COMPUTING SCIENCE

Rigorous Design and Implementation of an Emulator for EMV Contactless Payments

Martin Emms, Leo Freitas and Aad van Moorsel

TECHNICAL REPORT SERIES

No. CS-TR-1426 July 2014

TECHNICAL REPORT SERIES

No. CS-TR-1426 July, 2014

Rigorous Design and Implementation of an Emulator for EMV Contactless Payments

M. Emms, L. Freitas and Aad van Moorsel

Abstract

EMV is the contactless payment protocol supported worldwide by the major credit card companies in countries outside the USA. This paper presents a hybrid formal/non-formal design and implementation process for high integrity protocol emulators as well as a corresponding implementation of the EMV protocol and point of sale terminal. The objective of the EMV emulator is to test new cards and applications and to experiment with protocol attack and failure scenarios. The proposed design and implementation process includes a systemic inspection of the EMV natural language specification, the generation of a formal abstract model that represents the EMV protocol, the generation of test cases from the formal abstract model, continuous feedback from the implementation and the systemic documentation of the emulator code. We have applied the design and implementation process to the development of emulator code for Chip & PIN transactions, Visa contactless transactions and MasterCard contactless transactions.

Printed and published by Newcastle University,
Computing Science, Claremont Tower, Claremont Road,
Newcastle upon Tyne, NE1 7RU, England.

Bibliographical details

EMMS, M., FREITAS, F., VAN MOORSEL, A.

Rigorous Design and Implementation of an Emulator for EMV Contactless Payments
[By] M. Emms, L. Freitas and A. van Moorsel

Newcastle upon Tyne: Newcastle University: Computing Science, 2014.

(Newcastle University, Computing Science, Technical Report Series, No. CS-TR-1426)

Added entries

NEWCASTLE UNIVERSITY
Computing Science. Technical Report Series. CS-TR-1426

Abstract

EMV is the contactless payment protocol supported worldwide by the major credit card companies in countries outside the USA. This paper presents a hybrid formal/non-formal design and implementation process for high integrity protocol emulators as well as a corresponding implementation of the EMV protocol and point of sale terminal. The objective of the EMV emulator is to test new cards and applications and to experiment with protocol attack and failure scenarios. The proposed design and implementation process includes a systemic inspection of the EMV natural language specification, the generation of a formal abstract model that represents the EMV protocol, the generation of test cases from the formal abstract model, continuous feedback from the implementation and the systemic documentation of the emulator code. We have applied the design and implementation process to the development of emulator code for Chip & PIN transactions, Visa contactless transactions and MasterCard contactless transactions.

About the authors

Martin Emms is studying for a research PhD at Newcastle University's Centre for Cybercrime and Computer Security (CCCS). My research into potential vulnerabilities in the EMV payments system brought about by the introduction of Near Field Communications (NFC) payment technologies (i.e. NFC payment cards, mobile phone payments applications, NFC payment tags and NFC payment / top-up wrist bands). Supervised by Professor Aad van Moorsel with the School of Computing Science at Newcastle University. Martin has also been working with a local women's support centre in the North East of England to better understand the issues faced by survivors of domestic violence. The main focus of this research has been enabling survivors to access online / electronic domestic violence support services without the fear of being caught by their abuser. His role has been to design new applications that can help survivors access support services without leaving tell-tale electronic footprints.

Leo Freitas is a lecturer in Formal Methods working on the EPSRC-funded AI4FM project at Newcastle University. Leo received his PhD in 2005 from the University of York with a thesis on 'Model Checking Circus', which combined refinement-based programming techniques with model checking and theorem proving. Leo's expertise is on theorem proving systems (e.g. Isabelle, Z/EVES, ACL2, etc.) and formal modelling (e.g. Z, VDM, Event-B), with particular interest on models of industrial-scale. Leo has also contributed extensively to the Verified Software Initiative (VSTTE).

Aad van Moorsel is a Professor in Distributed Systems and Head of School at the School of Computing Science in Newcastle University. His group conducts research in security, privacy and trust. Almost all of the group's research contains elements of quantification, be it through system measurement, predictive modelling or on-line adaptation. Aad worked in industry from 1996 until 2003, first as a researcher at Bell Labs/Lucent Technologies in Murray Hill and then as a research manager at Hewlett-Packard Labs in Palo Alto, both in the United States. He got his PhD in computer science from Universiteit Twente in The Netherlands (1993) and has a Masters in mathematics from Universiteit Leiden, also in The Netherlands. After finishing his PhD he was a postdoc at the University of Illinois at Urbana-Champaign, Illinois, USA, for two years. Aad became the Head of the School of Computing Science in 2012.

Suggested keywords

EMV
CONTACTLESS
PAYMENT
EMULATOR
SOFTWARE QUALITY
HIGH-INTEGRITY DESIGN
IMPLEMENTATION.

Rigorous Design and Implementation of an Emulator for EMV Contactless Payments

Martin Emms
Newcastle University
martin.emms@newcastle.ac.uk

Leo Freitas
Newcastle University
leo.freitas@newcastle.ac.uk

Aad van Moorsel
Newcastle University
aad.vanmoorsel@newcastle.ac.uk

Abstract—EMV is the contactless payment protocol supported worldwide by the major credit card companies in countries outside the USA. This paper presents a hybrid formal/non-formaldesign and implementation process forhigh integrity protocol emulators as well as a corresponding implementation of the EMV protocol and point of sale terminal. The objective of the EMV emulator is to test new cards and applications and to experiment with protocol attack and failure scenarios. The proposed design and implementation processincludes a systemic inspection of the EMV natural language specification, the generation of a formal abstract model that represents the EMV protocol, the generation of test cases from the formal abstract model, continuous feedback from the implementation and the systemic documentation of the emulator code.We have applied the design and implementation process to the development of emulator code for Chip & PIN transactions, Visa contactless transactions and MasterCard contactless transactions.

Keywords-EMV, contactless, payment, emulator, software quality, high-Integrity design and implementation.

I. INTRODUCTION

The EMV specifications (EMV stands for Europay, MasterCard and Visa) control the operation of one and a half billion payment cards and twenty-one million point of sale terminals worldwide [6]. EMV payments can be contact mode (commonly termed Chip & PIN) and contactless mode, also known as NFC (Near Field Communication). Contact payments require the cardholder to insert the card into the point of sale terminal and enter their PIN to confirm the transaction. These Chip & PIN transactions can be any value up to the card limit / available balance on the card. Contactless payments are designed to be a convenient way to pay for low value (of the order of twenty euro, pounds or dollars, depending on the national arrangements) transactions with a card rather than cash. Designed to be faster than a traditional Chip & PIN transaction, the card is placed in close proximity to the point of sale terminal to authorise transactions. A PIN is only required once every so many purchases, depending on the specific implementation of the protocol.

