

Let us walk on the 3-isogeny graph: Step-by-step Artifact Walkthrough

Jesús-Javier Chi-Domínguez, Eduardo Ochoa-Jiménez and
Ricardo-Neftalí Pontaza-Rodas

August 17, 2025

Outline

Repository Overview

System Requirements

How to Install and How to Build?

How to Run Tests?

How to Benchmark?

Reproducing Manuscript's Graphics

Generating Technical Documentation

CI/CD Pipeline Overview

How to download our Docker Container? (1/2)

Additional Resources: CPU benchmarking

License and Contributions

Repository Overview (1/4)

- ▶ *Let us walk on the 3-isogeny graph: efficient, fast, and simple* is an open-source C framework for using 3-radical isogenies to improve some post-quantum cryptosystems (dCTIDH + QFESTA).
- ▶ This presentation summarizes the software structure and reproducibility workflow.

Repository Overview (2/4)

- ▶ Hosted on GitHub: <https://github.com/Crypto-TII/pqc-engineering-ssec-23>
- ▶ Modular design with components: Presentation Video, System Requirements, Build, Test, Benchmarks, Docs, Manuscript results replication, and CI/CD Pipeline.

Repository Overview (3/4)

Overview of our paper - YouTube video:

<https://www.youtube.com/watch?v=BjedMooSV30&list=PLFgwYy6Y-xWYCFruq66CFXXiWEWckEk6Q>



Figure 1: Overview of our paper - YouTube video.

Repository Overview (4/4)

(Full) Guided Tour of our Artifact:

https://www.youtube.com/watch?v=hLk_B5NpKRA&list=PLFgwYy6Y-xWYCFruq66CFXXiWEWckEk6Q&index=10



Figure 2: (Full) Guided Tour - YouTube video.

System Requirements

Our system requirements are extremely simple:

1. Out-of-the-box Linux (CPU Intel x86_64).
2. CMake + gcc
3. Python3:
 - ▶ Numpy
 - ▶ Matplotlib

How to Install?

Clone from GitHub.

Run:

```
git clone  
https://github.com/Crypto-TII/pqc-engineering-ssec-23.git
```

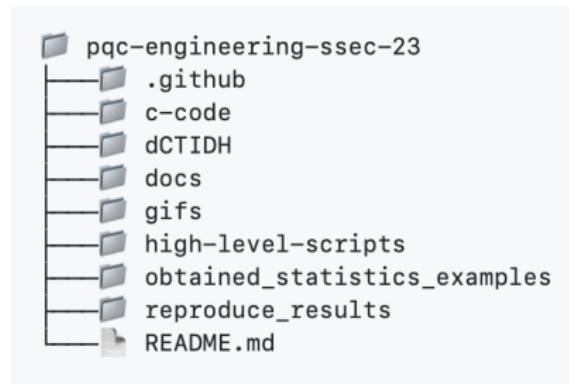


Figure 3: Downloaded project structure.

How to Build? (1/2)

Run:

```
cd c-code
cmake -DCMAKE_BUILD_TYPE=Release -B cmake-build-release
cd cmake-build-release
make -j
```

Figure 4: Build instructions.

How to Build? (2/2)

The screenshot shows a terminal window with two tabs open. The active tab displays the build logs for a C project. The logs show the compilation of various source files into objects and executables, including `munit.c.o`, `test_main.c.o`, `test_fq.c.o`, and `test_fp.c.o`. It also shows the linking of static libraries like `libsssec-p783.a` and the final executable `ssec-p783`. The build process includes several warning messages from the linker about missing `.note.GNU-stack` sections.

```
[ 76%] Building C object tests/CMakeFiles/tests-ssec-p592.dir/test_fq.c.o
[ 77%] Building C object benchmarks/CMakeFiles/benchmarks-ssec-p511.dir/benchmarks_main.c.o
[ 77%] Linking C executable benchmarks-ssec-p575
[ 77%] Linking C static library libsssec-p783.a
[ 78%] Building C object tests/CMakeFiles/tests-ssec-p511.dir/munit.c.o
[ 78%] Building C object tests/CMakeFiles/tests-ssec-p511.dir/test_main.c.o
[ 78%] Building C object tests/CMakeFiles/tests-ssec-p511.dir/test_fq.c.o
[ 79%] Building C object tests/CMakeFiles/tests-ssec-p511.dir/test_fp.c.o
[ 80%] Building C object tests/CMakeFiles/tests-ssec-p511.dir/test_isogeny_walks.c.o
[ 81%] Linking C executable benchmarks-ssec-p255
[ 82%] Linking C executable benchmarks-ssec-p383
[ 83%] Linking C executable benchmarks-ssec-p381
[ 84%] Linking C static library libsssec-p765.a
[ 84%] Built target ssec-p783
/usr/bin/ld: [ 84%] Building C object tests/CMakeFiles/tests-ssec-p783.dir/munit.c.o
warning: p254.s.o: missing .note.GNU-stack section implies executable stack
/usr/bin/ld: NOTE: This behaviour is deprecated and will be removed in a future version of the linker
[ 85%] Building C object tests/CMakeFiles/tests-ssec-p783.dir/test_fp.c.o
[ 85%] Linking C executable benchmarks-ssec-p398
[ 86%] Building C object tests/CMakeFiles/tests-ssec-p783.dir/test_main.c.o
[ 86%] Building C object tests/CMakeFiles/tests-ssec-p783.dir/test_fq.c.o
[ 87%] Building C object tests/CMakeFiles/tests-ssec-p783.dir/test_isogeny_walks.c.o
[ 87%] Building C object benchmarks/CMakeFiles/benchmarks-ssec-p783.dir/benchmarks_main.c.o
[ 87%] Built target ssec-p765
[ 88%] Building C object benchmarks/CMakeFiles/benchmarks-ssec-p765.dir/benchmarks_main.c.o
[ 88%] Built target benchmarks-ssec-p254
[ 89%] Building C object tests/CMakeFiles/tests-ssec-p765.dir/munit.c.o
[ 89%] Building C object tests/CMakeFiles/tests-ssec-p765.dir/test_main.c.o
[ 90%] Building C object tests/CMakeFiles/tests-ssec-p765.dir/test_fq.c.o
[ 90%] Building C object tests/CMakeFiles/tests-ssec-p765.dir/test_fp.c.o
[ 91%] Building C object tests/CMakeFiles/tests-ssec-p765.dir/test_isogeny_walks.c.o
/usr/bin/ld: warning: p575.s.o: missing .note.GNU-stack section implies executable stack
/usr/bin/ld: NOTE: This behaviour is deprecated and will be removed in a future version of the linker
[ 91%] Built target benchmarks-ssec-p575
```

Figure 5: Build process demo.

