USENIX Security '26 Artifact Appendix: Secure Protocol Composition under Dynamic Corruption: Scaling Up Symbolic Analysis for Real-World Security Properties

Cas Cremers
CISPA Helmholtz Center for Information Security

Erik Pallas
CISPA Helmholtz Center for Information Security

Aleksi Peltonen
CISPA Helmholtz Center for Information Security

A Artifact Appendix

A.1 Abstract

Although automated symbolic protocol verification has proven valuable and effective, current approaches begin to reach their limits: While small protocols can be analyzed automatically, the most complex case studies often require substantial expert time and resources. There have been many attempts to solve this problem by compositional verification, but they rely on unrealistic protocol assumptions and do not support real-world security properties like Forward Secrecy.

In this work, we enable compositional symbolic analysis for real-world security protocols with respect to modern security properties. We develop a composition result in the Applied π-Calculus that holds even in the presence of attackers capable of dynamic corruption if the protocols satisfy a disjointness requirement. We demonstrate the applicability and effectiveness of our result on the composition of a data exchange protocol with a Diffie-Hellman key exchange and a compositional analysis of Forward Secrecy in TLS 1.3 ECH.

This artifact contains the ProVerif models of both our case studies and the Full Version of our paper, including all proofs which did not fit into the space constraints of the conference version.

A.2 Description & Requirements

The artifact consists of a set of ProVerif models and the Full Version of our paper containing proofs and additional theory. Since the latter is a simple, self-contained PDF, the remainder of this Artifact Appendix refers to the ProVerif models unless specifically stated otherwise.

We provide models for two case studies. The first consists of an authenticated Diffie-Hellman key exchange and a simple data transfer protocol to demonstrate at a small scale how a compositional analysis can be set up in ProVerif. The second performs both a compositional and a monolithic analysis of Forward Secrecy in TLS 1.3 ECH while benchmarking and comparing the resource consumption of both types of analysis. It builds on prior work by Bhargavan, Cheval and Wood (cf. Karthikeyan Bhargavan, Vincent Cheval, and Christopher Wood. 2022. A Symbolic Analysis of Privacy for TLS 1.3 with Encrypted Client Hello. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security (CCS '22)*. ACM, New York, NY, USA, 365–379. doi:10.1145/3548606.3559360)

A.2.1 Security, privacy, and ethical concerns

There are no security or privacy risks associated with running the ProVerif models of our case studies. We also see no related ethical concerns; for a detailed analysis, see the Ethical Considerations appendix of our main paper.

Note, however, that a full benchmark of the TLS 1.3 ECH case study is very memory-intensive and can take up to several days. We therefore recommend to not run the benchmark on hardware used for day-to-day business but use a virtual machine, a remote server or similar instead.

A.2.2 How to access

The artifact is hosted in the Figshare repository identified by DOI 10.6084/m9.figshare.29958290. It contains two files:

- `secure_composition_under_dynamic_corruption_case_studies.pdf`: The Full Version of our paper, containing all proofs and theory omitted from the conference version due to space constraints.
- `secure_composition_under_dynamic_corruption_case_studies.zip`: An archive containing all models and build files needed for our case studies.

To recreate our experiments, download and unpack the `.zip` archive. All file paths in the following are relative to the root of this unpacked archive.

A.2.3 Hardware dependencies

There are no special hardware requirements to run the ProVerif models in this artifact. ProVerif itself requires

OCaml to be installed, which does not have official hardware requirements. However, we recommend to provide at least 200 GB of memory since benchmarking the TLS 1.3 ECH case study is memory-intensive. Alternatively, if omitting the scenarios that will likely time out after 48 hours, 70 GB of memory will suffice. For reference, we performed the benchmarks on a server featuring an Intel® Xeon® CPU E5-4650L with 2.60 GHz and 1 TB of RAM.

A.2.4 Software dependencies

The artifact requires the `make` build management tool to build the models for the TLS 1.3 ECH case study, ProVerif v2.05 or later to analyze the models, and OCaml and/or `opam` to install ProVerif. If you use the provided Dockerfile to run ProVerif, you also need a compatible container engine like Docker or Podman. At the time of this writing, ProVerif v2.05 is the most recent version; we also recommend using v2.05 after possible future releases to maintain consistency of the results.

A.2.5 Benchmarks

All models and benchmarking scripts are provided with the artifact.

A.3 Set-up

The following has been tested on Ubuntu 24.04. We assume that both `make` and Docker are already installed. For a detailed explanation of the structure of the case study files, see `\tls_ech\README.md`.

A.3.1 Installation

All dependencies can be either installed directly on a machine, or you can use the provided Dockerfile to run the case studies in a container.

ProVerif can be either installed from source or via the OCaml Package Manager `opam`; we recommend the latter. If not already present, install `opam` using the package manager of your choice (e.g., `apt install opam`) or follow the official installation instructions. Then, run `opam depext conf-graphviz` to install graphviz, `opam depext proverif` to install the remaining dependencies of ProVerif, and finally `opam install proverif.2.05` to install ProVerif v2.05.

