BRNO UNIVERSITY OF TECHNOLOGY
Faculty of Information Technology
Department of Intelligent Systems

SECURITY OF CONTACTLESS SMART CARD PROTOCOLS

Ing. Mgr. Martin Henzl

Extended Abstract of Ph.D. Thesis

Supervisor: doc. Dr. Ing. Petr Hanáček

Contents

1 Introduction

Contactless smart cards and near-field communication (NFC) devices are used in many modern applications worldwide. Most of these applications require high level of security. Contactless cards are more convenient for the user to perform transactions than contact cards; however, they yield new vulnerabilities due to the radio interface. Proper use of these technologies provides high level of security; however, some applications, especially for access control, may be developed by developers that are not security experts, so they can contain vulnerabilities.

Development of secure hardware is very expensive and very slow compared to development of software. Protocols used in security sensitive systems are usually very secure and sometimes even formally verified. The software implementation is usually developed faster than hardware and is usually not formally verified, which makes it the weakest link. The software implementation is as important as other parts of the system. The protocol must be carefully implemented using secure hardware in proper way, which is very difficult, thus there is space for potential mistakes leading to vulnerabilities. People writing such applications have to be perfectly aware of all weaknesses of the particular card type in order to implement the system properly. An automated tool for vulnerability search in contactless

communication applications would help them to verify their implementation on particular device.

When designing and verifying security protocols using informal techniques, some security errors may remain undetected. Formal verification methods provide a systematic way of finding protocol flaws. The protocol is specified in a formal way and the correctness of security properties is proved or disproved using formal methods and mathematics.

The motivation for this work is a massive spreading of new contactless technologies and development of many applications sometimes by developers that are not security experts. Due to the high number of systems using contactless technology worldwide and the possibility of gaining high financial profit from compromising such a system, there are efforts to find vulnerabilities in these systems on both sides, attackers are trying to compromise a system, while developers are trying to fix vulnerabilities and improve security.

2 Goals

The high level goal of this thesis is to investigate security of contactless smart card protocols and to find methods of improving security of these protocols.

This thesis is concerned with contactless smart card protocols, which are protocols, such as payment protocols, that use contactless smart cards to store some data, values, cryptographic keys, and to perform cryptographic operations. End users usually use these personalized cards for payments, access control, loyalty programs, etc. The focus here is on contactless smart cards which differ from smart cards with contact interface mainly in two aspects. Firstly, the contactless smart cards are usually simpler due to the power limitations, so they can be modeled more easily. Secondly, the contactless interface introduces threats due to the fact that all communication is wireless. These threats, which are not applicable for smart cards with contact interface, must also be considered when investigating security of contactless smart card protocols.

If we try to understand what security issues can occur in such a protocol, we have to look not only at one level of the communication, such as the RF link, or the high level protocol definition. We have to investigate possible vulnerabilities at all levels.

The focus in this thesis is on the high level attacks on the protocol level. Possibility of these attacks will be analysed and a method of semi-automated vulnerability finding using formal methods will be proposed.

The formal model can be created from the protocol definition or extracted from the eavesdropped communication. Unwanted states that constitute an attacks must also be specified. After analysing the protocol and creating the model including the attack states, the formal analysis methods, such as model checking, can be used.

However, not all kinds of attacks are covered by the proposed method, such as attacks specific to the contactless interface. One of the attack types that are not covered by the method, the relay attack, is investigated separately. A minor part of this thesis is therefore dedicated to relay attack investigation and countermeasure proposal.

Relay attack is one of the most dangerous attacks against contactless devices, because there is no practical countermeasure to it. There are so called distance bounding protocols; however, they are implemented only in some devices, keeping the rest of devices unprotected. In this thesis the relay attacks will be investigated and if possible, a countermeasure to relay attacks performed over network will be proposed.

3 Vulnerability Finding Method

This chapter introduces a method of semi-automated vulnerability finding in contactless smart card protocols, which is the main contribution of the thesis.

The concept puts together a man-in-the-middle (MITM) attack with verification methods to find vulnerabilities in a semi-automated way. Figure 1 shows the scheme of the proposed system. The process consists of a cycle of several steps that can be performed several times to make the protocol secure.

- The first step is a MITM attack that can be used to analyze the protocol. The MITM hardware will communicate with both PCD and PICC, and eavesdrop on the communication to extract the protocol. It can be also used to fuzz test the protocol by altering commands and data in an unanticipated way.
- The next step is the formal model creation. Results from the analysis can be used together with the protocol and smart card specifications to create a formal model. The MITM at the beginning of the process can be theoretically used to create a formal model when analyzing a third party protocol even without the precise protocol specification, the protocol specification can be extracted by eavesdropping on real communication. The developer of a protocol can skip the first step and create the model only from the protocol and smart card specifications.
- The model will be verified by the model checker. In this phase the potential vulnerabilities can be found.
- The attack vectors found by the model checker will be used to execute the attack on the device, using the MITM. If the attack is successful, the vulnerability is reported, otherwise the model is refined.
- The hardware for performing MITM is useful for trying to execute an attack and to figure out how the formal model should be refined after each run of the model checker.

This cycle will be repeated multiple times until a vulnerability is found or the model checker concludes that there is no attack on this model. When an attack is found by a model checker and is not confirmed using MITM on the real devices, the model is refined and model checker is executed again. When a vulnerability is found and confirmed by MITM, the protocol should be improved to fix this vulnerability. The model should be updated and model checker should be executed again. Although the process is not yet fully automated, the model checking can find a vulnerability in the model automatically. The following sections discuss hardware for the MITM, protocol analysis, and formal verification.

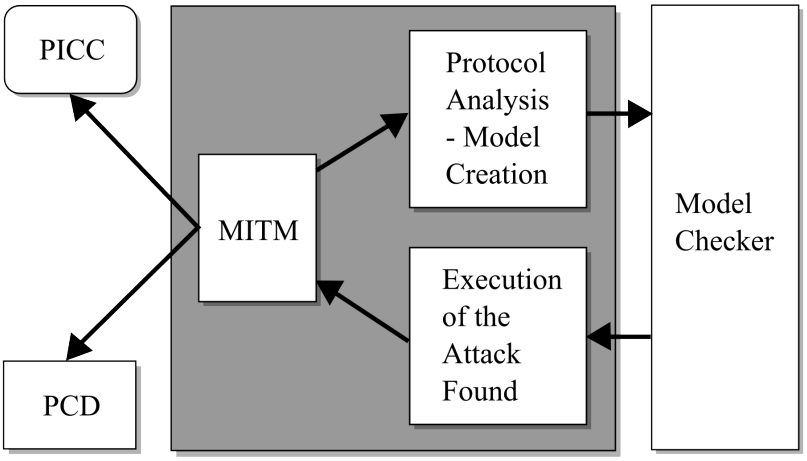

Figure 1: Scheme of semi-automated vulnerability search system

3.1 Hardware

In this section the hardware used to perform a man-in-the-middle (MITM) attack is discussed. In real environment, MITM can be done using relay attack. Our device will act as the MITM between two legitimate parties of the protocol. As mentioned earlier, there are two contactless devices needed for relay attack – a fake PCD and a fake PICC. We have connected both devices to one PC, which is the main hardware part of the system.

We have established a real relay attack using Proxmark 3, which is an open hardware platform for RFID research purposes. This device was developed by Jonathan Westhues for performing sniffing, reading and cloning of RFID tags. Proxmark 3 incorporates the FPGA unit, used for low level signal processing, and the ARM processor, that implements the transport layer. It can be used as a sniffer, as a reader or as a card, using various protocols. Proxmark 3 supports both low frequency (125 kHz – 134 kHz) and high frequency (13.56 MHz) signal processing.

The Proxmark 3 is connected to the PC via USB; however, the software developed by the community does not support realtime communication over USB, so we had to add it. With the original software the PC sends a command to Proxmark, which returns the result after processing it. We needed a realtime communication with the device, because each data packet received by the device requires its immediate transmission to the computer in order to get the response from the genuine PICC. In order to establish communication with just one party – PCD or PICC – we have implemented the anti-collision procedure.

Proxmark 3 acts as a fake PICC, communicates with the genuine PCD and forwards data blocks to the PC. It also transmits data blocks in the opposite direction. ACR122 acts as a fake PCD, doing the similar task with the genuine PICC. The PC controls both devices and relays data blocks between them. Data can be saved or altered in the PC.

3.2 Protocol Analysis

The goal of this part is to create a formal model of the implementation of the protocol. Smart card application issuers mostly don't publish their algorithms for any scientific feedback, hence there could be bugs that might remain hidden for a long time of using such a system. Furthermore, thanks to NFC, there are many more new applications being developed, and there is a great potential for the future. These applications also handle sensitive data and use contactless communication as well as smart cards. In order to be able to find any vulnerabilities in these closed source protocols, the wireless communication can be eavesdropped and the protocol can be extracted by analysing the data being exchanged. With the knowledge of the protocol, the formal model can be created. Limited knowledge of the protocol should therefore not entail a problem, the data needed for creating the protocol formal model can be extracted from the eavesdropped communication.

The eavesdropping of the protocol can be used to extract the protocol from real communication and the MITM also allows us to alter arbitrary data, change command order, communicate with just one of the legitimate parties and try various commands even with wrong parameters. In theory, the model creation and refinement could be done automatically from data gained by eavesdropping and fuzz testing, which would make the whole process of vulnerability finding fully automatic. Learning techniques allow automatic inference of behaviour of a system as a finite state machine. For example in [1] the authors showed that a Mealy machine representing a model of EMV smart card can be successfully extracted using protocol fuzzing. However, we did not try to make automatic protocol fuzzing, so the process of vulnerability finding is only semi-automatic. Automatic Mealy machine creation using protocol fuzzing was left for future work.

It is very beneficial to have the protocol and smart card description when creating the formal model of the system and not to rely only on data from MITM eavesdropping. The protocol can be described for example as a sequence diagram and the smart card as a Mealy machine. The information gained using MITM together with the protocol and smart card specification gives us an overall image of the system being observed. The creation of formal model from the protocol and smart card description is explained in chapter 4.

3.3 Verification

The formal model of the protocol can be used to automatically find vulnerabilities using formal verification methods. These methods are used for proving security properties of protocols such as authentication, integrity, confidentiality and anonymity. Not only they tell us whether the protocol meets these properties but they can also find the counterexample. These counterexamples can be considered possible attacks. Formal methods therefore provide us with the automated way of finding attacks and can also be used for proving that some attacks are not possible. In this part a model checker will search for possible attacks, which will later be evaluated on the hardware.

