## A ON RELIABLE PERFORMANCE MEASUREMENT

### A.1 Background

The execution time of a program can be affected by a myriad of factors, such as, the state of cache memory [Alameldeen and Wood 2003]; context switches [Pusukuri et al. 2012]; memory layout and OS environment variables [Curtsinger and Berger 2013; Mytkowicz et al. 2009]. Even when a platform is rebooted between runs there can be significant variation in benchmark run times [Kalibera et al. 2005a]. Furthermore, measurements of the benchmark operation can be subject to inherent and external random changes to the benchmark operation, as well as the system state prior to execution [Kalibera et al. 2005b; Kalibera and Jones 2013]. In addition, sensor readings themselves can drift over time [Bokhari et al. 2019]. Consequently, all these factors together result in measurement noise, which can affect both the sensitivity and specificity of performance measurements. One recent work, which surveyed and compared validation approaches in the context of energy consumption optimization [Bokhari et al. 2020], found that a number of measurement approaches failed to mitigate the effects of measurement noise. Only the proposed approach R3-validation was not misled: it exercised the tests in a rotated-round-robin fashion, while accommodating the necessary regular restarting and recharging of the test platform.

### A.2 Cost-Function Evaluation

Recall that CryptOpt generates solutions, mutates them, and measures their respective performance. Measuring this performance on physical machines is inherently noisy. To lower the effect of noise and enable more stable and fair comparisons, we base our measurement on R3-validation, which exercises tests in a rotated-round-robin fashion. This approach was used in previous work [Bokhari et al. 2020] in the context of energy consumption in order to mitigate the effects of measurement noise.

In our work, we adapt R3-validation in two ways. First, we forgo the restart of the computer as we do not observe any measurement drift over time. Second, we perform a random scheduling of program variants for performance measurement instead of strictly following a given order of measurements.

**Measuring Performance.** Algorithm 3 presents our performance-measurement routine. We now explain the rationale behind this from the inside out (inside the loop of the algorithm to the outside of the measure function).

Cycle counters are integers with a granularity of at best 1. Recall that CryptOpt can optimize functions of arbitrary size, ranging in our experiments from fewer than 100 to around 2000 instructions. That means that the accuracy of the fixed-granularity counter relative to the length-varying function changes. In other words, the same "stopwatch" is not appropriate to time an Olympic sprinter and a marathon runner.

The solution to that is to measure multiple executions of the same function instead of just one. We call multiple executions of the same function a *batch*. The batch size *bs* specifies how many executions of the same function are being measured with one measurement. It follows logically that small functions should have a bigger *bs* than long functions in order to get similar cycle accuracy.

Hence, we came up with the idea of the *cyclegoal*. That is, we want the measured cycles count for one batch across all functions to be in the order of $cyclegoal = 10\,000$ measured cycles. The reason for this particular number is that we then have four to five digits of accuracy. CryptOpt then scales the value for *bs* up or down by comparing the measured cycles against the *cyclegoal*.

This technique also mitigates another inherent problem that arises when measuring performance in cycles on hardware: different hardware platforms interpret "cycles" differently, and additional challenges arise due to different types of boosting. However, because CryptOpt adjusts *bs* in each

**Algorithm 3:** Execution Time Measurement

```
input  : A pointer to code A,
         B pointer to code B,
         bs size of a measurement batch,
         nob number of batches to measure
output : PA performance of code A in cycles,
         PB performance of code B in cycles
function measure(A, B, bs, nob)
begin
    /* initialize cycle lists */
    $cycles_A \leftarrow$ empty list
    $cycles_B \leftarrow$ empty list
    repeat
        /* f points to either A or B */
        $f \leftarrow \text{randomSelect}(A, B)$
        /* run bs times and get elapsed cycles */
        $cycles \leftarrow \text{countCyclesForNRuns}(bs, f)$
        /* append elapsed cycles to list A/B resp. */
        $\text{append}(cycles_f, cycles)$
    until both have been measured nob times;
    $PA \leftarrow \text{median}(cycles_A)$
    $PB \leftarrow \text{median}(cycles_B)$
    return PA, PB
end
```

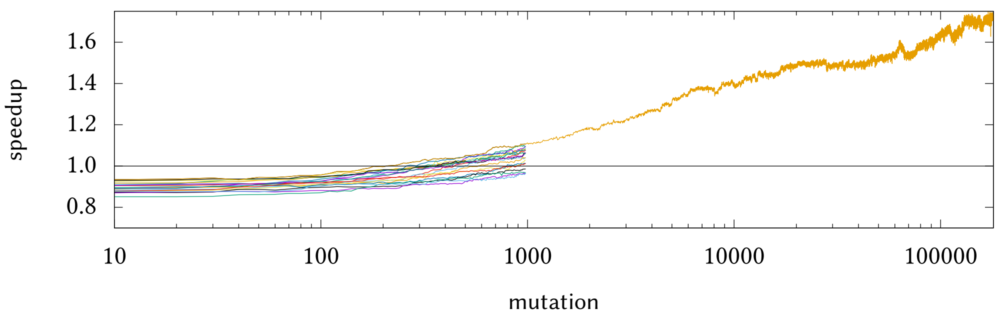
evaluation of the mutation (i.e. before each call to "measure") and it does so on each platform independently, we get comparable accuracy for all functions across platforms.

