USENIX

THE ADVANCED COMPUTING SYSTEMS ASSOCIATION

Fast Enhanced Private Set Union in the Balanced and Unbalanced Scenarios

Binbin Tu and Yujie Bai, *School of Cyber Science and Technology, Shandong University; Quan Cheng Laboratory; Key Laboratory of Cryptologic Technology and Information Security, Ministry of Education, Shandong University;* Cong Zhang, *Institute for Advanced Study, BNRist, Tsinghua University;* Yang Cao and Yu Chen, *School of Cyber Science and Technology, Shandong University; Quan Cheng Laboratory; Key Laboratory of Cryptologic Technology and Information Security, Ministry of Education, Shandong University*

https://www.usenix.org/conference/usenixsecurity25/presentation/tu

This artifact appendix is included in the Artifact Appendices to the Proceedings of the 34th USENIX Security Symposium and appends to the paper of the same name that appears in the Proceedings of the 34th USENIX Security Symposium.

August 13–15, 2025 • Seattle, WA, USA

978-1-939133-52-6

Open access to the Artifact Appendices to the Proceedings of the 34th USENIX Security Symposium is sponsored by USENIX.

USENIX'25 Artifact Appendix: Fast Enhanced Private Set Union in the Balanced and Unbalanced Scenarios

Binbin Tu[1,2,3], Yujie Bai[1,2,3], Cong Zhang[4], Yang Cao[1,2,3], and Yu Chen[1,2,3](✉)

[1]School of Cyber Science and Technology, Shandong University, Qingdao 266237, China
[2]Quan Cheng Laboratory, Jinan 250103, China
[3]Key Laboratory of Cryptologic Technology and Information Security, Ministry of Education, Shandong University, Qingdao 266237, China
[4]Institute for Advanced Study, BNRist, Tsinghua University, Beijing, China
{tubinbin,yuchen}@sdu.edu.cn, {baiyujie,caoyang24}@mail.sdu.edu.cn, zhangcong@mail.tsinghua.edu.cn

A Artifact Appendix

A.1 Abstract

The source code for our two enhanced PSU (ePSU) protocols in balanced and unbalanced settings, is available at https://github.com/real-world-cryptography/ePSU. We implement the hashing, pnMCRG, and one-time pad routines for our ePSU protocols. Our pnMCRG consists of two steps: pMCRG and nECRG. Additionally, pMCRG can be further divided into three steps: bOPRF + OKVS + pECRG in the balanced setting and bOPRF + FHE + pECRG in the unbalanced setting. All artifact evaluation results are provided in Section 8 of the paper and the Appendix.

A.2 Description & Requirements

The source code for our balanced and unbalanced ePSU protocols can be described as follows.

Balanced Setting. The code of our balanced ePSU is located in the "balanced_ePSU/" folder. In this folder, "ePSU/balanced_epsu.cpp" contains the complete implementation of ePSU, and "pnMCRG/pnMCRG.cpp" implements pnMCRG (Figure 18: pnMCRG from pMCRG and nECRG), including key components pMCRG (Figure 13: pMCRG from bOPRF, OKVS, and pECRG), pECRG (Figure 11: DDH-based pECRG), and nECRG (Figure 16: nECRG from ssPEQT and ROT). The "test/" folder contains test programs for balanced ePSU, pnMCRG, pMCRG, pECRG, and nECRG.

Unbalanced Setting. The code of our unbalanced ePSU is located in the "unbalanced_ePSU/" folder. In the "MCRG/" folder, we reuse part of APSU (https://github.com/real-world-cryptography/APSU) to implement steps 1–7 from Figure 14 (MCRG based on bOPRF and FHE). "pECRG_nECRG_OTP/pecrg_necrg_otp/pecrg_necrg_otp.cpp" implements the remaining parts of our unbalanced ePSU. "pECRG_nECRG_OTP/pnecrg/pnECRG.cpp" implements the key components pECRG and nECRG. The "test/" folder contains test programs for pECRG (step 8 in Figure 14) and nECRG (Figure 16: nECRG from ssPEQT and ROT). Additionally, "test.py" is a script to automate the MCRG and pECRG_nECRG_OTP processes.

