USENIX Security '25 Artifact Appendix: Learning from Functionality Outputs: Private Join and Compute in the Real World

Francesca Falzon
ETH Zürich, Switzerland

Tianxin Tang
Eindhoven University of Technology, Netherlands

A Artifact Appendix

A.1 Abstract

Private Join and Compute (PJC) is a two-party protocol recently proposed by Google for various use-cases, including ad conversion (Asiacrypt 2021) and which generalizes their deployed private set intersection sum (PSI-SUM) protocol (EuroS&P 2020). We analyze the risks associated with the PJC functionality output, and we describe and implement four practical attacks that break the other party's input privacy, and which are able to recover both membership of keys in the intersection and their associated values.

At a high level, this artifact consists of two parts:

1. An implementation of our search tree attack, located in the `search_tree` directory of our codebase.
2. Implementations of our maximum-likelihood estimation attack, the compressed sensing attack, and the discrete fourier transform attack, all of which are located in the `cs_dft_mle` directory.

This artifact appendix describes the necessary steps for reproducing the results presented in Section 7.

A.2 Description & Requirements

A.2.1 Security, privacy, and ethical concerns

This work is compliant with the Usenix ethics guidelines. Our work aligns with the intended use of PJC and therefore does not expose any vulnerabilities of the protocol or underlying cryptographic primitives.

Use of Synthetic Data. Our experiments exclusively use synthetic data generated by sampling numerical values from uniform and normal distributions. The use of synthetic data ensures that no real personal information is utilized or exposed, thereby eliminating any risks of harm to individuals or groups.

Disclosure. We reached out to Lepoint et al. (ASIACRYPT 2021) and the PJC code repository maintainers[1] on 19.11.2024 and 22.11.2024, respectively, to inform them about our analysis and suggest including information about our attacks in the README of the PJC code repository. They agreed to discuss the results internally and have updated their README[2] to include links to the prior attack works and additional information about the potential dangers of allowing multiple protocol invocations on the same or similar input set.

A.2.2 How to access

For this evaluation process, please access the artifact using https://doi.org/10.5281/zenodo.15147738.

A.2.3 Hardware dependencies

This artifact can be run on a commodity laptop with at least 16 GB of RAM and 32 GB of storage.

A.2.4 Software dependencies

This artifact does not require a specific OS but does rely on the proprietary software Gurobi Optimizer. The following steps describe how to obtain Gurobi Optimizer and the free academic license:

1. Go to the "Gurobi User Portal" at https://portal.gurobi.com/iam/login/ and register a new account using your university email address. When prompted, select "Academic".
2. After activating and logging into your academic account, generate your free academic license under the "Licenses" tab by selecting the "WLS Academic" option.
3. Download the generated lisense at https://license.gurobi.com/manager/licenses/
4. Edit the environment/shell path by running:
   `export GRB_LICENSE_FILE=path/to/gurobi.lic`
5. Download the Gurobi Optimizer (version 11.0 or later) that is suitable for your OS. (Tested with Gurobi Optimizer version 11.0.2 build v11.0.2rc0 for mac64[arm].)
6. To verify that Gurobi Optimizer and the license are installed correctly, please run
   `gurobi_cl --license`

[1] https://github.com/google/private-join-and-compute

[2] https://github.com/google/private-join-and-compute/commit/c0cb3f5b5b616caaaef80b7f7e9c335f4fe53ce7

A.2.5 Benchmarks

We do not require any external dataset for our experiments as we only use synthetic data, and we have included the data generation as part of our artifact.

A.3 Set-up

A.3.1 Installation

Ensure you have the Gurobi Optimizer and license installed, as described in Section A.2.4. The following instructions describe how to use a virtual python environment for the artifact evaluation. We tested our artifact using `python 3.10.15`.

1. If you do not have `virtualenv` installed, install it with:
   `pip3 install virtualenv`
2. Create a new virtual environment:
   `virtualenv myenv`
3. Activate the virtual environment:
   - Windows: `myenv\Scripts\activate`
   - MacOS/Linux: `source myenv/bin/activate`
4. Install the dependencies:
   `pip3 install -r requirements.txt`

A.3.2 Basic Test

1. To run a simple functionality test for the search tree attack, run the following command from the top-level directory:
   `python3 search_tree/test.py`
2. For the basic test for CS, DFT and MLE attacks, navigate to `cs_dft_mle/experiment/`, and run the following command:
   `python3 experiment.py ../config/test_non_sparse.json ../config/test_sparse.json`

A.4 Evaluation workflow

A.4.1 Major Claims

(C1): When carrying out the search-tree attack (described in Section 4 of our paper), increasing the partition size generally resulted in fewer queries with some exceptions. Notably, target set size $|T| = 10^3$ and partition set size $\ell = 2^6$ resulted in fewer queries than $\ell = 2^8$ for all values of intersection ratio ρ tested; additionally, when $\ell = 2^{10}$, we have that $\ell \geq |T|$, and thus all values are recovered with exactly one query. Similar trends are observed for $|T| = 10^4$ and $|T| = 10^5$. This is proven by the experiment (E1) described in Section 7.1 of our paper whose results are illustrated in Figure 3.

