A Methodology for Protocol Verification Applied to EMV® [3]

Leo Freitas[1], Paolo Modesti[2], and Martin Emms[1]

[1] School of Computing, Newcastle University, UK
[2] School of Computer Science, University of Sunderland, UK

Abstract. The EMVCo[3]organisation (*i.e.* MasterCard, Visa, *etc.*) protocols facilitate worldwide interoperability of secure electronic payments. Despite recent advances, it has proved difficult for academia to provide an acceptable solution to construction of secure applications within industry's constraints. In this paper, we describe a methodology we have applied to EMV1. It involves domain specific languages and verification tools targeting different analysis of interest. We are currently collaborating with EMVCo on their upcoming EMV®2^{nd} Generation (EMV2) specifications.

1 Introduction

In principle, payment protocols are designed to be secure, with adequate and effective cryptographic methods employed to ensure confidentiality, integrity, authentication, identification, *etc*. In practice, relevant attacks [9, 17, 19, 18] still occur in the industry, with financial fraud related to payment systems rising in the last few years: for example, in the UK, there has been a 80 percent increase in value of losses between 2011 and 2016, when the fraud losses were £618 million [24].

EMV, commonly termed *Chip & PIN*, is the dominant card based payment technology and is managed by EMVCo (www.emvco.com). In 2015, their protocols generated $US\$433$ billion in payments worldwide, protecting users from fraud and identity theft. Their protocols were designed to operate with cards being physically inserted into POS-terminal/ATM and used a wired connection to communicate. The introduction of EMV contactless made payments more convenient but created new security challenges as a wireless interface has been added to EMV cards and PIN entry has been waived.

More in general, with the recent publication of *PSD2* [13] and *Open Banking* APIs (openbanking.org.uk), regulation is pushing innovation. The payment/banking industry is being driven towards novel complex (cloud-based) protocols. Potential threats from systematic fraud are real. Thus, current development strategies would benefit from early safety-critical mindset.

Despite recent advances [15, 14], adequate solutions suitable for industry's problems, within its constraints, are still lacking. Solutions developed in academia are too complex to be used effectively by practitioners, who generally prefer to describe requirements in a way that is understandable by a wide range of IT professionals. Therefore, it is common to find in software requirement specification documents, even for

[3] EMV® is a registered trademark or trademark of EMVCo, LLC in the US and other countries.

large application such as EMV, an informal/semi-formal description based on natural language sentences and diagrams.

Analysis of EMV protocols is non-trivial due to the complexity of its requirements [22, 23]. They have to incorporate competing (and conflicting) interests from multiple issuers and from financial regulators worldwide. The introduction of contactless payments has significantly increased its complexity. While EMV contact (Chip & PIN) specification describes a unified payment protocol sequence (kernel) for all card types, the specification for contactless payments contains seven protocol sequences, one per card issuer. Complexity is reflected in expansion from four books (765 pages) for contact transactions [22], to additional ten books (1627 pages) for contactless [23].

This paper presents a new methodology used for the analysis of the safety and security of EMV's contactless protocols. It is illustrated by considering the security of contactless transaction protocols as stand-alone processes and the wider impact of contactless technology. Our key contribution is a structured analysis methodology involving various languages and tools that is tailored to EMV audiences/developers (*i.e.* acceptable to the industry partners we have been working with). Such methodology has identified and demonstrated the impact of vulnerabilities in the EMV1 protocols.

Related Work. Several works have investigated on various aspects of the EMV protocol suite. Some researchers focused on attacks on existing implementations. The discovery in [39] allows attackers to buy goods from retailers, whereas the discovery in [9] allows attackers to extract money from the victim's account. Relay attacks [17, 18] allow fraudulent transactions to be collected from contactless cards without the knowledge of the cardholder. In the area of formal methods, the first comprehensive formal description of EMV [15] used an F# model translated to Applied-pi [6], in order to make it amenable for analysis with the *ProVerif* verifier [7]. This analysis confirmed all known weaknesses without revealing any new vulnerabilities. [14] proposes an open specification of an EMV-compliant protocol that can be securely used on mobile devices, even if infected by malicious applications. The model is validated with Tamarin [40]. Other works [10, 30] have analysed cryptographic aspects, focusing on protocols for secure key agreement and channel establishment. Finally, we worked on a Z encoding of Kernel 3 [25] that lead to new discoveries [18, 20].

Contribution. Our work complements other formal approaches, as we benefit from existing verification techniques (*e.g.* model checking), but provides additional insights that cannot be captured by protocol verifiers. For example, the capability to formally specify user-defined functions in VDM-SL [33] and validate a model, in the protocol specification language *AnB* [36], against them. We also abstract from the underlying cryptographic aspects, assuming specific primitives are made available according to the protocol specification.

We use abstract (uninterpreted) functions in *AnB* to represent control flow and problem-related state updates, which the declarative nature of the *AnB* language lacks. Such functions can be formally specified, and are indeed implemented (*i.e.* function for public key of agents `pk` in *AnB* is only defined by different tools' implementations). This approach is inspired in their successful use by the CSP model checker FDR, and SMT-Lib decision procedures and predicate solvers. Beyond their implementation in

different tools, we formally specify the functions behaviour in VDM, the same language that is being used here for the formal specification of *AnB*. This is different from other approaches [1, 2, 5], which compile *AnB* to different languages (*e.g.* CSP [32], operational strands [31], IF [3]), with a formally specified compilation strategy/set of rules which consider uninterpreted functions only symbolically.

In our case, as with the semantics of *AnB*, these functions are also specified to be executable, hence enabling symbolic simulation of various protocols of interest. Presence of such functions does not compromise security goals checked, given that the intruder has access to them just like it has to access to symbolic public (non-cryptographic) functions in the abstract model. Therefore, all the intruder can do is to ask the environment to perform the computation and get the result. We think this is important to set the scene and clarify the scope of this methodology.