How to Run Tests? (1/2)

For running unit tests, simply execute:

- ▶ cd cmake-build-release
- ▶ ./tests/tests-ssec-p254
- ▶ ./tests/tests-ssec-p255
- ▶ ./tests/tests-ssec-p381
- ▶ ./tests/tests-ssec-p383
- ▶ ./tests/tests-ssec-p398
- ▶ ./tests/tests-ssec-p511
- ▶ ./tests/tests-ssec-p575
- ▶ ./tests/tests-ssec-p592
- ▶ ./tests/tests-ssec-p765
- ▶ ./tests/tests-ssec-p783

How to Run Tests? (2/2)

```
liwuen@liwuen-aero16:~/ter < liwuen@liwuen-aero16:~/proj < liwuen@liwuen-aero16:~/test < + <
tests/fp/cube_root
[ OK ] [ 0.00001464 / 0.00001443 CPU ]
[ OK ] [ 0.00732109 / 0.00721689 CPU ]
[ OK ] [ 0.00000041 / 0.00000040 CPU ]
[ OK ] [ 0.00020398 / 0.00020008 CPU ]
[ OK ] [ 0.00034651 / 0.00034255 CPU ]
[ OK ] [ 0.17325337 / 0.17127737 CPU ]
[ OK ] [ 0.00087555 / 0.00086566 CPU ]
[ OK ] [ 0.43777348 / 0.43282997 CPU ]
[ OK ] [ 0.00000081 / 0.00000080 CPU ]
[ OK ] [ 0.00040625 / 0.00040114 CPU ]
[ OK ] [ 0.00000165 / 0.00000163 CPU ]
[ OK ] [ 0.00082468 / 0.00081593 CPU ]
[ OK ] [ 0.000004197 / 0.000004155 CPU ]
[ OK ] [ 0.02898698 / 0.02877652 CPU ]
[ OK ] [ 0.00026474 / 0.00026214 CPU ]
[ OK ] [ 0.13237001 / 0.13106988 CPU ]
[ OK ] [ 0.00034206 / 0.00033878 CPU ]
[ OK ] [ 0.17103051 / 0.16934774 CPU ]
[ OK ] [ 0.00006664 / 0.00006599 CPU ]
[ OK ] [ 0.03331849 / 0.03299386 CPU ]
[ OK ] [ 0.00022046 / 0.00022002 CPU ]
[ OK ] [ 0.11823842 / 0.11000877 CPU ]
[ OK ] [ 0.00000928 / 0.00000918 CPU ]
[ OK ] [ 0.00463850 / 0.00459246 CPU ]
[ OK ] [ 0.02829769 / 0.02837272 CPU ]
[ OK ] [ 14.14884703 / 14.18636159 CPU ]
[ OK ] [ 0.00001137 / 0.00001158 CPU ]
[ OK ] [ 0.00568370 / 0.00579185 CPU ]
[ OK ] [ 0.01205582 / 0.01215694 CPU ]
[ OK ] [ 6.02790910 / 6.07846774 CPU ]
[ OK ] [ 0.24975477 / 0.24984901 CPU ]
[ OK ] [ 124.87738506 / 124.92450647 CPU ]

22 of 22 (100%) tests successful, 0 (0%) test skipped.
liwuen@liwuen-aero16:~/test_demo/pqc-engineering-ssec-23/c-code/cmake-build-release$ |
```

Figure 6: Test demo.

How to Benchmark? (1/4)

- ▶ **Important:** Need to use flags `-DCMAKE_BUILD_TYPE=Release` `-DBENCHMARKING=CYCLES` when building.
- ▶ If the flags were not used, the benchmarks will be empty.

```
liwuen@liwuen-aero16:~/test_demo/pqc-engineering-ssec-23/c-code/cmake-build-release$ benchmarks/benchmarks-ssec-p254
Numbers correspond for CGLHash2.
Average: 0

Q1:    0
Median: 0
Q3:    0

Min:    0
Max:    0

Numbers correspond for CGLHash3.
Average: 0

Q1:    0
Median: 0
Q3:    0

Min:    0
Max:    0
```

Figure 7: Benchmark errors

How to Benchmark? (2/4)

To **build** benchmarking, simply execute:

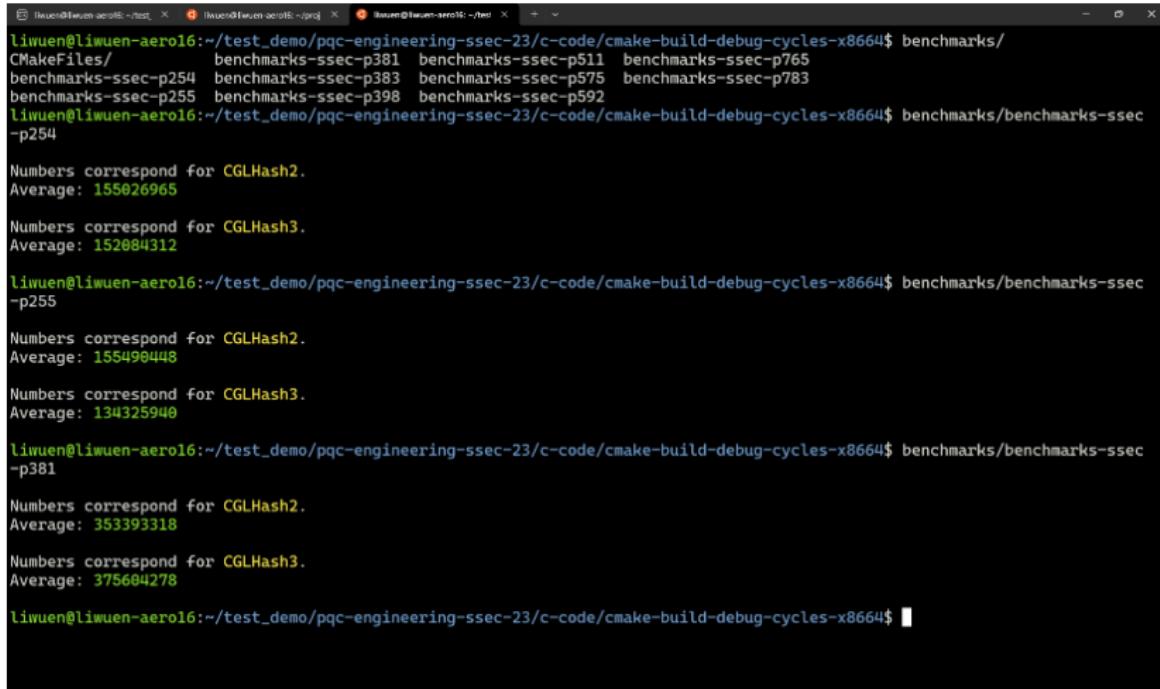
- ▶ `cmake -DCMAKE_BUILD_TYPE=Release
-DBENCHMARKING=CYCLES -DARCHITECTURE=x86_64 -B
cmake-build-release-cycles-x86_64`
- ▶ `cd cmake-build-release-cycles-x86_64`
- ▶ `make -j`

How to Benchmark? (3/4)