We run our experiments on a single Intel Core i7-13700 CPU @ 5.20GHz with 32 threads and 64GB of RAM, running Ubuntu 22.04.

A.2.1 Security, privacy, and ethical concerns

None.

A.2.2 How to access

We have open-sourced the implementation of our two enhanced PSU protocols at https://github.com/real-world-cryptography/ePSU. Besides, our artifact is available at https://doi.org/10.5281/zenodo.15128481.

A.2.3 Hardware dependencies

Our ePSU currently supports Ubuntu 22.04.

A.2.4 Software dependencies

Our ePSU requires the installation of G++ (version 11.4.0) and Python (version 3.10.12).

We leverage the constructions in [KKRT16, RS21, RR22, FV12] to implement bOPRF, OKVS, ssPEQT, and FHE (the building block of balanced/unbalanced pnMCRG) and use the code from the Vole-PSI library [Vol], the SEAL library [SEA], and the APSI/APSU library [APS, PSU]. We use an array of optimization techniques of FHE as [CLR17, CHLR18, CMdG+21, TCLZ23], such as batching (SIMD), windowing, and partitioning to significantly reduce the depth of the homomorphic circuit. As for ROT (the building block of nECRG) and DDH-based pECRG, we follow the libOTe library [lib]

and the OpenSSL library [Ope]. Our code supports multi-threading parallelism following the Vole-PSI library [Vol] and the OpenMP library [MP].

A.2.5 Benchmarks

We set the computational security parameter $\kappa = 128$, the statistical security parameter $\lambda = 40$ and use $\gamma = 3$ hash functions to insert sets X and Y into the cuckoo hash table and simple hash table, respectively. The item length is 64-bit following [LG23, DCZB24].

A.3 Set-up

A.3.1 Installation

Instructions for installing the required dependencies are given in https://github.com/real-world-cryptography/ePSU/blob/main/README.md. This code is conducted on Ubuntu 22.04, with G++ 11.4.0 and CMake 3.22.1.

A.3.2 Basic Test

ePSU Test. To run our balanced ePSU, evaluators can run the command in the balanced_ePSU/build directory. For example, when both set sizes are $|X| = |Y| = 2^{12}$ and thread number is 1, the command is `./test_balanced_epsu -nn 12 -nt 1 -r 0 & ./test_balanced_epsu -nn 12 -nt 1 -r 1`, where `nn`$\in \{10, 12, 14, 16, 18, 20, 22\}$ denotes the logarithm of set size $\{2^{10}, 2^{12}, 2^{14}, 2^{16}, 2^{18}, 2^{20}, 2^{22}\}$; `nt` denotes the number of threads$\{1, 2, 4, 8\}$; `r` denotes the index of involved parties (only 0 or 1).

Sub-protocols of ePSU test: run the commands in the balanced_ePSU/build directory. The symbols `nn`, `nt`, and `r` remain consistent with those in ePSU.

- **pnMCRG Test Command.**
  ```
  ./test_pnmcrg -nn 12 -nt 1 -r 0 &
  ./test_pnmcrg -nn 12 -nt 1 -r 1
  ```
- **pMCRG Test Command.**
  ```
  ./test_pmcrg -nn 12 -nt 1 -r 0 &
  ./test_pmcrg -nn 12 -nt 1 -r 1
  ```
- **pECRG Test Command.**
  ```
  ./test_pecrg -nn 12 -nt 1 -r 0 &
  ./test_pecrg -nn 12 -nt 1 -r 1
  ```
- **nECRG Test Command.**
  ```
  ./test_necrg -nn 12 -nt 1 -r 0 &
  ./test_necrg -nn 12 -nt 1 -r 1
  ```

eUPSU Test. To run our unbalanced ePSU, evaluators can run the command in the ePSU/unbalanced_ePSU directory. Here we fix the small set size $|X| = 2^{10}$. For example, when the large set size $|Y| = 2^{12}$ and thread number is 1, the command is `python3 test.py -pecrg_necrg_otp -cn 1 -nt 1 -nn 12`, where `nn`$\in \{12, 14, 16, 18, 20, 22\}$ denotes the logarithm of large set size $\{2^{12}, 2^{14}, 2^{16}, 2^{18}, 2^{20}, 2^{22}\}$; `nt` denotes the number of threads $\{1, 2, 4, 8\}$; `cn` denotes the column number of the matrix (if the large set size is less than 2^{20}, set to 1; otherwise set to 2).