Outline. In §2, we present an overview of our methodology, in §3 we introduce the specification language *AnB* through a simplified version of the EMV1 Kernel and present the method applied to the full Kernel 3 including key user-defined functions linked to the underlying verification environment in VDM. §4 presents our findings, and §5 summarises our results and discusses future work.

2 Methodology

Working with the payment protocols industry motivated the development of the methodology described here. It is a variation and extension of a successful application of rigorous/formal reasoning [18, 21] applied top-down.

Our approach applies model-based techniques for the formal specification and verification of protocol requirements and designs. It focuses on the construction by protocol designers of a declarative description of protocol sequences using the *AnB* (Alice & Bob) notation [36] (see §3.1). This model is used to investigate the consistency of requirements, identify descriptive errors, and generate test cases for our POS-terminal emulator capable of performing transactions with both EMV contact and contactless protocols.

This has proved to be effective in both documenting decisions precisely [20, 21, 25], and detecting significant protocol flaws early in the development process [19, 18], way before deployment or actual implementations. Specifically, it is a variation of a successful industry approach by Praxis (now Altran UK, see www.adacore.com/sparkpro/tokeneer), where formal specifications are used to clarify requirements and then later used to inform its design and implementations. That is, the sound rigour of the formalism does neither hamper the user's experience nor impose unrealistic expertise, yet provides a number of important verification outcomes and challenges (*i.e.* proof obligations) of interest. To illustrate the process, we analysed one of the EMV1 contactless kernels [23]. The process is depicted in Figure 1, and is explained below.

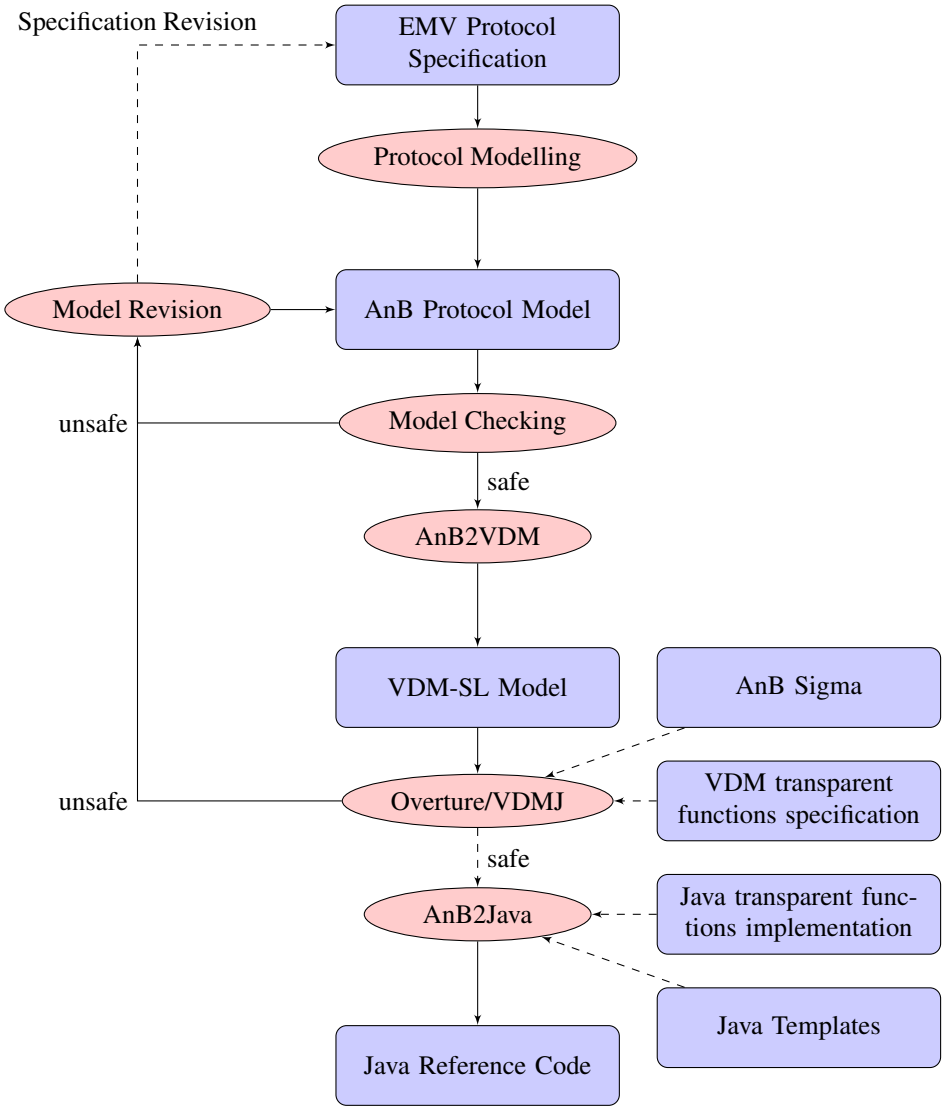

Fig. 1. Methodology

2.1 *AnB* Protocol modelling

Starting from the requirements in natural language (*EMV Protocol Specification*), protocol developers in industry produce flowcharts and UML-sequence diagrams describing protocol message exchanges and information flow. We introduce *AnB* to stake holders in order to systematically capture these exchanges between entities involved in a declarative fashion. Given *AnB*'s simplicity, our experience shows that the process is

generally fast and straightforward. The model (*AnB Protocol Model*) is then used to check message-format consistency (*i.e.* messages sent are within agents' knowledge), and to verify security properties such as secrecy and authentication goals.