To **run** benchmarking, inside the
cmake-build-release-cycles-x8664 folder, simply execute:

- ▶ benchmarks/benchmarks-ssec-p254
- ▶ benchmarks/benchmarks-ssec-p255
- ▶ benchmarks/benchmarks-ssec-p381
- ▶ benchmarks/benchmarks-ssec-p383
- ▶ benchmarks/benchmarks-ssec-p398
- ▶ benchmarks/benchmarks-ssec-p511
- ▶ benchmarks/benchmarks-ssec-p575
- ▶ benchmarks/benchmarks-ssec-p592
- ▶ benchmarks/benchmarks-ssec-p765
- ▶ benchmarks/benchmarks-ssec-p783

How to Benchmark? (4/4)



The screenshot shows a terminal window with three tabs open. The active tab displays a benchmarking session for CGLHash2 and CGLHash3. The session involves running benchmarks for different SSEC sizes (p254, p381, p511, p765, p575, p783, p383, p398, p592) and then calculating averages. The process is repeated for both hash functions.

```
liwuen@liwuen-aero16:~/test_demo/pqc-engineering-ssec-23/c-code/cmake-build-debug-cycles-x8664$ benchmarks/CMakeFiles/benchmarks-ssec-p254 benchmarks-ssec-p381 benchmarks-ssec-p511 benchmarks-ssec-p765 benchmarks-ssec-p575 benchmarks-ssec-p783 benchmarks-ssec-p383 benchmarks-ssec-p398 benchmarks-ssec-p592 liwuen@liwuen-aero16:~/test_demo/pqc-engineering-ssec-23/c-code/cmake-build-debug-cycles-x8664$ benchmarks/benchmarks-ssec-p254

Numbers correspond for CGLHash2.
Average: 155826965

Numbers correspond for CGLHash3.
Average: 152084312

liwuen@liwuen-aero16:~/test_demo/pqc-engineering-ssec-23/c-code/cmake-build-debug-cycles-x8664$ benchmarks/benchmarks-ssec-p381

Numbers correspond for CGLHash2.
Average: 155490448

Numbers correspond for CGLHash3.
Average: 134325940

liwuen@liwuen-aero16:~/test_demo/pqc-engineering-ssec-23/c-code/cmake-build-debug-cycles-x8664$ benchmarks/benchmarks-ssec-p381

Numbers correspond for CGLHash2.
Average: 353393318

Numbers correspond for CGLHash3.
Average: 375604278

liwuen@liwuen-aero16:~/test_demo/pqc-engineering-ssec-23/c-code/cmake-build-debug-cycles-x8664$ █
```

Figure 8: Benchmarking Demo

Reproducing Manuscript's Graphics

- ▶ Scripts located in reproduce_results folder.
- ▶ Need Python: Numpy and Matplotlib.

```
pqc-engineering-ssec-23
├── c-code
├── dCTIDH
├── docs
├── gifs
├── high-level-scripts
├── obtained_statistics_examples
└── reproduce_results
    ├── manuscript_figure_03
    │   ├── benchmark_graph_03.py
    │   └── generate_figure_03.sh      # <= NEED TO EXECUTE
    ├── manuscript_figure_04
    │   ├── benchmark_graph_04.py
    │   └── generate_figure_04.sh      # <= NEED TO EXECUTE
    └── manuscript_figure_05
        ├── dCTIDH_benchmarks_output  # <= AUTOMATICALLY GENERATED!
        ├── dCTIDH_builds            # <= AUTOMATICALLY GENERATED!
        ├── statistics_output         # <= AUTOMATICALLY GENERATED!
        ├── analyze_bench.py
        ├── benchmark_graph_05.py
        └── generate_figure_05.sh      # <= NEED TO EXECUTE

```

README.md

Figure 9: Location of bash scripts to reproduce manuscript's results.

Reproducing Manuscript's Graphics: Figure 3 (1/4)

Simply execute

- ▶ `cd reproduce_results/manuscript_figure_03`
- ▶ `chmod +x generate_figure_03.sh`
- ▶ `./generate_figure_03.sh`

Reproducing Manuscript's Graphics: Figure 3 (2/4)

```
ricardopontaza@ricardo-pontaza-P1G5-Linux:~/demo/pqc-engineering-ssec-23/reproduce_results/manuscript_figure_03
```

```
ricardopontaza@ricardo-pontaza-P1G5-Linux:~/demo/pqc-engine... ricardopontaza@ricardo-pontaza-P1G5-Linux:~/demo/pqc-engine... ricardopontaza@ricardo-pontaza-P1G5-Linux:~/demo/pqc-engine...
```

```
Max: 745024488
```

```
Numbers correspond for CGLHash3.
```

```
Average: 769622965
```

```
Q1: 763647668
```

```
Median: 765412958
```

```
Q3: 768687464
```

```
Min: 760214504
```

```
Max: 955131572
```

```
benchmarks/benchmarks-ssec-p511 | tee benchmarks_ssec-p511.output.txt
```

```
Numbers correspond for CGLHash2.
```

```
Average: 1913451503
```

```
Q1: 1764669066
```

```
Median: 1771467675
```

```
Q3: 2188514140
```

```
Min: 1757381764
```

```
Max: 9349978846
```

```
Numbers correspond for CGLHash3.
```

```
Average: 2308921455
```

```
Q1: 2295673904
```

```
Median: 2304333139
```

```
Q3: 2316145304
```

```
Min: 2274135246
```

```
Max: 2589707834
```

Figure 10: Generation script for Figure 3.

Reproducing Manuscript's Graphics: Figure 3 (3/4)