The EMV protocol is thus the mainstay of card based payment, including contact and contactless, and, increasingly, payment through phones [7]. An emulator allows one to test cards or applications that use the EMV protocol, and to try out attacks against EMV's security and dependability provisions. Especially in the latter case, the correctness of the emulator's protocol implementation is critically important. In this paper we introduce a practicalhybrid process for the design and implementation of protocol emulators, using a combination of formal techniques and careful implementation processes to gain and increase confidence in the correctness of the emulator implementation. In addition, we illustrate the use of this process to the development of an EMV point of sale terminal.

Our design and implementation approach combines informal and formal techniques. At the centre of our approach are UML sequence diagrams, which we use as the informal but precise description of the protocol fragments. It takes three main sources as input: (i) the EMV documents, (ii) feedback from the insights gained by the developers from coding the emulator, and (iii) feedback from the insights gained by the designers from constructing a formal model. To maintain control over the implementation, we rely on systemic code documentation and cross-referencing between code, UML diagrams and EMV specification, as we will explain.

The formal aspects of our approach are inspired on Praxis methodology [3], tailored to our needs. It focuses on the construction and proof of an abstract model using the Z notation [19]. This abstract model is used to investigate the consistency of the requirements, expose descriptive errors, and ultimately be used to generate test cases for the emulator code. Ultimately, if our abstract formal model correctly characterises the EMV requirements, then our test cases will be both minimal and wide-reaching, given they come from the mathematical characterisation of the EMV requirements for NFC.

The informal techniques are the main building blocks for the implementation, but, importantly, feedback from the formal modelling as well as the coding is included in the systemic process to build the emulator. The end result is a software emulator for the EMV transaction protocol sequence, and a demonstration that for generated test cases the emulator is an accurate representation of EMV.

The resulting emulator is the only emulator we know of that implements the Point of Sale terminal side of the EMV contactless protocol. Other emulators, such as [12][10], focus on the card device behaviour or are limited to the Chip & PIN contact protocol sequence. It should be noted that once the code is complete it will be made available as an open source emulator (the current code has significant functionality, including the contact and VISA contactless

sequences, with three others (MasterCard, Amex and JCB) still remaining).

This paper is organised as follows. We first explain the design process we followed in Section II. The soundness of the process is critical to gain confidence in the correctness of the emulator with respect to it implementing the EMV protocol correctly. Section III discusses the emulator design and implementation. Section IV introduces the formal model in some detail—it is not within the scope of this paper to discuss all details of the formal model. The role of the formal model in this paper is to generate test cases Section IV.C for the implementation and to feed insights back to the construction of the UML sequence diagrams. In Section VI the generated test cases and results of the tests are discussed. We conclude the paper with an evaluation of the quality of the code as well as the design process in Section VII, a discussion of related work in Section VIII and a conclusion in Section IX.

II. Design and Implementation Process

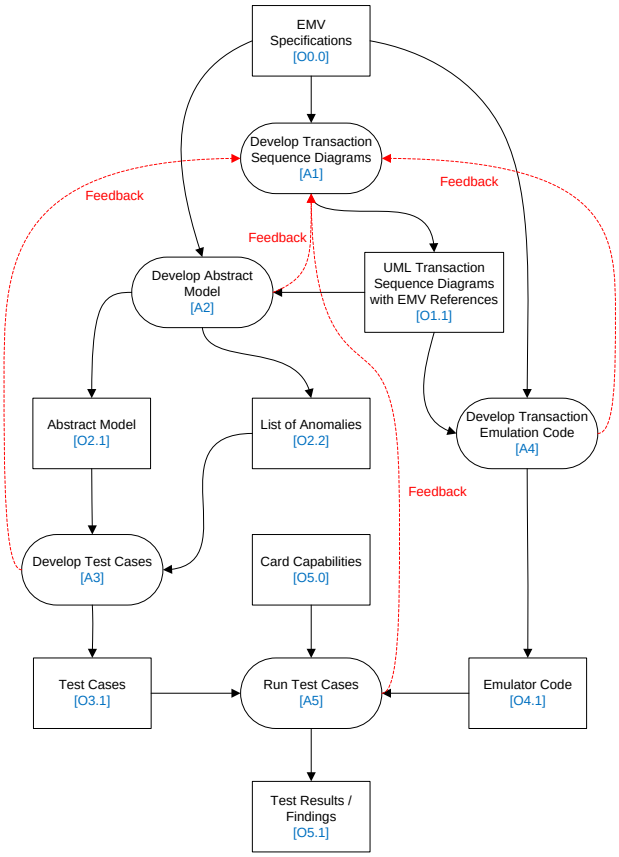
Ensuring that the emulation is an accurate representation of the EMV specification requires a systematic approach, which is documented at all stages, and which takes feedback from each stage of the process to refine the design and implementation of the emulator.The EMV requirement specifications [1][2] consist of over 2,000 pages of written English and flow diagrams. There are five distinct transaction sequences described in the EMV specifications: the original Chip & PIN contact transaction and four contactless transactions (one each for Visa, MasterCard, American Express and JCB), the latter four all being relevant for our work. Each transaction sequence is described in multiple sections spread across multiple volumes of the EMV requirements. For example Book 2 of the Chip & PIN specification [1] contains all of the security and cryptography functionality required for all of the contact and contactless transaction sequences described in the other books.

The process we follow in the implementation of the emulator is detailed in the UML activity diagram of Fig. 1The rounded boxes are activity nodes within the process, with reference numbers such as [A1]. The square boxes are object nodes with references such as [O1.0]: these are the data sources that drive the activities. Connecting edges, represented as black solid-arrows, indicate the default order in the flow of activities. However, the process is iterative, and the red dashed-arrows are connecting edges which indicate feedback within the process.

At the centre of our approach is the construction [A1] of UML sequence diagrams [O1.1]. Much of the process is about constructing these sequence diagrams as accurate as possible. To achieve this, we use a detailed analysis of the EMV requirements and a detailed working knowledge of the structure of the various specifications contributing to a single transaction. Moreover, we use feedback from the formal model construction [A2], the derivation of test cases [A3] and the coding [A4].

The EMV specifications [O0.0] are the originating source of all of the data in the process. Any data or assumption made in the emulator code or in the abstract model should be traceable back to its origin (i.e. the book/section/page within the EMV specifications). The EMV specifications are structured so that the complete description for a single transaction sequence is split across multiple sections and multiple books. The UML sequence diagrams [O1.1] are collate these multiple sources into a single easy to follow description of the transaction sequence. The transaction sequence diagrams [O1.1] are the initial stage of the iterative process that we used to create the concrete software implementation of the emulator [O4.1].