In our settings, we have used the OFMC model-checker [4]. If an error is found on the message-formats, it is likely that the developer has made one or more errors in the encoding of the *AnB* model. For example, some message exchanges specification could have been misinterpreted, w.r.t. the specification document, and therefore a model revision may be necessary.

If these consistency checks are successful, the model checker will try to verify the protocol against the security goals. If the goals are violated (i.e. the intruder may attack the protocol and the protocol is *unsafe*), a revision of the specification may be again necessary. If a fix is found amending the *AnB* model, then the same will have to be incorporated in the original specification.

Such iterative process may require several steps and it is aimed at building a correct and reliable *AnB* model that can be used in the next phases of the methodology.

2.2 VDM Protocol modelling

We compile *AnB* protocols to VDM-SL [33] to automatically obtain a *VDM-SL model* of protocols. These VDM models of *AnB* protocols can be symbolically evaluated by the formal language semantics of *AnB* described in VDM (*AnB Sigma*). This provides a formal characterisation of the *AnB* protocol, where the underlying program state in VDM represents the accumulated knowledge accreted as a result of performing a protocol run, which also includes what the intruder is capable of knowing in the worse case. This semantic encoding of *AnB* enables knowledge computation that is symbolic, and a notion of intruder model that is directly linked with (and limited by) the language semantics.

It should be noted that *AnB* is not expressive enough to capture control flow or to represent important computations explicitly. Instead, the user can define abstract function symbols to be used by the verification tools in implementation-dependant manners. For example, if the user defines a new function `foo` to perform some computation, what most tools will do is to ensure the function types/results are correct and execute as `Skip`. We make use of these user-defined functions (*VDM transparent functions specification*) to link the *AnB* model with the protocol's underlying required state and specific computations, by formalising their meaning.

Therefore, a protocol/formal-methods expert can write the library, providing executable formal specifications for functions involved in the protocol, and its underlying state once, which can then be reused across a number/family of protocols. For example, EMV1 contactless payment protocols are defined in kernels per issuer-specific implementation (*e.g.* Kernel 3 presented in §3 is Visa NFC).

The VDM model of the protocol is used for symbolic simulation of protocol runs, as well as test case specification (with the *Overture* and *VDMJ* tools).

2.3 Protocol implementation and tool support

We currently use the *AnBx* Compiler and Code Generator [37] to translate the *AnB* model to Java reference code. The compiler uses template files (*Java Templates*) which are instantiated with the protocol logic and the concrete Java implementation of the transparent functions. The compiler can also apply optimisation techniques in order to minimise the number of cryptographic operations and reduce the execution time [38].

The same tool is also used for translating the *AnB* model to VDM-SL. Moreover, in order to facilitate the adoption by practitioners, we have developed an Eclipse-based IDE [29] supporting many tools used in this methodology.

3 Case Study - EMV1 Kernel 3

In this section, we present the methodology through a case study.

3.1 AnB language

The *AnB* language [36] is a simple, abstract, and declarative language, where *AnBx* [11, 12] is a syntactic extension including various useful patterns of use and a stronger type system for user-defined (abstract) function symbols. Its key feature is the declaration of **`Agents`** representing protocol actors, **`Actions`** representing message exchanges between agents, and **`Goals`** representing desired properties of interest for the messages exchanged to have. Action messages are described in a simple expression language that include user-defined function symbols, some of which are security protocols primitives like private/public keys functions.

A protocol has always five sections: i) its name; ii) declaration of agents, types, and user/pre-defined functions; iii) declaration of initial knowledge for all declared agents, which implicitly include self-awareness (*i.e.*, agent `C` knows its own identity); iv) declaration of actions as message exchanges between agents, where various criteria/restrictions (described below) apply; and v) declaration of desirable security goals, which include messages being kept secret between agents and agents being authenticated by other agents on specific messages. An optional section with reusable (non-recursive) **`Definitions`** provides local (let-style) abstractions for names.

Figure 2 depicts a simplified version of our EMV1 contactless implementation: the simplification is the lack of many user-defined functions, which will appear in §3. This is a deliberate simplification (EMV1 has other complexities, as illustrated in [15], and elided here): our point is to illustrate that with an abstracted version of the protocol, certain security goals of interest are already breached (see §4.1).

To complete a payment, a card `C` needs to endorse the payment information (`ACPayload`) and this can be verified by both the terminal `T` and the card issuer `iss`. The card never exchanges messages directly with issuers, but only using the terminal as an intermediary.

In `Types` section we define the following identifiers: the `Agents` (`C`,`T`,`iis`), a component of the payload `PDOL` which abstracts information about the payment (e.g. amount, date, etc) and a `Nonce`. The issuer identifier (`iis`) is a constant (first letter lowercase), therefore, by convention in *AnB*, it is considered trusted. We assume that the

```
Protocol: emv_visa_k3_simple AnB
Types:
    Agent C,T,iss;
    Number PDOL,Nonce,empty;
    Function [Agent, Number -> SeqNumber] fcnSeqNo;
    Function [Agent -> PublicKey] sk;
    SymmetricKey ShkCiss
Definitions:
    ACPayload:  T,PDOL,fcnSeqNo(C,Nonce);
    AC:         {|ACPayload|}ShkCiss;
    SDAD:       {ACPayload}inv(sk(C))
Knowledge:
    C: C,iss,sk,inv(sk(C)),fcnSeqNo;
    T: T,sk;
    iss: iss,C,sk,empty;
    C,iss agree ShkCiss
Actions:
    T -> C: T, PDOL
    C -> T: SDAD,AC,C
    T -> iss: PDOL,AC,C
Goals:
    T weakly authenticates C on ACPayload
    iss weakly authenticates C on ACPayload
    ShkCiss secret between C,iss
    inv(sk(C)) secret between C
```

Fig. 2. *AnBx* Protocol Example

issuer systems are uncompromised. Variable identifiers (first letter uppercase) of type `Agents` can be impersonated by the intruder, while variables of type `Number` represent abstract values which are different at every run of the protocol.