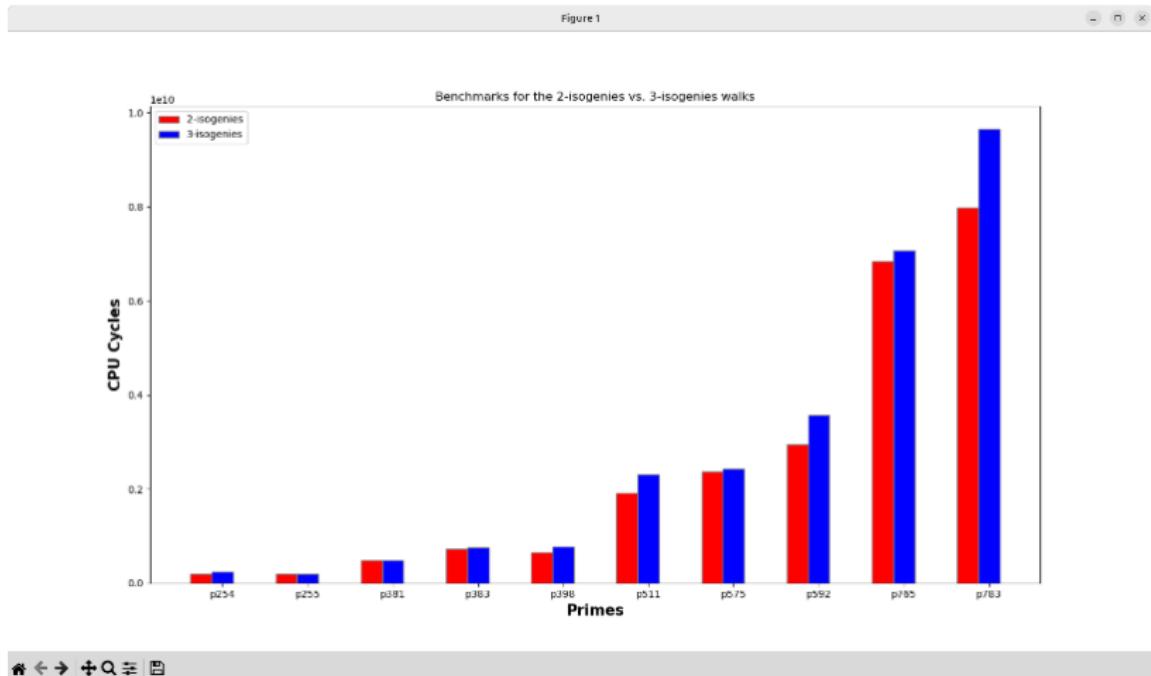


Figure 11: Generated statistical results from generate_figure_03.sh

Reproducing Manuscript's Graphics: Figure 3 (4/4)

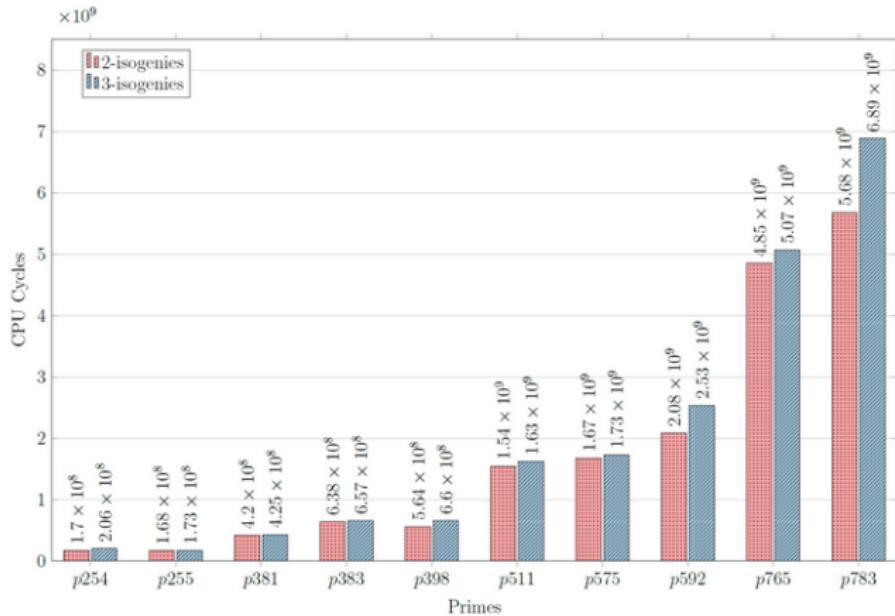


Figure 3: Benchmarks for the 2-isogenies vs. 3-isogenies walks, measured in CPU cycles.

Figure 12: Manuscript's Figure 3.

Reproducing Manuscript's Graphics: Figure 4 (1/4)

Simply execute

- ▶ `cd reproduce_results/manuscript_figure_04`
- ▶ `chmod +x generate_figure_04.sh`
- ▶ `./generate_figure_04.sh`

Reproducing Manuscript's Graphics: Figure 4 (2/4)

```
ricardopontaza@ricardo-pontaza-PiGS-Linux: ~/demo/pqc-engineering-ssec-23/reproduce_results/manuscript_figure_04
```

Numbers correspond for CGLHash2.
Average: 2368752706

Q1: 2358046436
Median: 2364409503
Q3: 2372597626

Min: 2339422190
Max: 2729886458

Numbers correspond for CGLHash3.
Average: 2443032114

Q1: 2431807688
Median: 2437604991
Q3: 2445513186

Min: 2415422662
Max: 2810899548

benchmarks/benchmarks-ssec-p592 | tee benchmarks_ssec-p592-output.txt

Numbers correspond for CGLHash2.
Average: 2940325885

Q1: 2928697408
Median: 2935591299
Q3: 2943246190

Min: 2904060464
Max: 3423052614

Numbers correspond for CGLHash3.
Average: 3583737646

Q1: 3564135148
Median: 3573225571
Q3: 3582969348

Min: 3538166126
Max: 4668548168

Figure 13: Generation script for Figure 4.

Reproducing Manuscript's Graphics: Figure 4 (3/4)

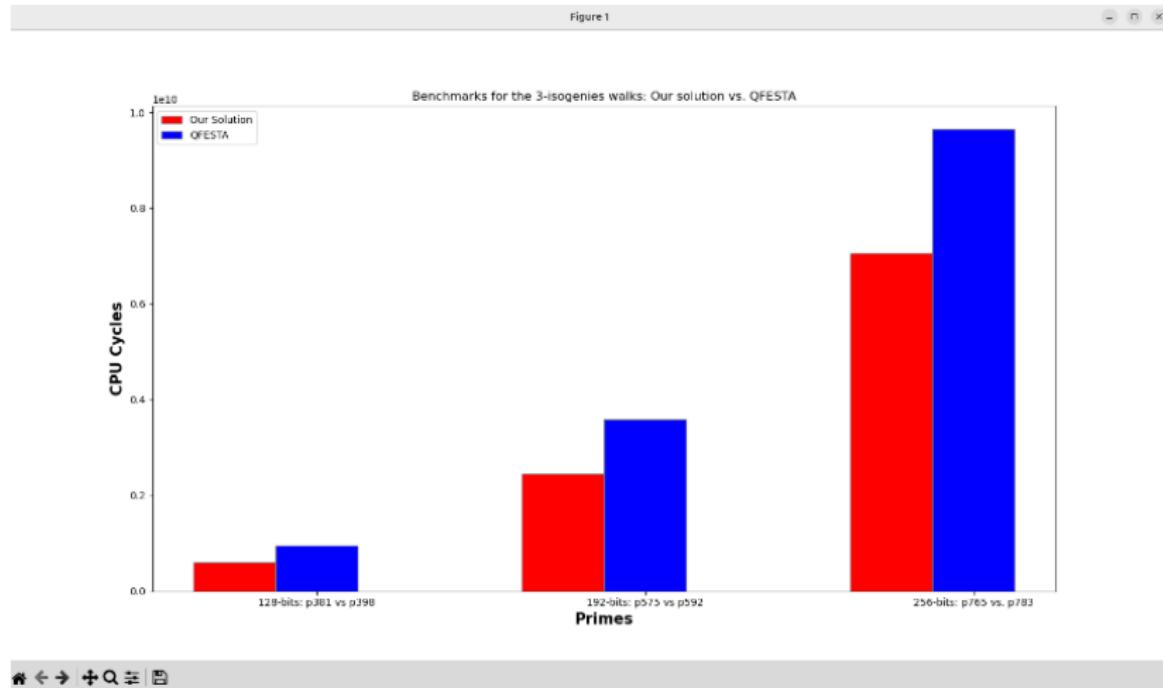


Figure 14: Generated statistical results from generate_figure_04.sh

Reproducing Manuscript's Graphics: Figure 4 (4/4)