Inherent with measuring time on hardware is measurement noise, which we as developers can hardly control. This noise can be due to the OS's interrupt event handlers or process scheduling. It can also be due to temperature rises the hardware limits itself. The technique above mitigates the problems having a bad "stopwatch" but at the same time increases the captured noise. Similar to the R3-validation, we mitigate this problem by simply measuring each batch multiple times, called *numbers of batches* or *nob* for short. That means that we have *nob* measurements for the same function. We take the median as the performance measurement for a function to drop outlier measurements. (Using the minimum rather than median did not have statistically significant effects – neither positive nor negative.) Empirically, setting $nob = 31$ works well on our experimental platforms.

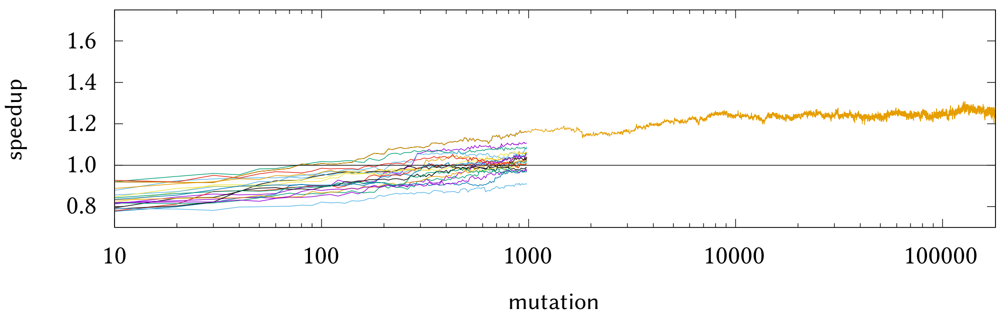
Complex processors nowadays can cause biases to one or another function by their speculative execution, prediction behavior and caching, just to name a few reasons. The R3-validation also mitigates this issue by using a random sequence of the functions to measure. We do this by randomly selecting either function to measure a batch of.

## A.3 AssemblyLine

CryptOpt uses AssemblyLine [0xADE1A1DE 2022] to assemble the generated initial and mutated code into memory positions *A* and *B*. AssemblyLine is a lightweight in-memory assembler for



(a) SIKEp434–square on i7 10G



(b) secp256k1–square on i7 11G

Fig. 9. CryptOpt optimization progress of different method on two platforms. Each line traces an optimization run over time (X-axis, log-scale), showing the relative performance gain over Clang. Bet-and-Run used for 20 runs with a budget of 1 000 mutations each. The best-performing candidate continues for another 180 000 mutations. Progress over time is roughly logarithmic in the number of mutations, achieving a performance gain of 52–86%. We generate one data point every ten mutations.

x86-64 assembly instructions. It takes an input string of instructions, assembles them into machine code and returns a pointer to the executable code. We use AssemblyLine to eliminate the overhead of writing to the file system and invoking a typical compiler tool chain for every evaluation. Additionally, CryptOpt uses AssemblyLine to assemble the code to the start of a memory page. This reduces noise in measurements because it always assembles aligned code for all functions, which, in turn, reduces memory-biased performance impacts [Mytkowicz et al. 2009]. Once assembled, CryptOpt calls the measure function with pointers to that code and values for batch size *bs* and number of batches *nob*.

## B OPTIMIZATION PROGRESS

The first evaluation question we answer is how the optimization progresses over time. Figure 9 shows examples of optimization runs on i7 10G and i7 11G, targeting the field-multiplication operation corresponding to the prime field for SIKEp434 and secp256k1 respectively.

The figures show performance over time, where performance is measured as gain over the baseline Clang-compiled code, while optimization time is measured in mutations (shown in log scale).

The figure shows the Bet-and-Run strategy in action. The optimization starts with 20 runs, each with a budget of 1 000 mutations. CryptOpt then selects the best-performing and continues optimizing it for a further 180 000 mutations. As the figure shows, in all runs, optimization progress is roughly logarithmic in the number of mutations.

A notable difference between the two optimizations is the overall performance gain. On the i7 11G platform the optimization converges rather quickly, whereas on i7 10G we see monotonic improvement until the very end. Finally, we note that these variations and performance gain are dependent on field, method and machine; here we see a gain 52–86%.

## C PLATFORM-SPECIFIC OPTIMIZATION

CryptOpt optimizes the execution time of the function on the platform it executes on. Because different platforms have different hardware components, the fastest code on one platform is not necessarily the fastest on another. Thus, in this section we ask whether CryptOpt overfits to the platform it executes on, as opposed to universally optimizing for all platforms. That is, we test how *native optimization*, where the code executes on the same platform it was optimized on, compares with *cross optimization*, where code is optimized on one platform and executes on another.