Fig. 1. Process of Creating and Validating an EMV Emulator



The UML sequence diagrams capture the sequence and logic of the transaction protocol, see Fig. 2 for an example. The implementation detail required to write the emulator code is captured using descriptive text and detailed references to the EMV specification, see TABLE I. for an example of the cross-referencing. During coding this list of references is used to retrieve the implementation details required to create the code. The references are also included in the code so that the link between the code and the EMV specification is documented.

The UML diagrams with the associated list of EMV references are used to guide the coding process [A4]. The diagrams guide the structure of the code whilst the detail of

the coding is taken from the referenced book / section / page in the EMV specifications.

At each stage of the process if additional information is found about the working of EMV it is fed back into the UML transaction sequence diagrams [O1.1]. The feedback is essential to refine our understanding of the EMV specifications and document it. Each time the diagrams are updated this drives the improvement of the emulator code [O4.1]. The completed Emulator Code is used in practical experiments [A5], running full or partial transaction sequences against bank cards.

Given the sheer scale of the EMV specification and its internal inconsistencies, we found it paramount to have a systematic, more rigorous, description of EMV specification details. These formal developments are depicted on the left hand side of Fig. 2. This results in an Abstract Model [O2.1], which has two important functions. First, we use it to develop test cases [O3.1]. Secondly, through the construction of the abstract model and test cases we create a deeper understanding of the EMV specification. This results in important feedback in establishing the correct transition diagrams in [A1].

The Test Cases [O3.1] can be used in two types of test. First, it helps provide confidence that the emulator is an accurate representation of the EMV specifications. Secondly, based on proof approaches detailed in Section IV, we can show that bank cards behave in an anomalous manner as identified in the List of Anomalies [O2.2]. To show that the emulator is an accurate representation of the EMV specification, we create the transaction data and terminal capabilities, which according to the abstract model [O2.1], should result in a particular outcome. Test [A5] can then be run against bank cards to investigate whether the cards under test implement the anomalous functionality we have identified in [O2.2]. Even though it may concern anomalous functionality, correct results of the tests [O5.1] provide the basis of our assertion that the emulator is an accurate representation of the EMV specification.

III. EMULATOR

In this section we review the main implications the design process had on the design and implementation of the EMV emulator. In particular, we discuss in Section III.A the UML sequence diagrams and explain how we documented the diagrams and the code with cross references to the EMV specifications. We also discuss in Section III.B the data dependencies within the code.

At a high level, the emulator code follows the Model View Controller design pattern. This has the advantage of allowing us to encapsulate the EMV transaction sequence logic as it is specified in [1][2] and keeping it separate from the user interface code and the contactless card reader device driver.

The code follows the structure of the EMV specifications with a kernel for Chip & PIN transactions and 4 contactless kernels for Visa, MasterCard, American Express and JCB. The contactless kernels are controlled by the Entry Point which decides which kernel to run based on the card presented to the reader, this exactly follows the structure of the EMV contactless specification [2].

A. *Documenting EMV Transaction Sequences*

The UML sequence diagram in Fig. 2collates information from multiple sections of the EMV specification to capture a transaction protocol sequence (in this case Visa fDDA transaction for offline transactions) in a single diagram. It shows the communication between a terminal and a card, with the messages that go back and forth. In this example, the card and terminal first determine the type of payment it will execute, defined through the Application identifier (AID). It then creates specific request in its Processing Options Data Object List (PDOL). The terminal then loops through several read request as indicated by the Application File Locator (AFL).

Fig. 2. fDDA Contactless Offline Transaction

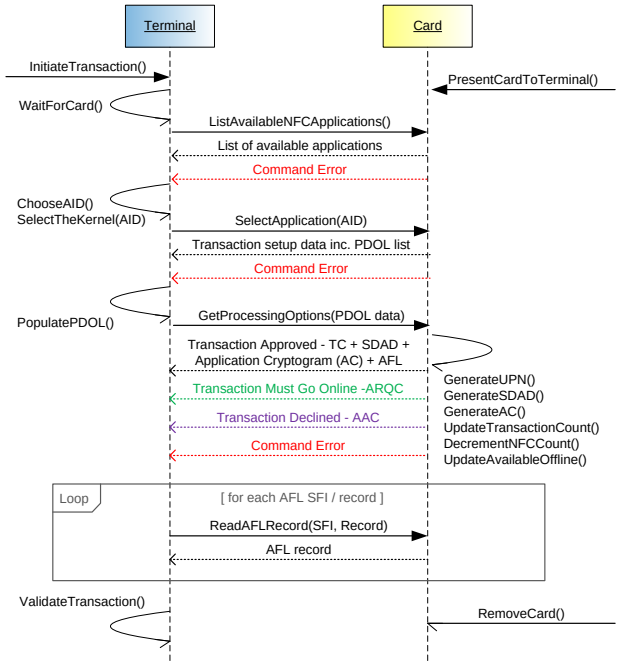


TABLE I. contains a description of each operation in the UML sequence diagram and associated references to the original EMV specification. This gives the level of detail required to keep control over the implementation of the emulator code.

Level of detail in the table means that providing the details of the entire transaction sequence would cover several pages.We have therefore restricted it to the entries for the GetProcessingOptions() command (see Fig. 2) and the card's responsesforthat command.

In TABLE I. the blue cells contain a description of a terminal command, the yellow describe the responses from

the card and the white cells contain the references to the EMV specification where the details of the preceding cell can be found.