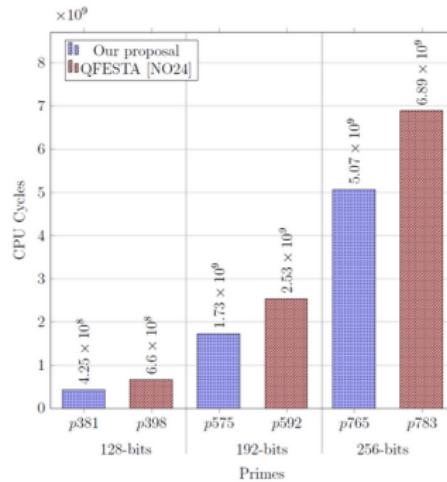


Figure 4: Benchmarks for the 3-isogenies walks for our proposed primes (p_{381} , p_{575} and p_{765}) vs. QFESTA [NO24] primes (p_{398} , p_{592} and p_{783}). Both p_{381} and p_{398} offer 128-bits security, while p_{575} and p_{592} offer 192-bits security, and p_{765} and p_{783} offer 256-bits security. For these six primes, the performance was measured in CPU cycles, having an improvement of 35.60% for 128-bits, 31.62% for 192-bits, and 26.41% for 256-bits, respectively.

Figure 15: Manuscript's Figure 4.

Reproducing Manuscript's Graphics: Figure 5 (1/5)

Simply execute

- ▶ `cd reproduce_results/manuscript_figure_05`
- ▶ `chmod +x generate_figure_05.sh`
- ▶ `./generate_figure_05.sh`

Reproducing Manuscript's Graphics: Figure 5 (2/5)

- ▶ The previous commands will (automatically) generate some folders.
- ▶ You can delete these (automatically-generated) folders between each run if necessary.

```
📁 manuscript_figure_05
├── 📁 dCTIDH_benchmarks_output      # <= AUTOMATICALLY GENERATED!
├── 📁 dCTIDH_builds                # <= AUTOMATICALLY GENERATED!
├── 📁 statistics_output           # <= AUTOMATICALLY GENERATED!
├── 📄 analyze_bench.py
├── 📄 benchmark_graph_05.py
└── 📄 generate_figure_05.sh        # <= NEED TO EXECUTE
```

Figure 16: Automatically-generated folders.

Reproducing Manuscript's Graphics: Figure 5 (3/5)

```
ricondepontazagricando.pontato@Ricardo-Pentax:~/Desktop/projects_dev/pqc-engineering-sse-23/reproduce_results/manuscript_Figure_05
```

```
�icondepontazagricando.pontato@Ricardo-Pentax:~/Desktop/projects_dev/pqc_eng... × ricondepontazagricando.pontato@Ricardo-Pentax:~/Desktop/projects_dev/pqc_eng... × ricondepontazagricando.pontato@Ricardo-Pentax:~/Desktop/projects_dev/pqc_eng... ×
```

```
action experiments 100 Mpc 1594.70106+-0.99399 mulsq 331392+-0 sq 678464+-0 addbs 474464+-0 mul 768546+-0 combo185 947543+-0
validate experiments 100 Mpc 1594.63372+-0.148899 mulsq 29673+-0 sq 8138+-0 addbs 17697+-0 mul 12493+-0 combo185 19882+-0
action experiments 100 Mpc 1594.65050+-0.85951 mulsq 331392+-0 sq 62946+-0 addbs 474464+-0 mul 768546+-0 combo185 324543+-0
validate experiments 100 Mpc 1594.63532+-0.18859 mulsq 29623+-0 sq 61238+-0 addbs 17697+-0 mul 12493+-0 combo185 19882+-0
action experiments 100 Mpc 1594.52652+-0.55987 mulsq 331392+-0 sq 62846+-0 addbs 474464+-0 mul 768546+-0 combo185 342543+-0
validate experiments 100 Mpc 1594.66687+-0.13848 mulsq 29673+-0 sq 8138+-0 addbs 17697+-0 mul 12493+-0 combo185 19882+-0
action experiments 100 Mpc 1594.63949+-0.97411 mulsq 331392+-0 sq 67846+-0 addbs 474464+-0 mul 768546+-0 combo185 342543+-0
validate experiments 100 Mpc 1594.59590+-0.07310 mulsq 331392+-0 sq 62046+-0 addbs 474464+-0 mul 768546+-0 combo185 342543+-0
action experiments 100 Mpc 1594.65461+-0.13441 mulsq 29623+-0 sq 8138+-0 addbs 17697+-0 mul 12493+-0 combo185 19882+-0
validate experiments 100 Mpc 1594.65461+-0.14150 mulsq 29623+-0 sq 8130+-0 addbs 17697+-0 mul 12493+-0 combo185 19882+-0
total action experiments 1000 Mpc 1594.68151+-0.98189 mulsq 331392+-0 sq 62846+-0 addbs 474464+-0 mul 768546+-0 combo185 342543+-0
echo .....
```

```
echo "Parsign benchmarking results for OPT_RAD_1 - 2047m61194 - keygen"
arsing benchmarking results for OPT_RAD_1 - 2047m61194 - keygen
#BNCH_KEYGEN_PRIM03-bench_keygen_194_opt_rad_1.out
#BNCH_KEYGEN_PRIM03-MH56D-analyzed_bench_keygen_194_opt_rad_1.txt
python3 analyze_bench_py
tee statistics_output/analyzed_bench_keygen_194_opt_rad_1.txt
tee statistics_output/analyzed_bench_act_194_opt_rad_1.txt
tee statistics_output/analyzed_bench_act_194_opt_rad_1.txt
ln -s statistics_output/analyzed_bench_keygen_194_opt_rad_1.txt
ln -s statistics_output/analyzed_bench_act_194_opt_rad_1.txt
ln -s statistics_output/analyzed_bench_act_194_opt_rad_1.txt
validate experiments 100 Mpc 16.10651+-0.05196 mulsq 15957+-0 sq 7605+-0 addbs 16133+-0 mul 11791+-0 combo185 18043+-0
action experiments 100 Mpc 10524.63549+-0.18174 mulsq 334323+-7418111 sq 764179+-841448 addbs 2459914+-2396273 mul 1598154+-1588666 combo185 2324485+-2381636
validate experiments 100 Mpc 16.13728+-0.18497 mulsq 15957+-0 sq 7605+-0 addbs 16133+-0 mul 11791+-0 combo185 188843+-0
action experiments 100 Mpc 10521.19758+-0.10913 mulsq 334323+-7418111 sq 764179+-841448 addbs 2459914+-2396273 mul 1598154+-1588666 combo185 2324485+-2381636
validate experiments 100 Mpc 16.12526+-0.11798 mulsq 15957+-0 sq 7605+-0 addbs 16123+-0 mul 11791+-0 combo185 188843+-0
validate experiments 100 Mpc 10523.03622+-0.09165 mulsq 334323+-2430111 sq 764179+-841448 addbs 2459914+-2396273 mul 1598154+-1588666 combo185 2324485+-2381636
validate experiments 100 Mpc 16.11994+-0.08499 mulsq 15957+-0 sq 7605+-0 addbs 16133+-0 mul 11791+-0 combo185 188843+-0
action experiments 100 Mpc 10517.19817+-0.08114 mulsq 334323+-7418111 sq 764179+-841448 addbs 2459914+-2396273 mul 1598154+-1588666 combo185 2324485+-2381636
validate experiments 100 Mpc 16.11644+-0.08848 mulsq 15957+-0 sq 8006+-0 addbs 16133+-0 mul 11791+-0 combo185 188843+-0
action experiments 100 Mpc 10526.87613+-0.08113 mulsq 334323+-7418111 sq 764179+-841448 addbs 2459914+-2396273 mul 1598154+-1588666 combo185 2324485+-2381636
validate experiments 100 Mpc 16.11760+-0.09771 mulsq 15957+-0 sq 7605+-0 addbs 16133+-0 mul 11791+-0 combo185 188843+-0
action experiments 100 Mpc 16.17002+-0.09809 mulsq 2354323+-7438111 sq 764179+-841448 addbs 2459914+-2396273 mul 1598154+-1588666 combo185 2324485+-2381636
validate experiments 100 Mpc 16.17434+-7.48356 mulsq 15957+-0 sq 7605+-0 addbs 16133+-0 mul 11791+-0 combo185 188843+-0
action experiments 100 Mpc 16.19117+-0.34065 mulsq 15957+-0 sq 7605+-0 addbs 16133+-0 mul 11791+-0 combo185 188843+-0
validate experiments 100 Mpc 16.10311+-0.115299 mulsq 15957+-0 sq 7605+-0 addbs 16133+-0 mul 11791+-0 combo185 188843+-0
action experiments 100 Mpc 10529.39521+-0.08006 mulsq 2354323+-7438111 sq 764179+-841448 addbs 2459914+-2396273 mul 1598154+-1588666 combo185 2324485+-2381636
validate experiments 100 Mpc 16.12081+-0.09072 mulsq 15957+-0 sq 8006+-0 addbs 16133+-0 mul 11791+-0 combo185 188843+-0
action experiments 100 Mpc 10538.38651+-10815.81557 mulsq 2354323+-2430111 sq 764179+-841448 addbs 2459914+-2396273 mul 1598154+-1588666 combo185 2324485+-2381636
validate experiments 1000 Mpc 06.11784+-0.78406 mulsq 15957+-0 sq 7605+-0 addbs 16133+-0 mul 11791+-0 combo185 188843+-0
total validate experiments 1000 Mpc 10520.17175+-10813.37398 mulsq 2354323+-2430111 sq 764179+-841448 addbs 2459914+-2396273 mul 1598154+-1588666 combo185 2324485+-2381636
echo .....
```

```
echo "Parsign benchmarking results for OPT_RAD_1 - 2047m61194 - act"
arsing benchmarking results for OPT RAD 1 - 2047m61194 - act
#BNCH_ACT_PRIM03-bench_act_194_opt_rad_1.out
#BNCH_ACT_PRIM03-MH56D-analyzed_bench_act_194_opt_rad_1.txt
python3 analyze_bench_py
tee statistics_output/analyzed_bench_act_194_opt_rad_1.txt
```

