Module 05: Introduction to Tamarin Proofs

Week-11: The Mastercard contactless protocol

David Basin, Dennis Jackson, Joseph Lallemand, and Jorge Toro

basin@inf.ethz.ch, djackson@inf.ethz.ch
joseph.lallemand@inf.ethz.ch, jorge.toro@inf.ethz.ch

November 2020

Welcome to the lab session for 11th week of the Information Security Lab. This session is part of the Module-05 on *Introduction to Tamarin Proofs*.

Instructions for Online Lab

We begin with the usual pointers and instructions. As you are aware, the lab sessions will be held online via Zoom. We ask that all students join the Zoom session via the following link:

`https://ethz.zoom.us/j/91450896871`

The first 30-ish minutes of the Zoom session will consist of a general presentation and discussion about the lab objectives and evaluation criteria. Afterwards, you may choose to stay on in the lab session to discuss specific problems you might be having with the lab. You are also welcome to post your questions and to join ongoing discussions on the Moodle forum at the following link:

`https://moodle-app2.let.ethz.ch/mod/forum/view.php?id=475218`

During the Zoom session, we will be hosting "break-out room". These are meant to facilitate one-on-one discussions with one of the TAs. During such discussions, you can share your screen and bring to our attention specific implementation issues that you might be facing. Please be aware that when you are sharing a screen with the session, you are responsible for the content that is presented to any other students in the breakout. In other words, *please be mature*.

Overview

EMV, founded by companies Europay, Mastercard, and Visa (thus the name) is the international protocol standard for in-store smartcard payments. In this lab we are going to develop a Tamarin security model and use it to perform a formal analysis of a simplified version of the EMV protocol for *Mastercard contactless transactions*.

The goal of this lab is for you to deepen your modeling and analysis skills, with the use case of a real-world protocol that is used in over 2 billion payment

cards and 40 million payment terminals worldwide. Specifically, we will train the modeling of (i) cryptographic mechanisms involving both asymmetric and symmetric crypto, (ii) communication channels with different levels of security, (iii) multiple roles, and (iv) execution branching (similar to *if* statements in imperative languages).

Please note that we expect you to use the Tamarin tool. We will **not** accept submissions written in any other languages or tools, including manually developed proofs.

Getting started

Before we dive into modeling, let's test the working environment. For this, follow the steps:

1. Install Tamarin by following the instructions given at:
   `https://tamarin-prover.github.io/manual/book/002_installation.html`.
2. Once you have installed the tool, create a file `Sig.spthy` with the code (watch out missing characters when copying/pasting):

```
theory Sig

begin
builtins: signing

rule LtkGen://PKI
  [ Fr(~ltk) ]
-->
  [ !Ltk($A, ~ltk), !Pk($A, pk(~ltk)), Out(pk(~ltk)) ]

//protocol rules
rule Send_Signature:
  [ Fr(~n), !Ltk($A, ltkA) ]
--[ Send($A, ~n) ]->
  [ Out(<~n, sign{~n}ltkA>) ]

rule Recv_Signature:
  [ !Pk($A, pkA), In(<n, sig>) ]
--[ Recv($A, n), Eq( verify(sig, n, pkA), true ) ]->
  [ ]

restriction equal://needed for signature verification
  "All a b #i. Eq(a, b)@i ==> a = b"

lemma executable://sanity check
  exists-trace "Ex A n #i #j. Send(A, n)@i & Recv(A,n)@j"

lemma signature_sent_by_agent://property to be verified
  "All A n #i. Recv(A, n)@i ==> Ex #j. Send(A, n)@j"

end
```

3. Run `tamarin-prover --prove Sig.spthy` and you should get the outcome:

```
...
```

```
analyzed: Sig.spthy

  executable (exists-trace): verified (6 steps)
  signature_sent_by_agent (all-traces): verified (5 steps)

==============================================================================
```

Overview of EMV contactless payment

Cashless means of payment are becoming more and more popular. These means include smartcards, smartphones, and smartwatches. In fact, almost every modern smart device is capable of transacting with payment terminals via Near Field Communication (NFC), a technology that allows for short-range wireless communication. In transactions using NFC, called *contactless*, the payment card (or the payment device, to be more general) is held near the payment terminal and hence the two parties exchange messages called commands and responses. For this lab, the encoding of commands and responses is irrelevant and we will simply consider them as messages.

