USENIX Security '26 Artifact Appendix: Memclave: Secure In-Memory Enclave for Untrusted Hosts

Amit Choudhari*
CISPA Helmholtz Center
for Information Security

Fabian van Rissenbeck*
TU Dortmund

Christian Rossow
CISPA Helmholtz Center
for Information Security

A Artifact Appendix

A.1 Abstract

The artifact contains the full source code of the Memclave framework, the ported Processing-In-Memory Benchmarks (PrIM) benchmark suite, and the plotting scripts for evaluation. Memclave includes (i) the Memclave *hypervisor* (`ci-switch` and a patched `QEMU`), (ii) a guest-side Linux kernel driver and In-memory Enclave (ime) client library that provide guest-side UPMEM-style abstractions, inside the guest VM, and (iii) the DRAM Processing Unit (DPU)-side runtime library (`ime`) contains Processing-in-Memory (PIM)-side subkernels, including the Memclave *loader*, First Stage Loader (FSL), key-exchange and messaging subkernels, and ported PrIM subkernels. The artifact additionally includes the unmodified PrIM benchmarks (native UPMEM baseline) and scripts required to reproduce the paper's plots and reported results.

A.2 Description & Requirements

The artifact is organized as follows.
`ci-switch`: A component of the Memclave hypervisor.
`qemu`: patched QEMU to interface with the CI-switch.
`driver`: Linux kernel module to interface with PIM.
`ime`: all DPU-side code (*loader*, FSL, key exchange and messaging subkernels, and the ported PrIM subkernels).
`ime-client-library`: guest-side library providing UPMEM-like abstractions and ported PrIM benchmarks.
`common`: shared headers used by Memclave components.
`mbedtls`: crypto library used both on-DPU and guest.
`plotting`: scripts used to generate figures for the paper.
`prim-benchmarks`: baseline PrIM suite for native UPMEM.
`scripts/hyp`: scripts for setting up the Memclave hypervisor
`README.md`: documentation and Reproduction guide.

Provided environments. Artifact contains (a) a VM image for execution and (b) two Docker images for development.

A.2.1 Security, privacy, and ethical concerns

The artifact is a confidential computation framework without any malicious components. We recommend evaluators execute the artifact on the provided test machine (Section A.2.3). It self-contained with benchmarking workloads.

A.2.2 How to access

We recommend evaluators use the Zenodo record (`https://doi.org/10.5281/zenodo.17986461`) to download the artifact. The source is mirrored on GitHub at `https://github.com/fabianvanrissenbeck/memclave`.

A.2.3 Hardware dependencies

For evaluation of Memclave, we need two linux systems (Sys_{Build} and Sys_{Run}). Sys_{Build} is a local linux based machine of the reviewer on which the docker image is deployed for building subkernels and ci-switch. Sys_{Build} does not have any specific configuration requirement as it is only used for building. Sys_{Run} is a remote machine with UPMEM PIM DIMMs (the paper evaluation runs on single PIM rank with 64 DPUs). To access the Sys_{Run}, SSH using the following credentials:

```
ssh -p 2293 reviewer@ios.inf.uni-osnabrueck.de
password: <Refer_HotCRP>
```

Tested configuration (from the paper, for reference): Sys_{Build}: intel Ultra 7 155H, Ubuntu 24.04.3 LTS (linux 6.11.11) Sys_{Run}: dual-socket Intel Xeon Silver 4216, Linux `5.15.0-142-generic`, UPMEM SDK `upmem-2025.1.0`, DPU frequency reported as 350 MHz per rank at runtime.

A.2.4 Software dependencies

Sys_{Run} has following requirements (already met). (1) x86 based Linux host with KVM support, (2) UPMEM SDK `upmem-2025.1.0`, (3) Toolchain for building, (4) Python 3.

Sys_{Build} has following requirements (pre-setup in the docker images). (1) Docker installed (to use the provided development images), (2) UPMEM SDK `upmem-2025.1.0` with a patched LLVM, (3) Toolchain for building the components (`gcc/g++`, `make`), (4) Python 3.

A.2.5 Benchmarks

The artifact evaluates Memclave using:

PrIM benchmark suite workloads. The PrIM suite contain 16 benchmarks, covering heterogeneous memory-access, compute, and synchronization patterns (native UPMEM baseline + Memclave-ported versions). The baseline suite is under `prim-benchmarks`, while Memclave ported guest-side and DPU-side components in `ime-client-library` and `ime`.
Microbenchmarks. Evaluate the subkernel load/unload, keyexchange, throughput for sealed data transfers and different transfer patterns (Table 3 and 4 from the main paper).