Figure 17: Generation script for Figure 5.

Reproducing Manuscript's Graphics: Figure 5 (4/5)

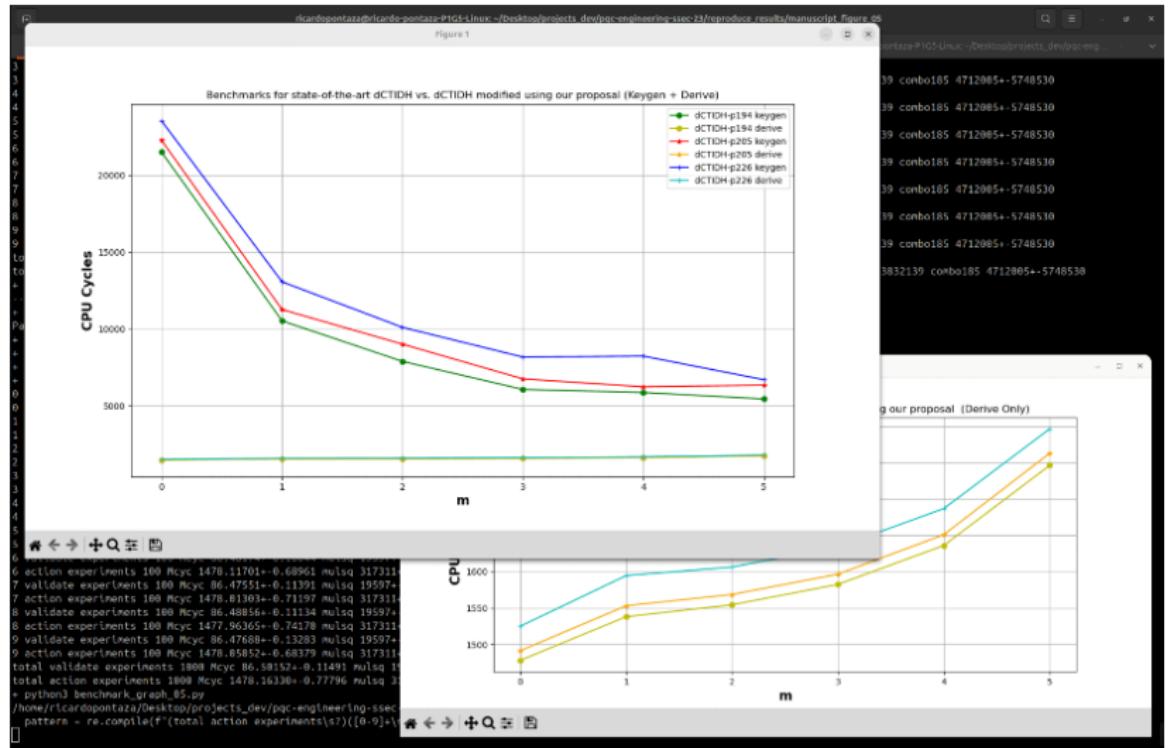


Figure 18: Generated statistical results from generate_figure_05.sh

Reproducing Manuscript's Graphics: Figure 5 (5/5)