(C2): We implemented our recovery methods, maximum likelihood estimation (MLE), discrete fourier transform (DFT), and compressed sensing (CS) using the Gurobi Optimizer. We then ran experiments for both noiseless and noisy recoveries using synthetic data with value ranges of $[-100, 100]$ and $[-1000, 1000]$. We report the recovery results over five runs, with statistics for the value range $[-1000, 1000]$ shown in Table 2 (noiseless) and Table 3 (noisy); results for the value range $[-100, 100]$ are presented in Table 4 (noiseless) and Table 5 (noisy) in the Appendix. We describe the steps to reproduce our results in Experiment E2 and briefly describe our observations below.

MLE. We focus on non-sparse recovery using MLE by setting the intersection size equal to the target set size. We observed that even after $0.95n$ queries for n targets, there is still some ℓ_1 loss in value recovery. This loss increases significantly when the value range is expanded to $[-1000, 1000]$ (see Tables 2 and 3). Moreover, adding noise to each query result (i.e., inner product) does not significantly affect the losses.

DFT. Our results indicate that DFT recovery performs very well in certain experiments. It consumes significantly less time compared with other methods. However, it can be unstable and is not robust to noise.

Compressed Sensing (CS). In the noiseless setting, both CS-ℓ_1 and CS-ℓ_2 achieve exact recovery for intersection ratios up to 0.2 with high probability. CS-ℓ_1 is more scalable than CS-ℓ_2; for CS-ℓ_2, we were only able to scale up to $n = 2000$ using Gurobi (with our table reporting $n = 1000$). For noisy recovery, both methods show robustness to truncated Gaussian noise with standard deviation $\sigma \in \{2.5, 5\}$.

A.4.2 Experiments

(E1): Search Tree Attack Experiment [5 human-minutes + 50 compute-minutes (Short Experiment)/7.34 compute-hours (Full Experiment) + 5GB disk]. Time estimates are based on running the experiments on a 16GB Apple M2 Pro.

Preparation: Make sure to follow the setup steps described in Appendix A.3 of this writeup. Before running this experiment, create the directory in which the results will be stored by running the command:
`mkdir results`

Execution: To run the experiments for the search tree attack, run the following commmand from the `search_tree` directory of the repository:
`python3 benchmark.py NUM-RUNS parameter`
where `NUM-RUNS` is a positive integer denoting how many times to run the attack for each parameter setting and `PARAMS` takes on the value of either `part` (which runs an abbreviated set of parameters and which can run in <1hr on a commercial laptop) or `full` (which runs the full set of parameters reported in our paper). For example, the command
`python3 benchmark.py 1 part`

specifies running the benchmarks for the abbreviated set of parameters for 1 run per parameter setting.
Results: To reproduce the graphs, first navigate to `results-part.tex` (if running the abbreviated parameter selection) or `results-full.tex` (if running the full parameter selection) and compile the LaTeX document. These documents pull the data from `results` and print the graphs depicted in Figures 3 and 7.

(E2): Noiseless and noisy recovery using CS, DFT and MLE [5 human-minutes + 4 compute-hours] To save evaluation time, we set the number of iterations to 1:
Preparation: Ensure that you have completed the setup steps described in Appendix A.3. Work within the `cs_dft_mle` directory. Navigate to the `experiment` directory. Remove the `results` directory if it exists due to basic test using `rm results`.
Execution:

1. (15 compute-minutes) Follow the following steps to produce results for target set size $n \in \{10^2, 10^3\}$:
   ```
   python3 experiment.py
   ../config/e1_non_sparse.json
   ../config/e1_sparse.json
   ```
   It outputs a `results` directory containing CSV files that report the results.
2. (3 compute-hours) The following step produces results for $n = 10^4$ (excluding the CS-ℓ_2 for noisy recovery, as explained in the paper):
   ```
   python3 experiment.py
   ../config/e2_non_sparse.json
   ../config/e2_sparse.json --exclude
   ```
   This updates the CSV files in the `results` directory.

Results: To generate the LaTeX tables from the results, run the following command:
```
python3 gen_tex_table.py 1000
```
(Optional) If `pdflatex` is installed, you can preview the generated tables using the following command:
```
pdflatex gen_table_pdf.tex
```
This generates `gen_table_pdf.pdf` that contains two tables similar to Table 2 and Table 3 presented in our paper.
Optional: For Table 4 and Table 5 in the Appendix, which differ in value range from $[-1000, 1000]$ to $[-100, 100]$ and noise magnitude, the evaluation steps are similar. First backup the results directory using
```
mv results results_1000
```
then run the following commands:
```
python3 experiment.py
../config/e3_non_sparse.json
../config/e3_sparse.json
```
and
```
python3 experiment.py
../config/e4_non_sparse.json
../config/e4_sparse.json --exclude
```
and finally
```
python3 gen_tex_table.py 100
```
(Optional) If you have `pdflatex` installed, you can preview the tables using the following command:
```
pdflatex gen_table_pdf.tex
```

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/usenixsec2025/.