To illustrate the use of function abstraction, `fcnSeqNo` is used to represent the capability of agents to generate sequence numbers (used typically in the interaction between the card and the issuer). Public key cryptography is modelled using a function `sk`, mapping agents to public keys, with the purpose of representing the public keys used for digital signature. Therefore, `sk(C)` is the public key of agent `C`, whilst `inv(sk(A))` is the corresponding private key. This key is is added to the initial knowledge of agent `C`, while function `sk` is known by all agents representing the capability of retrieving a certified public key from a keystore or public repository.

Use of cryptographic expressions is shown in section `Definitions`. `AC` represents the payload (`ACPayload`) encrypted with `ShkCiss`, the symmetric key agreed between the card and the issuer. We assume that the issuer and the card had agreed on a session key (`ShkCiss`) prior to the protocol run. An example of asymmetric encryption is `SDAD`, the digital signature of the payload, obtained encrypting the `ACPayload` with the private key of the card.

The actions are written in order of exchange, for a message from the source to the target agent. The next action must always be a response from the target of the previous action to another agent.

In the first action, the terminal sends its identity and the information about the payment options back to the card. We have shortened the (application selection) EMV1 protocol sequence here for simplicity. Then the card replies with the digital signature of the payload, which is encrypted with their pre-shared key (`ShkCiss`). The digital sig-

nature is used by the terminal to authenticate the card, while the ciphertext is forwarded to the card issuer, which uses the pre-shared key to authenticate the card, validate and authorise the payment request.

This description of the protocol actions, describes how knowledge is accreted as a result of the protocol execution. Crucial to this process is the intruder knowledge. It is characterised as what is knowable by a malicious party attempting to interfere with the protocol by either impersonation or passively listening to communication. Different tools will define different knowledge acquisition rules for the intruder, which will determine its threat capability, where a commonly implemented approach follows the Dolev-Yao model [16].

The `Goals` section can specify goals of the following type:

Weak Authentication : `B weakly authenticates A on M` and are defined in terms of non-injective agreement [34];

Authentication : `B authenticates A on M` and are defined in terms of injective agreement on the runs of the protocol, assessing the freshness of the exchange;

Secrecy : `M secret between A1,...,An` and are intended to specify which agents are entitled to learn the secret message `M` at the end of a protocol run.

For EMV1, we illustrate goals of interest based on our understanding of the protocol. The first two goals represent the terminal and the issuer authenticating the card on the payment information endorsement (`ACPayload`): payment authorisation request is made by a legitimate card and not by an attacker. Moreover, the protocols would like to achieve freshness, i.e., the same authorisation request cannot be used twice and the issuer is able to link each authorisation with the correct payment request. The two final goals states that various keys must remain secret.

3.2 From specification to the model

Following our methodology, after the user writes the *AnB* protocol that is well-formed (*i.e.* no type or name violations, knowledge provisos for message exchange are valid, *etc.*), we translate it to its VDM semantics. For the user-defined functions, VDM library implementations are required. In what follows, we describe the complete *AnB* model for EMV1, including these abstract functions, together with an excerpt of the underlying VDM state and abstract functions module.

EMV1 Visa kernel 3 contactless requirements are given in [23]. The document describes the Visa-specific contactless protocol, which is a variation/simplification of other common kernel features [22].

First, we read and understood these requirements. In the practice we envisage, protocol experts ought to know (or at least have a good idea of) what they are trying to describe. Second, we thought about what security goals would be of interest. Different from other protocols, such as Mondex [28], where security goals were clearly defined from the outset, such goals are not explicitly declared in the EMV1 requirements [22], yet we assume practitioners will know what goals to check; we added some we found relevant below in §4.1. Finally, before constructing the *AnB* model for EMV, we create UML sequence diagrams to illustrate the key stages/players as described by the requirements.

3.3 EMV1 Kernel 3

Our model of *EMV Visa kernel 3* considers three agents: a card issuer `iss`, a card `C` and a terminal (i.e. card reader) `T`. When the card is issued, it is preloaded with a unique pre-shared asymmetric key that is used to run a key agreement protocol which generates the session key `ShkCiss`, whose computation is based on the transaction counter. The pre-shared key is known only by the card issuer and the card itself. The session key can be used to ensure that the communication between card and issuer is secure, even though the two agents during a protocol run never exchange messages directly but only through the terminal. The protocol, using the DDA authorisation technology, assumes that the issuer can be trusted. In other words, it is assumed that the issuer systems are not compromised by the intruder, and that for legitimate cards pre-shared keys and session keys with the issuer are stored securely and kept secret.

Along with the ability to encrypt data with the session key, the card is also able to digitally sign messages that can be used to authenticate the card with the terminal. We use a number of abstraction functions in *AnB* actions to represent protocol functionality beyond simple message exchanges between parties through user-defined functions (*e.g.* `fcnAgree, fcnCVM, fcnUsage`, etc). We also use definitions to name commonly used message expressions (*e.g.* `CardVisaCap, TermTransVisaCap`, etc). We illustrate here the actions performed during the protocol run, and we will describe the result of the security analysis in §4.1.

