USENIX Security '25 Artifact Appendix: CoVault: Secure, Scalable Analytics of Personal Data

Roberta De Viti[1], Isaac Sheff[1*], Noemi Glaeser[2,4*], Baltasar Dinis[3*], Rodrigo Rodrigues[3], Bobby Bhattacharjee[4], Anwar Hithnawi[5*], Deepak Garg[1], and Peter Druschel[1]

[1]Max Planck Institute for Software Systems (MPI-SWS), Saarland Informatics Campus
[2]Max Planck Institute for Security and Privacy (MPI-SP)
[3]Instituto Superior Técnico (ULisboa), INESC-ID
[4]University of Maryland
[5]ETH Zürich

A Artifact Appendix

A.1 Abstract

There is growing awareness that the analysis of personal data, such as individuals' mobility, financial, and health data, can provide significant societal benefits. However, liberal societies have largely refrained from such analytics, arguably due to the absence of secure platforms that can scale to billions of records while operating under a very strong threat model. We argue that a key missing piece is an architecture capable of scaling (actively-)secure multi-party computation (MPC) horizontally without weakening security. To bridge this gap, we introduce CoVault, an analytics platform that combines server-aided MPC with trusted execution environments (TEEs). This approach enables the colocation of MPC parties in a single datacenter without reducing security while scaling MPC horizontally up to the datacenter's available resources.

Our artifact includes the source code of our prototype implementation of CoVault, along with the scripts to execute the code and produce all the major evaluation results that are presented in the technical paper. Additionally, we provide Python scripts for visualizing the results.

A.2 Description & Requirements

We outline the hardware and software requirements for running our artifact in § A.2.3 and § A.2.4, respectively. The `README.md` file in our artifact includes step-by-step instructions to help users configure their own machines for running CoVault provided they meet the requirements. We refer to this scenario as the "generic setup", which is relevant to a generic user.

To facilitate artifact evaluation, we provide USENIX Security evaluators with SSH access to a set of pre-configured virtual machines (VMs) on Google Cloud Compute Engine (GCE). These VMs satisfy all hardware and software dependencies, and are set up for intercommunication, as well as mutual authentication. We refer to this scenario as the "GCE setup", which is relevant to the USENIX Security evaluators.

The remainder of this section and § A.3 provide further details on hardware and software dependencies (§ A.2.3, § A.2.4) and installation steps (§ A.3), However, evaluators with SSH access to the GCE setup may skip these and proceed directly to § A.4.

A.2.1 Security, privacy, and ethical concerns

The artifact poses no security, privacy, or ethical concerns. It does not compromise system integrity, and all datasets are synthetic, hence devoid of any sensitive personal data. (In fact, CoVault uses data-oblivious computation protocols, whose performance characteristics are independent of the actual data.)

A.2.2 How to access

The artifact is available at `https://zenodo.org/records/14736568`. During the evaluation phase, evaluators will receive the SSH keys to access our pre-configured GCE setup, as described above.

A.2.3 Hardware dependencies

Generic setup. Our CoVault prototype runs on an interconnected set of machines (physical or virtual), where half support one second-generation TEE type (e.g., Intel TDX) and the other half support a different second-generation TEE type

*Affiliation at time work was done.

(e.g., AMD SEV-SNP). Second-generation TEEs encapsulate entire VMs, unlike first-generation TEEs (e.g., Intel SGX). The minimum requirements to run CoVault is one machine of each TEE type, each with at least 2 CPU cores.

To run all the scripts reproducing the experiments in the paper, each machine must have at least 4 CPU cores, 32 GB RAM, and 50 GB of free disk space. Insufficient disk space may result in truncated output logs (including latency results) and the impossibility to run tests with the AG2PC protocol, which requires to store large precompiled MPC circuits.

The results in the evaluation section of the paper were obtained in a setup with at least 15Gbps interconnect bandwidth among the machines (Google gVNIC on GCE). While CoVault can run with lower bandwidth, this may cause network bottlenecks, increased latencies, and, in the case of our prototype, timeouts.

An experiment in the paper measures the cost of TEE overhead in CoVaultṪo this end, it compares CoVault to an otherwise identical setup without the TEEs. To run this experiment, the user must either (i) have the ability to enable and disable TEE encapsulation on the same machine pair, or (ii) have access to two identical (physical or virtual) machines pairs – one with the two active TEEs of different types and one without TEEs.