3.3.1 Tool Selection

There are many papers describing and comparing various formal verification tools, such as NRL and FDR comparison [14], Casper/FDR, ProVerif, Scyther and Avispa comparison [7], or OFMC, Cl-Atse and ProVerif comparison [13]. Various tools have been studied and tested for the purpose of this thesis to find out which one is the best for security verification of protocols using contactless smart cards. During the process of selecting the right tool, various aspects of the tool had to be considered, such as performance, how difficult it would be to model desired features in the particular modeling language, and published results with the tools.

The AVANTSSAR tool was chosen for the security verification, mainly because the fact that the high level ASLan++ language can be easily used to model the desired features and because three different back-end model checkers can be used to verify the model. Also there are published papers suggesting that this tool and its back-end model checkers have good results in the field of security protocol verification. The performance seems to be good and for example performance comparison [13] of ProVerif with AVANTSSAR back-end model checkers Cl-Atse and OFMC shows better results of AVANTSSAR back-end model checkers; however, the difference is not significant. AVANTSSAR developed from AVISPA and both tools seem to be proved and used by the community.

AVANTSSAR (Automated VAlidatioN of Trust and Security of Service-oriented ARchitectures) is a follow-up project of AVISPA, introducing new languages for describing models, the AVANTSSAR Specification Languages ASLan++ and ASLan. ASLan++ [16] is a high level formal language similar to the HLPSL, used for specifying security-sensitive service-oriented architectures, their associated security policies, and their trust and security properties. The semantics of ASLan++ is formally defined by translation to ASLan, the low-level specification language that is the input language for the back-ends of the AVANTSSAR Platform – OFMC, CL-AtSe, and SATMC.

OFMC [5] combines a number of techniques to enable the efficient analysis of security protocols. First, OFMC uses lazy data types as a simple way of building efficient on-the-fly model checkers for protocols with very large, or even infinite, state spaces. A lazy data type is one where data constructors build data without evaluating their arguments. Second, OFMC models the adversary in a lazy fashion, where adversary communication is represented symbolically and solved during search. Third, while OFMC performs verification for a bounded number of sessions, it works with symbolic session generation, which avoids enumerating all possible ways of instantiating possible sessions. Fourth, OFMC exploits a state-space reduction technique, inspired by partial-order reduction, called constraint differentiation [15]. Constraint differentiation works by eliminating certain kinds of redundancies that arise in the search space when using constraints to represent and manipulate the messages that may be sent by the adversary. Finally, OFMC also provides some limited support for handling different equationally specified operators on messages [6]. [4]

Cl-Atse [18] represents protocol states symbolically as a collections of non-ground facts, which record the states of different threads, the messages sent to the network, and the adversary knowledge. In particular, constraints are used to

describe what the different agents know and a constraint calculus is used to solve for what they can know, from messages previously exchanged, i. e., the calculus is used to solve a variant of the non-ground intruder deduction problem. CL-Atse was designed to allow the easy integration of new deduction rules and operator properties. [4]

SATMC [3] is an open platform for model checking of security services. SATMC reduces the problem of checking whether a protocol is vulnerable to attacks of bounded length to the satisfiability of a propositional formula which is then solved by a state-of-the-art SAT solver. This is done by combining a reduction technique of protocol insecurity problems to planning problems and SAT-reduction techniques developed for planning and LTL that allows for leveraging state-of-the-art SAT solvers. SATMC provides a number of distinguishing features, including the ability to check the protocol against complex temporal properties (e.g. fair exchange); analyze protocols (e.g. browser-based protocols) that assume messages are carried over secure channels (e.g. SSL/TLS channels). [17]

4 Formal Model

This chapter provides a description of the proposed method that can be used to create a model of a contactless smart card and a terminal and to define states representing attacks. This model can be then used in model checking to find attack traces in the protocol. The model takes into account the implementation details of a particular smart card which could be possibly avoided in a high level protocol verification. These details are important because wrong use of smart card commands may introduce a vulnerability even if the high level definition of the protocol is secure. The ASLan++ language was chosen for protocol modeling, it can be used as an input for multiple back-end model checkers of the AVANTSSAR Platform.

A model of protocol in ASLan++ is defined by roles that can be played either by a legitimate party or by an adversary called intruder. We establish two main roles in the model description to represent the implementation – the first role represents the smart card with its functionality and settings, the second role represents the protocol. The protocol is executed by the terminal, the smart card only responds to commands from the terminal. The protocol can be therefore identified with the terminal in our model. The intruder model that is used is the well-known Dolev-Yao intruder model [8]. All communication is synchronous with the intruder, the intruder intercepts the messages from the legitimate user and each legitimate user receives messages only from the intruder. The intruder can be therefore identified with the network. Figure 2 shows the configuration of subjects in the model. The PCD executes the protocol and communicates with the PICC via the intruder, who is a man-in-the-middle. The goal of our vulnerability finding method is to find out if the intruder would be able to perform some attack in this configuration and find an attack trace.

The state explosion problem has to be addressed. If we create precise model of the smart card and the terminal functionality, the model will be too complex for the model checker, the number of states will be so high that the model checking execution time will be unacceptable. The goal of this thesis is to create modeling

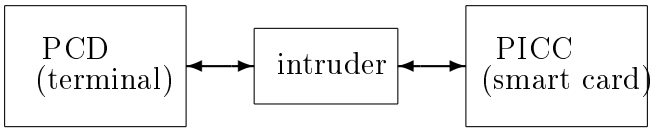

Figure 2: Intruder model

method that will create models which can be computed using model checking in acceptable time and which describe the functionality sufficiently. We create simplified models that are weaker than the precise model would be, so more attacks can be found. Attacks that are found by the model checker can be tested and in case of false positive the model can be adjusted to be more precise and not contain the particular false vulnerability. The resulting model will be a trade-off between precision and model checker execution time.

Since smart cards are usually used in applications where high level of security is required, confidentiality, integrity and authentication should be provided by the protocol to protect data that are being transferred between the smart card and the terminal. Confidentiality of data is achieved by encrypting the data using any of the state-of-the-art ciphers, which are strong enough to be relied on. In this thesis the strength of the cipher is supposed to be sufficient to resist attacks focused merely on breaking the cipher rather than finding vulnerabilities in the protocol. We therefore consider all ciphers unbreakable for purposes of this thesis so that we can focus on vulnerabilities in the protocol.

Integrity of messages exchanged between smart card and terminal can be ensured in multiple ways, such as computing the cyclic redundancy check (CRC) of plaintext and encrypting it together with data, or by using message authentication code (MAC), which is a cryptographic hash. MAC can be used to cryptographically secure the integrity of data even if these data are not encrypted.

Contactless smart cards usually require terminal authentication which ensures that the data will not be revealed to unauthorized entities. Each file in the smart card has usually access permissions that are used to authorize operations on these files. The access rights are determined according to the symmetric key that was used for authentication.

4.1 Modeling Tool

ASLan++ is the specification language used in AVANTSSAR. It is a high-level formal language for specifying security-sensitive service-oriented architectures. It is easy for system designers to use, because it is close to the way in which they think about systems. It can be used also by users who are not experts on formal specification language. The AVANTSSAR platform provides conversion from high-level ASLan++ to ASLan, which is a low-level specification language used by back-end model checkers to perform verification of security properties.

ASLan++ document consists of four parts: Entities, Declarations, Statements, and Goals. General schematic architecture of ASLan++ is shown in figure 3. An ASLan++ model is a hierarchical structure of entities. The top-level entity is called Environment, its sub-entity is called Session. Sub-entities of Session are used to describe characteristics of different agents or roles. The entity contains a collection

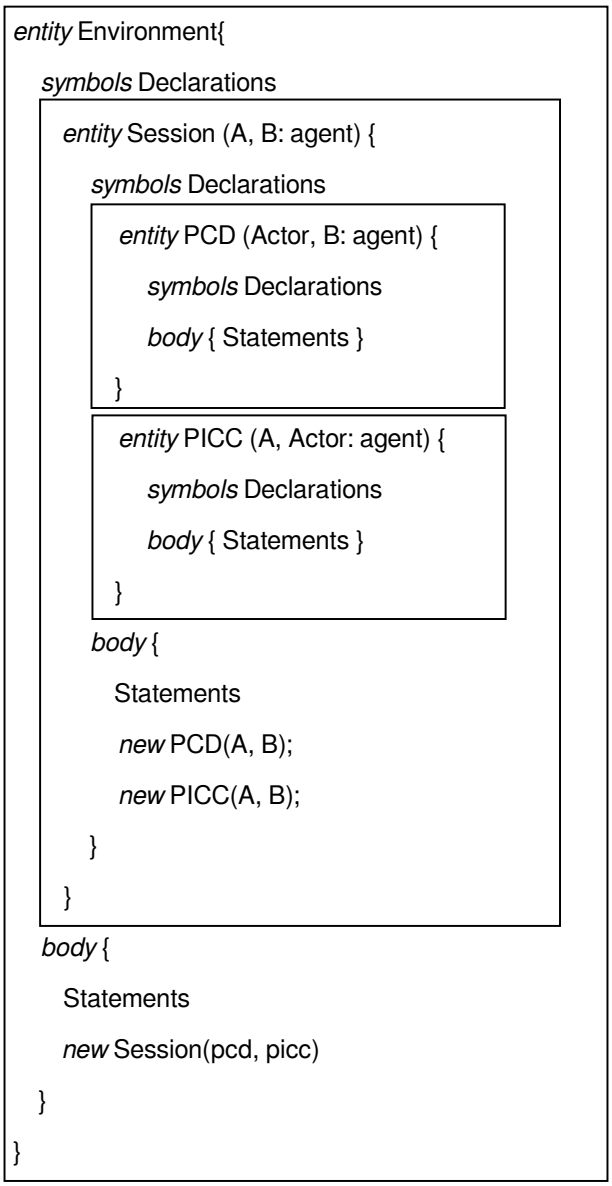

Figure 3: Schematic architecture of ASLan++

of declarations, starting with keyword *symbols*, and a series of statements, starting with keyword *body*. Declarations are used to define types, variables, constants, and functions. They are the static part of the entity, while statements describe the dynamic part of the entity. Goals are used to formalize the desired security properties. The most general way to formalize security properties is to use extended Linear Temporal Logic (LTL) [2] formulas. Validation goals have a name and a LTL formula that is checked by the validation back-ends. Another way to define a goal is to define an assertion. Assertions are given as a statement in the body of an entity. They are expected to hold only at the given point of execution of the current entity instance.

The ASLan++ model can be checked by any of the AVANTSSAR back-ends. The back-end model checker will then give a counterexamples when an attack is found, which can be used to deduce the security flaw of the system. When

no attack is found, it doesn't necessarily mean there is no vulnerability in the protocol. The reason may be that the model checker explores the search space to the maximal depth which was previously set in the back-end without finding any attack.