To test this, we first optimize each of the functions on each of the platforms. We then measure the performance of the produced code from all platforms on all platforms (as per evaluation described in Appendix A). Table 6 shows the results for the operations multiply and square on all nine fields we tested. Rows correspond to the platform the code was optimized on and columns to the platform the code is executed on. The value in a cell shows the ratio between the execution time of the cross-optimized code and that of the native code. An example value of 1.10 indicates that the cross-platform code (from the "row" platform) needs 1.10 times as many cycles as the native code (from the "column" platform). In other words, a blue cell means that code from the column architecture outperforms code from the row architecture, and the number indicates by how much.

The additional column G.M. shows the geometric mean of the numbers in the row. This gives a measure of "how platform-specific" one optimization is: A value larger than one (blue) in column G.M. in the row for platform X indicates that on average, code from other platforms outperform the code from X, if ran on platforms other than X. In turn, a value smaller than one means that the code from platform X tends to outperform platform-specific code on other platforms. Please note that a "large number" can arise for two reasons: (a) The platform-specific code is very bad (thus easy to outperform), or (b) the generated code is very generic and can thus compete with the platform-specific ones. To see which is applicable per platform, one needs to see how well the code compares to off-the-shelf compilers. Note that this is also not a universal measure, as off-the-shelf compilers generate different code per platform. Alternatively, running the same code on all platforms would also not give a fair baseline, as compilers would not be enabled to generate platform-specific code, whereas CryptOpt would be.

As can be seen in Table 6 for Curve25519, optimizing and running on the same architecture tends to outperform cross-architecture optimizations (which would mean the 8-by-8 is mostly blue, and the G.M. column is fully blue, too). However, there are cases where cross-architectural optimizations are better. In particular it appears that optimization of SIKEp434 functions on AMD is not ideal, and the results underperform code optimized on Intel processors. (Still, faster than compiling with off-the-shelf compilers).

Another effect that the table highlights is that optimization overfits for processor families, for some families more than others. In particular, Intel 6th and 10th generations show little differences in the respective inter-platform performance. For example, the related 3x3 sub-matrices for Curve25519 in Table 6 have values in the small interval [0.99, 1.05] (mul), [0.90, 1.05] (square);

Table 6. Optimization results. We show the relative improvements in % for the multiplication (top) and squaring (bottom) operations; time savings are marked in blue. First, to observe hardware-specific optimization, the 8-by-8 matrix shows the performance the optimized operation that have been optimized on one machine and then run on another. The subsequent two rows (Clang/GCC) then show the time savings of our optimized operations over off-the-shelf-compilers. Lastly, "Final" shows the time savings of our best-performing implementation over the best-performing compiler-generated version.

Curve25519

| run on | 1900X | 5800X | 5950X | i7 6G | i7 10G | i9 10G | i7 11G | i9 12G | G.M. |
|---|---|---|---|---|---|---|---|---|---|
| opt on | | | | | | | | | |
| 1900X | | 1.09 | 1.07 | 1.12 | 1.13 | 1.24 | 1.19 | 1.21 | 1.13 |
| 5800X | 1.11 | | 0.98 | 1.14 | 1.15 | 1.15 | 1.15 | 1.20 | 1.11 |
| 5950X | 1.12 | 1.01 | | 1.12 | 1.16 | 1.18 | 1.14 | 1.22 | 1.12 |
| i7 6G | 1.06 | 1.16 | 1.13 | | 1.01 | 1.00 | 1.05 | 1.15 | 1.07 |
| i7 10G | 1.07 | 1.12 | 1.10 | 1.05 | | 1.00 | 1.09 | 1.18 | 1.07 |
| i9 10G | 1.07 | 1.13 | 1.17 | 0.99 | 1.00 | | 1.14 | 1.29 | 1.09 |
| i7 11G | 1.08 | 1.11 | 1.09 | 1.10 | 1.20 | 1.12 | | 1.20 | 1.11 |
| i9 12G | 1.03 | 1.03 | 1.01 | 1.04 | 1.05 | 1.12 | 1.04 | | 1.04 |
| Clang | 1.14 | 1.16 | 1.13 | 1.19 | 1.20 | 1.19 | 1.22 | 1.27 | 1.19 |
| GCC | 1.04 | 1.12 | 1.10 | 1.13 | 1.14 | 1.15 | 1.18 | 1.28 | 1.14 |
| Final | 1.04 | 1.12 | 1.12 | 1.15 | 1.14 | 1.15 | 1.18 | 1.27 | 1.14 |
| | | | | | | | | | |
| 1900X | | 1.09 | 1.11 | 1.15 | 1.04 | 1.20 | 1.20 | 1.31 | 1.14 |
| 5800X | 1.10 | | 1.00 | 1.13 | 1.04 | 1.21 | 1.18 | 1.15 | 1.10 |
| 5950X | 1.16 | 1.00 | | 1.13 | 1.03 | 1.15 | 1.21 | 1.18 | 1.10 |
| i7 6G | 1.25 | 1.17 | 1.18 | | 0.91 | 1.02 | 1.19 | 1.30 | 1.12 |
| i7 10G | 1.23 | 1.17 | 1.18 | 1.04 | | 1.05 | 1.15 | 1.37 | 1.14 |
| i9 10G | 1.20 | 1.17 | 1.15 | 0.99 | 0.90 | | 1.11 | 2.39 | 1.18 |
| i7 11G | 1.18 | 1.10 | 1.09 | 1.06 | 0.97 | 1.10 | | 1.40 | 1.11 |
| i9 12G | 1.06 | 1.05 | 1.04 | 1.03 | 0.94 | 1.05 | 1.09 | | 1.03 |
| Clang | 1.18 | 1.12 | 1.11 | 1.15 | 1.04 | 1.16 | 1.14 | 1.27 | 1.14 |
| GCC | 1.15 | 1.14 | 1.13 | 1.18 | 1.08 | 1.21 | 1.26 | 1.31 | 1.18 |
| Final | 1.15 | 1.12 | 1.11 | 1.16 | 1.16 | 1.16 | 1.14 | 1.27 | 1.16 |