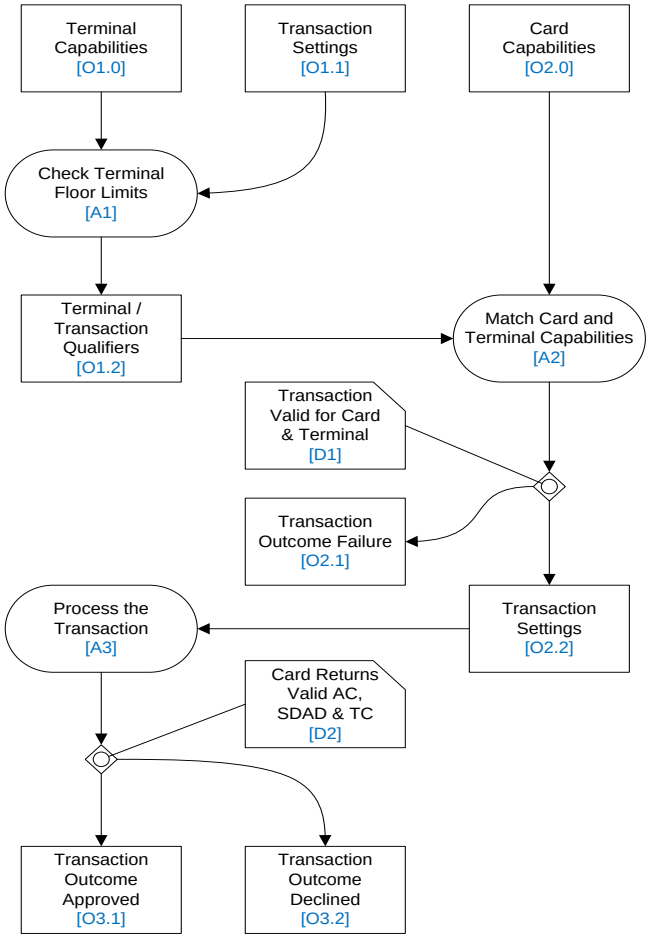
TABLE I. REFERENCES FOR UML DIAGRAM FIG. 2

GetProcessingOptions(PDOL data) In the Visa fDDA transaction Get Processing Options (GPO) is used to request completion of the transaction. The PDOL data must contain all of the data elements requested by the card otherwise the transaction will be rejected.
EMV v2.2 Book C-3 - 2.4.1 Initiate Application Processing p12 EMV v2.2 Book C-3 - 5.2 Initiate Application Processing p40 EMV v2.2 Book C-3 - 5.2.1 Get Processing Options (GPO) Command p40 EMV v2.2 Book C-3 - 5.2.2 Initiate Application Processing p40 – 46 EMV v4.3 Book 3 - 6.5.8.4 Data Field Returned in the Response p60
Transaction Approved – TC + SDAD + AC + AFL If the card approves the completion of the transaction in offline mode, it will return Transaction Cryptogram (TC) in the Cryptogram Information Data (CID). The card also returns all of the data elements required by the terminal to complete the transaction: Signed Dynamic Application Data (SDAD) used by the terminal to verify that the card has approved the same transaction that the terminal sent. Application Cryptogram (AC) used in the completion of the transaction with the Bank to validate that a valid card completed the transaction. Application File Locator (AFL) contains the location in the card's file structure where the terminal can read the data elements required to complete the transaction.
EMV v2.2 Book C-3 - 5.4.3 Determine the Card Disposition p50 EMV v2.2 Book C-3 - 5.2.2.2 GPO Response SW1 SW2 p43 EMV v2.2 Book C-3 - 5.2.2.3 Contactless Path Determination p46 EMV v2.2 Book C-3 - A.2 Data Elements by Name p97 EMVv2.2 Book C-3 Annex C Fast Dynamic Data Authentication p127 EMV v2.2 Book C-3 - C.1 Dynamic Signature Verification p128 EMV v4.3 Book 3 - 6.5.8 Get Processing Options APDUs p59 EMV v4.3 Book 3 - 6.5.8.4 Data Field Returned in the Response p60
Transaction Must Go Online –ARQC If the card requires online completion of the transaction it will return ARQC in the Cryptogram Information Data (CID). Online completion is required when the amount of the transaction exceeds the cards offline transaction limit or offline cumulative limit or when the number of offline transactions exceeds the number of consecutive offline transactions.
EMV v2.2 Book C-3 - 5.4.3 Determine the Card Disposition p50 EMV v2.2 Book C-3 - A.2 Data Elements by Name p97 EMV v2.2 Book C-3 - 2.4.7 Online Processing (EMV Mode) p14 EMV v2.2 Book C-3 - 2.4.8 Completion (EMV Mode) p15
Transaction Declined - AAC If the card declines the transaction it returns AAC in the CID. The card declines the transaction when it cannot be completed as requested by the terminal.
EMV v2.2 Book C-3 - 5.4.3 Determine the Card Disposition p50 EMV v2.2 Book C-3 - A.2 Data Elements by Name p97 EMV v2.2 Book C-3 - 2.4.7 Online Processing (EMV Mode) p14 EMV v2.2 Book C-3 - 2.4.8 Completion (EMV Mode) p15
Command Error The possible error codes returned by Get Processing Options are: 6984- Try Another Interface. The transaction should be reattempted using either the contact interface or the magnetic strip interface. 6985 - Select Next. The transaction should be reattempted using the next combination of Kernel / AID (if any). 6986 - Try Again, reattempt the transaction with the same parameters. The card must be represented, this may occur if the card is removed too early.
EMV v2.2 Book C-3 - 5.2.2.2 GPO Response SW1 SW2 p43 EMV v2.2 Book C-3 - 2.4.8 Completion (EMV Mode) p15 EMV v2.2 Book B - 3.3 Combination Selection p24

B. *Documenting the Data Dependencies*

The EMV specifications contain a number of complex data structures that control the operation of the transaction sequence. We have therefore documented the data dependencies of the transaction sequences as it is important to understand what the critical data elements are when coding the emulator. An example can be seen in Fig. 3The same object and activity symbols have been used as in Fig. 2with the additional decision elements e.g. [D1] to specify the detail of the decision nodes.

Fig. 3. Data Dependencies of the fDDA Transaction



The outcome of the transaction is dependent on the settings / status of the terminal [O1.0], the value of the transaction [O1.1] and the settings / status of the card [O2.0].

As an example, the significant terminal settings [O1.0] are the Terminal floor limits which specify the maximum

value for offline transactions. The Terminal Transaction Qualifiers (TTQ) which define modes in which the terminal can operate (online / offline), the reader interfaces supported (EMV / magnetic strip), the cardholder verification modes supported (PIN / Signature / No Verification) and the card authentication cryptographic methods supported (SDA / DDA / CDA).

The significant transactions are the transaction amount and the transaction date. The amount can be greater than the terminal floor limits for this transaction or cumulatively over a number of transactions performed on the terminal. The amount can also be greater than the card limits for this transaction or cumulatively over a number of transactions performed by the card. The date of the transaction is checked against the start and expiry dates of the card.

The significant card settings [O2.0] are the start / expiry dates, value limit for offline transactions, cumulative value limit for offline transactions, limit for the number of consecutive offline transactions, authentication methods supported (SDA / DDA / CDA), operational modes (online / offline), supported reader interfaces (EMV / magnetic strip), cardholder verification modes (PIN / Signature / No Verification).

