USENIX Security '26 Artifact Appendix: M-Step: A Single-Stepping Framework for Side-Channel Analysis on TrustZone-M

Cristiano Rodrigues[1], Marton Bognar[2], Sandro Pinto[1], and Jo Van Bulck[2]

[1]*Centro ALGORITMI, Universidade do Minho,* [2]*DistriNet, KU Leuven*

A Artifact Appendix

A.1 Abstract

This artifact provides *M-Step*, the first open and extensible single-stepping attack framework specifically designed for Arm TrustZone-M. The package includes the source code, tooling, and documentation required to reproduce the paper's experiments and key results, specifically: (i) the M-Step microbenchmarks; (ii) the leakage analysis of various microarchitectural components; and (iii) the end-to-end attacks on Mbed TLS. We release the M-Step framework with the all the core mechanisms and plugins defined in the paper to enable future replication, engage the system security community, and allow future users to use M-Step as a Swiss army knife for side-channel analysis on Arm MCUs.

A.2 Description & Requirements

A.2.1 Security, privacy, and ethical concerns

All the attacks are conducted on the target microcontroller, there is no security or privacy concerns for the host machine or the evaluators. The host machine is only used to obtain and analyze the results of experiments and running tools to help visualize the collected traces.

A.2.2 How to access

All artifacts are archived for long-term access on Zenodo at `https://doi.org/10.5281/zenodo.17910184`. Additionally, we also release them on GitHub as an open-source project at `https://github.com/M-Step-Framework`.

A.2.3 Hardware dependencies

Using M-Step and reproducing the experiments requires an STM32 Nucleo-144 development board (NUCLEO-L552ZE-Q), which includes the target MCU, an STM32L552ZE with an Arm Cortex-M33 core.

A.2.4 Software dependencies

The artifacts have been successfully tested on Ubuntu 22.04 LTS, Ubuntu 24.04 LTS, and NixOS (version 25.05). While the artifact is expected to be compatible with most modern Unix-based systems, we recommend using one of the tested distributions for guaranteed reproducibility.

Development Environment. To ensure a reproducible environment, we use Nix to automatically manage all required dependencies at their specific versions. The provided Nix development shell contains all the necessary tools to compile the source code, run the evaluation scripts, and use the third-party programs required by M-Step. For users on non-NixOS systems, Nix can be installed by following the instructions for the package manager at `https://nixos.org/download/`.

After installation, Nix Flakes and the new `nix` command must be enabled by adding the following line to the Nix configuration file (`~/.config/nix/nix.conf` or `/etc/nix/nix.conf`):

```
experimental-features = nix-command flakes
```

STM32 Programmer. The STM32 Programmer is required to flash binaries onto the target board. It can be downloaded from the official STMicroelectronics website (a free user account is required). After installation, ensure that the `STM32_Programmer_CLI` binary is available in your system's `PATH`. To grant the STM32 Programmer user-level permissions to access the board, create the file `/etc/udev/rules.d/99-stlink.rules` if it does not already exist, and add the following Udev rules. Without these rules, `sudo` will be required for every deployment.

```
SUBSYSTEMS=="usb", ATTRS{idVendor}=="0483",
    ATTRS{idProduct}=="374e", MODE="0666",
    GROUP="plugdev"
SUBSYSTEMS=="usb", ATTRS{idVendor}=="0483",
    ATTRS{idProduct}=="374b", MODE="0666",
    GROUP="plugdev"
```

A.2.5 Benchmarks

None. The artifact is self-contained and does not require any external benchmarks or datasets.

A.3 Set-up

A.3.1 Installation

First, clone the artifact's source code from the Git repository and initialize its submodules:

```
git clone --recurse-submodules \
https://github.com/M-Step-Framework/m-step
cd m-step
```

Development Shell. All subsequent commands must be executed within the Nix development shell, which provides a reproducible environment with all necessary dependencies. To enter the shell, run the following command from the root of the repository. This command needs to be executed each time a new terminal session is started.

```
nix develop
```

A.3.2 Basic Test

To verify that the toolchain is functional and all dependencies are correctly installed, perform a basic end-to-end test of the M-Step framework. This test involves configuring, compiling, and deploying a simple application to the target board. Navigate to the `ns-world-bare` directory and execute the following steps:

1. Configure. Run the script to configure the Non-Secure and Secure world projects:

```
./0-config.sh
```

2. Compile. Compile the source code for both worlds:

```
./1-compile.sh
```

3. Deploy. Deploy the compiled binaries to the target board:

```
./2-deploy.sh
```

4. Monitor. Reset the board and capture its output:

```
./3-monitor.sh
```

Upon successful execution, you should observe the board booting, first into the Secure world (i.e., TF-M) and then transitioning to the Non-Secure world, which will print the message "*Hello NS World!!*". This confirms that the development environment is set up correctly.

