USENIX

THE ADVANCED COMPUTING SYSTEMS ASSOCIATION

A Formal Analysis of Apple's iMessage PQ3 Protocol

Felix Linker, Ralf Sasse, and David Basin, *ETH Zurich*

https://www.usenix.org/conference/usenixsecurity25/presentation/linker

This artifact appendix is included in the Artifact Appendices to the Proceedings of the 34th USENIX Security Symposium and appends to the paper of the same name that appears in the Proceedings of the 34th USENIX Security Symposium.

August 13–15, 2025 • Seattle, WA, USA

978-1-939133-52-6

Open access to the Artifact Appendices to the Proceedings of the 34th USENIX Security Symposium is sponsored by USENIX.

USENIX Security '25 Artifact Appendix: A Formal Analysis of Apple's iMessage PQ3 Protocol

Felix Linker
Department of Computer Science, ETH Zurich

Ralf Sasse
Department of Computer Science, ETH Zurich

David Basin
Department of Computer Science, ETH Zurich

A Artifact Appendix

A.1 Abstract

We present the formal verification of Apple's iMessage PQ3, a highly performant, device-to-device messaging protocol offering strong security guarantees even against an adversary with quantum computing capabilities. PQ3 leverages Apple's identity services together with a custom, post-quantum secure initialization phase and afterwards it employs a double ratchet construction in the style of Signal, extended to provide post-quantum, post-compromise security.

We present a detailed formal model of PQ3, a precise specification of its security properties, and machine-checked security proofs using the TAMARIN prover. Our analysis covers both key ratchets, including unbounded loops, which was believed by some to be out of scope of symbolic provers like TAMARIN (it is not!). Our artifact provides a pseudocode specification of PQ3, our model of both PQ3 and its security properties, and case studies illustrating how one can prove protocols like PQ3 using TAMARIN.

A.2 Description & Requirements

A.2.1 Security, privacy, and ethical concerns

None.

A.2.2 How to access

Our artifact is available at: https://doi.org/10.5281/zenodo.14710687.

The artifact is structured as follows. We omit files and folders that are only included for technical reasons. Instructions on how to use the scripts are provided in Section A.4.2.

`case-study` Contains the example theories used to illustrate our proof methodology.
`proofs` Contains proofs for lemmas that cannot be proven automatically and a script to check individual proofs.
`spec` Contains an iMessage PQ3 pseudocode specification.
`model.spthy` Our formal model of iMessage PQ3.
`oracle.py` A proof heuristic aiding proof construction.
`prove-auto.sh` A script that constructs proves for all automatically provable lemmas.
`prove-expensive.sh` A script that constructs the proof for an automatically provable but computationally expensive lemma.
`wellformedness.sh` A script that checks that our model is well-formed.

A.2.3 Hardware dependencies

Using Tamarin requires no special hardware. However, some proofs require significant time and memory for construction or verification. Ideally, the machine used should have 140 GB or more of memory. Proofs were constructed and timed on a server with 252 GB of memory and two Intel Xeon E5-2650 v4 CPUs, i.e., a 48-thread server.

However, note that Tamarin uses RAM inefficiently. Tamarin stores the entire proof tree, but a proof only needs to access a path in the proof tree. Therefore, memory compression can help you check proofs on machines with much less memory than indicated. For example, some proofs were constructed on a MacBook Pro with only 32 GB of memory and an Apple M2 Max CPU, where we observed 100 GB of virtual memory usage but only 5-10 GB of physical memory usage.

A.2.4 Software dependencies

All major operating systems can be used. We used Ubuntu 24.04 when timing proofs (see Section A.4.2).

To use our artifact, you must have the following tools installed:

- Tamarin v1.8.0 or higher. Installation instructions are provided at: https://tamarin-prover.com/manual/master/book/002_installation.html. We recommend using `brew`.
- Maude v3.1 or higher. When installing Tamarin with `brew`, you will also install Maude. In any other case, the Tamarin manual provides more information on how to install Maude.

- Python 3. Python 3 is a major, well-supported programming language. We therefore do not provide instructions on how to install it.

A.2.5 Benchmarks

None.

A.3 Set-up

A.3.1 Installation

Download our artifact and extract it.

A.3.2 Basic Test

Open a shell in the artifacts directory and run the following script:

```
./wellformedness.sh
```

This should take no more than 5 minutes. If the command terminated successfully, you will see a list of lemmas (e.g., `Auto_ChainKeySources`) all marked as `analysis incomplete`. Above that, you will see the Tamarin and Maude versions used. Above that, you should see the text "All wellformedness checks were successful."

A.4 Evaluation workflow

A.4.1 Major Claims

(C1): Proofs for all security properties and auxiliary lemmas of our PQ3 Messaging Protocol can be constructed or verified.

(C2): Proofs for all lemmas of our case study models can be constructed.

A.4.2 Experiments

For experiments E2 and E3, our artifact lists approximate time and memory requirements for each script or proof file in the respective README. As noted in A.2.3, you may need much less physical memory than indicated when utilizing your operating system's memory compression.

Experiments (E1)-(E3) together prove iMessage PQ3's security, as described in Section 5 of our paper. (E4) constructs proofs for the case study models described in Appendix A of our paper.

(E1): [Verify Wellformedness] [5 human-minutes + 5 compute-minutes]:
How to: Execute the script `wellformendness.sh`.
Results: Observe that the output contains "All wellformedness checks were successful."

(E2): [Prove auto-provable lemmas] [5 human-minutes]:
How to: Execute the scripts `prove-auto.sh` and `prove-expensive.sh`. Note that the latter script may take a day or more to construct the proof.
Results: Observe that (i) the output of `prove-auto.sh` shows every lemma prefixed with `Auto_*` as "verified", and (ii) the output of `prove-expensive.sh` shows the lemma prefixed with `Expensive_Auto_*` as "verified."

(E3): [Verify constructed proofs] [5 human-minutes]:
How to: Navigate to the `proofs` directory. Execute the script `check.sh` for each `.spthy` file in the `proofs` directory. For example:

```
./check.sh CkCompromise.spthy
```

Results: Observe that when checking every proof file, the lemma named like the proof file is shown as "verified." Also, observe that every lemma in the original file `model.spthy` was shown to be verified in this experiment or experiment E2.

(E4): [Prove case studies] [5 human-minutes + 1 compute-minute]:
How to: Navigate to the `case-study` directory. For each `.spthy` file in the directory, run:

```
tamarin-prover --prove <FILE>
```

Results: Observe that for each file, all lemmas are shown as "verified."

A.5 Notes on Reusability

Our artifact is not intended for reuse. We hope, however, that our case studies serve as educational examples for how to prove protocols secure, which exhibit similar looping behavior to the PQ3 Messaging Protocol.

A.6 Version

Based on the LaTeX template for Artifact Evaluation V20231005. Submission, reviewing and badging methodology followed for the evaluation of this artifact can be found at https://secartifacts.github.io/usenixsec2025/.