Formal Analysis of SPDM: Security Protocol and Data Model version 1.2

Cas Cremers, Alexander Dax, and Aurora Naska,
CISPA Helmholtz Center for Information Security

https://www.usenix.org/conference/usenixsecurity23/presentation/cremers-spdm

This artifact appendix is included in the Artifact Appendices to the Proceedings of the 32nd USENIX Security Symposium and appends to the paper of the same name that appears in the Proceedings of the 32nd USENIX Security Symposium.

August 9–11, 2023 • Anaheim, CA, USA

978-1-939133-37-3

Open access to the Artifact Appendices to the Proceedings of the 32nd USENIX Security Symposium is sponsored by USENIX.

USENIX'23 Artifact Appendix: Formal Analysis of SPDM: Security Protocol and Data Model version 1.2

Cas Cremers
CISPA Helmholtz Center for Information Security

Alexander Dax
CISPA Helmholtz Center for Information Security

Aurora Naska
CISPA Helmholtz Center for Information Security

A Artifact Appendix

A.1 Abstract

This document contains the description of the formal models and proofs of the three modes of Security Protocol and Data Model (SPDM) protocol version 1.2.1. We provide four models that capture the main device attestation mechanism, and the three modes of key exchange with (i) preshared symmetric keys, (ii) preshared public keys, and (iii) public key pair, certificates over the public key, and a root of trust. During our analysis we prove the main security guarantees of each of the models, such as Responder Authentication, Measurements Authentication, Handshake Secrecy, etc. Our proofs and models are formalized using the Tamarin Prover's input language.

We provide the artifacts in a public Github repository for inspection and reproduction with instructions on how to replicate each of the proofs. In addition, the repository includes a Python programm to obtain all our results automatically.

A.2 Description & Requirements

A.2.1 Security, privacy, and ethical concerns

None.

A.2.2 How to access

All our files are public and can be inspected and reused by accessing the following GitHub repository https://github.com/FormalAnalysisOf/SPDM/tree/V1.

A.2.3 Hardware dependencies

The hardware dependencies are inherited from the Tamarin Prover, although the manual of the latter does not mention any hardware dependencies. To the best of our knowledge, any modern notebook should be sufficient to run Tamarin.

A.2.4 Software dependencies

In the following we list all software dependencies:

1. Tamarin Prover[1] which depends on `haskell-stack`, `graphviz`, and `maude`. Note that Tamarin does not run on Windows systems. To run Tamarin on Windows refer to WSL[2].
2. Python3 - install `pip`, and use it to install `tabulate` and `matplotlib`.

A.2.5 Benchmarks

None.

A.3 Set-up

A.3.1 Installation

Clone the repository using

```
$ git clone
 ↪  https://github.com/FormalAnalysisOf/SPDM.git
```

To install the Python dependencies, please install *Python3, pip, tabulate*, and *matplotlib*. For instance, on an Ubuntu system you can install them using

```
$ apt install python3
$ apt install pip3
$ pip3 install tabulate matplotlib
```

Afterwards install the development branch of Tamarin.

Installing Tamarin Some package managers let you install Tamarin directly. We suggest compiling it from scratch (develop branch) using the manual https://tamarin-prover.github.io/manual/book/002_installation.html. for instructions. The exact commit we used to obtain our proofs in the develop branch is:

[1] https://tamarin-prover.github.io/manual/book/002_installation.html

[2] https://learn.microsoft.com/en-us/windows/wsl/install

```
7c980321158ebf7c7c03c53cee93248507584065
```

Our models can also be executed using the latest development version of Tamarin, however, that might affect the execution's runtime.

A.3.2 Basic Test

Make sure that the *tamarin-prover* executable is in the *$PATH*. To test if you correctly installed the Tamarin Prover, execute

```
$ tamarin-prover test
```

You should see a message containing
- a check for `maude`,
- a check for `Grapviz`, and
- a test for the unification structure (0 errors and 0 failures).

In the end you should see the following:

```
All tests successful.
The tamarin-prover should work as intended.
        :-) happy proving (-:
```

A.4 Evaluation workflow

A.4.1 Major Claims

Our artifact contains formal models of the SPDM specification and the means to execute them. The models include at the end of the file the main security properties of our analysis, and sanity checks expressed by the so-called *lemmas*, which are a formal representation using the Tamarin Porver's input language. The user can automatically reproduce the proofs for all lemmas by either using the python program or the instruction sets in our artifact. We state the security properties investigated in our analysis in the following:

(C1) Device attestation: For the part of SPDM that aims to provide device attestation, we prove **Responder Authentication** and **Measurement Authentication**. Definitions of those properties can be found in Section 4 of our paper. Our claims are proven with experiment **(E1)**.

(C2) Certificate KEX: For the key exchange based on certificates and public key cryptography, we prove **Responder Authentication**, **Mutual Authentication**, and **Handshake Secrecy**. Definitions of those properties can be found in Section 4 of our paper. Our claims are proven with experiment **(E2)**.

(C3) Pre-Shared KEX: For the key exchange based on preshared public keys, we prove **Mutual Authentication** and **Handshake Secrecy**. Further, for a restricted model (see section 4.5 in the paper), we also prove **Forward Secrecy**. Definitions of those properties can be found in Section 4 of our paper. Our claims are proven with experiment **(E3)**.

(C4) PSK Exchange: For the key exchange based on preshared symmetric keys, we prove **Mutual Authentication** and **Handshake Secrecy**. Definitions of those properties can be found in Section 4 of our paper. Our claims are proven with experiment **(E4)**.

A summary of our proof results and runtime can be found in Table 1 of our paper.