P-224

| run on | 1900X | 5800X | 5950X | i7 6G | i7 10G | i9 10G | i7 11G | i9 12G | G.M. |
|---|---|---|---|---|---|---|---|---|---|
| opt on | | | | | | | | | |
| 1900X | | 1.01 | 1.02 | 1.12 | 1.12 | 1.06 | 1.06 | 1.09 | 1.06 |
| 5800X | 1.06 | | 1.01 | 1.16 | 1.16 | 1.10 | 1.11 | 1.20 | 1.10 |
| 5950X | 1.08 | 1.00 | | 1.22 | 1.19 | 1.12 | 1.20 | 1.25 | 1.13 |
| i7 6G | 1.06 | 1.04 | 1.03 | | 1.01 | 0.95 | 1.04 | 1.05 | 1.02 |
| i7 10G | 1.03 | 1.01 | 1.00 | 1.00 | | 0.95 | 1.04 | 1.04 | 1.01 |
| i9 10G | 1.10 | 1.04 | 1.04 | 1.07 | 1.06 | | 1.34 | 1.84 | 1.16 |
| i7 11G | 1.03 | 1.01 | 1.01 | 1.06 | 1.09 | 1.01 | | 1.05 | 1.03 |
| i9 12G | 1.04 | 1.02 | 1.02 | 1.05 | 1.08 | 1.00 | 1.04 | | 1.03 |
| Clang | 1.28 | 1.26 | 1.26 | 1.35 | 1.35 | 1.28 | 1.26 | 1.43 | 1.31 |
| GCC | 1.71 | 1.56 | 1.56 | 2.06 | 2.06 | 1.96 | 1.91 | 2.22 | 1.87 |
| Final | 1.28 | 1.27 | 1.26 | 1.36 | 1.35 | 1.35 | 1.26 | 1.43 | 1.32 |
| | | | | | | | | | |
| 1900X | | 1.02 | 1.08 | 1.03 | 1.09 | 1.11 | 1.16 | 1.20 | 1.08 |
| 5800X | 1.15 | | 1.04 | 1.06 | 1.10 | 1.14 | 1.18 | 1.25 | 1.11 |
| 5950X | 1.10 | 0.96 | | 1.04 | 1.11 | 1.12 | 1.14 | 1.78 | 1.14 |
| i7 6G | 1.13 | 1.08 | 1.14 | | 1.02 | 1.12 | 1.13 | 1.12 | 1.09 |
| i7 10G | 1.11 | 1.03 | 1.09 | 0.97 | | 1.04 | 1.11 | 1.08 | 1.05 |
| i9 10G | 1.09 | 1.04 | 1.09 | 0.93 | 0.96 | | 1.08 | 1.05 | 1.03 |
| i7 11G | 1.09 | 1.00 | 1.06 | 1.01 | 1.04 | 1.08 | | 1.06 | 1.04 |
| i9 12G | 1.07 | 1.03 | 1.08 | 0.99 | 1.02 | 1.08 | 1.06 | | 1.04 |
| Clang | 1.23 | 1.15 | 1.21 | 1.20 | 1.24 | 1.30 | 1.22 | 1.37 | 1.24 |
| GCC | 1.77 | 1.53 | 1.61 | 1.89 | 1.95 | 2.02 | 1.95 | 2.08 | 1.84 |
| Final | 1.23 | 1.20 | 1.21 | 1.29 | 1.29 | 1.30 | 1.22 | 1.37 | 1.26 |