A.3 Set-up

We separate the artifact workflow into a development machine (Sys_{Build}), an execution machine (Sys_{Run}) and memclave environment (Sys_{Run_VM}). On Sys_{Build}, we build Memclave's host components and bundle them into `hyp/` using `./scripts/hyp/setup.sh`. We then copy `hyp/` and subkernels to Sys_{Run}. On Sys_{Run}, we launch the guest VM (Sys_{Run_VM}) and compile guest-side programs inside this VM.

A.3.1 Installation

Step 0: Obtain and the split of artifact. We download and unpack the artifact archive on both Sys_{Build} and Sys_{Run}. The directories relevant to Sys_{Build} are `ci-switch`, `hyp`, `ime`, `scripts/hyp`, `plotting`. And directories relevant to Sys_{Run} are `qemu`, `ime-client-library`, `prim-benchmarks`.

Step 1: Build hyp [on Sys_{Build}]. We use the docker image which includes build environment required to build PIM-side components and `ci-switch`. First, load docker images.

```
cd memclave-package
docker load -i memclave.tar
docker load -i memclave-qemu.tar
```

Build Hypervisor and copy to Sys_{Run}:

```
./scripts/hyp/setup.sh
scp hyp.tar.xz reviewer@<Sys_run>:~
```

Step 2: Boot and setup Sys_{Run_VM} [on Sys_{Run}]. On Sys_{Run}, extract transferred `hyp.tar.xz`, enter `hyp/` directory and run `boot.sh` specifying one from available Rank_IDs {0, 23, 26, 36}. Note: do not use {6, 7, 24, 25, 28, 31, 34, 35, 38, 39}:

```
cd hyp;
./boot.sh --exact-rank=<Rank_ID>
# login/password: memclave/memclave
```

Start `ci-switch` and launch the guest VM (Sys_{Run_VM}) using the patched QEMU. `setup-vm` loads `vpim` kernel module and setup reverse proxy. [Note]: Command runs in foreground and does not return. Keep the terminal open.

```
memclave@memclave-vm:~$ setup-vm
```

After loading, the driver creates device nodes `/dev/vpimN`.

Step 3: Setup reverse proxy to connect to Sys_{Run_VM} [on Sys_{Build}]. [Note]: This command stays in foreground and does not return. Keep the terminal open.

```
ssh -NL 26173:localhost:26173 -p 2293 \
    reviewer@ios.inf.uni-osnabrueck.de
```

Login to Sys_{Run_VM}.

```
ssh -p 26173 memclave@127.0.0.1
# user/pass: memclave/memclave
```

Step 4: Build subkernels [on Sys_{Build}] Build all Memclave subkernels and copy to Sys_{Run_VM}

```
cd ime; ./build.sh; cd ..
scp -r -P 26173 ime-client-library \
    memclave@127.0.0.1:~/
scp -P 26173 ime/subkernels.tar \
    memclave@127.0.0.1:~/ime-client-library/
```

Step 5: Build Memclave client apps [on Sys_{Run_VM}]. Build Memclave ported PrIM apps

```
cd ime-client-library
./build.sh
```

Step 6: Verify setup Confirm `boot.sh` successfully launches the Sys_{Run_VM}. The `setup-vm` successfully allocated ranks. Rank N, if allocated successfully, is mapped to /dev/vpimN. SSH to Sys_{Run_VM} is successful.

A.3.2 Basic Test

Run `bs` example on Sys_{Run_VM}.

```
cd build
./ime-bs-example
# Output: ...[OK] Outputs are equal
cd ..
```

A.4 Evaluation workflow

This section describes how to reproduce the key evaluation results of the paper (refer `reproduction.dox`).

A.4.1 Major Claims

(C1): Memclave is compatible with the full PrIM benchmark suite (Figure. 4). We port all 16 PrIM benchmarks and evaluate Memclave against CPU and native UPMEM (PIM-insecure) baselines. This is proven by experiment (E1), producing Figure. 4.