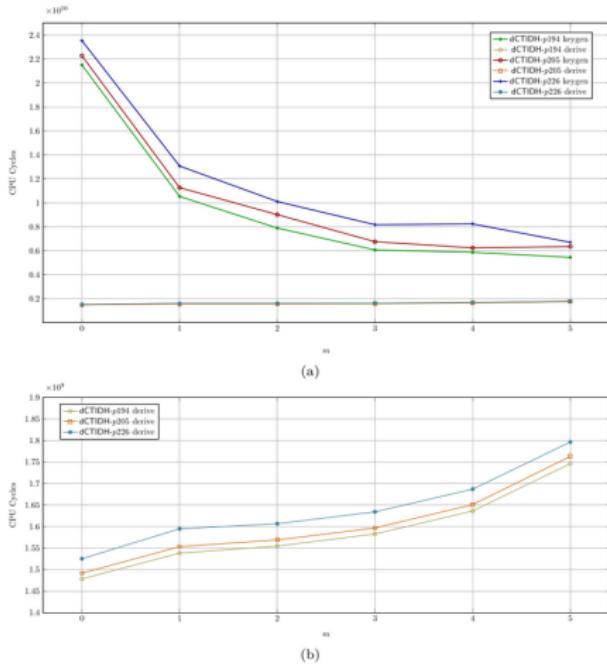


Figure 5: Benchmarks for state-of-the-art dCTIDH vs. dCTIDH modified using our proposal. Both the key generation (keygen) and the shared key derivation (derive) were tested. From

Figure 19: Manuscript's Figure 5.

Generating Technical Documentation (1/2)

We use Doxygen to generate the technical documentation.

- ▶ Configuration file: `Doxyfile`
- ▶ To generate, simply execute:
 - ▶ `cd docs`
 - ▶ `doxygen Doxyfile`
- ▶ Output in `docs/html/index.html`

Public link:

<https://crypto-tii.github.io/pqc-engineering-ssec-23/>

Generating Technical Documentation (2/2)

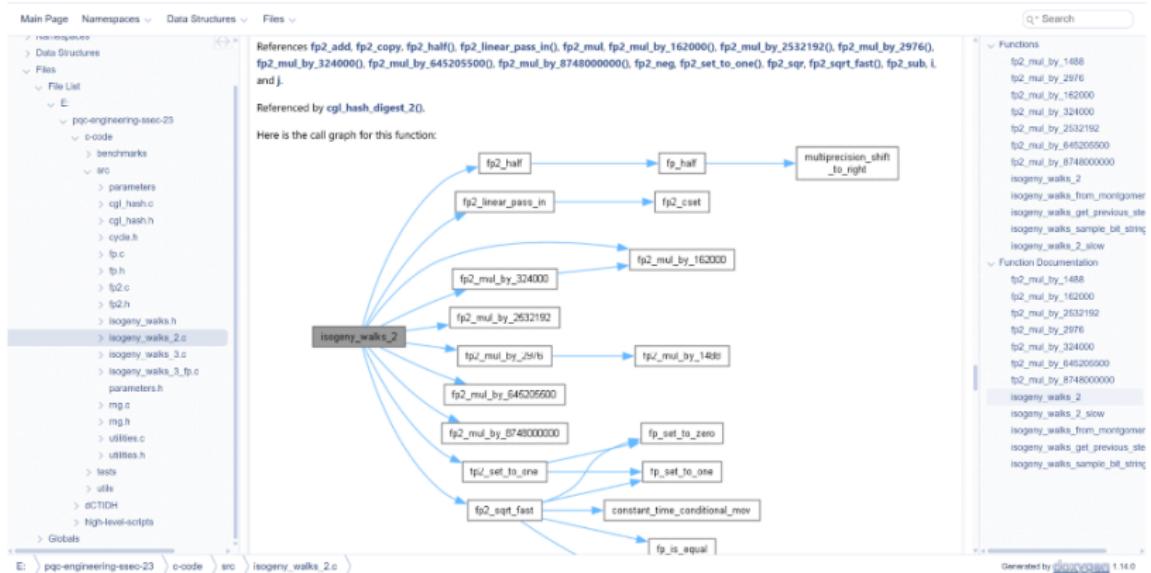


Figure 20: Technical documentation generated using Doxygen.

CI/CD Overview (1/3)

In order to show that our project can be integrated in a Real-World industrial environment, we provide a CI/CD pipeline.

- ▶ We use GitHub Actions for CI/CD.
- ▶ Pipeline includes **Build**, **Test**, **Benchmark**, and **Reporting** stages.
- ▶ YAML config:
`.github/workflows/cmake-multi-platform.yml`

CI/CD Overview (2/3)

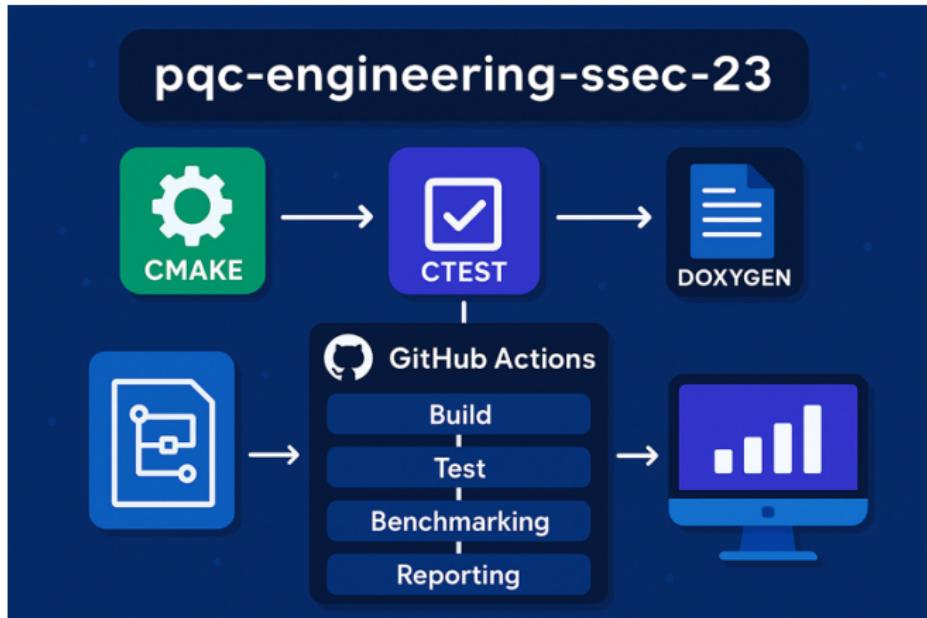


Figure 21: Designed CI/CD pipeline with **Build**, **Test**, **Benchmarking**, and **Reporting** stages.