IV. Formal Abstract Model

In this work, we studied the EMV requirements documents [1][2] to produce a formal abstract model of its properties and functionalities, specifically for the fDDAcontactless transaction protocol (summarised in Fig. 2The motivation is to mathematically capture these requirements enabling checking properties of interest to hold (i.e.the requirements documents are consistent), and to produce test cases for our EMV emulator derived from formal proof. We first elaborate in Section A on the use of formal methods in the design of high-integrity software, and then discuss in Section B the specific use of the abstract model for the implementation of the emulator.

A. Formal Models in High-Integrity Software

The motivation for our approach comes from the development of embedded systems in the safety-critical domain, where both UML diagrams and formal models are used. Benefits include the creation of meaningful abstractions without implementation details, and early discovery of faults. There is a distinction between rigorous hand-written mathematics and formal mechanically checked mathematics descriptions. High assurance is achievable via correctness-by-construction and a rigorous evaluation of risks and costs [3].

A mathematical characterisation of requirements is created as an abstract model. A proof that the security or other properties is established by the model serves as evidence that properties are ensured by the abstract model, and similarly for the concrete design and code. Concrete designs are created rigorously or formally, with or without refinement proofs linking layers of development, depending on the desired assurance level. From the concrete formal design, a minimal set of code test cases with high coverage is generated.

In the highest-assurance scenarios, functional correctness proofs of code are performed to check if code behaviour corresponds to specifications. Available code-generators can take the formal design and produce provably correct code. System design represents the (mathematically) documented application programming interfaces (APIs). A landmark example is by the US National Security Agency on access control [3] it introduces a new software standard with efforts quantitatively measured and justified. In our case, we created a formal model of the EMV NFC requirements as way of validating the current implementation and of introducing a model-based testing methodology to the study of EMV emulator code. That is, if our formal model properly captures the requirements, we can instantiate its variables with specific values to create test cases that will exercise all classes of tests/problems possible within the EMV spectrum. The idea is to have a minimal set of tests that captures all classes of tests possible.

In [20], we provide an abstract model of the complete operation of the EMV fDDAcontactless transaction, with proof ensuring all the conditions to establish the feasibility of operations involved in the sequence diagram of Fig. 2These proofs are then used to produce the test cases [Fig. 1O3.1] for the EMV emulator.In this way, we ensure that the emulator code does indeed satisfy the conditions underpinned by the model. The test cases are described in this Section C.

Ultimately, this model is to be used to either derive code, or to be used as code annotation to be used by static analysis tools to determine its correctness with respect to both EMV specification and formal model, and ultimately the EMV requirements documents themselves. Code contracts [5] are used to detect run-time errors like division by zero. Functional correctness requires contracts often twice the code size. Properties analysed depend on the contract provided and on the levels of assurance or correctness needed. Contract provenance is thus an important aspect of the code verification process.

B. Abstract Model

Our abstract model uses the Z notation. Proof obligations in Z are usually of three kinds: well-formedness of models, where partial functions are applied within their domains, and unique existential quantifiers are sound; operational feasibility, where specified operations have (implicitly defined) preconditions strong enough to establish (explicitly defined) postconditions; and data reification via (usually forward) simulation, where the use of (concrete) data structure representations in operations closer to an implementation language are shown to respect the abstract representation and operations.

Our models have 49type definitions,61 Z schemas representing the NFC operations of the protocol,and 79 proofs in total, of which 49 are theorems representing properties of interest for the whole model[20]. Feasibility

proofs are useful in deducing formal model-based test cases as they characterise the complete space of behaviours for all operations of interest, including successful and all possible error cases, both determined by mathematical predicates representing disjoint behaviours of the protocol. That is, feasibility proofs characterise a set of disjoint predicates with non-overlapping conditions that when accumulated lead to true (e.g. precondition op = x < 0 or x > 0 or x = 0). Thus, each disjunct represents a unique class of behaviours for the functionality being proved. Moreover, we also prove that these disjunct predicates amount to true, hence we guarantee all behaviours are accounted for.

The formal model follows the methodology advocated in [3], which enumerate requirements realised by each piece for formal specification. Then, if all elements of the requirements are accounted for within the abstract model in away that conveys the indented behaviour described in English. We state this formal model is a more accurate representation of the EMV protocol than the EMV specification books.

These efforts correspond to theterminal side of Fig. 2The mechanisation of a formal concrete design, together with a proof of refinement these designs faithfully satisfy the abstract model linked to the requirements is under construction. Refinement proofs are perhaps the most costly aspect of any proof exercise, as it needs to establish that the implementation details do not breach any of the contractual requirements established by the abstract model.

We derive our test cases from this abstract model. We also derive a systematic code-annotation technique, whereby the same principle of enumerating what aspect of the requirements each piece of code within the emulator is realised. These test cases represent a test-oracle based on requirements testing, rather than testing for any implementation issues. The former is likely to capture potential (major) errors, as well as inconsistencies within the protocol itself, as the next section discusses. Errors from the concrete design are more likely to expose problems with implementation choices, and that is our aim in the future as well, where the emulator code will be annotated with formal specification amenable to static analysis of properties of the behaviour of the code.

C. The Current Emulator Implementation

The code of the emulator is structured according to the structure of the EMV specification, the core modules are (i) Kernel 0 contact Chip & PIN transactions for all card types (ii) Kernel 1 JCB contactless transactions (iii) Kernel 2 MasterCard contactless transactions (iv) Kernel 3 Visa contactless transactions (v) Kernel 4 American Express contactless transactions (vi) Entry Point which initiates the transaction and decides which Kernel to run based on the type of card presented to the emulator (vii) the RSA cryptography module which is shared by all of the Kernels. So far we have implemented the Kernel 0, Kernel 2, Kernel 3, Entry Point and Cryptography modules.

V. TEST CASES

Feasibility proofs were used to derive test cases for the emulator to satisfy. In [20], a complete list of such cases is discussed. Here, we present the key results from two main steps of the fDDA protocol: PDOL quests from the terminal to the card, and the card's SDAD response, both of which are responsible for establishing the requested transaction is valid and financially fulfilled.

A. PDOL Test Cases

For the PDOL, there are six tests of interest, five of which are error cases where we expect the emulator to demonstrate exceptional behaviour. The successful case (s1) relies on: i) valid card capability (i.e. user chose a Visa transaction on a Visa card); (ii) valid transaction with all the conditions for a fDDAcontactless transaction met (see [6] for details); and (iii) the right payload on the PDOL response. Error cases include: e1) when the capability is not available (i.eterminal choose MasterCard for a Visa card); e2) when the terminal's transaction usage cannot match any available usage from the card (i.e. card is cash-only); e3) when the card's and terminal's capabilities are incompatible (i.e. the fDDA is an offline transaction terminal and the terminal only supports online). Finally, trivial error cases are when the transaction type is not NFC.