A.4.2 Experiments

In the following we list all the different models and explain how to execute them individually. Afterwards we introduce the python program to execute all models in experiments **(E1)-(E4)** at once.

We ran our models initially on an Intel(R) Xeon(R) CPU E5-4650L 2.60GHz machine with 756GB of RAM using 4 threads and the estimated runtime is based on this machine.

First navigate in the cloned repository's model folder

```
$ cd SPDM/TamarinModels
```

and then make the oracle file executable

```
$ chmod +x oracle
```

(E1): *[1 human-minute + ∼ 4 compute-minutes]:*
Device attestation is modelled in the file: *device_attestation.spthy*. The lemma *ResponderAuth* models Responder Authentication and the lemma *MeasurementAuth* models Measurement Authentication. The model file further contains several sanity lemma and helper lemmas to prove the above property. With the following instruction all of them will be executed.

Preparation: After cloning the repository and installing the software dependencies, navigate into the *TamarinModels* folder within the cloned repository.
Execution: Open the terminal in this folder and execute the following command:

```
$ python3 tamarin_wrapper.py
↪ device_attestation.spthy -p
↪ "auth,Sanity" -c 4 -t 1800
```

Results: The results are printed into the terminal, and a *.csv* file is also stored within the results folder.

(E2): *[1 human-minute + ∼ 52 compute-minutes]:*
The certificate based key exchange is modelled in the file: *key_exchange.spthy*. The lemma *resp_authentication_at_finish* models Responder Authentication and the lemma *mutual_authentication* models Mutual Authentication. Handshake Secrecy is modelled via 2 lemma: *secret_major_init_side* and *secret_major_resp_side*. The model file further contains several sanity lemma and certain helper lemma to prove

the above property. With the following instruction all of them will be executed.

Preparation: After cloning the repository and installing the software dependencies, navigate into the Tamarin-Models folder within the cloned repository.
Execution: Open the terminal in this folder and execute the following command:

```
$ python3 tamarin_wrapper.py
↪ key_exchange.spthy -p "Sanity"
↪ -c 4 -t 1800
```

Results: The results are printed into the terminal, and a *.csv* file is also stored within the results folder.

(E3): *[1 human-minute + ∼ 29 compute-minutes]:* The preshared public key based key exchange is modelled in two file: *preshared_pk.spthy* and *preshared_pk_copy.spthy*. The lemma *mutual_authentication* models Mutual Authentication and is executed in *preshared_pk.spthy*. The other file executes Handshake Secrecy is modelled via 2 lemma: *secret_major_init_side* and *secret_major_resp_side*. The model file further contains several sanity lemma and certain helper lemma to prove the above property. With the following instruction all of them will be executed.

Preparation: After cloning the repository and installing the software dependencies, navigate into the Tamarin-Models folder within the cloned repository.
Execution: Open the terminal in this folder and execute the following commands:

```
$ python3 tamarin_wrapper.py
↪ preshared_pk.spthy -p
↪ "auth,Sanity" -c 4 -t 1800
```

```
$ python3 tamarin_wrapper.py
↪ preshared_pk_copy.spthy -p "fs"
↪ -c 4 -t 1800
```

Results: The results are printed into the terminal, and *.csv* files are also stored within the results folder.

(E4): *[1 human-minute + ∼ 3 compute-minutes]:* The preshared public key based key exchange is modelled in two files: *preshared_psk.spthy* and *refl.spthy*. The lemma *mutual_authentication* models Mutual Authentication and is executed in *preshared_psk.spthy*. The other file executes Handshake Secrecy is modelled via 2 lemma: *secret_major_init_side* and *secret_major_resp_side*. The model file further contains several sanity lemma and certain helper lemma to prove the above property. With the following instruction all of them will be executed.

Preparation: After cloning the repository and installing the software dependencies, navigate into the Tamarin-Models folder within the cloned repository.
Execution: Open the terminal in this folder and execute the following commands:

```
$ python3 tamarin_wrapper.py
↪ attack_refl_preshared_psk.spthy
↪ -c 4 -t 1800
```

```
$ python3 tamarin_wrapper.py
↪ preshared_psk.spthy -p "Sanity"
↪ -c 4 -t 1800
```

Results: The results are printed into the terminal, and *.csv* files are also stored within the results folder.

Easier alternative
Instead of running all models independently, it is also possible to run all of them at once. On our computing device this took ∼ 1,5h.
Preparation: After cloning the repository and installing the software dependencies, navigate into the TamarinModels folder within the cloned repository.
Execution: open the terminal in this folder and execute the following command:

```
$ python3 tamarin_wrapper.py -f
↪ case_studies.tamjson
```

Results: While the results will be printed into the terminal, *.csv* files are also stored within the results folder.

Timeouts
Depending on the computing device it can happen that single lemmas do not terminate in the given timeout. You can either change the timeout at the `-t` parameter or if one is running all at once, change the `"timeout"` field within the `case_studies.tamjson` file.

A.5 Notes on Reusability

We conducted a first in-depth formal analysis of the three models of SPDM, and proved their main security properties. Ideally, we would verify all security properties on the complete model, however this seems beyond reach of the current state-of-the-art symbolic analysis tools. However, we modelled the protocol in a modular fashion s.t. models can be reused and adapted as the specification and standard evolve. We hope that our models can serve as a starting point for a unified model and encourage future work on the SPDM protocol.

A.6 Version

Based on the LaTeX template for Artifact Evaluation V20220926. Submission, reviewing and badging methodology followed for the evaluation of this artifact can be found at https://secartifacts.github.io/usenixsec2023/.