Sub-protocols of eUPSU test: run the commands in the ePSU/unbalanced_ePSU directory. The symbols are the same as above.

- **pECRG Test Command.**
  ```
  python3 test.py -pecrg -cn 1 -nt 1 -nn 12
  ```
- **pnECRG Test Command.**
  ```
  python3 test.py -pnecrg -cn 1 -nt 1 -nn 12
  ```

A.4 Evaluation workflow

A.4.1 Major Claims

(C1): We implement our enhanced PSU in the balanced setting (ePSU), and its subprotocols, including pnMCRG, pMCRG, pECRG, and nECRG.

(C2): We implement our enhanced PSU in the unbalanced setting (eUPSU), and its subprotocols, including pnMCRG and pMCRG.

A.4.2 Experiments

(E1): *[ePSU performance] : This experiment benchmarks the communication and runtime of ePSU with 10Gbps bandwidth, 0.4ms RTT; 100Mbps, 10Mbps and 1Mbps bandwidth, 160ms RTT.*

How to: Open a terminal, and execute the following command: `tc qdisc add dev lo root netem delay 0.2ms rate 10Gbit` to config the local network as 10Gbit bandwidth with 0.4ms RTT. Evaluators can adjust the network settings using different parameters as needed, such as 100Mbps, 10Mbps and 1Mbps bandwidth, 160ms RTT. Then run the command `./test_balanced_epsu -nn 20 -nt 1 -r 0 & ./test_balanced_epsu -nn 20 -nt 1 -r 1` in the balanced_ePSU/build directory. The command will run balanced_ePSU with set size $|X| = |Y| = 2^{20}$, and print the information of communication and running time. Evaluators can set `nn` to test balanced_ePSU under different set sizes, `nt` to test balanced_ePSU with different numbers of threads.

Results: For each run of balanced_ePSU, this experiment print information as follows:

- `Comm cost = 277.402 MB`, indicating the communication cost of the balanced_ePSU is 277.402 MB.
- `end 78144.0 78143.895 **********`, indicating the running time of the balanced_ePSU is 78144 ms.

- `Balanced_ePSU functionality test pass! And union size is: 1048577`, indicating the Balanced_ePSU functionality tests passed.

(E2): *[eUPSU performance]: This experiment benchmarks the communication and runtime of eUPSU with 10Gbps bandwidth, 0.4ms RTT; 100Mbps, 10Mbps and 1Mbps bandwidth, 160ms RTT.*

How to: Open a terminal, and execute the following command: `tc qdisc add dev lo root netem delay 0.2ms rate 10Gbit` to config the local network as 10Gbit bandwidth with 0.4ms RTT. Evaluators can adjust the network settings using different parameters as needed, such as 100Mbps, 10Mbps and 1Mbps bandwidth, 160ms RTT. Then run the command `python3 test.py -pecrg_necrg_otp -cn 1 -nt 1 -nn 20` in the unbalanced_ePSU directory. The command will run unbalanced_ePSU with set size $|X| = 2^{10}, |Y| = 2^{20}$, and print the information of communication and time. Evaluators can set `nn` to test unbalanced_ePSU under different set sizes, `nt` to test unbalanced_ePSU with different numbers of threads.

Note: The total running time of unbalanced_ePSU is the sum of the MCRG execution time and the pECRG_nECRG_OTP execution time. Similarly, the total communication cost of unbalanced_ePSU is the sum of the MCRG communication cost and the pECRG_nECRG_OTP communication cost.