1. `T` $\rightarrow$ `C: cmdListApps`
   The terminal asks the card the list of applications (`cmdListApps`) (i.e. different EMV Kernels) it is able to run (*e.g.* Visa, MasterCard, etc.).
2. `C` $\rightarrow$ `T: C,respVisa`
   The card replies with its identity and the application it intends to run, in this case Visa - `respVisa`. A card could store different kernels for different card issuers.
3. `T` $\rightarrow$ `C: fcnSelect(C,respVisa)`
   The terminal tells the card to start running the Visa program, by sending the selection message `fcnSelect(C,respVisa)`. `fcnSelect` is a function implemented by the EMV kernel, and it abstracts the request as part of the message exchange.
4. `C` $\rightarrow$ `T: CardVisaPDOL,CardVisaCap`
   The card replies with the meta information it requires for engaging in a transaction (*i.e.* `CardVisaPDOL` contains transaction date, amount, currency, country, etc.) and what verification methods the card is capable of performing. That is, `CardVisaCap` contains the card verification methods (CVM) to be used (*e.g.* online pin number, or offline signature, etc). It is defined as `fcnCVM(C,respVisa)`, where `fcnCVM` abstracts the card computing its verification method for the requested application.
5. `T` $\rightarrow$ `C: PDOL,TermTransVisaCap`
   The terminal replies with the actual `PDOL`, a list of values corresponding to the `CardVisaPDOL` request list (`PDOLDate`, `PDOLAmount`, `PDOLCountry`, `PDOLCurrency`, etc). It includes the unpredictable number `PDOLUPN` generated by the terminal and used to identify the transaction. Moreover, the terminal send to the card `TermTransVisaCap`, which is the intersection between the general terminal capabilities as well as transaction specific capabilities. It is defined by the abstraction function `fcnCVM(T,fcnCVM(T,TermUsageValid))`, where `TermUsageValid` is defined as

`fcnAgree(fcnUsage(C,respVisa),fcnUsage(T,respVisa))`. That is, `fcnUsage` returns the usage scenarios for both card and terminal for the requested Visa Kernel 3 application, whereas `fcnAgree` ensures that there is an agreeable choice between them. Usage examples include information where the transaction is taking place (*e.g.* shop, ATM, etc.) as well as kernel-specific limits (*e.g.* maximum value limit before going online, maximum overall limit per contactless transaction, *etc.*). The innermost application of `fcnCVM` to the result of usage agreement between card and terminal determines the transaction-specific capabilities between both parties, whereas the outermost application of `fcnCVM` ensures this transaction-specific agreement is within what's generally possible for the terminal.

6. `C` $\rightarrow$ `T: fcnAgree(CardVisaCap,TermTransVisaCap),SDAD,AC`
   The card is now ready to start the transaction, provided the capabilities between card and terminal within chosen transaction are agreed (`fcnAgree(CardVisaCap,TermTransVisaCap)`). The other two components are `SDAD` and `AC`. `SDAD` is a message digitally signed by the card containing the application cryptogram payload (`ACPayload`), which is composed by the terminal transaction data required by the card (`PDOL`), the card unique sequence number (`CSN`) per application, and the card unpredictable number `CUPN`. The second component is the application cryptogram (`AC`), which is a ciphertext of the `PDOL` and `CSN` encrypted with the session key `ShkCiss`, and it serves as an acknowledgement from the card held by the terminal. Since the key is only known to the card and the issuer, only the issuer will be able to decrypt this message.
7. `T` $\rightarrow$ `iss: PDOL,AC,C`
   In the last step, the terminal forwards the information to the issuer including the `PDOL`, `AC` and the card identity. This represents the terminal "cashing" in its "I owe you"'s given by the card for the transaction.

3.4 EMV1 user-defined functions in VDM

The user defined functions for our EMV1 model are interpreted/implemented within our VDM formal semantics of *AnB* abstract functions, which include cryptographic primitives (*i.e.*, the default environment for our *AnB* semantics specify `pk`, `sk`, *etc.*). They exist in the context of certain types and a global state.

Listing 1.1 provides the highlights of types defined for our *AnB* abstract functions for EMV1. We abuse the VDM notation slightly here: record fields of the same type are separated by commas to save space; "..." represent types/invariants that may have more fields/predicates. These types (and functions), have been thoroughly investigated in [25] and key findings were presented in [18].

For instance, the `Card` type models the card identity (*i.e.* its 16-digit number), its card verification methods per application, and if a card has more than one application (*e.g.* Visa Debit, MasterCard Credit). The verification method is defined by choosing the adequate `CVM` (Cardholder Verification Method) value (*e.g* online pin-verified, offline signature, *etc.*) and usage value (*e.g.* POS-terminal, ATM, *etc.*).

Note this abstracts away underlying cryptographic verification technologies actually employed like CDA, SDA, or DDA. That means we are assuming the *AnB* cryptographic primitives' implementation would be assigned to one of these schemes, which makes

them transparent (and orthogonal) to what we are capturing, which is the flow of control and information between protocol entities in order to ensure certain security goals.

```
types Id = seq1 of char;
  Card :: id: Id    cvm: map App to CardCVM ...
    inv mk_Card(-, cvm) == cvm <> {|->};
  CardCVM ::  x, y: nat     cap: seq1 of (CVM * Usage)
    inv mk_CardCVM(x, y, cap) ==  x <= y and
      (forall i in set domain[CVM, Usage] (elems cap) &
        i subset VALID_CVM) and
        range[CVM, Usage] (elems cap) subset VALID_USAGE;
  Terminal ::   id: Id    use: set1 of Usage
    cvm: map App to CVM
    inv mk_Terminal(-, cvm, use) == cvm <> {|->} and
      (forall i in set rng cvm & i subset VALID_CVM) and
      use subset VALID_USAGE;
```

Listing 1.1. VDM types used by EMV1 *AnB* user-defined functions

The *AnB* type system is too weak to represent what we need, hence the need for using VDM types. These types provide an interpretation for the space of possible values for the parameters of *AnB* functions (see Listing 1.2) like `fcnSelect`, `fcnUsage`, *etc*. Other details aside, the point of Listing 1.2 is to illustrate how we concretely represent *AnB* abstract functions within our method framework.