In the EMV contactless transaction, after the card-terminal interaction is successfully completed and typically when the amount is small, the terminal may accept the transaction offline. The funds are *not* immediately transferred from the cardholder's account to the merchant's, but typically at the end of the day. If the transaction is otherwise required to be authorized by the card issuer (which is indeed the most common scenario), then the terminal goes online, establishes a communication channel with the issuer, and sends the authorization request. Upon reception of the request, the card issuer undergoes a number of checks, including that the cardholder's account has sufficient funds for the amount requested and that the card-produced cryptographic proof for the transaction is correct. If everything checks out, the issuer sends back to the terminal the authorization response. See below a graphical representation of the interaction between the card, the terminal, and the issuer in a contactless transaction.

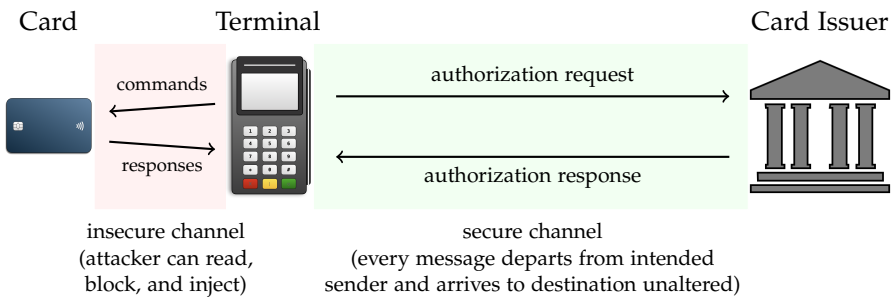

Figure 1: The three agents and their interaction during a contactless transaction.

The card-terminal interaction is determined by one of the six EMV contactless protocols (called *kernels* in EMV's terminology), each of them corresponding to one of the card brands: Mastercard, Visa, American Express, JCB, Discover, and UnionPay. As said before, we will focus on a simplified variation of the Mastercard protocol. The 600-page specification of the full protocol is available at

https://www.emvco.com/wp-content/uploads/documents/C-2-Kernel-2-V2.9-final_3.pdf. There is no need to read the full specification for this lab!

The (simplified) Mastercard protocol

This section provides technical background on the Mastercard contactless protocol. For the purpose of this lab, substantial parts of the protocol have been omitted and also some messages have been simplified. Omissions include all but one authentication mode and all cardholder verification methods. An overview of the resulting execution flow of the Mastercard contactless transaction is given in Figure 2. The crypto-related notation in the figure and also in the rest of this material is the following:

- mk is a (master) key only known to the card and its issuer.
- f is a key derivation function.
- $(privC, pubC)$ is the private/public key pair of the card. $pubC$ is publicly known.
- $sign_{priv}(m)$ is the digital signature on m with the private key $priv$.
- $verify(sig, m, pub)$ equals true if and only if $sig = sign_{priv}(m)$ and $(priv, pub)$ is a valid private/public key pair.
- $MAC_k(m)$ and $MAC'_k(m)$ are Message Authentication Codes (MAC) on m with the key k.

The remainder of this section describes in detail the steps of the protocol.

1. *Application Selection*: An EMV contactless transaction is performed using one of the six EMV contactless protocols/kernels mentioned before. The negotiation of the kernel to be used for the transaction is the first phase of it. This negotiation is done via two exchanges of '`SELECT`' commands and responses.

   This brief description of the application selection phase is merely informative. We will *not* model this phase since we are focusing on the Mastercard kernel.

2. *Offline Data Authentication (ODA)*: After a kernel has been selected (Mastercard in our case), the terminal sends the '`GET PROCESSING OPTIONS`' command with the Processing Data Object List (PDOL) as payload. The PDOL is composed of the *amount* of the transaction and a terminal-generated random number un, called the Unpredictable Number (UN) in EMV's terminology.

   To this command, the card responds with the Application Interchange Profile (AIP). The AIP informs the terminal of the card's authentication capabilities. In this lab we will assume that AIP = '`DDA`' and will explain later what this entails.