4.2 Smart Card Model

The PICC can be seen as a state machine. The PICC reads commands from PCD, changes its internal state according to these commands, and responds back. States of the machine are determined by the internal state of the PICC logic and by the value of internal variables of the PICC, such as content of files and used cryptographic keys. Since the logic must have finite number of states and the files and keys can only have finite number of values, the number of states of the machine will be finite. The transition rules of the automaton are defined by the set of commands and parameters of these commands. Although the set of parameters will be high, it will be finite, so the number of transition rules will be finite as well. We can therefore model the PICC behavior using a finite-state automaton or, more specifically, a Mealy machine, whose output is determined by the current state and the current input. Another state machine concepts can be used instead, such as UML state machine, which is an enhanced realization of the finite-state automaton mathematical concept with characteristics of Mealy machine. UML state machine diagrams are convenient for describing contactless smart card behavior, because they support enhanced methods for simple picturing of complex behavior, such as extended states, hierarchically nested states, and orthogonal regions.

The automaton should describe behavior of a PICC in the level of detail suitable for model checking, which means the simpler the better. It should be designed to be simpler than the real implementation and allow false positives rather than false negatives. It should be as simple as possible, because the model checking could take unbearable amount of time due to the state explosion problem, if the number of states was not kept low. The model can allow false positives because it can be iteratively refined, but it should not allow false negatives, which would result in false belief that the system is secure. The automaton can be refined if false positives are found by the model checker.

Figure 4 shows a sample UML state machine diagram describing logic of the Mifare DESFire MF3ICD40, which is one of the cards later used to demonstrate the verification method. Mifare DESFire is a memory card, so the logic is quite simple. The card shown in the figure has three applications, the default application number 0 and two standard applications with numbers 1 and 2, and uses two keys for authentication, so the user can be authenticated using $key1$, $key2$, or not authenticated ($noKey$). Only basic commands needed for a payment protocol are modeled, the authentication command ($auth$), select application command ($select$), read file ($read$), and write file ($write$). Two actions of 1) putting the card to the proximity of the reader which starts the communication and 2) taking the card away from the reader to end the communication are represented by *activate* and *deactivate* transitions respectively.

The model should represent the behavior of a personalized issued card that is ready to be used in the protocol, which means that it does not have to support all commands which are used for the smart card personalization or commands that

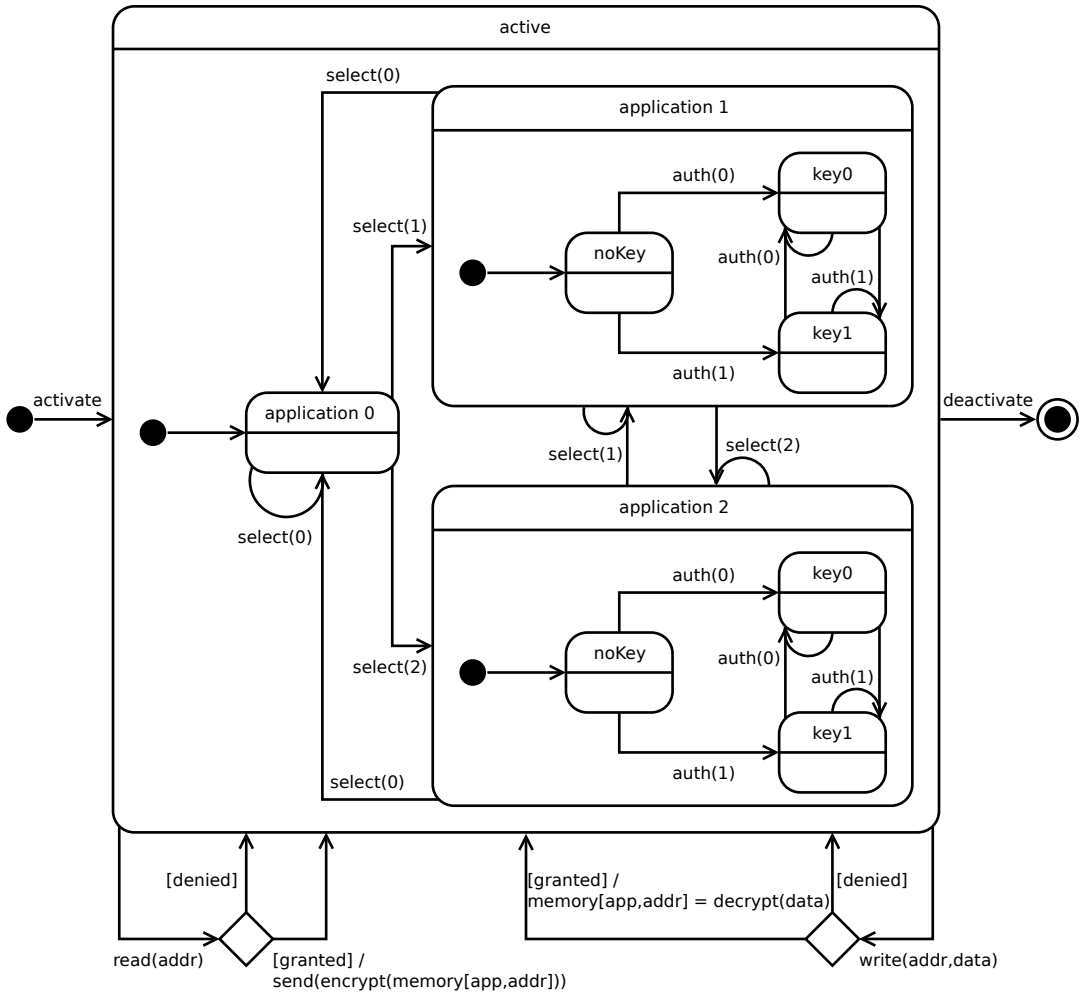

Figure 4: UML state machine describing basic Mifare DESFire behavior

are not enabled after the smart card is issued. This approach results in simpler models and shorter model checking computation times. The Mifare DESFire smart card supports more commands than the commands shown in figure 4, but these commands will not be used after the card is personalized in secure environment, so they are useless in the model. Also the application 0 in the model does not allow authentication, because it is used only during personalization for operations related to creating and setting up other applications.

When the contactless smart card is put to the proximity of the reader, it is activated and an anti-collision procedure is performed. The anti-collision procedure is used to allow multiple cards to communicate with the terminal without interference. After the anti-collision procedure, the terminal communicates only with one smart card at a time, the order of smart cards is negotiated during the anti-collision procedure. There is no reason for modelling the anti-collision procedure, so in the model the card gets immediately into the *active* state. When the card is taken away from the reader, the communication is terminated and the card is deactivated.

The diagram in figure 4 uses features of UML state machine diagrams to simply picture complex behavior. The diagram uses nested states. If a system is in the

nested state (called substate), it is implicitly also in the surrounding state (called superstate). The state machine will attempt to handle any event in the context of the substate, but if the substate does not prescribe how to handle the event, the event is automatically handled at the higher level context of the superstate.

The figure describes an extended state machine which uses extended states to describe memory of the card. The extended state is a combination of the state and the extended state variables. This feature is very useful, because state machines without extended states need large number of states to implement variables. The machine from figure 4 can be pictured without extended states using orthogonal region implementing memory, as shown in figure 5. Each state can contain two or more orthogonal regions and being in such a state means being in all its orthogonal regions simultaneously. The number of states in the memory region is very large, so only a couple of states are depicted to show the notion. We could define the state machine without orthogonal regions, such machine would have states from the cartesian product of states in the current orthogonal regions.

The memory cards will result in very simple diagrams, while smart cards with more complex logic like Java Cards or BasicCards, which allow execution of arbitrary code, will result in more complex diagrams. Examples in this thesis are based on Mifare DESFire, but models of other card types can be also created.

Although UML state machines are very useful for depicting behavior of contactless smart cards, the behavior can also be described using simple finite-state automatons and Mealy machines. Such description is more formal and can provide more detailed insight.

We can create the Mealy machine representing the PICC by combining an automaton describing the PICC logic and an automaton representing the state of memory (the two machines that were combined using orthogonal regions in figure 5). The formal definition of the PICC Mealy machine will be provided later. We can analyse the logic and memory automatons separately.

The PICC logic automaton should describe behavior of PICC as a response to the commands sent by PCD. Let M_{logic} be a deterministic finite automaton defined as a quintuple $(Q_{logic}, \Sigma_{logic}, \sigma_{logic}, q_{logic0}, F_{logic})$, consisting of:

- a finite set of states Q_{logic}
- a finite set of input symbols Σ_{logic}
- a transition function $\sigma_{logic} : Q_{logic} \times \Sigma_{logic} \to Q_{logic}$
- a start state $q_{logic0} \in Q$
- a set of accept states $F_{logic} = Q_{logic}$ (PICC may end in all states)

Figure 6 shows an example of M_{logic} automaton describing logic of the Mifare DESFire based on 4.

In this example the card has three applications and uses two keys for authentication. The states are denoted by a pair of application number and authenticated key respectively. The initial state is the state where default application number 0 is selected and no authentication was performed – authenticated key 0. Only basic commands needed for a payment protocol are modeled, the select application command (*select*), the authentication command (*auth*), the read file command