P-256

| run on | 1900X | 5800X | 5950X | i7 6G | i7 10G | i9 10G | i7 11G | i9 12G | G.M. |
|---|---|---|---|---|---|---|---|---|---|
| opt on | | | | | | | | | |
| 1900X | | 1.02 | 0.99 | 1.11 | 1.11 | 1.14 | 1.13 | 0.67 | 1.01 |
| 5800X | 1.07 | | 0.97 | 1.15 | 1.12 | 1.19 | 1.16 | 0.70 | 1.03 |
| 5950X | 1.08 | 1.01 | | 1.15 | 1.12 | 1.18 | 1.15 | 0.68 | 1.03 |
| i7 6G | 1.05 | 1.01 | 0.99 | | 0.98 | 1.03 | 1.05 | 1.03 | 1.02 |
| i7 10G | 1.06 | 1.00 | 0.98 | 1.03 | | 1.08 | 1.07 | 0.64 | 0.97 |
| i9 10G | 1.04 | 0.98 | 0.96 | 0.97 | 0.95 | | 1.04 | 0.61 | 0.93 |
| i7 11G | 1.04 | 0.97 | 0.95 | 1.06 | 1.03 | 1.13 | | 0.61 | 0.96 |
| i9 12G | 1.03 | 1.01 | 0.98 | 1.04 | 1.02 | 1.08 | 1.09 | | 1.03 |
| Clang | 1.32 | 1.30 | 1.27 | 1.38 | 1.34 | 1.43 | 1.33 | 0.89 | 1.27 |
| GCC | 1.72 | 1.61 | 1.56 | 2.08 | 2.02 | 2.14 | 2.01 | 1.36 | 1.79 |
| Final | 1.32 | 1.34 | 1.34 | 1.42 | 1.42 | 1.43 | 1.33 | 1.47 | 1.38 |
| | | | | | | | | | |
| 1900X | | 1.08 | 1.06 | 1.02 | 1.12 | 1.08 | 1.15 | 1.11 | 1.08 |
| 5800X | 1.06 | | 1.00 | 1.00 | 1.12 | 1.06 | 1.10 | 1.80 | 1.12 |
| 5950X | 1.06 | 1.01 | | 1.05 | 1.11 | 1.08 | 1.12 | 1.09 | 1.06 |
| i7 6G | 1.09 | 1.08 | 1.05 | | 1.04 | 1.02 | 1.10 | 1.06 | 1.06 |
| i7 10G | 1.04 | 1.04 | 1.02 | 0.91 | | 0.97 | 1.07 | 1.04 | 1.01 |
| i9 10G | 1.11 | 1.06 | 1.03 | 0.96 | 1.06 | | 1.15 | 1.06 | 1.05 |
| i7 11G | 1.06 | 1.04 | 1.02 | 0.97 | 1.08 | 1.03 | | 1.76 | 1.10 |
| i9 12G | 1.06 | 1.09 | 1.06 | 1.00 | 1.09 | 1.06 | 1.07 | | 1.05 |
| Clang | 1.28 | 1.31 | 1.28 | 1.23 | 1.35 | 1.32 | 1.26 | 1.43 | 1.30 |
| GCC | 1.75 | 1.65 | 1.61 | 1.87 | 2.04 | 1.98 | 1.98 | 1.98 | 1.85 |
| Final | 1.28 | 1.31 | 1.28 | 1.35 | 1.35 | 1.36 | 1.26 | 1.43 | 1.32 |

P-384

| run on | 1900X | 5800X | 5950X | i7 6G | i7 10G | i9 10G | i7 11G | i9 12G | G.M. |
|---|---|---|---|---|---|---|---|---|---|
| opt on | | | | | | | | | |
| 1900X | | 0.96 | 0.96 | 1.59 | 1.60 | 1.64 | 1.02 | 0.97 | 1.18 |
| 5800X | 1.25 | | 1.00 | 1.82 | 1.86 | 1.88 | 1.10 | 1.08 | 1.32 |
| 5950X | 1.16 | 1.00 | | 1.85 | 1.83 | 1.87 | 1.12 | 1.06 | 1.31 |
| i7 6G | 0.76 | 0.71 | 0.71 | | 0.98 | 1.03 | 0.67 | 0.60 | 0.79 |
| i7 10G | 0.76 | 0.71 | 0.70 | 0.99 | | 1.02 | 0.67 | 0.60 | 0.79 |
| i9 10G | 0.78 | 0.71 | 0.71 | 0.97 | 0.98 | | 0.67 | 0.61 | 0.79 |
| i7 11G | 1.04 | 0.99 | 0.98 | 1.68 | 1.68 | 1.74 | | 0.99 | 1.22 |
| i9 12G | 1.13 | 1.05 | 1.05 | 1.79 | 1.82 | 1.84 | 1.05 | | 1.29 |
| Clang | 0.98 | 0.94 | 0.94 | 1.52 | 1.53 | 1.56 | 0.85 | 0.90 | 1.12 |
| GCC | 1.33 | 1.29 | 1.28 | 2.28 | 2.30 | 2.35 | 1.56 | 1.34 | 1.66 |
| Final | 1.29 | 1.34 | 1.35 | 1.56 | 1.56 | 1.56 | 1.27 | 1.49 | 1.42 |
| | | | | | | | | | |
| 1900X | | 1.02 | 1.00 | 1.76 | 1.77 | 1.67 | 1.08 | 0.98 | 1.24 |
| 5800X | 1.21 | | 0.98 | 1.74 | 1.70 | 1.63 | 1.11 | 0.97 | 1.26 |
| 5950X | 1.05 | 1.01 | | 1.78 | 1.75 | 1.68 | 1.10 | 0.98 | 1.25 |
| i7 6G | 0.74 | 0.71 | 0.70 | | 0.99 | 0.94 | 0.71 | 0.57 | 0.78 |
| i7 10G | 0.74 | 0.73 | 0.71 | 0.99 | | 0.94 | 0.70 | 0.56 | 0.78 |
| i9 10G | 0.77 | 0.73 | 0.72 | 1.06 | 1.05 | | 0.73 | 0.60 | 0.82 |
| i7 11G | 1.02 | 1.00 | 0.98 | 1.62 | 1.62 | 1.53 | | 0.92 | 1.18 |
| i9 12G | 1.15 | 1.10 | 1.08 | 1.92 | 1.92 | 1.83 | 1.15 | | 1.34 |
| Clang | 0.98 | 0.95 | 0.93 | 1.51 | 1.49 | 1.42 | 0.84 | 0.81 | 1.08 |
| GCC | 1.36 | 1.16 | 1.14 | 2.38 | 2.36 | 2.25 | 1.62 | 1.19 | 1.60 |
| Final | 1.32 | 1.33 | 1.34 | 1.53 | 1.52 | 1.52 | 1.21 | 1.44 | 1.40 |