   The terminal then sends the '`READ RECORD`' command to read the card's records, which are static data stored in the card. We will assume these records to be

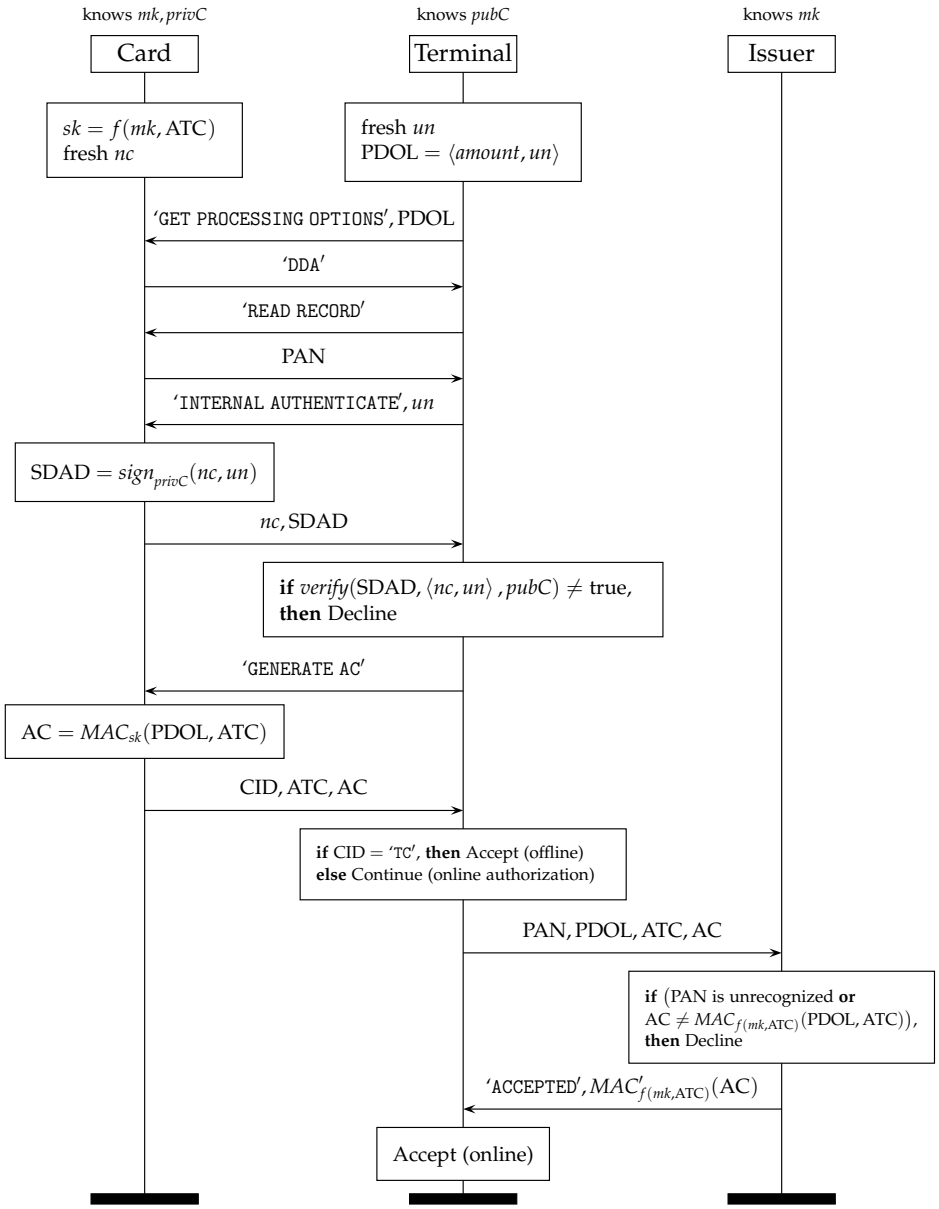

Figure 2: Overview of the (simplified) Mastercard contactless transaction. While in the full protocol further actions and messages follow the Accept and Decline actions, for this lab we will assume that they simply entail a stop in the execution.

composed solely of the Primary Account Number (PAN, commonly known as the *card number*).

The terminal then proceeds to cryptographically authenticate the card, using the method indicated by the AIP. Our assumption on the AIP being 'DDA' entails that the Dynamic Data Authentication (DDA) method will be used. In this method, the terminal sends the 'INTERNAL AUTHENTICATE' command along with the terminal's random number *un*. The card replies with the Signed Dynamic Authentication Data (SDAD), which is a signature by the card on its own random number *nc* and that of the terminal. The full Mastercard protocol features two further Offline Data Authentication methods: the Static Data Authentication (SDA) and the Combined Dynamic Data Authentication (CDA) methods, but we will **not** consider them in this lab.