Results: For each run of unbalance_ePSU, this experiment print information as follows:

- `receiver all time15778.6`, indicating the running time of MCRG in unbalanced_ePSU is 15778.6 ms.
- `end 178.4 178.382 **********`, indicating the running time of pECRG_nECRG_OTP in unbalanced_ePSU is 178.5 ms.
- `Communication total: 1983 KB`, indicating the communication cost of MCRG in unbalanced_ePSU is 1983 KB.
- `Comm cost = 0.482 MB`, indicating the communication cost of pECRG_nECRG_OTP in unbalanced_ePSU is 0.482 MB.

(E3): *[balanced pnMCRG performance]: This experiment benchmarks the communication and runtime of pnMCRG in a balanced setting with 10Gbps bandwidth, 0.4ms RTT.*

How to: Open a terminal, and execute the following command: `tc qdisc add dev lo root netem delay 0.2ms rate 10Gbit` to config the local network as 10Gbit bandwidth with 0.4ms RTT. Then run the command `./test_pnmcrg -nn 20 -nt 1 -r 0 & ./test_pnmcrg -nn 20 -nt 1 -r 1` in the balanced_ePSU/build directory. The command will run the balanced pnMCRG with set size $|X| = |Y| = 2^{20}$, and print the information of communication and time. Evaluators can set `nn` to test the balanced pnMCRG under different set sizes, `nt` to test the balanced pnMCRG with different numbers of threads.

Results: For each run of balanced pnMCRG, this experiment prints information as follows:

- `Comm cost = 256.355 MB`, indicating the communication cost of the balanced pnMCRG is 256.355 MB.
- `end 78065.2 78065.166 **********`, indicating the running time of the balanced pnMCRG is 78065.2 ms.
- `pnMCRG functionality test pass!`, indicating the balanced pnMCRG functionality tests passed.

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/.

References

[APS] https://github.com/microsoft/APSI.

[CHLR18] Hao Chen, Zhicong Huang, Kim Laine, and Peter Rindal. Labeled PSI from fully homomorphic encryption with malicious security. In *ACM CCS*, 2018.

[CLR17] Hao Chen, Kim Laine, and Peter Rindal. Fast private set intersection from homomorphic encryption. In *ACM CCS*, 2017.

[CMdG+21] Kelong Cong, Radames Cruz Moreno, Mariana Botelho da Gama, Wei Dai, Ilia Iliashenko, Kim Laine, and Michael Rosenberg. Labeled PSI from homomorphic encryption with reduced computation and communication. In *ACM CCS*, 2021.

[DCZB24] Minglang Dong, Yu Chen, Cong Zhang, and Yujie Bai. Breaking free: Efficient multi-party private set union without non-collusion assumptions. *IACR Cryptol. ePrint Arch.*, 2024.

[FV12] Junfeng Fan and Frederik Vercauteren. Somewhat practical fully homomorphic encryption. *IACR Cryptol. ePrint Arch.*, 2012.

[KKRT16] Vladimir Kolesnikov, Ranjit Kumaresan, Mike Rosulek, and Ni Trieu. Efficient batched oblivious PRF with applications to private set intersection. In *ACM CCS*, 2016.

[LG23] Xiang Liu and Ying Gao. Scalable multi-party private set union from multi-query secret-shared private membership test. In *ASIACRYPT*, 2023.

[lib] https://github.com/osu-crypto/libOTe.

[MP] https://www.openmp.org/.

[Ope] https://github.com/openssl/openssl.

[PSU] https://github.com/real-world-cryprography/APSU.

[RR22] Srinivasan Raghuraman and Peter Rindal. Blazing fast PSI from improved OKVS and subfield VOLE. In *ACM CCS*, 2022.

[RS21] Peter Rindal and Phillipp Schoppmann. VOLE-PSI: fast OPRF and circuit-psi from vector-ole. In *EUROCRYPT*, 2021.

[SEA] https://github.com/microsoft/SEAL.

[TCLZ23] Binbin Tu, Yu Chen, Qi Liu, and Cong Zhang. Fast unbalanced private set union from fully homomorphic encryption. In *ACM CCS*, 2023.

[Vol] https://github.com/Visa-Research/volepsi.