Finally, the horizontal scaling experiments in the paper use varying VM and core configurations: (i) 2 VMs with at least 2 cores each, (ii) 2 VMs with at least 4 cores each, (iii) 4 VMs with at least 2 cores each, (iv) A total of 22 cores per TEE type, preferably distributed across multiple VMs. The `README.md` file specifies the required configuration for each experiment. Reproducing a scaling experiment requires its corresponding resources.

GCE setup. The GCE setup meets all of the above requirements. In particular, it is preconfigured with a sufficient number of Cloud VMs encapsulated into either Intel TDX or AMD SEV-SNP TEEs to run all experiments. Furthermore, it includes additional VMs with identical hardware and software configurations but without active TEEs for measuring TEE overhead. The Intel VMs are c3-standard-8 and the AMD VMs are n2d-standard-8; they are deployed in the us-central1-a zone and use the gVNIC (15Gbps). The `README.md` file specifies which set of GCE VMs to use for each experiment.

A.2.4 Software dependencies

Generic setup. Our codebase has been tested on Ubuntu 22.04 LTS and Debian 12. While it is expected to work on other Unix-based environments, dependency names may vary from those automatically installed by our setup scripts. The artifact requires C++ build tools, Python3, and pip3 for installing Python packages. All other dependencies are installed by `setup.sh`, as detailed in the `README.md` file.

GCE setup. The guest OS on all GCE VMs is Ubuntu 22.04 LTS, and all the software dependencies have already been installed.

A.2.5 Benchmarks

As noted in § A.2.1, CoVault runtime cost is a function of the input dataset size and fixed integer width, but it is independent of the actual data. Therefore, our experiments use only synthetic datasets, which are generated by our code.

Generic setup. The `README.md` file provides step-by-step instructions for generating these datasets and populating a Redis database.

GCE setup. On the GCE setup, this data generation process has already been completed.

A.3 Set-up

Generic setup. Once all hardware and software dependencies are met, the user can obtain the artifact from the Zenodo URL in § A.2.2. The `README.md` file provides instructions for installing dependencies, setting up the CoVault prototype, writing `CONFIG.me` files, and troubleshooting common issues.

GCE setup. On the GCE setup, the artifact is already installed and available at `/home/covault`. Additionally, all required configuration files are already set up.

A.3.1 Installation

Generic setup. As mentioned in § A.2.4, our software dependencies are the standard C/C++ build tools, Python3, and pip3. Our setup scripts intentionally do not install these dependencies, as the process is OS-specific.

On Ubuntu 22.04 LTS, the required software can be installed with: (i) `sudo apt install build-essential`; (ii) `sudo apt install python3`; (iii) `sudo apt install python3-pip`. Additionally, running `sudo apt update` may be useful in some cases. (Note that our scripts should not be sensitive to specific versions of Python3, but we have only tested with Python 3.11.5.)

After downloading the artifact (§ A.2.2), users can follow the instructions in `README.md` to install CoVault and its additional dependencies.

GCE setup. On the GCE setup, CoVault is preconfigured for evaluation – no additional installation is required.

A.3.2 Basic Test

The `README.md` file provides step-by-step instructions for running basic tests like the `primitives.sh` script.

A.4 Evaluation workflow

A.4.1 Major Claims

Next, we summarize the main experimental results claimed in Sec. 6 of the CoVault technical paper. These claims, (C1)–(C5) below, can be verified by running the corresponding experiments (E1)–(E5) in § A.4.2. We begin with claims related to the choice of the underlying MPC protocol and the performance of basic query primitives in CoVault:

(C1) DualEx is a compelling choice for the protocol underlying CoVault, when a 1-bit leak is acceptable. On one core pair, it runs twice as long as semi-honest garbled circuits (GCs) by design and it is about 10x faster than the fully maliciously-secure protocol, AG2PC. The overhead of TEEs in CoVault is negligible compared to MPC (e.g., under 1s in our experiment), with MPC being the main bottleneck. This claim is verified by experiment (E1) in § A.4.2 and described in Sec. 6.2, Figure 3 of the technical paper.

(C2) The runtime of linear table scans increases linearly with the input size, while sorting, sorted merge, and compaction exhibit slightly super-linear growth. Sorting is significantly more expensive than compaction (which is why our reduce trees sort only in the first stage and merge-compact in subsequent stages). This claim is established using experiment (E2) in § A.4.2 and described in Sec. 6.2, Figures 4a and 4b of the paper.