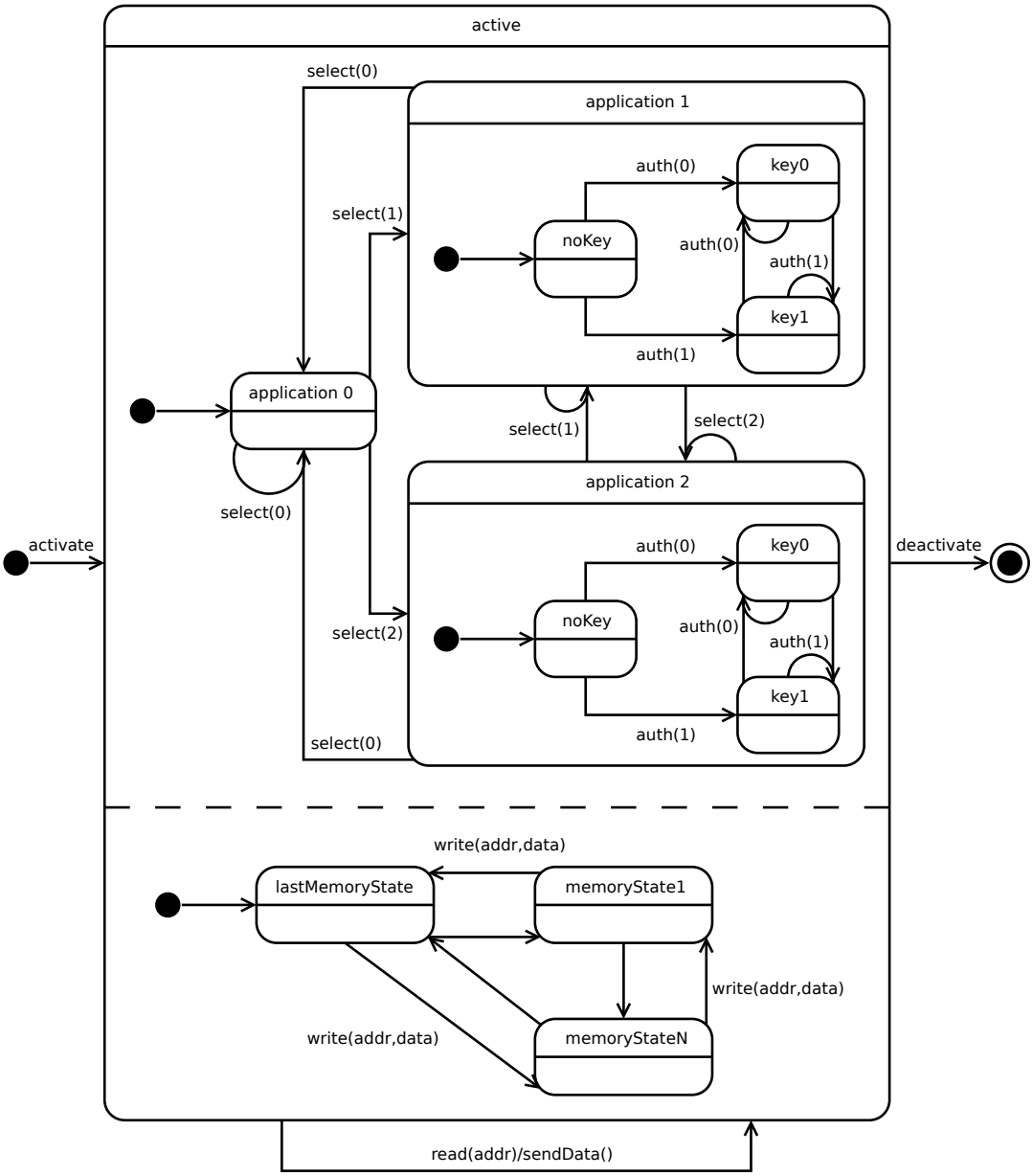

Figure 5: Mifare DESFire UML state machine with memory states

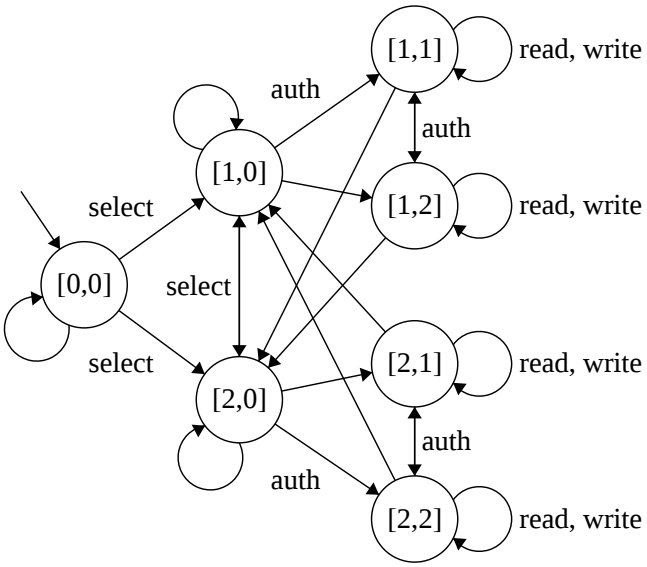

Figure 6: FSM describing smart card behavior for some basic commands

(*read*), and the write file command(*write*). Read and write commands do not change state of the automaton, for these operation the memory automaton will be needed. The diagram does not contain description of all transitions, which are same as in 4, and does not show final states. All states are potentially final, since the communication with the card can be ended or interrupted in arbitrary state.

The automaton describing the state of the PICC memory has states determined by the content of files, values of cryptographic keys, and values of all other variables that are persistent in the PICC memory and that can be changed during the life of the card. It can be defined similarly as the M_{logic}. Let $A = a_1, a_2, ...a_n$ denote all memory blocks (files, keys, etc.), n is the number of memory blocks. Let D be a set of all possible data that can be stored in a block. Let $C_{write} = A \times D$ be a set of all write command parameters, which consist of memory address and data to be written and let $C_{read} = A$ be a set of read command parameters consisting of memory address and let c_{noop} be a command for no operation. Let M_{memory} be a deterministic finite automaton defined as a quintuple, $(Q_{memory}, \Sigma_{memory}, \sigma_{memory}, q_{memory0}, F_{memory})$, consisting of:

- a finite set of states $Q_{memory} = D_1 \times D_2 \times ... \times D_n$, where n is the number of memory blocks
- a finite set of input symbols $\Sigma_{memory} = C_{write} \bigcup C_{read} \bigcup \{ c_{noop} \}$
- a transition function $\sigma_{memory} : Q_{memory} \times \Sigma_{memory} \rightarrow Q_{memory}$ (commands for writing data C_{write} change state appropriately, C_{read} and c_{noop} do not change state)
- a start state $q_{memory0} \in Q$ (initial content of memory)
- a set of accept states $F_{memory} = Q_{memory}$ (PICC may end in all states)

The automaton describing the PICC is the combination of the automaton describing the PICC logic and the automaton representing the state of memory.

Let M be a Mealy machine defined by a 6-tuple $(Q, Q_0, \Sigma, \Lambda, T, G)$ consisting of the following:

- a finite set of states $Q = Q_{logic} \times Q_{memory}$
- a start state $Q_0 = (q_{logic0}, q_{memory0})$, which is an element of Q
- a finite set of input symbols $\Sigma \subseteq \Sigma_{logic} \times \Sigma_{memory}$; input alphabet will contain only meaningful commands:
  $(write, c_i)$, where $write \in \Sigma_{logic}, c_i \in C_{write}$
  $(read, c_i)$, where $read \in \Sigma_{logic}, c_i \in C_{read}$
  (c_i, c_{noop}), where $c_i \in \Sigma_{logic} \setminus \{write, read\}$, $c_{noop} \in \Sigma_{memory}$
- a finite set called the output alphabet $\Lambda = D \bigcup R$, where R is a set of PICC status responses and D will be used for read command responses
- a transition function $T : Q \times \Sigma \to Q$ mapping pairs of a state and an input symbol to the corresponding next state
- an output function $G : Q \times \Sigma \to \Lambda$ mapping pairs of a state and an input symbol to the corresponding output symbol

An intuitive interpretation of a Mealy machine is following. At any point in time, the machine is in some state $q \in Q$. It is possible to give inputs to the machine by supplying an input symbol $i \in \Sigma$. The machine then responds by producing an output symbol $G(q, i)$ and transforming itself to a new state $T(q, i)$.

The read and write commands will be processed only after correct authentication, which is determined by the state of the logic automaton. The read command will return the file content based on the state of the memory automaton, and write command will change the state of the memory automaton. All other transitions will return only status of the command execution.

4.2.1 States reduction

The model checking execution time strongly depends on the total number of states. In order to keep the model checking time short, the number of states of the state machine that simulates the smart card should be as low as possible, so some optimization should be performed. To reduce the number of states in the state machine we can reduce the number of states used for logic (M_{logic}), or for memory (M_{memory}), or both.

To reduce the number of states that describe logic of the smart card, we can keep only states that has any side effect, for instance send data to the reader (read command) or make persistent changes in the memory (write command), and join them with the supporting states that represent the chain of commands. We can create optimized commands that are combination of multiple real commands. Each combined command has a side effect. We simulate commands for data transfer – *read* and *write*. This approach reduces execution time of the model checker. Figure 7 shows the state machine from figure 4 with reduced number of states. There are only two commands:

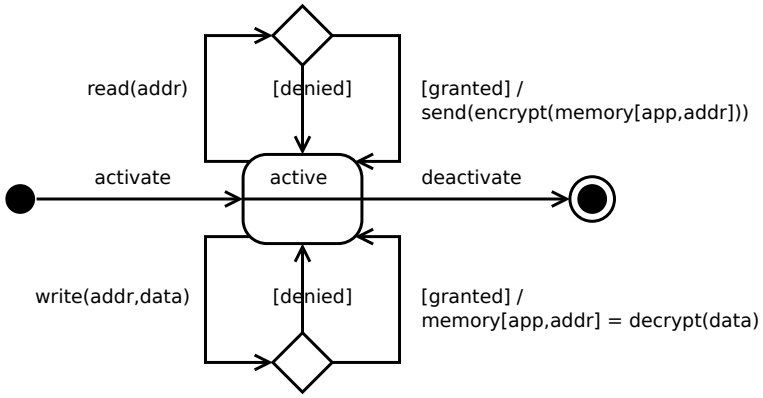

Figure 7: Reduced number of states

- read: this command is a combination of select application, authenticate, and read command
- write: this command is a combination of select application, authenticate, and write command

This reduction is possible and has no impact on attack finding results, because the supporting commands for selecting application and authentication can be performed multiple times and only the last performed command has impact on the following read or write command. The internal state is determined by the last select and authenticate commands, the previous commands are forgotten. The read and write commands will contain parameters for selected application number, which will be consequently part of the memory address, and other parameter for authentication and determining the authentication key. The figure contains the authentication token *auth*, which will be described later together with the authentication mechanism.

To reduce the number of states in the M_{memory}, we have to reduce the number of memory blocks that can be written to, and/or reduce the number of possible data that can be stored. If the card supports addressing of data blocks by application, file ID, offset and length, the number of possible write locations can be tremendous. Better approach is to have only memory locations that the application is supposed to write to or read from and one undesired location for each file that will be used to simulate writing or reading to bad location that will corrupt the result. Using this approach the total number of states will be reduced dramatically, which will also reduce the model checker execution time.

4.2.2 PICC Entity

When the PICC behavior is known and modeled for example using UML state diagram, the PICC role in ASLan++ can be created. The ASLan++ general schematic was shown in figure 3, the PICC behavior is defined in the $entityPICC$, which contains *symbols* declarations and *body*. The *body* of the PICC role can be created based on the UML state diagram. The basic PICC functionality that is created in the *body* is an infinite loop that reads commands from PCD, processes

them, and sends responses back to the PCD, as shown in figure 8. The states of the PICC (as defined in the UML state diagram) are determined by values of state variables, that are defined in the *symbols* declarations part.

States can be defined in several ways. There can be one PICC state variable or there can be multiple state variables. In the latter case the PICC state is determined by values of all state variables together. The state variables may represent for instance the selected application and authenticated key.