A.4 Evaluation workflow

A.4.1 Major Claims

(C1): M-Step is a reliable single-stepping framework for Arm TrustZone-M that can adapt to various victim code patterns and instruction types, consistently stepping over functions with a negligible number of multi-steps. This claim is substantiated by experiment E1, which reproduces the microbenchmarks from §6.1, with results presented in Table 5.

(C2): M-Step is an extensible framework with modular, on-demand plugins (§5) that facilitate side-channel analysis across different Arm Cortex-M microarchitectural components. We demonstrate that M-Step can establish covert channels by exploiting various microarchitectural leakages, as proven by experiments E2–E5, which reproduce the results reported in §6.2, Figure 6.

(C3): M-Step's high temporal resolution enables two end-to-end proof-of-concept attacks that extract RSA private keys from Mbed TLS's key generation and import functions in a single trace. This claim is substantiated by E6, which reproduces the results reported in §6.3.

A.4.2 Experiments

(E1): *[M-Step Microbenchmarks] [5 human-minutes + 30 compute-minutes]: replicates the single-stepping reliability microbenchmarks from §6.1.*
Preparation: Navigate to the experiment directory:

```
cd evaluation/t1-mstp-metrics
```

Execution: Execute the evaluation script:

```
./1-run-test.sh
```

Results: The script will print the M-Step reliability metrics to the terminal, which can be compared against the results in Table 5 of the paper. The results are also saved to `./logs/0_raw_trace.txt` for later use.

(E2): *[DIV Instruction Covert Channel] [5 human-minutes + 2 compute-minutes]: replicates the data-dependent covert channel in the DIV instruction.*
Preparation: Navigate to the experiment directory:

```
cd evaluation/t2-covert-udiv
```

Execution: Execute the evaluation script:

```
./1-run-test.sh
```

Results: The plot, saved at `./logs/06-div.png`, demonstrates the covert channel and should resemble Figure 6a in §6.2.

(E3): *[Instruction Timing Covert Channel] [5 human-minutes + 2 compute-minutes]: replicates the covert channel based on varying instruction execution times.*

Preparation: Navigate to the experiment directory:

```
cd evaluation/t3-covert-inst
```

Execution: Execute the evaluation script:

```
./1-run-test.sh
```

Results: The plot at `./logs/06-inst_diff.png` demonstrates the covert channel and should resemble Figure 6b in §6.2.

(E4): *[ICache-based Covert Channel] [5 human-minutes + 2 compute-minutes]: replicates the instruction cache-based covert channel.*

Preparation: Navigate to the experiment directory:

```
cd evaluation/t4-covert-cache
```

Execution: Execute the evaluation script:

```
./1-run-test.sh
```

Results: The plot, saved at `./logs/06-cache.png`, demonstrates the covert channel and should resemble Figure 6c in §6.2.

(E5): *[Bus Contention Covert Channel] [5 human-minutes + 2 compute-minutes]: replicates the covert channel based on bus contention.*

Preparation: Navigate to the experiment directory:

```
cd evaluation/t5-covert-cont
```

Execution: Execute the evaluation script:

```
./1-run-test.sh
```

Results: The plot at `./logs/06-contention.png` demonstrates the covert channel and should resemble Figure 6d in §6.2.

(E6): *[End-to-End PoC Attacks] [5 human-minutes + 1 compute-hour]: replicates the two end-to-end attacks against Mbed TLS.*

Preparation: Navigate to the experiment directory:

```
cd evaluation/t6-pocs
```

Execution: Execute the evaluation script:

```
./1-run-test.sh
```

Results: The script will print the recovered RSA private keys for both proof-of-concept attacks to the terminal.

A.5 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/.