CI/CD Overview (3/3)

cmake-multi-platform.yml

on: push

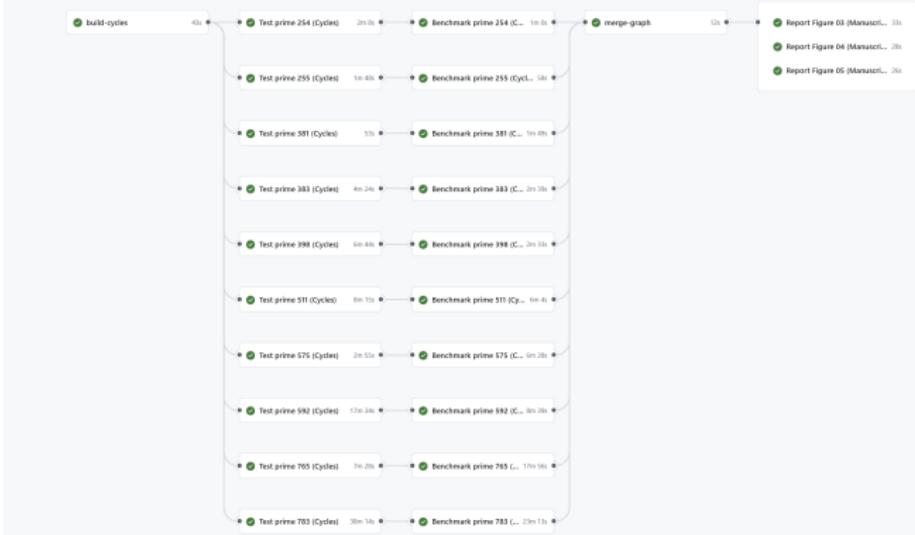


Figure 22: Pipeline in action, running with GitHub actions.

CI Stage: Build

- ▶ Triggers on push and pull request
- ▶ Any linux (Ubuntu example), Intel x86_64 CPU.
- ▶ Uses CMake caching for speed.

CI Stage: Test

- ▶ Runs unit and integration tests
- ▶ Stores artifacts for future analysis
- ▶ Automatic failure reports

CI Stage: Benchmark

- ▶ Executes performance benchmarks in both **CPU cycles** and **execution nanoseconds**.
- ▶ Benchmarking for every proposed prime.

CI Stage: Reporting (1/2)

- ▶ All the scripts used to reproduce our results reported in the manuscript are tested.
- ▶ The generated statistical data and the generated graphs are uploaded as public artifacts in our GitHub pipeline so they can be used freely.
- ▶ This allows collaborators, scientists, and anyone in general to reproduce, validate, and expand our research project.

CI Stage: Reporting (2/2)

Summary			
Job	Artifacts		
Run details	Name	Size	Digest
● build-cycles	benchmarks_sec	9.23 kB	sha256:71f1e590c550d6a40f0f03d17045202ce6cf564aef7a52d494088bfef...
● Test prime 254 (Cycles)	benchmarks_sec-p254-output	562 Bytes	sha256:fe1330ffba75e7ff00d1085125e10e4f13dbaa42f28ffcc5e731bb77...
● Test prime 181 (Cycles)	benchmarks_sec-p255-output	952 Bytes	sha256:e7e3446490139324f607759d679e243c8f717aefdf3720a1817355...
● Test prime 183 (Cycles)	benchmarks_sec-p381-output	981 Bytes	sha256:ee58798ef949c351d479795ac4a187fe0260e5493c26c1b67b2e02...
● Test prime 196 (Cycles)	benchmarks_sec-p383-output	562 Bytes	sha256:1baa0a52f07a72019822891e6d8327d1e2517a1c305e08040080...
● Test prime 511 (Cycles)	benchmarks_sec-p398-output	560 Bytes	sha256:11548783d00822a7446377a250801781d41350e275cc072e56176181...
● Test prime 575 (Cycles)	benchmarks_sec-p511-output	963 Bytes	sha256:03ab47372017520b1433865247188c75cc072e56176181f904...
● Benchmark prime 254 (Cycles)	benchmarks_sec-p575-output	566 Bytes	sha256:e7c52f89a6d3318d36608f7288320b46c386d41c761c38f52f...
● Benchmark prime 255 (Cycles)	benchmarks_sec-p592-output	970 Bytes	sha256:c8f29fbcd77159434793c0f08954ab5646e704384032c45f77015...
● Benchmark prime 381 (Cycles)	benchmarks_sec-p765-output	971 Bytes	sha256:1e0730553d48d56c7b0091e2290842086ab5a521cb3796b75e7038...
● Benchmark prime 383 (Cycles)	benchmarks_sec-p783-output	973 Bytes	sha256:17713c70e808d72232e7061c4951131ecac2d520adafab309334ab32a...
● merge-graph	cmake-build-release-cycles-x8664	1.92 MB	sha256:cb7ed5c17108031575d4832e0108795d451d2515c4075de20167...
● Report Figure 03 (Manuscript)	generated-figure-03	12.6 kB	sha256:10274e51a15080abac0bf0076bd421f172e790803802ca2777a0f...
● Report Figure 04 (Manuscript)	generated-figure-04	13.7 kB	sha256:a8b25a565f30f7163f487cb129212bb0345471b3b8473fc49558...
● Report Figure 05 (Manuscript)	generated-figure-05	29.4 kB	sha256:c841c9e20f78c6b729aae972242fe314704a7139f80e374e05...
● Usage	reproduce-results-code	7.14 kB	sha256:3fb197979b9d59e53a0826448762734e80294a5133a402074485...
● Workflow file			

Figure 23: Publicly available artifacts.

Docker Container

Simply execute

- ▶ docker pull
tiicrc/github-selfhosted-runner-pqc:latest
- ▶ docker images | grep pqc

Docker Container (2/2)

To mount, first locate your terminal at the artifact's root folder (*pqc-engineering-ssec-23*) and execute

- ▶ `docker run --rm -ti -v $PWD:/src -w /src
tiicrc/github-selfhosted-runner-pqc:latest bash`

After mounting, the terminal will change to

- ▶ `/src# <insert commands here>`

Industrial Readiness Proof-of-Concept

- ▶ Simulates real deployment environments
- ▶ Documented logs, errors, and benchmarking outputs

Additional Resources: CPU benchmarking

- ▶ Included details on how to:
 - ▶ Turn off turbo-boost.
 - ▶ Assembly instructions used in our benchmarking.
- ▶ Automated benchmarking scripts under
`high-level-scripts/benchmark_02_20250408.sh`

License and Contributions

- ▶ Open-source under Apache License.
- ▶ License guidelines in LICENSE file.
- ▶ Issues and PRs welcome!

About the Authors

- ▶ Jesús-Javier Chi-Domínguez,
- ▶ Eduardo Ochoa-Jiménez,
- ▶ Ricardo-Neftalí Pontaza-Rodas.

Thank you

LET US WALK ON THE 3-ISogeny GRAPH: EFFICIENT, FAST, AND SIMPLE



THANK YOU

Thank you

Thank you!