SIKEp434

| run on | 1900X | 5800X | 5950X | i7 6G | i7 10G | i9 10G | i7 11G | i9 12G | G.M. |
|---|---|---|---|---|---|---|---|---|---|
| opt on | | | | | | | | | |
| 1900X | | 0.99 | 0.95 | 1.95 | 1.83 | 1.71 | 1.27 | 1.01 | 1.28 |
| 5800X | 1.32 | | 0.96 | 2.02 | 1.91 | 1.79 | 1.37 | 1.04 | 1.37 |
| 5950X | 1.39 | 1.03 | | 2.04 | 1.92 | 1.81 | 1.40 | 1.05 | 1.40 |
| i7 6G | 0.70 | 0.66 | 0.63 | | 0.95 | 0.89 | 0.80 | 0.60 | 0.77 |
| i7 10G | 0.72 | 0.67 | 0.64 | 1.04 | | 0.94 | 0.80 | 0.60 | 0.79 |
| i9 10G | 0.72 | 0.69 | 0.66 | 1.12 | 1.06 | | 0.88 | 0.94 | 0.87 |
| i7 11G | 0.84 | 0.83 | 0.80 | 1.42 | 1.36 | 1.27 | | 0.77 | 1.00 |
| i9 12G | 1.36 | 1.06 | 1.02 | 1.99 | 1.91 | 1.78 | 1.37 | | 1.39 |
| Clang | 1.15 | 1.11 | 1.06 | 1.84 | 1.75 | 1.65 | 1.01 | 1.12 | 1.30 |
| GCC | 1.40 | 1.16 | 1.12 | 2.55 | 2.43 | 2.28 | 2.08 | 1.32 | 1.70 |
| Final | 1.64 | 1.68 | 1.68 | 1.84 | 1.84 | 1.85 | 1.27 | 1.87 | 1.70 |
| | | | | | | | | | |
| 1900X | | 0.99 | 0.98 | 1.88 | 1.79 | 1.93 | 1.36 | 1.02 | 1.31 |
| 5800X | 1.17 | | 0.98 | 1.88 | 1.79 | 1.93 | 1.39 | 1.00 | 1.34 |
| 5950X | 1.21 | 1.01 | | 1.89 | 1.80 | 1.96 | 1.46 | 1.06 | 1.37 |
| i7 6G | 0.67 | 0.66 | 0.65 | | 0.95 | 1.01 | 0.82 | 0.57 | 0.77 |
| i7 10G | 0.70 | 0.70 | 0.68 | 1.06 | | 1.09 | 0.89 | 0.62 | 0.82 |
| i9 10G | 0.68 | 0.68 | 0.66 | 0.98 | 0.93 | | 0.83 | 0.59 | 0.78 |
| i7 11G | 0.85 | 0.80 | 0.79 | 1.32 | 1.26 | 1.35 | | 0.77 | 0.99 |
| i9 12G | 1.30 | 1.03 | 1.02 | 1.93 | 1.82 | 1.96 | 1.47 | | 1.39 |
| Clang | 1.10 | 1.10 | 1.08 | 1.76 | 1.68 | 1.82 | 1.04 | 1.06 | 1.29 |
| GCC | 1.38 | 1.45 | 1.43 | 2.49 | 2.37 | 2.56 | 2.25 | 1.28 | 1.83 |
| Final | 1.65 | 1.67 | 1.67 | 1.80 | 1.81 | 1.82 | 1.28 | 1.85 | 1.68 |