3. *Transaction Authorization (TA)*: For the transaction authorization, the terminal must have the card generate and supply the Application Cryptogram (AC), which is a cryptographic proof of the transaction. For this, the terminal sends the '`GENERATE AC`' command, to which the card responds with the Cryptogram Information Data (CID), the Application Transaction Counter (ATC), and the AC itself.

   The CID, which can be '`TC`' or '`ARQC`', indicates the type of transaction authorization: offline or online. The Application Cryptogram (AC) is a MAC computed over the PDOL (i.e. $\langle amount, un \rangle$) and the ATC. The computation by the card (and verification by the issuer) of the AC uses the session key *sk* derived from a shared master key *mk* and the ATC. The key *mk* is only known to the issuer and the card.

   The transaction continues as follows:

   - if CID = '`TC`', then the transaction is approved offline by the terminal. In this case, the AC is called the Transaction Cryptogram (TC).
   - else (i.e. if CID = '`ARQC`'), then the terminal forwards the transaction data to the issuer for online authorization. The data to be forwarded is composed of the card number (i.e. the PAN), the PDOL, the transaction counter ATC, and the Application Cryptogram (AC). In this case, the AC is called the Authorization Request Cryptogram (ARQC).

     The issuer authorizes if the AC is correct, in which case it sends back to the terminal the authorization code '`ACCEPT`' and an issuer-generated MAC, called the Authorization Response Cryptogram (ARPC). The ARPC serves, among other things, as a cryptographic proof for the terminal that the issuer accepted the transaction.

Notice that the terminal **cannot** verify the AC, nor the ARPC, because the key *mk* and *sk* by extension are unknown to terminal.

Tasks

Download the complementary material from:

`https://moodle-app2.let.ethz.ch/mod/resource/view.php?id=512291`

which contains two files:

- `Mastercard.spthy`, which is a skeleton model for the (simplified) Mastercard contactless protocol we just described, and
- `Makefile`, which is a shell script that you must use to produce the analysis results.

Task 1. Complete the skeleton model by *adding* the appropriate Tamarin code

(e.g. rules, lemmas, restrictions) to it such that the resulting model satisfies the following requirements:

1. It must model the three roles present in a transaction: the *card*, the *terminal*, and the *issuer*.
2. It must allow for the execution of the two types of accepted transactions: *offline* and *online*.
3. Every rule where the *card completes a transaction* must have the form:

```
rule Name:
  [ ... ]
--[ Running(~PAN, 'Terminal', <'Card', 'Terminal', transaction>),
    Running(~PAN, $Issuer, <'Card', 'Issuer', transaction>), ... ]->
  [ ... ]
```

4. Every rule where the *terminal completes its interaction with the card for a transaction requiring online authorization* must have the form:

```
rule Name:
  [ ... ]
--[ Running($Terminal, $Issuer, <'Terminal', 'Issuer', transaction>), ... ]->
  [ ... ]
```

5. Every rule where the *terminal accepts a transaction offline* must have the form:

```
rule Name:
  [ ... ]
--[ Commit('Terminal', ~PAN, <'Card', 'Terminal', transaction>), ... ]->
  [ ... ]
```

6. Every rule where the *terminal accepts an online-authorized transaction* must have the form:

```
rule Name:
  [ ... ]
--[ Online(),
    Commit('Terminal', ~PAN, <'Card', 'Terminal', transaction>),
    Commit($Terminal, $Issuer, <'Issuer', 'Terminal', transaction>), ... ]->
  [ ... ]
```

7. The `Commit` and `Running` facts in the above rules must be used for the agreement-based authentication properties, which will be described later.
8. The `~PAN`, `$Terminal` and `$Issuer` terms in the above rules must be the identities of the agents instantiating the card, terminal, and issuer roles, respectively. The term `transaction` must be a message composed of the PAN, the PDOL, the ATC, and the AC.
9. The *card-terminal channel* must be modeled to be under full control of the adversary who can read, block and inject messages (recall that this is kwnon as the Dolev-Yao adversary model). For this you must use Tamarin's `In` and `Out` facts.
10. There are no rules to model the *terminal-issuer channel*, the *compromise of agents*, or the *issuer role*, other than the ones provided in the skeleton model.

11. The `executable_...` lemmas provided in the skeleton model must verify. They are sanity check lemmas to verify that your model is operable, i.e. it admits the completion of "clean" transactions without adversarial intervention other than the delivery of messages sent over the card-terminal channel.