PICC response is based on the current state and the received command. Both state and command variables are declared in the *symbols* part of the PICC entity. ASLan++ allows new type definition, so the state variable may be of type *state* and the command variable may be of type *command*. These types can be declared in the *symbols* part of the *Environment*. These types should be declared as subtypes of the basic type *text*. Variables could also be declared as *text* without creating new types.

For creating a model of Mifare DESFire with reduced set of commands as shown in 7 no states are necessary, because the model has only one state. The PICC responses are then based only on the received commands.

This section is dealing only with the logic automaton and shows only the basic structure of the PICC role. The PICC behavior is more complex when the memory automaton is taken into account. The memory automaton is not created in the same way by modeling its states, it is created in a more natural and straightforward way by introducing variables that represent the memory of the PICC and the state of the memory automaton is determined by the values of these variables. In other words, the state of the memory automaton is determined by the content of the PICC memory. The PICC will also have other variables for example for authentication purposes as described later, and we will consider it as part of the memory automaton.

The *body* part of the PICC entity can access the memory for read and write, so the resulting model will be the combination of the logic and memory automatons.

4.2.3 Basic Concepts

There are some basic concepts that can be put together to form a smart card model. These concepts are general and can be used to create a model of arbitrary smart card with pre-defined set of commands. We describe modeling of the following concepts:

- Applications
- Authentication
- Encryption
- Files and Permissions
- Personalization
- Integrity

```
entity PICC (A, Actor: agent) {

  symbols
  State: state;
  Command: command;
  ...

  body {
  ...
    while(true) {

      % read command
      A -> Actor: ?Command;

      select {
        on(State = state1): {
          select {
            on(Command = command1): {
              ...
            }
            on(Command = command2): {
              ...
            }
            ...
          }
        }
        on(State = state2): {
          select {
            on(Command = command1): {
              ...
            }
            on(Command = command2): {
              ...
            }
            ...
          }
        }
        ...
      }
      % send response
      Actor -> A: ok;
    }
  }
}
```

Figure 8: PICC role in ASLan++

The following sections describe the method of creating a PICC role in the ASLan++ for these concepts, how to implement basic commands (commands of the PICC automaton) and also how to implement the simplified commands (commands of the PICC automaton with reduced number of states).

Applications

Multi-application contactless smart cards support multiple applications even from different vendors on a single card. The application on cryptographic memory card is not an executable program, it is rather a set of resources dedicated to application outside the card. The application on the card can consist of files used to store data and symmetric keys used for authentication and data encryption. The application outside the card can securely store data in the card and read them back later. This can be used for instance for payment applications or loyalty program applications, where some credit is stored on the card.

To simulate the application selection in the PICC role, we can use a state variable which is set by the PCD using a *select* command. The value of selected application is then used for file access. If we use the automaton with reduced number of states, the application selection is part of another command, such as the *read* or *write* command.

Authentication

The authentication process between smart card and terminal is usually mutual, both parties must prove possession of a common secret. In case of Mifare DESFire contactless smart card, the three-pass authentication is executed and the common secret is the DES/3DES key. When creating a model of a smart card, the authentication does not have to have precisely three message exchanges, it can be simplified in order to keep the number of states low. The simple way of simulating the mutual authentication process and modeling in ASLan++ is a fresh session key generation performed by one of the parties and sending it encrypted using the authentication key to the other party. The other party must check that the session key is fresh and was never used before during the protocol run. This approach uses a trick based on the fact that we can be certain of things that we cannot in the real environment. We can have a secret key shared only by legitimate entities and we can be sure that the intruder does not know the key. So if something is encrypted using this secret key, such as the fresh random session key, the receiving party can be sure that the message was encrypted by the legitimate counterpart, and also the sending party can be sure that only the legitimate counterpart can decrypt the message. The sending party generates the fresh session key to simulate new session key generation performed during the three-pass authentication, the receiving party must check that the key is really fresh and was never used before during the protocol run. The fresh session key generation and checking by the other party will prevent replay attack on the authentication. Figure 9 shows example of a three-pass authentication. $\{A.B\}_K$ means concatenation of A and B encrypted using encryption key K.

Thanks to the fact that in the model we can be certain of things that we cannot in the real environment and that the PICC can remember all previously used session keys and check that the new session key is really fresh, we can simulate the authentication using only one message exchange, as shown in figure 10.

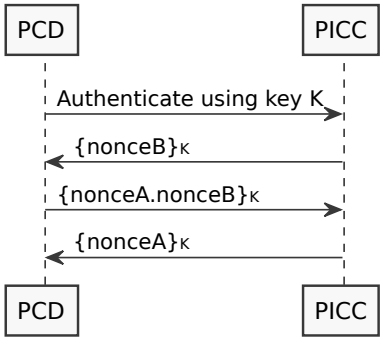

Figure 9: Three-pass authentication example

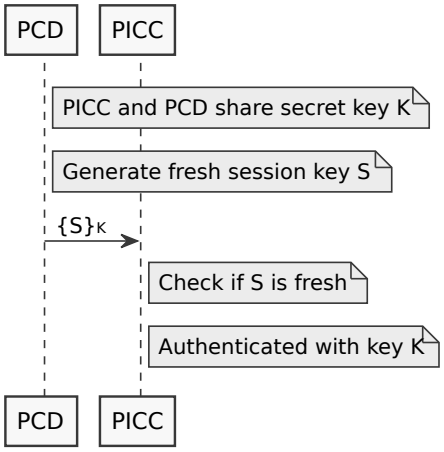

Figure 10: Simplified authentication used in model

After the one-pass authentication, PCD and PICC share the common session key, which could not be eavesdropped by the attacker, because it was encrypted with key not known by the attacker. The PICC knows which authentication key was used and can grant access to files accordingly. The authentication needs to be implemented in both PCD and PICC roles. The PCD always starts the communication and sends commands, so it will also generate a random session key.

In case of the automaton with reduced number of commands, the authentication can be part of another message. Figure 11 shows possible ASLan++ source of one-pass authentication, where the authentication token is part of the *readFile* command. The PCD generates fresh *SessionKey* and sends it in the *auth* token with authentication key *key*1, which is not known by the intruder. PICC checks that the session key was never used before (authentication resulting in fresh session key) or that it is the current session key, in which case the protocol continues with the old session key (no new authentication). The current session key is stored in variable *SessionKey* and the set of all used session keys is *UsedSessionKeys*. In case of successful authentication, the current session key is stored in *UsedSessionKeys* set for later use.

Encryption

The high level language ASLan++ already supports modeling of communication encryption, but it does not consider various modes of encryption algorithms. In ASLan++ any data can be encrypted using symmetric or asymmetric cipher. These ciphers are considered unbreakable for purposes of protocol modeling, therefore the intruder cannot learn the plaintext of the encrypted data unless he knows the corresponding key. The complexity of breaking the encryption algorithm is out of scope of this thesis. But there are different modes of encryption that must be taken into account when creating a model even if the cipher algorithm itself is considered unbreakable. Symmetric ciphers are used in the following modes:

- ECB – Electronic Codebook
- CBC – Cipher Block Chaining
- CFB – Cipher Feedback
- OFB – Output Feedback
- CTR – Counter

The ECB mode encrypts each block of data in the same way independently on the other blocks. The initialization vector is same for each block. The other modes are more secure, because each block encryption depends on the previous blocks, which makes the cryptanalysis more difficult. The initialization vector of the cipher is changed after each block encryption, so each block is encrypted using different initialization vector. Mifare DESFire MF3ICD40 specification states that DESFire uses CBC mode. Although each block of data is encrypted in CBC mode, same initialization vector is used for each block, which means that for short data blocks the data is encrypted using ECB and we will consider it as ECB mode for purposes of this thesis. This mode is prone to replay attacks, because each

```
entity PCD (Actor, B: agent) {
  ...
  body {
    % fresh session key generation
    SessionKey := fresh();
    % read name
    Actor -> B: readFile(addressName, auth(key1, SessionKey));
    B -> Actor: enc(SessionKey, ?Data);}
  }
}
entity PICC (A, Actor: agent) {
 ...
  body {
    while(true) {

      % read command
      A -> Actor: ?Command;

      select {
        on(Command = readFile(?DataAddress, auth(?AuthenticatedKey,
              ?SessionKeyTemp))): {
          % authentication
          select {
            on(!UsedSessionKeys->contains(SessionKeyTemp) |
                    SessionKey = SessionKeyTemp): {
              % store current session key
              UsedSessionKeys->add(SessionKeyTemp);
              SessionKey := SessionKeyTemp;

              % authenticated
              ...
            }
          }
        }
      }
    }
  }
}
```

Figure 11: One-pass authentication in ASLan++

```
% ECB mode
encryptedECB := enc(SessionKey, Data);

% CBC mode
encryptedCBC := enc(SessionKey, nextIV(lastIV), Data);
```

Figure 12: ECB and CBC encryption modes in ASLan++

data block is encrypted using the same initialization vector and the same key. In ASLan++ each block is encrypted using same key and there are no initialization vectors, so we can consider it the ECB mode.

From the protocol modeling perspective, the CBC, CFB, OFB, and CTR modes do not differ. They use the initialization vector which is different for each block. The strength of these modes is out of scope of this thesis. We can model these modes by adding fresh number (not used before and not known by the intruder) to the data being encrypted, simulating the changing initialization vector. This approach will provide resistance to replay attacks.

Encryption in ECB mode can be written in ASLan++ as $enc(SessionKey, Data)$, a non-invertible function representing $Data$ encrypted using key $SessionKey$. Non-invertible means that although it may be overheard by the intruder, the intruder is not able to invert the function to get the $SessionKey$ or $Data$.

In case of CBC, we can use initialization vectors that are chained using custom function $nextIV()$ so that fresh initialization vector is used each time. The first initialization vector is custom vector $zeroIV$, the next one is $nextIV(zeroIV)$, the next one is $nextIV(nextIV(zeroIV))$, etc. The encryption in the CBC mode can then look like this: $enc(SessionKey, nextIV(lastIV), Data)$, where $lastIV$ is the last initialization vector. Other encryption modes can be modeled along the same lines.

Figure 12 shows encryption in ECB and CBC modes.

Files and Permissions

Smart cards provide file system with permissions that can control access to each file based on the key that was used for authentication. We can model files and permissions in ASLan++ either as variables or as facts. If the structure of files is static and will not change during the life of the smart card, it is possible to model files using variables in PICC role. Each file would be a variable and file permissions would be variables as well. Better approach is to use ASLan++ facts. Facts are global and more flexible, so when using facts it is possible to check content of PICC files even from the PCD role, and it is possible to add new facts and retract existing facts, which can be used to simulate flexible file system where files can be created and deleted. Figure 13 shows how the file system can be declared in ASLan++ as fact *fileSystem* with four parameters for data address, authentication keys to get read and write permission, and data itself.