Curve448

| run on | 1900X | 5800X | 5950X | i7 6G | i7 10G | i9 10G | i7 11G | i9 12G | G.M. |
|---|---|---|---|---|---|---|---|---|---|
| opt on | | | | | | | | | |
| 1900X | | 1.14 | 1.12 | 1.40 | 1.34 | 1.44 | 1.25 | 1.04 | 1.21 |
| 5800X | 0.85 | | 1.00 | 1.24 | 1.19 | 1.23 | 1.03 | 1.07 | 1.07 |
| 5950X | 0.86 | 1.01 | | 1.20 | 1.16 | 1.26 | 1.02 | 1.05 | 1.06 |
| i7 6G | 0.78 | 0.97 | 0.95 | | 0.97 | 1.03 | 0.94 | 0.89 | 0.94 |
| i7 10G | 0.81 | 0.96 | 0.94 | 1.03 | | 1.09 | 0.94 | 0.95 | 0.96 |
| i9 10G | 0.78 | 0.94 | 0.92 | 0.99 | 0.95 | | 0.89 | 0.91 | 0.92 |
| i7 11G | 0.90 | 1.08 | 1.06 | 1.25 | 1.18 | 1.25 | | 1.02 | 1.08 |
| i9 12G | 0.91 | 1.17 | 1.15 | 1.44 | 1.36 | 1.46 | 1.18 | | 1.19 |
| Clang | 0.80 | 0.92 | 0.91 | 1.22 | 1.18 | 1.25 | 1.03 | 0.98 | 1.02 |
| GCC | 0.74 | 0.90 | 0.88 | 1.10 | 1.07 | 1.13 | 0.92 | 0.91 | 0.95 |
| Final | 0.95 | 0.96 | 0.96 | 1.12 | 1.13 | 1.13 | 1.03 | 1.02 | 1.04 |
| | | | | | | | | | |
| 1900X | | 1.05 | 1.02 | 1.24 | 1.19 | 1.22 | 1.07 | 1.00 | 1.10 |
| 5800X | 1.00 | | 0.97 | 1.29 | 1.26 | 1.26 | 1.10 | 1.06 | 1.11 |
| 5950X | 1.05 | 1.02 | | 1.35 | 1.27 | 1.31 | 1.22 | 1.09 | 1.16 |
| i7 6G | 0.95 | 1.05 | 1.01 | | 0.97 | 0.99 | 1.07 | 1.50 | 1.06 |
| i7 10G | 0.96 | 1.01 | 0.99 | 1.04 | | 1.00 | 1.00 | 0.97 | 1.00 |
| i9 10G | 0.98 | 1.04 | 1.00 | 1.06 | 0.98 | | 1.02 | 0.98 | 1.01 |
| i7 11G | 1.01 | 1.05 | 1.02 | 1.23 | 1.18 | 1.23 | | 1.63 | 1.15 |
| i9 12G | 1.05 | 1.12 | 1.09 | 1.33 | 1.29 | 1.32 | 1.24 | | 1.17 |
| Clang | 0.88 | 0.95 | 0.92 | 1.10 | 1.07 | 1.09 | 1.04 | 0.93 | 1.00 |
| GCC | 0.90 | 0.96 | 0.93 | 1.11 | 1.07 | 1.10 | 0.99 | 0.91 | 0.99 |
| Final | 0.93 | 0.95 | 0.95 | 1.10 | 1.10 | 1.11 | 0.99 | 0.94 | 1.01 |

