
```

We can write these properties with the previous writting. The first one, $P_3$, means that if the selection of the application ('A4') has failed, then the GPO ('A8') command and the VERIFY ('20') command can't be accepted.

$$\begin{aligned}&((SW1(1) \neq 90)\ and\ (SW2(1) \neq 00)\ and\\ &\quad (SW1(3) \neq 90)\ and\ (SW2(3) \neq 00)\ and\\ &\quad (SW1(5) \neq 90)\ and\ (SW2(5) \neq 00))\ or\\ &\overline{(INS(0) = A4)\ and\ (INS(2) = A8)\ and\ (INS(4) = 20)}\end{aligned} \tag{3}$$

The second property, the property $P_4$, means we can select the application twice in a row. For this one, we have to capture the parameters to specify which application can be selected (we have verified the parameters used, i.e. the AID of the application is the correct one in the log file).

$$\begin{aligned}&((SW1(1) = 90)\ and\ (SW2(1) = 00)\ and\\ &\quad (SW1(3) = 90)\ and\ (SW2(3) = 00)\ or\\ &\overline{(INS(0) = A4)\ and\ (INS(2) = A4)}\end{aligned} \tag{4}$$

The computation time is reasonable, as it takes approximately 2 seconds to generate these two properties. The produced XML file is valid, and works well with the observer tool, as it is intended.

### C. Discussion

The generation of properties using genetic algorithm is functional. It is possible to generate as many properties as required, and it is time effective.

However, there is still some place for improvements. The first one concerns the generation function. For now, only the INS byte and the two status word bytes are used. A proper generation function should manipulate all possible byte, such as CLA, P1, P2 as well as optional fields UDC, LE and UDR. The cross-over and mutation functions should be modified accordingly.

The second improvement concerns the fitness function. The effectiveness of the genetic algorithm relies on this function. Improving it would enable to have more meaningful generated properties. For example, it could take into account same additional data, provided by the user, that would give more weight to some field value that are meaningful in a given specifications. Moreover, it should not only take into account the log file, but also the previously generated properties. This would guarantee that generated properties are not redundant.

## VI. Conclusion

We have seen that smart card industries, and more particularly the payment one, rely their security on the standards and specification, such as EMV. Before being released, a smart card following these specifications must be validated by specialized firms.

This validation is possible thanks to an observer, listening the communication between a terminal and a smart card. The product might be validated as long as all the defined properties are validated by the observer. However, it is complicated to generate theses properties, as it is a combinatorial problem.

The proposed approach enables the automated creation of properties, thanks to genetic algorithms, to create a relevant partial oracle to know if there is an anomaly on the smart card application. However, there is still some place for improvements. The perspectives are to improve the functions used by the genetic algorithm. Particularly, the improvements of the fitness function would give us more useful properties.

## References


[1] *ISO/IEC 7816 - Identification cards*, International Organization for Standardization and the International Electrotechnical Commission Std.

[2] Eurosmart, http://www.eurosmart.com/publications/market-overview, 2014.

[3] EMVCo, http://www.emvco.com/, 2014.

[4] S. c. d. l. s. d. s. d. S. Carlos Martin, "évaluation et certification."

[5] K. Vedder and F. Weikmann, "Smart cardsrequirements, properties, and applications," in *State of the Art in Applied Cryptography*. Springer, 1998, pp. 307–331.

[6] *ISO/IEC 14443 - Identification cards – Contacless intergrated circuit cards*, International Organization for Standardization and the International Electrotechnical Commission Std.

[7] *EMV Integrated Circuit Card Specifications for Payment Systems, version 4.3*, EMVCo Std., 2011.

[8] *M/Chip 4 Card Application Specifications for Credit and Debit*, MasterCard International Std.

[9] *Visa Integrated Circuit Card Specifications*, Visa Std., 2009.

[10] S. J. Murdoch and R. Anderson, "Security protocols and evidence: Where many payment systems fail," in *Financial Cryptography and Data Security*. Springer, 2014, pp. 21–32.

[11] Joint Interpretation Librabry, "Application of attack potential to smartcards," 1 2013, version 2.9.

[12] A. C. Noubissi, A. Séré, J. Iguchi-Cartigny, J.-L. Lanet, G. Bouffard, and J. Boutet, "Cartes à puce: Attaques et contremesures," *MajecSTIC*, vol. 16, p. 1112, 2009.

[13] W. Rankl, *Smart Card Applications: Design models for using and programming smart cards*, Wiley, Ed., 2007.

[14] M. Bond, O. Choudary, S. J. Murdoch, S. Skorobogatov, and R. Anderson, "Chip and skim: cloning emv cards with the pre-play attack," in *Security and Privacy (SP), 2014 IEEE Symposium on*. IEEE, 2014, pp. 49–64.

[15] L. Francis, G. Hancke, and K. Mayes, "A practical generic relay attack on contactless transactions by using nfc mobile phones," *International Journal of RFID Security and Cryptography (IJRFIDSC)*, vol. 2, pp. 92–106, 2013.

[16] S. J. Murdoch, S. Drimer, R. Anderson, and M. Bond, "Chip and pin is broken," in *Security and Privacy (SP), 2010 IEEE Symposium on*. IEEE, 2010, pp. 433–446.

[17] W. Rankl and W. Effing, *Smart card handbook*. John Wiley & Sons, 2010.

[18] Joint Interpretation Librabry, "Guidance for smartcard evaluation," 2010.

[19] J. Radatz, A. Geraci, and F. Katki, "Ieee standard glossary of software engineering terminology," *IEEE Std*, vol. 610121990, no. 121990, p. 3, 1990.

[20] S. C. Alliance, "What makes a smart card secure?" 2008.

[21] D. R. Wallace, D. R. W. L. M. Ippolito, B. B. Cuthill, and L. M. Ippolito, *Reference information for the software verification and validation process*. DIANE Publishing, 1996.

[22] http://www.emvco.com/approvals.aspx, 2015.

[23] E. T. Barr, M. Harman, P. McMinn, M. Shahbaz, and S. Yoo, "The oracle problem in software testing: A survey," 2015.

[24] D. K. Peters and D. L. Parnas, "Using test oracles generated from program documentation," *Software Engineering, IEEE Transactions on*, vol. 24, no. 3, pp. 161–173, 1998.

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

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

[27] C. Pacheco and M. D. Ernst, *Eclat: Automatic generation and classification of test inputs*. Springer, 2005.

[28] P. Loyola, M. Staats, I.-Y. Ko, and G. Rothermel, "Dodona: automated oracle data set selection," in *Proceedings of the 2014 International Symposium on Software Testing and Analysis*. ACM, 2014, pp. 193–203.

[29] G. Fraser, M. Staats, P. McMinn, A. Arcuri, and F. Padberg, "Does automated white-box test generation really help software testers?" in *Proceedings of the 2013 International Symposium on Software Testing and Analysis*. ACM, 2013, pp. 291–301.

[30] H. D. Foster, A. C. Krolnik, and D. J. Lacey, *Assertion-based design*. Springer Science & Business Media, 2004.

[31] G. Jolly, S. Vernois, and J.-L. Lambert, "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*. Springer, 2014, pp. 192–201.

[32] L. Davis *et al.*, *Handbook of genetic algorithms*. Van Nostrand Reinhold New York, 1991, vol. 115.