(C2): End-to-end overhead on representative workloads is modest. For MLP, Memclave converges toward the native UPMEM baseline as compute grows and incurs at most $\leq 1.5\times$ overhead at practical widths (Figure. 5); for BFS, overhead depends on graph/levels and is $\approx 1.1\times$ on some graphs (Figure. 6). Experiment (E2) proves this and generates Figure. 5 and Figure. 6.

(C3): Microbenchmarks reproduce the reported transfer costs and patterns (Tables 3–4). We reproduce (i) sealed-transfer and transfer-pattern throughput and (ii) profile subkernel loading and key exchange. Experiments (E3) and (E4) prove this and generate Tables 3–4.

A.4.2 Experiments

We provide a standard template (Run_Baseline, Run_Memclave, Plot) for running each experiment.

Run_Baseline: Run <Expt> **[on Sys_{Run}]**

```
cd memclave-package/prim-benchmarks
./run.sh <Expt: prim, bfsmlp, mram, subk, all>
# Output: output.tar
```

Run_Memclave: Run <Expt> **[on Sys_{Run_VM}]**

```
cd ime-client-library
./run.sh <Expt: prim, bfsmlp, micro, subk, all>
# Output: output.tar
```

Plot: Plots figures **[on Sys_{Build}]**. [Note]: Plotting dependencies are satisfied in `plotting/requirements.txt`

```
scp -P 2293 reviewer@ios.inf.uni-osnabrueck.de:\
~/output.tar output_baseline.tar
scp -P 26173 memclave@localhost:~/output.tar \
  output_memclave.tar
cd plotting; ./plot_all.sh \
  ../output_memclave.tar ../output_baseline.tar
```

(E1): PrIM suite compatibility (Reproduce Figure. 4) [10 human-mins + 10 compute-min].
Goal: Run all 16 PrIM benchmarks in native UPMEM and in Memclave; join results; generate Figure. 4 plot.
Step E1.1 : Run the baseline PrIM. Follow `Run_Baseline` with <expt> as `prim`
Step E1.2: Run the Memclave PrIM. Follow `Run_Memclave` with <expt> as `prim`
Step E1.3: Generate plots. Follow `Plot`.
Result: Figure. 4-style breakdown across all 16 benchmarks.

(E2): MLP and BFS overhead (Reproduce Figure. 5,6) [10 human-min + 10 compute-min]
Goal: run MLP and BFS on Memclave and native UPMEM and plot the comparison.
Step E2.1: Run MLP and BFS on UPMEM baseline. Follow `Run_Baseline` with <expt> as `bfsmlp`
Step E2.2: Run MLP and BFS on Memclave. Follow `Run_Memclave` with <expt> as `bfsmlp`
Step E2.3: Generate plots. Follow `Plot`.
Result: Generates Figure. 5, Memclave approaches PIM-insecure for larger N, with $\leq 1.5\times$ overhead at practical widths. Generates Figure. 6, comparison across graphs (LiveJournal1, loc-gowalla, roadNet-PA).

(E3): Sealed transfer and transfer-pattern throughput to/from PIM (Reproduce Table 3, 4) [10 human-mins + 30 compute-min].
Goal: build the crypto benchmark and the subkernel, then run the benchmark to emit CSV.
Step E3.1: Run baseline. Follow `Run_Baseline` with <expt> as `mram`
Step E3.2: Run Memclave. Follow `Run_Memclave` with <expt> as `micro`
Step E3.3: Generate plots. Follow `Plot`.
Result: CSV outputs corresponding to Table 3 and 4.

(E4): subkernel loading and key exchange (Reproduce Table 3) [10 human-mins + 10 compute-min].
Goal: Measure subkernel loading/unloading and key exchange timings.
Step E4.1: Run baseline. Follow `Run_Baseline` with <expt> as `subk`.
Step E4.2 [on $\text{Sys}_{\text{Run\_VM}}$ *]:* Restart Sys_{Run_VM} for subkernel profiling.

```
sudo poweroff
```

Step E4.3 [on Sys_{Run} *]:* Rerun Installation Step 2 with boot-stats.sh.

```
cd hyp;
./boot-stats.sh --exact-rank=<Rank_ID>
# continue steps 2 from installation
```

Step E4.4 [on $\text{Sys}_{\text{Build}}$ *]:* Rerun Installation Step 3
Step E4.5: Run Memclave. Follow `Run_Memclave` with <expt> as `subk`
Step E4.6: Generate plots. Follow `Plot`.
Result: CSV outputs corresponding to Table 3.

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/.