The first parameter of the fact represents the address of the file and is of type *text*, which is the most simple type in ASLan++. The second parameter

```
fileSystem(text, symmetric_key, symmetric_key, message): fact;
```

Figure 13: PICC file system in ASLan++

represents authentication key that must be used to obtain read permission to this file and is of type *symmetric_key*, which is an ASLan++ type for symmetric keys. Analogously, the third parameter is the authentication key for write permission. The fourth parameter represents data stored in the file and is of type *message*, which is a compound type that can store any combination of data of any other type.

Although address has a simple type, it represents a number of values that constitute the address on a real card, such as selected application number, file ID, offset, and length of data. We decided to have a separate fact for each data block that can be addressed instead of one fact per file, which results in more than one fact per file. Blocks of different lengths and offsets may overlap, so not all blocks will contain meaningful data. Such blocks will contain the message *corrupted* to easily recognize unwanted data.

Long files will contain many fact definitions, but for modeling purposes we can reduce the number of possible file addresses by defining only the desired addresses and one invalid address instead of all possible invalid addresses. Reading from this invalid address will return *corrupted* and writing to this location will save *corrupted*.

Personalization

Behavior of each smart card type can be modeled using basic principles of applications, authentication, encryption, files, and permissions. All cards of one type has the same behavior. For using in a protocol, such as payment protocol or loyalty program, the smart card must be personalized. Personalization is a process when the smart card is initially populated with data of an intended smart card user, such as the name or the account number. Consequently, each smart card will contain different data in files. This process should be taken into consideration when modeling the smart card protocol. The personalization process does not have to be modeled, since it usually takes place in a trusted environment. The smart card can be used in the modeled protocol only after the personalization, so we can create the model of a card which is already personalized. To create the model of a personalized smart card, all files must be created and populated as they would be during the personalization process.

Integrity

Integrity of data exchanged between the PCD and the PICC is important, but it is not always possible for the PCD or the PICC to check the integrity. The attack definitions described later will cover these attacks so that any attack on integrity will be reported by the model checker.

There are situations in which the PCD or the PICC can check the integrity of data to avoid an attack, such as if some mechanism providing integrity assurance

is used or if the integrity of data is protected by itself. The integrity protecting mechanisms can be for example message authentication code (MAC) or encryption in CBC mode. The data with its own protection mechanism are for example certificates, which are digitally signed. For data with this property we can implement integrity check in the ASLan++ source so that the PCD or PICC can find out that the data has been altered and perform a response to such attack. Otherwise the PCD or the PICC cannot distinguish between genuine data and forged data, so the integrity assurance depends on the inability of attacker to send forged data. The model checker may find an attack on integrity, in such case some integrity mechanism should be implemented.

4.3 Application Logic Model

There are two interacting roles in the ASLan++ model, the PICC, representing the card, and the PCD, representing the terminal. The PICC is only executing commands sent to it from the PCD, so we model the application logic of the protocol in the PCD role. The PCD role contains the application logic of the terminal and of the back-end systems. It issues commands to the PICC and decides what to do next when the response from PICC is received. The PCD represents the protocol run.

During the development, the developer can use the sequence diagram of the protocol or the flow diagram of the application as the basis for the PCD model. The PCD role should contain the logic (or simplified logic) of the application. The intruder can also play the PCD role, but he does not have to follow the logic in the role definition, he can perform arbitrary actions. The role definition is good only for the legitimate entity behavior.

Figure 14 shows the diagram of a sample payment protocol that will be used to demonstrate the protocol logic modeling. The diagram shows only the communication between two legitimate parties where no error occurs. A flow diagram can be used to better describe the logic of the PCD. The PCD role in ASLan++ should reflect the PCD logic shown in the diagram.

The previously described states reduction of the PICC role will reduce the number of commands by making them more complex. So for example the three-pass authentication followed by the *select* command for selecting application and then by the *read* command will result in only one command combining them together. This fact must be taken into account when translating the model checker results into the applicable attack paths.

Figure 15 shows how the PICC role implementation of the protocol may look like when the number of Mifare DESFire commands is reduced only to *read* and *write* in order to reduce model checking execution time. First two parameters of both commands are same. The first parameter is in both cases the address of data to be read or written. Mifare DESFire uses application number, file ID, offset of data in file, and length to address particular data block, so the address will represent the combination of these values. For modeling purposes, each of these combinations will be named according to the variable it will store. So for example the cardholder's name will be stored in application number 1, in file with file ID 1, with offset 0 and length 20; this particular data block address will be named *addressName* to indicate that this address is used to store the name. Other

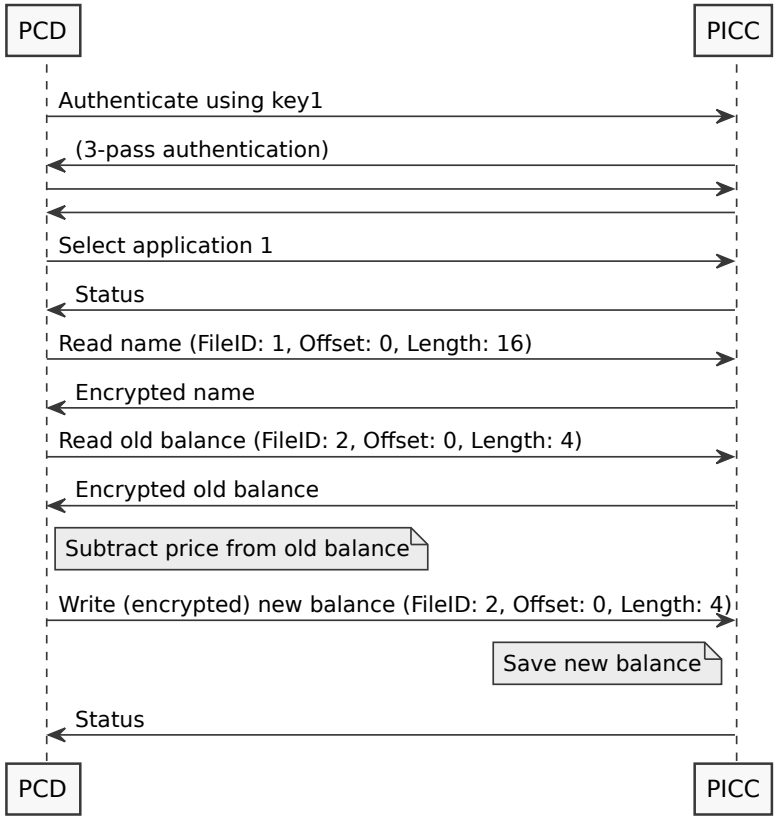

Figure 14: Sample payment protocol

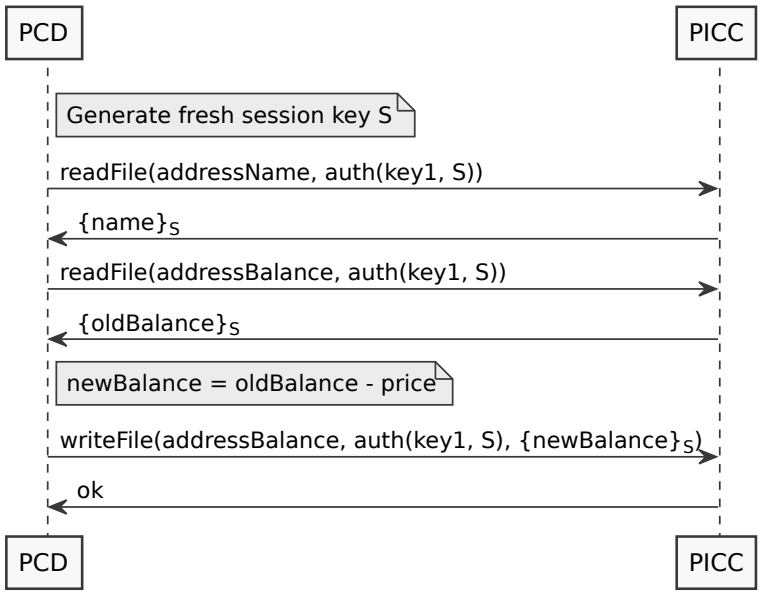

Figure 15: Payment protocol with reduced set of commands

addresses will be named in the same manner. Addresses not intended to store data will also have some name.

The second parameter $auth(key1, S)$ is an authentication token. It is a session key S encrypted using private key $key1$ ($key1$ is shared between legitimate entities and not known by the intruder). The PICC checks whether S is the current session key (no new authentication) or S is a fresh session key (authentication using $key1$). Every old session key (invoked by replay attack) is rejected by the PICC.

The third parameter in the *write* command is the data to be written encrypted using the session key from the second parameter. The response of the *read* command is the data encrypted using the session key from the second parameter, the response of the *write* command is only a status message. Symmetric encryption of $oldBalance$ using key S is denoted $\{oldBalance\}_S$.

4.4 Attack Definition

In the previous sections the model creation was described. The model is written in ASLan++ language, which can be automatically translated to the ASLan language, which is an input format for the back-end model checkers. The attack definition must be provided for the model checker to find any attack traces. The attack is defined as a condition that should never happen in normal protocol run and that means that the intruder learned something that he should not have learned (confidentiality), or that he changed something that he should not have changed (authentication, integrity). These conditions are defined in the ASLan++ model and then translated to states that mean an attack. If the model checker finds a path to one of the attack states, a possible attack is reported. The attack trace should be evaluated and in case of false positive, refinements should be made to the model. The model checker should be run again and this process should be repeated until real attack is found or the model checker concludes that there is no

attack.

Although there are means for defining security goals of confidentiality and authentication in ASLan++, these do not fit well for the purposes of our attack definitions. We will use assertions that will always hold unless an attack is under way. We can easily set goals that the protocol should achieve, covering all desired security goals, by defining assertions in the PCD role that can contain information from PICC which would not be available in real environment, such as content of files (because files are modeled as global facts). Example in figure 16 shows an assertion that can be used at some point in the PCD or PICC role to check content of some file on the card.

```
assert ok: fileSystem(addressBalance,key1,key1,newBalance)
```

Figure 16: Attack definition in ASLan++

We can interpret this assertion as follows: if the file at address *addressBalance* contains the value *newBalance*, it is ok, otherwise the model checker will stop and an attack will be reported.

5 Protocol Modeling Limitations

5.1 Attacks not Covered

Although formal verification methods are useful for finding vulnerabilities on the protocol level, the usability of this technique on other attacks on contactless smart cards is limited. Other attacks, such as physical attacks, side-channel attacks, and attacks specific for contactless communication are out of scope of this method, since this method is not suitable for them and there is no way how to model properties that would be necessary to find such attacks.