For all these error cases the EMV requirements do not specify what kind of PDOL should the protocol return. That leaves the option of returning a valid PDOL (manufactured by mistake or design), which itself is something that needs tightening in the EMV documents. We derived tests for these cases to see whether the emulator suffers from this (mis)behaviour. Fortunately, this was not present in the code. Nevertheless, this is indeed a hole in the EMV specification found through formal reasoning.

B. Foreign Currency Transaction Limits

The transaction currency is one of the important fields in the transaction data as it is one of the data elements signed by the card in the SDAD to verify the transaction to the terminal. However the EMV specifications do not specify the process required when the terminal and the card have different currencies, the specifications contain a description of a reference currency which is to be used in this situation but no information about its use. This omission in the EMV specification was discovered as part of the process to formulate the pro-conditions for the abstract model. It was clear that the currency was one of the preconditions that should be included in the model but we could not establish correct process or outcome when the terminal currency != card currency from the EMV specification.

The only specific reference to foreign currency transactions is in a MasterCard specific document OneSMART Pre-Authorized Solution Guide [8] which on page 2-10 states "The card can only authorize transactions offline if they are expressed in the currency that the issuer specifies on the card application at personalization. Transactions in all other currencies will be driven online for authorization by the issuer."

VI. Results of the Test Cases

A. Methodology and Environment

The emulator allows us to configure the terminal settings so that we can create the specific conditions required to run the test cases generated by the abstract model. The important terminal settingsfor the specific fDDA protocol sequence we discuss here are the Terminal floor limits and the Terminal Transaction Qualifiers (TTQ). The Terminal floor limits specify the value above which transactions must be performed online. The TTQ defines modes in which the terminal can operate (online / offline), the reader interfaces supported (EMV / magnetic strip), the cardholder verification modes supported (PIN / Signature / No Verification) and the card authentication cryptographic methods supported (SDA / DDA / CDA).

For our tests the currency used is important. We used UK issued contactless bank cards in the test cases. We know the card's contactless transaction limit is £15 (at time of testing) and the number of consecutive contactless transactions is 5 (after which a PIN must be entered for authentication). We know that the currency of the card is UK pounds. Some of the card limits are unknown as the card does not divulge them (in particular the card's offline transaction limit and consecutive transaction limit).

B. PDOL Test Cases

At the start of a transaction the card issues the Processing options Data Object List (PDOL) which lists all of the data elements that the card requires the terminal to pass in the Get Processing Option command. The PDOL request contains the data element tag and the expected length of the data element. In test case (s1), for successful completion of the GPO command, the PDOL data returned by the terminal must (i) contain all of the data elements requested by the card (ii) all of the data elements must be of the correct length (iii) the data elements must be in the same order as requested in the list (iv) there must not be any additional data elements in the PDOL data.

Test (e1) sends a standard MasterCard PDOL response to a Visa card. MasterCard commonly uses an empty PDOL for Get Processing Options, which is represented by the command data field 83 followed by a length indicator of 00.

Test case (e2) involves sending transaction data in the PDOL that conflicts with the available usage settings of the card. This can be achieved by sending a cash back amount in the PDOL which conflicts with the usage of a contactless card.

Test case (e3) requires that the terminal capabilities exclude the valid transaction being agreed given the cards capabilities. The Terminal Transaction Qualifiers TTQ is included in the PDOL for Visa fDDA transactions to inform the card of the terminals capabilities. The card should decline a transaction where the TTQ values exclude the circumstances required for anfDDA transaction. Setting TTQ to include "Online cryptogram required" and/or setting the TTQ to exclude "EMV mode supported" will exclude the capabilities of the card which are required for an fDDA transaction.

Test case (e4) involves the terminal sending PDOL data which does not conform to the PDOL list requested by the card. The PDOL data can be corrupted in the following ways: (i) not including all of the data elements requested (ii) including data elements that are the incorrect length (iii) sending invalid data in the PDOL such as alpha characters where numeric data is expected (iv) including additional data elements in the PDOL data. In all cases, the card should produce an error code in response to the GPO command containing the erroneous PDOL data.

Results for Test Case (s1) - This is the success test case, as predicted by the abstract model the card approves the transaction and returns a valid SDAD.

```
Applications Available on Card

A0000000031010 <0>

SELECT(A0000000031010)

Contactless Transaction

Date     = 011212

Amount   = 5.00

Cashback = 0.00

Currency = 0826

--------- Kernel 3 Processing ----------------------

PDOL

8321BA20C0000000000005000000000000000826000000000008261212010061A0366B

-- TC Returned by Get Processing Options (9F10) ---

--------- Validating SDAD -------------------------

SDAD

0F4970C0FEAD97A63445C28A866760B449F20D51BC49161CD0A
598482FDDE5D35557D1C0B4B759262DEB8132AB66744EF69A5B
D196BF6C105C7E47FC735D72B2D2ABD2035049B3DA1F1ACC6A4
33F51ED71CD42E94D391B6A1B06CD98D50F37035689D4ECA1AA
F244AFE729B3AE7CD704

--------- TRANSACTION SUCCESSFUL ------------------
```

Results for Test Case (e1) - Sendinga valid MasterCard PDOL to a Visa card. The card rejects the PDOL with the response 6A80 Invalid Data which is as per the abstract model.

```
Applications Available on Card

A0000000031010 <0>

SELECT(A0000000031010)

Contactless Transaction

Date     = 000000

Amount   = 0.00

Cashback = 0.00

Currency = 0000

--------- Kernel 3 Processing ----------------------
```

```
PDOL

8300

Error [6A80] returned by GetProcessingOptions()

--------- TRANSACTION FAILED ----------------------
```

Results for Test Case (e2) - Transaction settings that conflict with the card usage settings, in this casethe cashbackvalue of the PDOLwas set to £5.00 which is not allowed for contactless transactions. The card acts incorrectly it ignores the cashback and approves the transaction for £5.02 in the amount field.This demonstrates that the test cases generated by the abstract model have found an anomaly.

```
Applications Available on Card

A0000000031010 <0>

SELECT(A0000000031010)

Contactless Transaction

Date     = 011212

Amount   = 5.02

Cashback = 5.00

Currency = 0826

--------- Kernel 3 Processing ----------------------

PDOL

8321BA20C000000000000502000000000500082600000000000
82612120100198E9611

-- TC Returned by Get Processing Options (9F10) ---

--------- Validating SDAD -------------------------

SDAD

67E1584CE27464EE9FE72A56B8898E50BE5F5B1BF43EC08B9DE
CB069BE75B54165E39FD4C761B4DD6ADB1CADC571AA31E2B3ED
755F54F68E47F7BFFB15FF8B549EE2A98D26836191FB14D45EA
B2705B4DC3AEC3D69150404FF9A3B9BC8BCF81B7C54C13A4A1B
4393D9FCE63C9C7C7F14

--------- TRANSACTION SUCCESSFUL ------------------
```