If you prefer to use a container, navigate to the root directory of the unpacked artifact; there, you find a Dockerfile and an archive `proverif2.05.tar.gz` which contains the ProVerif source code. From there, run `docker build -t composition:latest -f ./Dockerfile .` to build the Docker image, and afterwards `docker run -it composition:latest` to run the resulting container. If you use a container engine other than Docker, use the corresponding `build` and `run` commands instead. Note that the Dockerfile contains instructions for installing `vim` and `emacs` for convenient inspection of text files within the container.

A.3.2 Basic Test

To check whether ProVerif is properly installed, run `proverif -help`. The resulting output should start with `Proverif 2.05. Cryptographic protocol verifier, by Bruno Blanchet, Vincent Cheval, and Marc Sylvestre` followed by explanations for several input parameters. For testing the TLS 1.3 ECH benchmark navigate to `\tls_ech` and run `bash run_bench one pfs 2 2 2 2` to perform a monolithic analysis of the first of the 32 scenarios considered, and `bash run_bench -c one pfs 2 2 2 2` for a compositional analysis. The four digits at the end of the command indicate the scenario code and can be used to separately analyze individual scenarios (for an overview of relevant scenarios, see Appendix E in our paper). On a sufficiently powerful machine the analyses of scenario 2222 will take around 10 and 5 minutes, respectively.

A.4 Evaluation workflow

A.4.1 Major Claims

(C1): All requirements of a compositional analysis of an authenticated Diffie-Hellman key exchange and a simple data exchange protocol are met, and it yields the same results as a monolithic analysis.

(C2): Compositional analysis of Forward Secrecy of the master secret in TLS 1.3 ECH decreases resource consumption by up to two orders of magnitude in ProVerif.

A.4.2 Experiments

(E1): [5 human-minutes + 100 compute-milliseconds + 10MB disk]: Compositional and monolithic analysis of the Diffie-Hellman/Data Exchange case study; cf. Section 6.2 of the paper. Validates (C1).

How to: Perform a monolithic and a compositional analysis of the Diffie-Hellman/Data Exchange case study; Forward Secrecy of the session keys will be proven true in both analyses.

Preparation: We recommend to open two terminal windows side-by-side, although this not strictly necessary. Navigate to
`dh` in both or the single terminal.

Execution:
1. If you use two terminal windows, perform this step in the first. Run `bash run_bench` for a monolithic analysis.
2. In `dh_auth.pv`, comment out lines 121 and 133; this is done in ProVerif by surrounding the line with `(* ... *)`.

3. If you use two terminal windows, perform this step in the second. Run `bash run_bench` again for a compositional analysis.

Results: ProVerif outputs some proof information followed by the results of five so-called queries in a block titled 'Verification summary'. These correspond to the theorems we want to proof.

The first two queries are sanity checks whether the modeled protocols can actually be executed; in the monolithic analysis both should evaluate to `false`, while in the compositional analysis the first should evaluate to `false` and the second to `true` (note that for these two queries, ProVerif attempts to prove the negated statement here as indicated by the `not` directly following the `Query` keyword, meaning that `false` actually indicates a successful proof).

The third query verifies the *set-up guarantee* of the Diffie-Hellman part of the model, which needs to hold for a compositional analysis. In both analyses, this should evaluate to `true`.

The fourth and fifth queries state Forward Secrecy of the parties' sessions keys generated by the Diffie-Hellman key exchange. These, too, should evaluate to `true` in both analyses.

(E2): [30 human-minutes + 460 compute-hours + 200GB disk]: Benchmarks of a compositional and monolithic analysis of Forward Secrecy in TLS 1.3 ECH; cf. Section 6.3 of the paper. Validates (C2).

How to: Run the provided benchmarking script and compare the measured resource consumption.

Preparation: Navigate to `\tls_ech`. If you have run the script before, remove all temporary files and directories (`\tls_ech\generated_configs`, `\tls_ech\generated_libraries`, `\tls_ech\tests`, `\tls_ech\ongoing.txt`) and the files containing the results (`\tls_ech\security_properties \forward_secrecy\result[_composition].txt`); otherwise, the script will skip some or all of the scenarios and not produce new data.

Execution:
1. Run `bash run_bench -c reach pfs` for the compositional analysis. On our reference hardware, this takes roughly 53 h.
2. Run `bash run_bench reach pfs` for the monolithic analysis. On our reference hardware, this takes roughly 405 h, or about 213 h when ignoring scenarios that time out.

Note that `run_bench` evaluates all scenarios sequentially and does not do any parallelization. It is possible to evaluate scenarios in parallel by analyzing them individually as described in Appendix A.3.2; if you attempt this, make sure to provide the individual processes with sufficient resources.

Results: In both cases, the benchmarking script outputs a `.txt` file containing the resource consumption. The compositional analysis results are written to `\tls_ech\security_properties \forward_secrecy\result_composition.txt`, the results of the monolithic analysis to `result.txt` in the same directory.

The results of our own benchmarking can be found in `\tls_ech\pfs_results`.

A.5 Notes on Reusability

The composition mode we added to the TLS 1.3 ECH models can be used to analyze other security properties than Forward Secrecy. However, this requires additional theoretical work to verify that the respective properties can be defined over one of the two parts in which we split the protocol, and that the requirements of the composition theorem we describe in our paper still hold. The protocol can also be split up at another position depending on the property and whether the requirements still hold.

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/usenixsec2026/.