We further evaluate CoVault in the context of an epidemic analytics scenario:

(C3) Query latency scales almost inversely with the number of available core pairs (of the two TEE types). Even with only 4 core pairs (i.e., 4 cores on 2 VMs), our implementation of basic FGA queries completes in a reasonable time. For instance, queries q1 and q2 in the technical paper complete within 20min and 3h in our experiments on 4 core pairs, and their execution time improves in proportion to the number of core pairs available to our MapReduce setup. This claim is established by experiment (E3) and is described in Sec. 6.3, Figure 6 (blue lines) of the technical paper.

In addition to our empirical tests, we developed a performance model based on detailed measurements of basic units (individual mappers, reducers, and other query primitives) to extrapolate query latency at scale (Sec. 6.3.1 of the technical paper):

(C4) Our performance model correctly predicts the empirical results of q2 in small deployments, provided that the measurements of basic units are taken on the same testbed as full queries. This claim is established by experiments (E4) and (E5), and the corroboration between the extrapolated results and empirical results on our small deployment is shown in Sec. 6.3 of the technical paper, Figure 6b (yellow line).

(C5) We claim that colocation of the computing parties is key to scaling MPC-based computations. Our performance model indicates that executing epidemic analytics for a country with 80M people in a reasonable amount of time requires substantial bisection bandwidth, which is difficult to achieve across datacenters. This claim is based on results from our performance model (experiments (E4) and (E5)), as discussed in Sec. 6.3.1 of the technical paper.

A.4.2 Experiments

The artifact includes a `README.md` file detailing the steps to configure the environment, run experiments, collect results, and generate figures supporting claims C1–C5 in § A.4.1. While the GCE setup is preconfigured for immediate use, the `README.md` also provides guidance on how to configure a generic setup for running the experiments successfully.

We summarize the experiments below, linking them to the claims they support. The human hours assume access to our GCE setup. The compute hours are approximate, but we recommend planning conservatively. In fact, using a testbed on a public Cloud such as GCE introduces higher variability than an in-house controlled setup due to factors like resource contention, network fluctuations, background load from other tenants, and potential hardware allocation variations by the Cloud provider.

For this reason, although a single experiment may not take long, multiple repetitions are required to obtain stable and repeatable results, extending overall runtime. The compute hours below refer to the default number of repetitions set in the scripts. Our evaluation in Sec. 6 of the technical paper aggregated results over two months of execution.

(E1) [0.1 human-hour + 24 compute-hours per configuration]: Microbenchmarks comparing the performance of the MPC protocols used in our evaluation, and breaking down the costs of different system components (C1).

(E2) [0.1 human-hour + 24 compute-hours per configuration]: Microbenchmarks evaluating the cost of basic query primitives as a function of the input size (C2).

Evaluation in the context of an epidemic analytics scenario:

(E3) [0.1 human-hour + 24–72 compute-hours per configuration]: Direct measurement of horizontal scaling with a small number of cores for two queries described in the paper (C3).

Extrapolation of query latency to a very large number of cores and big data:

(E4) [0.1 human-hour + 24 compute-hours per configuration]: Measurements of basic execution units and latency extrapolation using the performance model (C4 and C5).

(E5) [0.1 human-hour + 0.1 compute-hours per configuration]: Execution of the performance model and analysis of results (C4 and C5).

A.5 Notes on Reusability

The `README.md` file provides guidelines for tuning various parameters in the scripts, such as the number of repetitions, whether to execute DualEx or a semi-honest protocol for comparison, the roles of generator and evaluator in a semi-honest execution, and input sizes.

Users can also modify the C++ code to adjust parameters like the size of intermediate results between mappers and reducers and attribute bit-width. While our prototype evaluates a specific set of queries, users can implement custom FGA queries by leveraging the building blocks in `src/primitives.cpp` and replicating the MapReduce execution logic from our test files.

Note that if the input size for certain experiments is significantly increased, additional RAM may be required to process the data in-memory. Otherwise, the experiments may run out of memory and terminate prematurely.

Additionally, although our prototype evaluates queries in the context of an epidemic analytics scenario, is it possible to change the database schema and the queries.

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/.