In this chapter another method that can increase the security of contactless smart cards is proposed. This method is focused on possible attack that is not covered in the protocol modeling method and cannot be found using formal verification, because it is an attack on low level communication, where timing is of importance.

This chapter is dedicated to preventing relay attacks, which is a type of attack that cannot be prevented on the application level. Relay attacks are possible due to the contactless communication link and were described in section **??**. Two countermeasures are proposed in this chapter. These methods can be used to prevent real attacks that induce delays significantly longer than the delay caused by the time travelling longer distance. They can be used against most likely attacks, which are not expensive and can be easily performed by attackers with moderate skills, which makes them very dangerous.

5.2 Relay Attack Mitigation

We propose a method to prevent real-world attacks that induce delays significantly longer than the delay caused by the time travelling longer distance. This method is described in the first subsection. In the second subsection we show a method that is a countermeasure to the overclocking attacks. The method is based on overclocking the legitimate reader to the limit the communicating card can still reliably operate, which reduces to minimum the time the attacker can gain by overclocking the forged reader. We have implemented the overclocking method in the reader and show the results. The signal was analysed on the oscilloscope. The communication time was reduced while the card was still able to reliably operate.

5.2.1 Passive Detection

The reader can monitor the communication and detect anomalies. It does not make any changes to the transmitted signal or data being sent, so we call it passive detection. Alternatively, the reader monitoring can be provided by an external device such as Proxmark 3, which can be used to eavesdrop on the communication and which provides precise timing data.

This method can be used against relay attacks where significant delays are induced for instance by buffered communication link between attackers' devices. The passive detection is based on precise measuring the responses of all commands. Initially, the fingerprint of each type of smart card is made, all response times are measured and saved for later use. During the communication, all response times are continuously measured and compared to the times saved in the smart card's fingerprint. In case of any anomaly, the possible attack is reported.

Additionally, the reader should have much shorter delay restrictions. The Frame Waiting Time should be restricted to minimal values for which the smart card can operate reliably, and the Frame Waiting Time Extension should be disabled by default and allowed only in reasonable situations.

The relay attack over short distance performed with custom made hardware would not be detected by passive detection. However, attacks over computer network or attacks using off-the-shelf USB readers could be detected, because they induce much bigger delays, as discussed in the previous section. These attacks are not expensive and can be easily performed by attackers with moderate skills, which makes them very dangerous. This countermeasure is quite easy to implement compared to distance bounding protocols. It can be worth implementing such countermeasure even if it does not protect against all theoretical attacks, because it protects against the most likely attacks.

5.2.2 Overclocking

As mentioned earlier, attackers can reduce the round-trip time by overclocking the communication with the legitimate smart card, while communicating on the normal frequency of 13.56 MHz with the legitimate reader. The result is that they get the response from the card faster that the legitimate reader would get it, so they can send the response back sooner than the reader expects and reduce the delay caused by the relay attack. The distance bounding protocol could therefore be circumvented.

The proposed method is based on overclocking the legitimate reader to frequency as high as possible, where the smart card is still reliably operating, which reduces chances for the attackers to perform successful relay attack. The timing is shown in figure 17. The first row depicts the time the ordinary communication takes. This is the time the attacker must not exceed in order to keep the relay attack undetected by the round-trip time measurements. The second line shows the relay attack time, which consists of the delay caused by the relay attack, which is the time of flight of the signal and delays on intermediate devices, and time needed by the attacker to execute the command, which is equal to the time needed in the standard communication. In this case the total time exceeds the time of the standard communication. The third line is the case of overclocking attack, which reduces the time of the command execution by the attacker. In this situation the total time is same as the time of the standard communication, which will likely make the relay attack successful. The last line shows the proposed method of overclocking the legitimate reader, which will result in reducing the time of the standard communication, establishing new time limit. So even if the attacker is overclocking the communication with the legitimate card as well, he will exceed the new time limit.

Implementation with Proxmark

We have implemented the reader that communicates with the smart card on the frequency 16 MHz using Proxmark 3. Figure 18 compares the response times between standard communication at 13.56 MHz and communication of our overclocked reader running at 16 MHz. Mifare DESFire smart card was used and the depicted command is the polling command, which is periodically sent by the reader. By increasing the frequency, approximately $53\mu s$ was spared on this basic command. The response is not clearly visible in the signal, because it is modulated on a subcarrier 848 kHz, so all parts of the communication are marked in the graph.

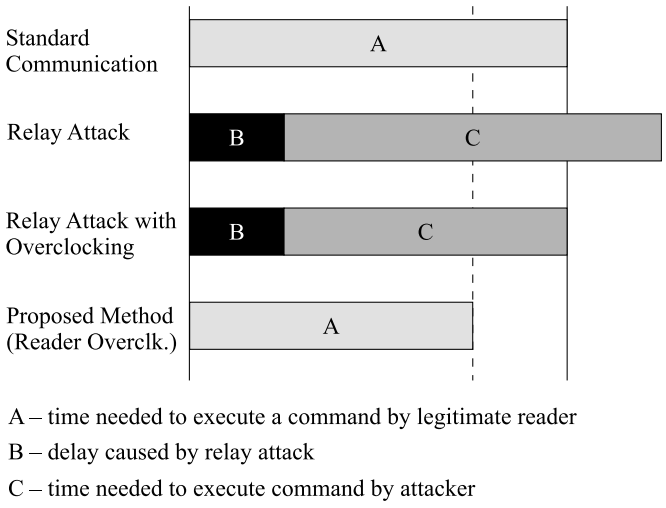

Figure 17: Time consumption

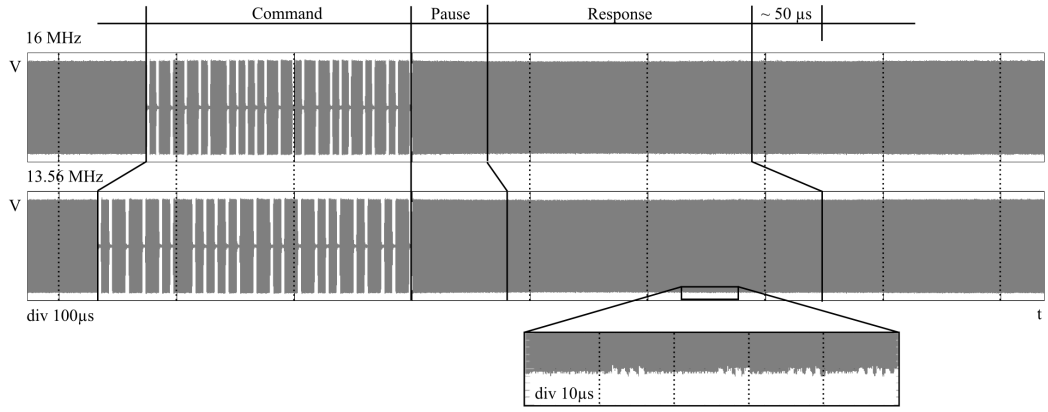

Figure 18: Response times comparison

6 Conclusions

This thesis analyses contactless smart card protocol threats and presents a method of semi-automated vulnerability finding in contactless smart card protocols using model checking. The high level goal of this thesis was to investigate security of contactless smart card protocols and to find methods of improving security of these protocols. The contribution of this thesis is twofold: 1) the method of semi-automated vulnerability finding using formal methods, which can be used for finding high level attacks on the protocol level, and 2) the countermeasures to relay attacks performed over a network, which were created after relay attacks investigation.

The focus in this thesis is on the high level attacks on the protocol level. Possibility of these attacks was analysed and a method of semi-automated vulnerability finding using formal methods was proposed. The formal model can be created from the protocol definition or extracted from the eavesdropped communication. Unwanted states that pose an attacks are specified. After analysing the protocol and creating the model including the attack states, model checking can be used to automatically find vulnerabilities.

AVANTSSAR platform is used for the formal verification, the models are written in the ASLan++ language. Examples demonstrate the usability of the proposed method.

This thesis deals mainly with simple smart cards with fixed file structure and pre-defined set of commands. These smart cards provide authentication based on symmetric keys, multiple applications and file system with access permissions. Access control is based on keys that are used for authentication, data may be encrypted using some symmetric cipher. One of the most popular and widespread contactless smart cards that uses this scheme is Mifare DESFire, which was used in examples in this thesis. Other smart cards have more sophisticated operating system and can execute applications on their chip, such as Java Cards, MULTOS cards or BasicCards. Their application logic can be modeled as well, but this thesis is focused mainly on smart cards with fixed file structure and pre-defined set of commands.

The method presented in this thesis was used to find a previously unpublished weakness of the Mifare DESFire MF3ICD40 contactless smart card. Some features of the Mifare DESFire MF3ICD40 were found to be very dangerous and it may be very difficult to implement protocol using this card in a secure way. Although these features are not considered vulnerabilities of the smart card itself, they help to introduce vulnerabilities into the implementation.

We have shown how the inappropriate protocol implementation can yield new vulnerabilities even if the protocol itself is secure and the communication with the hardware is considered secure too. We have demonstrated a sample attack on fictional payment protocol implementation on Mifare DESFire smart card. There is a potential for adversaries to perform similar attacks on real systems. We have introduced a concept of automated vulnerability search using formal verification methods to find complex attack traces which are not likely to be found manually. There is a possibility to use the source code to get an overall image of the protocol and to create the model which is as close to reality as possible, or a man-in-the-middle attack can be used to get information about the protocol from the implementation.

Not all kinds of attacks are covered by the proposed method, so one type of the remaining attack types – the relay attack – was investigated separately. A minor part of this thesis was dedicated to relay attack investigation and countermeasure proposal.

We have proposed a method based on passive detection to prevent real attacks that induce delays significantly longer than the delay caused by the time travelling longer distance. It can be used against most likely attacks, which are not expensive and can be easily performed by attackers with moderate skills, which makes them very dangerous. This countermeasure is quite easy to implement compared to distance bounding protocols. It can be worth implementing such countermeasure even if it does not protect against all theoretical attacks.

We have shown a possible countermeasure to the overclocking attacks. The method is based on overclocking the legitimate reader to the maximal limit where the communicating card can still reliably operate. This method reduces to minimum the chances of the attacker to gain time by overclocking the communication with the legitimate card and hence to circumvent the time limit. We have implemented the reader that communicates with a smart card on the frequency 16 MHz and tested it with a real card.

Further research may be focused on finding more automatic methods of creating formal model from the analysed protocol. Learning techniques allow automatic inference of behaviour of a system as a finite state machine and can be used to extract such formal models from software on smart cards or to extract the protocol. Such automated reverse-engineering takes little effort and is fast. The finite state machine models obtained can be used in the method presented in this thesis. This approach would improve this method by making it more automatic.

The results presented in this thesis were published in journal [10] with impact factor and international conferences [11], [9], and [12].

