USENIX'23 Artifact Appendix: Automated Analysis of Protocols that use Authenticated Encryption: How Subtle AEAD Differences can impact Protocol Security

Cas Cremers[1], Alexander Dax[1,3], Charlie Jacomme[2], and Mang Zhao[1,3]

[1]CISPA Helmholtz Center for Information Security, Germany
[2]Inria Paris, France
[3]Saarland University

A Artifact Appendix

A.1 Abstract

This artifact appendix presents a description of the experiments and case studies conducted in the research paper *Automated Analysis of Protocols that use Authenticated Encryption: How Subtle AEAD Differences can impact Protocol Security*. One of the core objectives of this research is to analyze the impact of subtle differences in AEADs on the security of a variety of protocols.

We provide means to reproduce our case studies of 8 distinct security protocols, namely YubiHSM, Facebook's Message Franking, SFrame, WebPush, Whatsapp Group Messaging, GPG, saltpack, and Scuttlebutt. These protocols are analyzed under different various AEAD models. We structured and defined those models in a *library* file. All models in the library, along with the case studies, are implemented using the Tamarin Prover, a symbolic analysis tool for security protocols.

The AEAD models and case studies are made available in a public Github repository with detailed instructions and automated means to replicate the experiments discussed in the original research paper. Additionally, a Docker image with the necessary software is made available for easy setup and execution.

A.2 Description & Requirements

A.2.1 Security, privacy, and ethical concerns

None.

A.2.2 How to access

All our files are publicly available and can be accessed in the following GitHub repository https://github.com/AutomatedAnalysisOf/AEADProtocols/tree/V1.

A.2.3 Hardware dependencies

Our artifact does not require any specific hardware. However, as the used software (e.g. the Tamarin Prover [1]) does also scale with computation power and memory, we recommend to at least use a modern notebook or similar modern computing devices. GPUs are not required.

A.2.4 Software dependencies

We provide access to a docker[2] image which has all the necessary software dependencies pre-installed.

(Optional) Dependencies for manual installation In case the reviewers choose to manually install the dependencies, they should install the following

1. Tamarin Prover[3] (depends on `haskell-stack`, `graphviz`, and `maude`.) Note that Tamarin **does not run on Windows** systems and a virtual machine/WSL may be needed. Additionally, we added a .zip file with the correct Tamarin version to the GitHub repository.
2. Python3 - install `pip`, and use it to install `tabulate` and `matplotlib`.

A.2.5 Benchmarks

None.

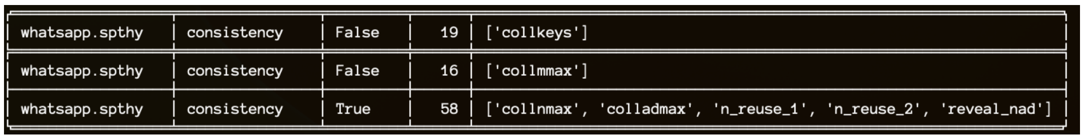
A.3 Set-up

A.3.1 Installation

Clone the repository using:

```
$ git clone https://github.com/
  ↪ AutomatedAnalysisOf/AEADProtocols.git
```

[1]https://tamarin-prover.github.io
[2]https://docs.docker.com/engine/install/
[3]https://tamarin-prover.github.io/manual/book/002_installation.html



```
| whatsapp.spthy | consistency | False | 19 | ['collkeys']                                                    |
| whatsapp.spthy | consistency | False | 16 | ['collmmax']                                                    |
| whatsapp.spthy | consistency | True  | 58 | ['collnmax', 'colladmax', 'n_reuse_1', 'n_reuse_2', 'reveal_nad'] |
```

Figure 1: Exempt of the terminal output produced by tamarin_wrapper.py.

and navigate inside the repository.

After installing `docker`, pull the docker image using

```
$ docker pull aeads/tamarin
```

Now, you can run the image using:

```
$ docker run -it -v $PWD:/opt/case-studies
 ↪  aeads/tamarin bash
```

(Optional) Hints for manual installation

In the case you want to set up the experiments **without** docker here are some hints:

- Make sure to use the *tamarin prover* version provided in the GitHub repository.
- Follow the instruction on `https://tamarin-prover.github.io/manual/book/002_installation.html` to install the tamarin prover. Common problems are missing Haskell dependencies or outdated versions of Maude
- Make sure that the *tamarin-prover* executable is in the *$PATH*.

A.3.2 Basic Test

Execute

```
$ tamarin-prover test
```

to see whether the docker started successfully (or whether your manual installation worked).

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

One of the primary objectives of the case study analysis is to identify the most robust AEAD model that preserves the desired security property for each protocol.

The provided models in the artifact appendix include formal representations, so-called *lemmas*, expressed in the input language of the Tamarin Prover, which capture the desired security properties of the protocols.

Through automated execution of Tamarin with different AEAD models using a provided Python script, the lemma results are checked to determine if they hold or provide counterexamples, facilitating efficient analysis. Further details on the Python script and the core idea can be found in the original paper.

Table 1 shows an output of the Python script for the Whatsapp group messaging protocol model. Here, for instance, the lemma marked as *consistency* does not hold under the `collkeys` or the the `collmmax` AEAD models. The name tags are explained in the `README.md` file and defined in the original paper.

Our focus lies on identifying the weakest AEAD model that ensures the security property proven by the lemma, as well as determining the strongest AEAD models that lead to potential attacks. However, it is important to note that certain models may not terminate within the specified timeout. In such cases, we still identify AEAD models that demonstrate both secure properties and attack possibilities. In this case, we do not claim that these models represent the strongest or weakest models where the property still holds or yields a counterexample.

We give details on the reported results in the original paper and provide the concrete results in the GitHub `https://github.com/AutomatedAnalysisOf/AEADProtocols/tree/V1`.

A.4.2 Experiments

Instead of running all models independently, we provide a python program to run all of them at once. For that we used a computing cluster with Intel® Xeon® Gold 6244 CPUs and 1TB RAM. In case you do not have access to a computing cluster, you may need to increase the timeout in the `case_studies.tamjson` file. Open the file using the editor of your choice and navigate to the line `"timeout": 60,` and increase the number slightly. The number after the *timeout* is defined as seconds.

In the GitHub repository we also provide the results of our case studies when running them on our machine. We had a total evaluation time of 17 hours and 29 minutes with a total of 1404 tamarin prover calls. While we tested the case studies also on a modern notebook, we cannot guarantee the same precise result, as specific lemmas using certain AEAD models may not terminate within the timeout. This mostly concerns the protocol models if YubiHSM and SFrame. All others should finish rather fast (< 1hour), also on normal notebooks.

Preparation: After following the installation instruction in Section A.3.1, enter the `Models` folder within the cloned repository/docker image.

Execution: Execute

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

Note, that depending on your machine the results may differ. You can increase the timeout in the `case_studies.tamjson`

Results: While the results will be printed into the terminal (see Figure 1), *.csv* files of the results are also stored within the newly created `results` folder. They can be compared to our provided results in the `results_precomputed` folder in the `Models` directory. We also refer to Table 3 and Table 4 in the original paper to confirm that your run did find the same attacks

A.5 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/.