12. It must allow for the analysis of the three authentication properties:

    (a) All transactions accepted offline by the terminal are authenticated to it by the card. The lemma for this must be named: `auth_to_terminal_offline`.

    (b) All online-authorized transactions accepted by the terminal are authenticated to it by *both* the card and the issuer. The lemma for this must be named: `auth_to_terminal_online`.

    (c) All transactions accepted by the issuer are authenticated to it by *both* the card and the terminal. The lemma for this must be named: `auth_to_issuer`.

13. The three authentication properties must be implemented using the **non-injective agreement** property as defined in `https://tamarin-prover.github.io/manual/book/007_property-specification.html` with the message to agree on being $\langle \text{PAN}, \text{PDOL}, \text{ATC}, \text{AC} \rangle$, as explained before in item 8.

Please name your model file `Mastercard_<number>.spthy`, where `<number>` is your matriculation number. This naming convention will allow us to run reporting/grading scripts. Submit your model file.

Task 2. Put the `Makefile` in the same folder as your model, open a terminal at that folder and then run:

```
make model=Mastercard_<number>.spthy,
```

which will create a file named `Mastercard_<number>.proof` with the Tamarin-produced analysis results of your model. Submit this file.

For each of property violations that the tool finds, if any, write in a complementary `README_<number>.txt` file a possible exploit of it by criminals in real world. For this you can inspect the constraint system found by the tool for the violated properties. For this, use Tamarin in interactive mode by running:

```
tamarin-prover interactive Mastercard_<number>.spthy.
```

If applicable, submit the `README_<number>.txt` file too.

Hints

Below we give a few hints that you might find be useful when developing your model:

1. The skeleton model offers the rule `Generate_ATC`, which models the "pool" of the Application Transaction Counters (ATC). From this pool is where the card looks the counter up in every transaction. Notice that the `Once(...)` facts of the provided `Card_Responds_To_GPO` and `Issuer_Receives_AC` rules along with the `once` restriction make sure that the ATC is different per session, from a card's perspective (note that two different cards can have the same ATC, which is a realistic scenario). You should not need further code to model the generation or lookup of the ATC as your card rules can carry it in the state facts across the full transaction.

2. To use the terminal-issuer secure channel provided in the skeleton model, you should think of a way to instantiate the `channelID` term within the rules. A good practice is to have it be of the form `<id, label>` where `id` is a session identifier and `label` is the number of the message in said session. This is indeed the way that the skeleton model uses the `channelID` term, see the `Send` and `Recv` facts in the issuer rules.

3. To deal with execution branching in Tamarin, it is a common practice to "multiply" rules. For example, say you have a protocol where an agent goes to two different states, depending on what it receives. This can be modeled with the two rules:

```
rule Go_To_1:
  [ State_0(), In('go_to_1') ]-->[ State_1() ]

rule Go_To_2:
  [ State_0(), In('go_to_2') ]-->[ State_2() ]
```

   You might use this idea to model the **if** statement for offline vs. online authorization.

4. Finally, decline actions need not be modeled explicitly (at least not in this lab). For example, say you are modeling an agent that stops the execution, provided that some condition C is met. Then you should simply have a rule that models the continuation of the execution, provided that the **negation** of C holds. See this idea implemented in the provided `Issuer_Receives_AC` rule where the issuer checks that the AC is correct and that the PAN indicates one of its cards. This is indeed the negation of the condition for the issuer's decline action in Figure 2.

Good luck!

Acronyms

AC Application Cryptogram. 6, 7, 9

AIP Application Interchange Profile. 4, 5

ARPC Authorization Response Cryptogram. 6

ARQC Authorization Request Cryptogram. 6

ATC Application Transaction Counter. 6, 7, 9

CDA Combined Dynamic Data Authentication. 5

CID Cryptogram Information Data. 6

DDA Dynamic Data Authentication. 5

MAC Message Authentication Code. 4, 6

NFC Near Field Communication. 3

ODA Offline Data Authentication. 4, 5

PAN Primary Account Number. 5–7

PDOL Processing Data Object List. 4, 6, 7

SDA Static Data Authentication. 5

SDAD Signed Dynamic Authentication Data. 5

TA Transaction Authorization. 6

TC Transaction Cryptogram. 6

UN Unpredictable Number. 4