Results for Test Case (e3) – The terminal capabilities TTQ is set to exclude the offline capabilities of the card in an fDDA transaction. In this the card rejects the transaction when the terminal and card have incompatible capabilities with the error code 6984 Try Another Interface. This is as predicted by the abstract model.

```
Applications Available on Card

A0000000031010 <0>

SELECT(A0000000031010)

Contactless Transaction

Date     = 011212

Amount   = 5.00

Cashback = 0.00

Currency = 0826

--------- Kernel 3 Processing ----------------------

PDOL

80A800002383218080000000000000050000000000082608260
00000000000001212010067D0F6ED

Error [6984] returned by GetProcessingOptions()

--------- TRANSACTION FAILED ----------------------
```

Results for Test Case (e4)

We ran several test cases where the PDOL was not in the format requested by the card, in each case the card responded with the error code 6984 Try Another Interface. This is as predicted by the abstract model.

C. Foreign Currency Transaction Limits

The cards used in our testing were UK issued credit and debit cards operating in GB pounds, none of which were MasterCard.We set up transactions in Australian Dollars with values of $50.00 (£32.39), $101.00 (£65.43),$200.00 (£129.56), $300.00 (£194.34). All of the transactions were approved even thoughvaluewas greater than the £15.00 contactless transaction limit. The following log entry shows aAUD$300.00 transaction being approved by a contactless Visa card, approval is signified by the card returning a valid SDAD and Application Cryptogram for each transaction.

```
Applications Available on Card

A0000000031010 <0>

SELECT(A0000000031010)

Contactless Transaction

Date     = 120328

Amount   = 300.00

Cashback = 0.00

Currency = 0036

--------- Kernel 3 Processing ----------------------

PDOL = 8321BE20C000000000030000000000000000036000

000000000361203280020CF96C4

-- TC Returned by Get Processing Options (9F10) ---

--------- Validating SDAD -------------------------

9F4B =>7E1795D15F95EB7CB97F2550894B3871AB1DA56BA194

CD9217ACE931FC895FC8EA6B4A13732EA04CE6E96760785DE0F
67BDCDC1B243825206C91B178ED56474361A4E08EE555873998
AE2A3FA4DF260ABDFFB6B40E07D19898829AA2BE8B88118E9FA
2261D71E8B7F1C847E61F4490E5C2DE66D0CB3C0E64AE680D13
076DB871

--------- TRANSACTION SUCCESSFUL ------------------
```

This foreign currency example demonstrates that by creating the abstract model, we found an anomaly. This allowed us to generate a test case, to see how the emulator implementations handled this situation and in this case the emulator indeed handled foreign currencies as predicted by the formal model.

VII. EVALUATION

Evaluation of the research reported in this paper pertains to two facets: correctness of the emulator itself, and an assessment of the benefits and limitations of the process we

used in the design of the emulator. By the nature of this line of work, the evaluation is largely qualitative, based on the experiences of the researchers, although of course some objective results for the tests exist.

With respect to correctness of the emulator, the formally derived test casesreveal that the emulator code correctly handles the test suite. The formally derived test both show correctness with respect to the EMV standard as well as to observed anomalies in this specification. Additionally, we used the emulator throughout our research intopotential security vulnerabilities of the EMV protocols, and revisedthe code as needed for this use. Further confidence in the code can gained throughdeveloping with more extensive test suites, but this has been outside the scope of the current work.

Additional confidence in the quality of the code comes from the use of the design process we depicted in Fig. 1The systemic documentation of the code allows us to maintain a clear overview of the link between UML diagrams, the emulator code and EMV documents. This is especially important since transactions captured in a single UML sequence diagrams often span multiple EMV documents. Hence, code documentation assures that the link between code, UML and EMV remains firmly established.

Another important element in the design process is the creation of a formal abstract model that represents the EMV protocol, in our specific case the fDDA transaction. Construction of the formal model facilitated the identification of anomalies in the EMV specification (e.g., about limits in foreign currency) as well as the generation of test cases for fDDA. This demonstrates the importance of the feedback the emulator implementation received from the process of constructing an abstract model.

At the centre of our methodology is capturing the EMV transactions as UML sequence diagrams. As indicated in Fig. 2correctness (albeit informal) of the sequence diagrams is the key to correctness of the code. Three efforts feed into identifying the sequence diagrams, in an iterative process: (i) the EMV documents themselves; (ii) the insights obtained when constructing the formal model, and (iii) the insights obtained when coding the emulator. We believe that the combination of the three provide a scalable approach to developing an accurate and reliable emulation of the EMV protocol.

VIII. Related Work

The related work for the current paper can be naturally subdivided in work that relates to payment protocols such as EMV and work that relates to the design process of emulators and related software, including that based on the use of formal methods.

EMV is documented in the various EMV specification [1][2]documents, to which we have made reference throughout the paper. In addition, there is a body of research in the security of EMV protocol, including the search for vulnerabilities such as in [8][9]. However, for the current paper, the most relevant literature related to EMV and similar payment protocols is that on research that establishes simulators or emulators for EMV.

Several authors have used emulators as a tool to test applications early in the life cycle and to try out attacks, but none of the authors went to much length to argue the quality of the resulting emulators. It seems that this comes from the fact that the development of the emulator has a more specific purpose, and therefore a generic ‘soundness’ of the software is less sought for. As an example, Mohorko et al [10] present an emulator for a payment system called Margento, but do not discuss the underlying design method. This is understandable, since the purpose of the work was to demonstrate the practicability of the voice-based authentication system of Margento.

Our approach is inspired by elements of the Praxis approach to high-integrity development [3][17], but does not go to, for instance, the length of formal proofs. See Section III for a detailed scoping discussion. The prime use of formal methods is in generating test cases. However, as important is the deep and precise understanding we gained from constructing the formal model—this was an important source of feedback that we believe strongly improved the quality of our code. Obviously, elements of the Praxis approach have seen extensive use in industry, and also for EMV code there is a case for increased rigour in hardware and software development of systems and applications. This paper is a first step in that direction.

Closest related to our work is the SmartLogic tool set by De KoningGans and De Ruiter[14]. It implements a suite of hardware and software tools for the particular objective of executing attack scenarios. The solution results in a publicly available set of tools, including emulator software for some aspects of the protocol. There is no description of a (rigorous) design such as in this paper, and arguably, this is the consequence of SmartLogic’s aim to demonstrate it is a useful tool to analyze attack scenarios. Nevertheless, also in these cases, the design methods followed in this paper may be appropriate to help improve the quality of the emulator code.