Abstrakt

Tato práce analyzuje hrozby pro protokoly využívající bezkontaktní čipové karty a představuje metodu pro poloautomatické hledání zranitelností v takových protokolech pomocí model checkingu. Návrh a implementace bezpečných aplikací jsou obtížné úkoly, i když je použit bezpečný hardware. Specifikace na vysoké úrovni abstrakce může vést k různým implementacím. Je důležité používat čipovou kartu správně, nevhodná implementace protokolu může přinést zranitelnosti, i když je protokol sám o sobě bezpečný. Cílem této práce je poskytnout metodu, která může být využita vývojáři protokolů k vytvoření modelu libovolné čipové karty, se zaměřením na bezkontaktní čipové karty, k vytvoření modelu protokolu a k použití model checkingu pro nalezení útoků v tomto modelu. Útok může být následně proveden a pokud není úspěšný, model je upraven pro další běh model checkingu. Pro formální verifikaci byla použita platforma AVANTSSAR, modely jsou psány v jazyce ASLan++. Jsou poskytnuty příklady pro demonstraci použitelnosti navrhované metody. Tato metoda byla použita k nalezení slabiny bezkontaktní čipové karty Mifare DESFire. Tato práce se dále zabývá hrozbami, které není možné pokrýt navrhovanou metodou, jako jsou útoky relay.

Abstract

This thesis analyses contactless smart card protocol threats and presents a method of semi-automated vulnerability finding in such protocols using model checking. Designing and implementing secure applications is difficult even when secure hardware is used. High level application specifications may lead to different implementations. It is important to use the smart card correctly, inappropriate protocol implementation may introduce a vulnerability, even if the protocol is secure by itself. The goal of this thesis is to provide a method that can be used by protocol developers to create a model of arbitrary smart card, with focus on contactless smart cards, to create a model of the protocol, and to use model checking to find attacks in this model. The attack can be then executed and if not successful, the model is refined for another model checker run. The AVANTSSAR platform was used for the formal verification, models are written in the ASLan++ language. Examples are provided to demonstrate usability of the proposed method. This method was used to find a weakness of Mifare DESFire contactless smart card. This thesis also deals with threats not possible to cover by the proposed method, such as relay attacks.

References

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

[2] Armando, A.; Carbone, R.; Compagna, L.: LTL Model Checking for Security Protocols. In *Computer Security Foundations Symposium, 2007. CSF '07. 20th IEEE*, July 2007, ISSN 1940-1434, s. 385–396, doi:10.1109/CSF.2007.24.

[3] Armando, A.; Carbone, R.; Compagna, L.: SATMC: a SAT-based Model Checker for Security-critical Systems. In *TACAS'14: Proceedings of the 20th international Conference on Tools and Algorithms for the Construction and Analysis of Systems*, Springer, 2014, ISBN 978-3-642-54861-1, s. 31–45.
URL `http://www.ai-lab.it/armando/pub/tacas2014.pdf`

[4] Basin, D.; Cremers, C.; Meadows, C.: Model checking security protocols. *Handbook of Model Checking*, 2011.

[5] Basin, D.; Mödersheim, S.; Viganò, L.: OFMC: A symbolic model checker for security protocols. *International Journal of Information Security*, ročník 4, č. 3, 2004: s. 181–208, ISSN 1615-5270, doi:10.1007/s10207-004-0055-7.
URL `http://dx.doi.org/10.1007/s10207-004-0055-7`

[6] Basin, D.; Mödersheim, S.; Vigano, L.: Algebraic intruder deductions. In *Logic for Programming, Artificial Intelligence, and Reasoning*, Springer, 2005, s. 549–564.

[7] Cremers, C. J.; Lafourcade, P.; Nadeau, P.: Comparing State Spaces in Automatic Protocol Analysis. In *Formal to Practical Security*, *Lecture Notes in Computer Science*, ročník 5458/2009, Springer Berlin/Heidelberg, 2009, s. 70–94.

[8] Dolev, D.; Yao, A. C.: On the security of public key protocols. *Information Theory, IEEE Transactions on*, ročník 29, č. 2, 1983: s. 198–208.

[9] Henzl, M.; Hanacek, P.: Modeling of Contactless Smart Card Protocols and Automated Vulnerability Finding. In *Biometrics and Security Technologies (ISBAST), 2013 International Symposium on*, July 2013, s. 141–148, doi:10.1109/ISBAST.2013.26.

[10] Henzl, M.; Hanacek, P.: A Security Formal Verification Method for Protocols Using Cryptographic Contactless Smart Cards. *Radioengineering*, ročník 25, č. 1, April 2016: s. 132–139, ISSN ISSN 1210-2512, doi:10.13164/re.2016.0011.

[11] Henzl, M.; Hanacek, P.; Jurnecka, P.; aj.: A concept of automated vulnerability search in contactless communication applications. In *Security Technology (ICCST), 2012 IEEE International Carnahan Conference on*, Oct 2012, ISSN 1071-6572, s. 180–186, doi:10.1109/CCST.2012.6393556.

[12] Henzl, M.; Hanacek, P.; Kacic, M.: Preventing real-world relay attacks on contactless devices. In *Security Technology (ICCST), 2014 International Carnahan Conference on*, Oct 2014, s. 1–6, doi:10.1109/CCST.2014.6987031.

[13] Lafourcade, P.; Terrade, V.; Vigier, S.: Comparison of cryptographic verification tools dealing with algebraic properties. In *Formal Aspects in Security and Trust*, Springer, 2009, s. 173–185.

[14] Meadows, C. A.: *Computer Security — ESORICS 96: 4th European Symposium on Research in Computer Security Rome, Italy, September 25–27, 1996 Proceedings*, kapitola Analyzing the Needham-Schroeder public key protocol: A comparison of two approaches. Berlin, Heidelberg: Springer Berlin Heidelberg, 1996, ISBN 978-3-540-70675-5, s. 351–364.

[15] Mödersheim, S.; Vigano, L.; Basin, D.: Constraint differentiation: Search-space reduction for the constraint-based analysis of security protocols. *Journal of Computer Security*, ročník 18, č. 4, 2010: s. 575–618.

[16] von Oheimb, D.; Mödersheim, S.: ASLan++ – A Formal Security Specification Language for Distributed Systems. In *Formal Methods for Components and Objects*, *Lecture Notes in Computer Science*, ročník 6957, editace B. Aichernig; F. de Boer; M. Bonsangue, Springer Berlin Heidelberg, 2012, ISBN 978-3-642-25270-9, s. 1–22.

[17] SATMC: A SAT-based Model-Checker for Security Protocols and Security-sensitive Applications. [Online] Cited 2016-04-12. Available at: `http://www.ai-lab.it/satmc/`.

[18] Turuani, M.: The CL-Atse Protocol Analyser. In *Term Rewriting and Applications - Proc. of RTA*, *Lecture Notes in Computer Science*, ročník 4098, Seattle, WA, USA, 2006, ISBN 3-540-36834-5, s. 277–286.

MARTIN HENZL

Úvoz 52, 60200 Brno | +420 608 828 796 | martin.henzl@gmail.com

EDUCATION

Ph.D. candidate **2009 - Present**
Brno University of Technology, Faculty of Information Technology
Doctoral Degree Programme Computer Science and Engineering
Ph.D. Thesis: "Security Protocols of Contactless Smart Cards"

Mgr. **2007 - 2009**
Masaryk University, Faculty of Informatics
Master Degree Programme Information Technology Security

Ing. **2007 - 2009**
Brno University of Technology, Faculty of Information Technology
Master Degree Programme Computer Graphics and Multimedia

Bc. **2004 - 2007**
Masaryk University, Faculty of Informatics
Bachelor Degree Programme Informatics

PROFESSIONAL EXPERIENCE

R&D Scientist - Aerospace **2013 - Present**
Honeywell International, s.r.o.

Security Auditor **2013 - Present**
Security audits

Technical and Research Worker **2009 - 2013**
Brno University of Technology, Faculty of Information Technology
Teaching (lectures and seminars) in courses: Cryptography; Cryptography and Information Security; Information System Security; Wireless and Mobile Networks

Freelance Programmer **2009 - 2013**
Various piece works, mainly C#, .NET.

PUBLICATIONS AND PAPERS

Henzl, M., Hanáček, P. *A security formal verification method for protocols using cryptographic contactless smart cards.* Radioengineering journal, IF 0.653. 2016

Drahanský, M., Hanáček, P., Zbořil, F., Henzl, M., Zbořil, F. V., Yim J., Shim, K. *Cryptomodules in Wireless Networks using Biometric Authentication: Securing Nodes in Wireless Networks.* Improving Information Security Practices through Computational Intelligence. Book: IGI Global 2015

Henzl, M., Barabas, M., Janča, R., Hanáček, P. *Bezpečnost bezkontaktních platebních karet.* DSM Data Security Management journal. 2014

Henzl, M., Hanáček, P., Kačic M. *Preventing Real-world Relay Attacks on Contactless Devices.* In: International Carnahan Conference on Security Technology. Rome, Italy. 2014

Henzl, M., Hanáček, P. *Modeling of Contactless Smart Card Protocols and Automated Vulnerability Finding*, International Symposium on Biometrics and Security Technologies, Chengdu, China 2013

Jurnečka, P., Hanáček, P., Barabas, M., Henzl, M., Kačic, M. *A method for parallel software refactoring for safety standards compliance.* In: System Safety 2013 collection of papers. Cardiff, UK 2013

Jurnečka, P., Hanáček, P., Barabas, M., Henzl, M., Kačic, M. *A method for parallel software refactoring for safety standards compliance.* Resilience, Security & Risk in Transport. The Institution of Engineering and Technology. London, UK 2013

Kačic, M., Henzl, M., Hanáček, P. *A Concept of Behavioral Reputation System in Wireless Networks*, 47th International Carnahan Conference on Security Technology, Medellin, Colombia 2013

Kačic, M., Henzl, M., Jurnečka P., Hanáček, P. *Malware injection in wireless networks*, The 7th IEEE International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications, Berlin, Germany 2013

Henzl, M., Hanáček, P., Jurnečka, P., Kačic, M. *A Concept of Automated Vulnerability Search in Contactless Communication Applications*, In: Proceedings 46th Annual IEEE International Carnahan Conference on Security Technology, Boston, USA 2012

Henzl, M., Hanáček, P. *NFC z pohledu bezpečnosti*, In: DSM Data Security Management, Czech Republic 2011

Henzl, M. *Security of Contactless Smart Cards*, In: Proceedings of the 17th Conference STUDENT EEICT 2011, Brno, Czech Republic 2011

ADDITIONAL SKILLS

Driving Licence – Category B
Private Pilot Licence - PPL(A)