This VDM model needs to be written once (most likely by a formal methods expert) for a variety of EMV1 protocols. Given EMV invariants are not mathematically deep, but rather simple if numerous (something common in industrial models), the required expertise is not onerous. We tested this hypothesis by having a good MSc student without formal background develop most of the necessary EMV1 user-defined libraries for another EMV protocol (relay resistance) [35].

```
fcnSelect(a: Agent, app: App) r: App
pre (is_Card(a) => app in set dom a.cvm) and    post r = app;
    (is_Terminal(a) => app in set dom a.cvm)

fcnUsage: (a: Agent, app: App) r: (App * Usage)
pre pre_fcnSelect(a, app)
post r.#1 = app and (is_Card(a) => r.#2 in set (range (elems a.cvm(app).cap))) and
      (is_Terminal(a) => r.#2 in set a.use);
```

Listing 1.2. *AnB* user-defined functions for EMV1 given in VDM

For instance, function `fcnSelect` insists that the chosen agent application must have cardholder verification methods (CVM) validation criteria in order for it to be selected, and given this test passes, the result is the given application. This illustrates the underlying checking, in this case rather simple, under which conditions the first stage of the protocol can operate.

In general, the precise documentation of various conditions of all EMV protocol stages is at the heart of our methodology. Cumulatively, this forms the compound conditions for every specific successful protocol run. More importantly, it can also be used for further investigation through test case generation that is minimal with accountable

coverage, or to have proof obligations about the individual satisfiability of each step (*i.e.* are the given contracts sound?).

For instance, in order to query the expected usage for a chosen agent application (`fcnUsage`), the input parameters must have passed the conditions for selection (`pre_fcnSelect`) first. That being the case, the result is the usage present within the corresponding agent type structure. This use of (precondition) referencing makes protocol state dependencies easily accounted for.

Overall, we model these functions and their underlying state (*i.e.* card, terminal, and transaction internal states), hence encoding all necessary invariants, pre/postconditions for the protocol's functional correctness.

This is different from OFMC because we can make claims about the underlying expected protocol specification via the VDM functional correctness model of protocol transparent functions and state. Moreover, when problems are discovered by a VDM simulation or proof, we have to identify whether this is a problem for *AnB* (*i.e.* something about the information flow that OFMC missed), or a problem within the transparent function's behaviour themselves. The former strengthens the verification capability of *AnB* tools, whereas the latter strengthens and precisely documents the underlying assumptions about required transparent functions.

Given the nature of their use, the specification of these transparent functions are a more onerous job. That is because they are capturing the underlying system state and updates, and their verification will require theorem proving, yet these can be done once for a family of *AnB* security protocols. A concrete example of such verification of transparent functions for a family of protocols are the public-key cryptography primitive functions in *AnB* (*e.g.* `pk` and `inv`), which must represent an injective relationship between public and private keys.

Because we have a direct formal semantics of *AnB* in VDM as well, a number of further verification and specification opportunities arise, and we are working on integrating those within our methodology in Figure 1.

4 Results and Extensions

In our simulation environment for the *AnB* language semantics, we have explicit definitions for the pre-defined (cryptographic) and user-defined (EMV1 protocol specific) transparent functions, as well as the other parts of the EMV1 common kernel can perform. This enables us to perform a number of interesting analyses. This model is used to investigate the consistency of requirements, identify descriptive errors, and generate test cases for our POS-terminal emulator capable of performing transactions with both EMV contact and contactless protocols.

The *AnB* protocol semantics, defined in VDM, provides the knowledge accumulated by each agent, including our model of the intruder capabilities, as a result of executing the protocol according to the language semantics. This is a different strategy from other *AnB* tools [4, 5]: we explicitly model allowed behaviours by each *AnB* program constructs as defined by the language semantics in VDM, rather than by observing exchanged messages in a translated (IF-notation [4]) format. Arguably, our approach prioritises safety before security. Another way to look at such differences is that we do not

encode *AnB* in other languages (*e.g.* CSP [34] or IF-notation) for these languages' tool consumption and limitations. Instead, we represent the underlying *AnB* semantic state and operational semantics transitions using VDM (*i.e.* our approach consists among other things in defining the semantics of *AnB* in VDM-SL).

We derive VDM test cases to exercise key (or if possible all) specification entities (*e.g.,* type invariants, pre and postconditions, *etc.*). We also derive proof obligations that if/when discharged demonstrate the correctness of the model as a whole: without the proofs we have a debugging/testing tool, whereas with such proofs we have a verified functionally correct abstract execution environment for the specific EMV1 protocol of interest.

We have already worked with the Isabelle/HOL theorem prover to discharge VDM proofs, and are currently translating key aspects of EMV's infrastructure to be proved. To ensure proofs in different logics (*i.e.,* VDM's LPF and Isabelle's HOL) can be addressed properly, we follow ideas from [41, 27].

Moreover, we can also use other available tools to perform further security analysis (e.g. the *AnBx* compiler translates *AnB* to ProVerif [8]), as well as code generation for a concrete implementation in Java that we plan to deploy to real terminals and run on real cards, all this within our Eclipse-based *AnBx* IDE [29]. Overall, all this gives a valuable exploration and testing platform for protocol experimentation and prototyping. These implementations and their generated test cases can also serve as oracles to real implementations including the myriad complexities involved in an actual EMV1 protocol stack.

4.1 Security Analysis

The security goals we considered in our modelling (§3.3) are exactly the four goals described in Figure 2. The only difference is that for the authentication goals we consider `SDADPayload`, which contains few more components than just `ACPayload`.