Pushing the envelope with respect to the use of emulators is the work by Balfe and Paterson [12]. It provides a highly sophisticated use of an emulator as part of a payment system to assure that security properties of EMV extend to online payments. The emulator is embedded in a Trusted Platform Module. Again, the methods used for designing and implementing the emulator are secondary; the main trust of the paper is the design of a system to provide point of sale terminal equivalent security properties for online transactions (in which the card is not physically read).

For completeness, we discuss a quite different type of use of emulator or simulator code, such as by Bokai and Mahammadi[16]. Whereas an emulator substitutes for or replaces the physical incarnation of a protocol, simulation is a purely logic version of the protocol. The focus in simulation is not on direct equivalence with the protocols but

on finding the right level of abstraction at which to analyze the protocol. [16]does this for a micro payment scheme. Our emulator could potentially be used for usability studies or for visualization, but that is not its primary purpose. Therefore, the focus in the current paper is on directly mimicking the EMV protocol, without a desire to abstract away any details.

Related to the current paper in yet other ways are papers that are concerned with generating test cases. Devos et al [13] use software patterns for the identification of reusable test cases. Arguing that formal approaches are too expensive, reuse of test cases across application with similarities is being explored. Although the researchers constructed four versions of an implementation (from 'toy' to one with 20 commands), the focus of the paper was not on the methods behind the implementation of the emulator. Certainly, identifying patterns, for instance derived from generalization of our UML descriptions, would be an interesting avenue to pursue to identify test cases with good coverage. However, this was not within the scope of the current work.

Finally, we like to point the reader to related work in the formal verification of security properties of payment protocols such as EMV. This has a rich history, as becomes clear from the paper by Stepney et al on Mondex in 1996[4]. More recent, the EMV protocol has been subject to formal approaches. In the current paper, we used a formal abstract model that is detailed in a technical report [20] and can be expected to be published separately in future work. Related formal treatment of the EMV protocol can be found in [14], giving a formal analysis using F# and the ProVerif and FS2PV tools. Similarly, Pasupathinathan and colleagues [15]studied online authorization approaches such as Verified by VISA using Casper and FDR2 tools. We believe that the publications are only the start of a significant amount of additional formal treatment of protocols work that should be pursued for EMV and related protocols.

To summarize, the work in this paper is the first emulator for the EMV protocol that seriously considered high-integrity software development methods. Other emulators exist, but these serve a specific purpose (e.g., analyzing attack scenarios or particular protocol extensions) and possibly because of that are less concerned with the methods underlying the software development.

IX. CONCLUSION

In this paper we presented the design and implementation of an EMV emulator. The purpose of the emulator is to aid in the development of applications and new cards through rapid testing and to aid in the research of dependability (including security) related anomalies and properties. Therefore, we went through great length to follow a rigorous but practicable design and implementation process in the development of the emulator software. At the centre of the process is the identification of a set of UML sequence diagrams that as precisely as possible (within the limitations of UML) captures the EMV transactions. Three important sources work together to assist in constructing correct sequence diagrams: the EMV specifications, the insights gained from constructing a formal model and the insights gained from coding the emulator. The formal model also allowed for identification of anomalies in the EMV protocol and generation of relevant test cases and we showed that our code passed the test suite. The result of the work is an EMV point of sale terminal emulator that represents a highly accurate implementation of EMV protocols, with functionality as well as quality characteristics that is currently not present in any other EMV software.

REFERENCES

[1] EMV, Specifications for Payment Systems, Version 4.3, Books 1,2,3 and 4, November 2011.

[2] EMV, Contactless Specifications for Payment Systems, Version 2.2, Books A,B,C-1,C-2,C-3,C-4 and D June 2012.

[3] D. Cooper and J. Barner. Tokeneer ID station EAL5 demonstrator. Technical Report S.P1229.81.1, Altran Praxis, 2008.

[4] S. Stepney, et al. Mondex: An electronic purse. Program Research Group (PRG) 126, Oxford University, 1996.

[5] B. Meyer. Design by contract and the component revolution. In TOOLS (34), pages 515–518, 2000.

[6] Worldwide EMV Deployment. 2011. http://www.emvco.com/about_emvco.aspx?id=202. Accessed: 2012-04-11.

[7] EMVCo, White Paper on Contactless Mobile Payment, September 2011, avalable from http://www.emvco.com.

[8] S. Murdoch, S. Drimer, R. Anderson and M. Bond, Chip and PIN is Broken, IEEE Symposium on Security and Privacy, pp. 433-446, 2010.

[9] M. Pasquet and S. Gerbaix, The Complexity of Security Studies in NFC Payment System, 8th Australian Information Security Management Conference, 2010

[10] J. Mohorko, A. Chowdury and P. Planinsic, Emulation of Mobile Payment System, International Conference on Systems, Signals and Image Processing, pp. 21-24, 2008.

[11] G. deKoningGans and J. deRuiter, The SmartLogic Tool: Analysing and Testing Smart Card Protocols, 5th International Conference on Software Testing, Verification and Validation, pp. 864-871, 2012

[12] S. Balfe and K. Paterson, Emulating EMV for Internet Payments, Workshop on Scalable Trusted Computing, pp. 81-92, 2008

[13] N. Devos, C. Ponsard, J Deprez, R. Bauvin, B. Moriau and G. Anckaerts, Efficient Reuse of Domain-Specific Test Knowledge, International Conference on Software Engineering, 2012

[14] J. deRuiter and E. Poll, Formal Analysis of the EMV Protocol Suite, Theory of Security and Applications, LNCS Vol. 6993, 2012

[15] V. Pasupathinathan, J. Piperzyk, H. Wang and J. Cho, Formal Analysis of Card-Based Payment Systems in Mobile Devices, Fourth Australasian Information Security Workshop, 2006.

[16] M. Bokai and S. Mahammadi, Exploring Adoption of NetPay Micro-Payment: A Simulation Approach, Educationald and Information Technology, Vol. 3, pp. 245-250, 2010

[17] J. Barnes, Experiences in the Industrial Use of Formal Methods, 11th International Workshop on Automated Verification of Critical Systems, 2011.

[18] MasterCard, OneSMART Pre-Authorized Solution Guide, January 2005, available at http://www.m2mgroup.ma/livresetdocs/MasterCard/Smart%20Card/ZW-Entire_Manual.pdf

[19] J. Woodcock and J. Davies. Using Z. Prentice Hall 1998.

[20] [details of reference to technical report removed to facilitate double blind review]