We analysed the protocol with the OFMC model checker. This tool uses the AVISPA Intermediate Format IF [3] as "native" input language, which allows to describe security protocols as an infinite-state transition system using set-rewriting. The major techniques employed by OFMC are the lazy intruder, which is a symbolic representation of the intruder, and constraint differentiation, which is a search-reduction technique that integrates the lazy intruder with ideas from partial-order reduction achieving a reduction of the search space associated without excluding attacks (or introducing new ones).

As the terminal is capable of engaging only in one session at a time, we tested initially the protocol run only for one session. For each goal, our findings were:

- Goal: `T authenticates C on SDADPayload`
  This goal implies that the terminal is able to authenticate the transaction request endorsed by the card. This goal seems problematic to achieve for two reasons. First, the terminal does not always have an identifier that can be sent by the terminal to the card. Second, the `ACPayload` that is signed with the private key of `C` does not include the terminal identity `T`, therefore it is not possible to prove the injective agreement, i.e. the intention of the card to endorse a message intended for the terminal. A possible fix, if the terminal has an identifier, is simply to add `T` to the

definition of `ACPayload`. Moreover, at the first step (as in the protocol in Figure 2) the terminal should send its identifier to the card. This vulnerability can be exploited as shown in attacks, such as [9]: the attacker pre-generates authorisation codes on its own terminal, then goes to a merchant and replies that authorisation code. This will trick the merchant terminal to accept the transaction, hence enabling goods to be taken for free.

- Goal: `iss authenticates C on ACPayload`
  This goals implies that the card issuer is able to authenticate the transaction request endorsed by the card. We found that this goal is satisfied. Therefore, the asymmetric encryption mechanism (using the session key `ShkCiss`) seems sufficiently robust.
- Goal: `ShkCiss secret between C,iss`
  This goal implies that the session key `ShkCiss` is kept confidential during the protocol run. This goal is also satisfied, as the key never leaves the card during the protocol execution. Therefore, as long as the card is uncompromised (*i.e.* keys are stored securely) this goal is satisfied.
- Goal: `inv(sk(C)) secret between C`
  This goal implies that the private asymmetric key of the card used for signature remain secret after the protocol run. Again, this goal is satisfied for the same reason of the previous goal.

We also tested the protocol for two parallel sessions, and found that the second goal can be satisfied only for the weak authentication, but not for the injective agreement. Since a terminal does not engage in parallel session, this is not a problem for now, but if, in the future, terminals will be able to handle contactless payments in parallel, this might be a matter of concern, as a transaction authorisation can be used twice, unless mechanisms of prevention are enforced (e.g. counters).

5 Conclusion and Future Work

In this paper, we presented a new methodology that puts industry-accepted languages (*AnB*) and state-of-the-art formal reasoning tools (OFMC, ProVerif, Overture/VDM, Isabelle/HOL, *etc*) to the analysis of payment protocols. In particular, this is motivated by our current work collaborating with EMVCo on the development of their upcoming EMV®2^{nd} Generation (EMV2) specifications. Given the current confidentiality of EMV2, we illustrated the methodology with EMV1 Kernel 3 for contactless payments. Many of the tools and techniques presented are not new. The key of what is novel is a mixture between how we put these tools together, what new theories (*e.g.* executable formal semantics of *AnB* in VDM), and new tools (*e.g.* formal simulation of EMV1&2 kernels) are being used by industry.

With the upcoming developments in the "FinTech" industry as a result of not only EMV2 but also PSD2 and Open Banking APIs, our aim is to enable the dependable development of payment protocols faster and cheaper, with an accountable demonstration of why that is the case. This follows in the footsteps of a proven approach by Altran Praxis: we took considerable inspiration from their work on Tokeneer, as well as our own work on Mondex [28].

We are currently working on publishing the details of the novel *AnB* semantics, which will enable an extended set of goal verifications automatically beyond what is possible at the moment. Moreover, we could also work on the automatic generation of the reference implementation, including the user-defined functions, from the VDM model rather than directly from the *AnB* model. The workflow presented in Figure 1 is currently being applied within the EMV2 protocol. Further details appear in another paper in these proceedings [26], yet a number of technical details had to be removed until EMV2 becomes public due to non-disclosure agreement restrictions.

References

1. Almousa, O., Mödersheim, S., Modesti, P., Viganò, L.: Typing and compositionality for security protocols: A generalization to the geometric fragment. In: ESORICS, Proceedings, Part II. pp. 209–229. Springer (2015)
2. Almousa, O., Mödersheim, S., Viganò, L.: Alice and Bob: Reconciling formal models and implementation. In: Programming Languages with Applications to Biology and Security, pp. 66–85. LNCS 9465, Springer (2015)
3. AVISPA: Deliverable 2.3: The Intermediate Format (2003), avispa-project.org
4. Basin, D., Mödersheim, S., Viganò, L.: OFMC: A symbolic model checker for security protocols. International Journal of Information Security 4(3), 181–208 (2005)
5. Basin, D., Keller, M., Radomirovic, S., Sasse, R.: Alice and Bob meet equational theories. In: Logic, Rewriting, and Concurrency, pp. 160–180. LNCS 9200, Springer (2015)
6. Bhargavan, K., Fournet, C., Gordon, A.D., Tse, S.: Verified interoperable implementations of security protocols. In: IEEE Computer Security Foundations Workshop (2006)
7. Blanchet, B.: An efficient cryptographic protocol verifier based on Prolog rules. In: Computer Security Foundations Workshop, IEEE. pp. 0082–0082. IEEE Computer Society (2001)
8. Blanchet, B., Smyth, B., Cheval, V.: ProVerif 2.00: Automatic cryptographic protocol verifier, user manual and tutorial (2018)
9. Bond, M., Choudary, O., Murdoch, S.J., Skorobogatov, S., Anderson, R.: Chip and skim: cloning EMV cards with the pre-play attack. In: S&P. pp. 49–64. IEEE (2014)
10. Brzuska, C., Smart, N.P., Warinschi, B., Watson, G.J.: An analysis of the EMV channel establishment protocol. In: CCS. pp. 373–386. ACM (2013)
11. Bugliesi, M., Modesti, P.: AnBx-Security protocols design and verification. In: Automated Reasoning for Security Protocol Analysis and Issues in the Theory of Security: Joint Workshop, ARSPA-WITS 2010. pp. 164–184. Springer (2010)
12. Bugliesi, M., Calzavara, S., Mödersheim, S., Modesti, P.: Security protocol specification and verification with AnBx. Journal of Inf. Security and Applications 30, 46–63 (2016)
13. Cortet, M., Rijks, T., Nijland, S.: Psd2: The digital transformation accelerator for banks. Journal of Payments Strategy & Systems 10(1), 13–27 (2016)
14. Cortier, V., Filipiak, A., Florent, J., Gharout, S., Traoré, J.: Designing and proving an EMV-compliant payment protocol for mobile devices. In: EuroS&P. pp. 467–480. IEEE (2017)
15. De Ruiter, J., Poll, E.: Formal analysis of the EMV protocol suite. In: Joint Workshop on Theory of Security and Applications. pp. 113–129. Springer (2011)
16. Dolev, D., Yao, A.: On the security of public-key protocols. IEEE Transactions on Information Theory 2(29) (1983)
17. Drimer, S., Murdoch, S.J., et al.: Keep your enemies close: Distance bounding against smartcard relay attacks. In: USENIX security symposium. vol. 312 (2007)

18. Emms, M., Arief, B., Freitas, L., Hannon, J., van Moorsel, A.: Harvesting high value foreign currency transactions from EMV contactless credit cards without the PIN. In: CCS. pp. 716–726. ACM (2014)
19. Emms, M., Arief, B., Little, N., van Moorsel, A.: Risks of offline verify pin on contactless cards. In: Financial Cryptography and Data Security. pp. 313–321. Springer (2013)
20. Emms, M., Freitas, L., van Moorsel, A.: Rigorous design and implementation of an emulator for EMV contactless payments. Tech. rep., Newcastle University (2014)
21. Emms, M.J.: Contactless payments: usability at the cost of security? Ph.D. thesis, Newcastle University (2016)
22. EMVCo: EMV integrated circuit card specifications for payment systems [books 1 to 4] (Dec 2011), https://www.emvco.com/emv-technologies/contact/
23. EMVCo: EMV contactless specifications for payment systems [books a,b,c-1,c-2,c-3,c-4,c-5,c- 6,c-7 and d] (Feb 2016), https://www.emvco.com/emv-technologies/contactless/
24. Financial Fraud Action: Fraud the fact. the definitive overview of payment industry fraud and measures to prevent it (2017), https://www.financialfraudaction.org.uk/fraudfacts17/
25. Freitas, L., Emms, M.: Formal specification of EMV protocol. Tech. rep., Newcastle University (2014)
26. Freitas, L.: VDM at large: Modelling the EMV(R) 2nd Generation Kernel. In: Formal Methods: Foundations and Applications - 21th Brazilian Symposium, SBMF 2018, Salvador, Brazil, November 28-30, 2018, Proceedings. Lecture Notes in Computer Science, vol. 11254. Springer (2018)
27. Freitas, L., Jones, C.B., Velykis, A., Whiteside, I.: How to say why (in AI4FM). Tech. rep., Newcastle University (2013)
28. Freitas, L., Woodcock, J.: Mechanising mondex with Z/Eves. Formal Aspects of Computing 20(1), 117 (Jan 2008)
29. Garcia, R., Modesti, P.: An IDE for the design, verification and implementation of security protocols. In: ISSRE Workshops. pp. 157–163. IEEE (2017)
30. Garrett, D., Ward, M.: Blinded diffie-hellman. In: International Conference on Research in Security Standardisation. pp. 79–92. Springer (2014)
31. Guttman, J., Herzog, J., Ramsdell, J., Sniffen, B.: Programming cryptographic protocols. LNCS 3705, 116 (2005)
32. Hoare, C.A.R.: CSP – Communicating Sequential Processes. Prentice-Hall (1985)
33. Jones, C.B.: Systematic software development using VDM, vol. 2. Prentice Hall Englewood Cliffs (1990)
34. Lowe, G.: A hierarchy of authentication specifications. In: CSFW'97, pp. 31–43. IEEE Computer Society Press (1997)
35. Maiden, J.: EMV's Relay Resistance Protocol in MasterCard Contactless Specification. Master's thesis, School of Computing Science, Newcastle University (2017)
36. Mödersheim, S.: Algebraic properties in Alice and Bob notation. In: Int. Conf. on Availability, Reliability and Security (ARES 2009). pp. 433–440 (2009)
37. Modesti, P.: AnBx: Automatic generation and verification of security protocols implementations. In: Foundations & Practice of Security. LNCS 9482, Springer (2015)
38. Modesti, P.: Efficient Java code generation of security protocols specified in AnB/AnBx. In: Security and Trust Management STM, Proceedings. pp. 204–208 (2014)
39. Murdoch, S.J., Drimer, S., Anderson, R., Bond, M.: Chip and pin is broken. In: S&P. pp. 433–446. IEEE (2010)
40. Schmidt, B., Meier, S., Cremers, C., Basin, D.: Automated analysis of Diffie-Hellman protocols and advanced security properties. In: CSF. pp. 78–94. IEEE (2012)
41. Woodcock, J., Freitas, L.: Linking VDM and Z. In: Engineering of Complex Computer Systems, IEEE International Conference on. pp. 143–152. IEEE (2008)