P-521

| run on | 1900X | 5800X | 5950X | i7 6G | i7 10G | i9 10G | i7 11G | i9 12G | G.M. |
|---|---|---|---|---|---|---|---|---|---|
| opt on | | | | | | | | | |
| 1900X | | 1.05 | 1.07 | 1.32 | 1.31 | 1.31 | 1.15 | 1.08 | 1.15 |
| 5800X | 0.94 | | 1.01 | 1.26 | 1.26 | 1.25 | 1.15 | 1.06 | 1.11 |
| 5950X | 0.94 | 0.99 | | 1.29 | 1.27 | 1.31 | 1.14 | 1.59 | 1.18 |
| i7 6G | 0.86 | 0.96 | 0.99 | | 0.99 | 0.99 | 0.93 | 1.00 | 0.97 |
| i7 10G | 0.87 | 0.97 | 0.97 | 1.01 | | 1.03 | 0.91 | 0.98 | 0.97 |
| i9 10G | 0.87 | 0.97 | 0.99 | 1.00 | 1.04 | | 0.91 | 1.51 | 1.02 |
| i7 11G | 0.93 | 1.02 | 1.04 | 1.26 | 1.20 | 1.21 | | 0.98 | 1.07 |
| i9 12G | 0.98 | 1.08 | 1.10 | 1.38 | 1.37 | 1.36 | 1.22 | | 1.18 |
| Clang | 0.98 | 1.07 | 1.08 | 1.38 | 1.37 | 1.37 | 1.20 | 1.19 | 1.20 |
| GCC | 0.89 | 1.03 | 1.04 | 1.18 | 1.18 | 1.18 | 1.03 | 1.03 | 1.06 |
| Final | 1.04 | 1.06 | 1.08 | 1.18 | 1.18 | 1.19 | 1.13 | 1.05 | 1.11 |
| | | | | | | | | | |
| 1900X | | 1.02 | 1.05 | 1.18 | 1.12 | 1.13 | 1.07 | 1.16 | 1.09 |
| 5800X | 1.10 | | 1.03 | 1.28 | 1.20 | 1.23 | 1.10 | 1.15 | 1.13 |
| 5950X | 1.07 | 0.98 | | 1.26 | 1.17 | 1.19 | 1.08 | 1.12 | 1.10 |
| i7 6G | 1.03 | 1.01 | 1.04 | | 0.96 | 0.97 | 1.02 | 1.89 | 1.09 |
| i7 10G | 1.09 | 1.05 | 1.11 | 1.04 | | 1.21 | 1.06 | 1.23 | 1.10 |
| i9 10G | 1.08 | 1.07 | 1.10 | 0.98 | 0.94 | | 1.05 | 1.16 | 1.05 |
| i7 11G | 1.08 | 1.06 | 1.08 | 1.17 | 1.16 | 1.13 | | 1.14 | 1.10 |
| i9 12G | 1.10 | 1.04 | 1.05 | 1.19 | 1.14 | 1.15 | 1.10 | | 1.09 |
| Clang | 1.10 | 1.13 | 1.15 | 1.38 | 1.32 | 1.34 | 1.39 | 1.21 | 1.25 |
| GCC | 1.10 | 1.04 | 1.07 | 1.15 | 1.10 | 1.12 | 1.06 | 1.21 | 1.11 |
| Final | 1.10 | 1.07 | 1.07 | 1.17 | 1.17 | 1.15 | 1.06 | 1.21 | 1.12 |

Poly1305

| run on | 1900X | 5800X | 5950X | i7 6G | i7 10G | i9 10G | i7 11G | i9 12G | G.M. |
|---|---|---|---|---|---|---|---|---|---|
| opt on | | | | | | | | | |
| 1900X | | 1.09 | 1.14 | 1.12 | 1.11 | 1.10 | 1.10 | 1.15 | 1.10 |
| 5800X | 1.23 | | 1.10 | 1.11 | 1.10 | 1.09 | 1.12 | 1.17 | 1.11 |
| 5950X | 1.14 | 0.96 | | 1.13 | 1.18 | 1.10 | 1.09 | 1.08 | 1.08 |
| i7 6G | 1.26 | 1.15 | 1.24 | | 1.00 | 0.99 | 1.14 | 1.35 | 1.13 |
| i7 10G | 1.25 | 1.09 | 1.16 | 1.01 | | 0.99 | 1.05 | 1.37 | 1.11 |
| i9 10G | 1.34 | 1.12 | 1.18 | 1.01 | 1.01 | | 1.06 | 1.29 | 1.12 |
| i7 11G | 1.24 | 1.09 | 1.15 | 1.07 | 1.06 | 1.23 | | 1.35 | 1.14 |
| i9 12G | 1.25 | 1.09 | 1.15 | 1.13 | 1.06 | 1.05 | 1.09 | | 1.10 |
| Clang | 1.18 | 1.05 | 1.09 | 1.10 | 1.09 | 1.07 | 1.07 | 1.20 | 1.10 |
| GCC | 1.19 | 1.09 | 1.14 | 1.15 | 1.15 | 1.14 | 1.12 | 1.24 | 1.15 |
| Final | 1.18 | 1.09 | 1.09 | 1.10 | 1.09 | 1.09 | 1.07 | 1.20 | 1.11 |
| | | | | | | | | | |
| 1900X | | 1.19 | 1.17 | 1.14 | 1.14 | 1.14 | 1.10 | 1.14 | 1.13 |
| 5800X | 1.13 | | 1.00 | 1.11 | 1.11 | 1.11 | 1.08 | 1.08 | 1.08 |
| 5950X | 1.15 | 1.00 | | 1.11 | 1.11 | 1.11 | 1.08 | 1.14 | 1.09 |
| i7 6G | 1.30 | 1.17 | 1.14 | | 1.00 | 1.00 | 1.04 | 1.40 | 1.12 |
| i7 10G | 1.23 | 1.16 | 1.16 | 1.00 | | 1.00 | 1.17 | 1.41 | 1.13 |
| i9 10G | 1.26 | 1.18 | 1.16 | 1.00 | 1.00 | | 1.06 | 2.51 | 1.21 |
| i7 11G | 1.24 | 1.13 | 1.11 | 1.09 | 1.09 | 1.09 | | 1.35 | 1.13 |
| i9 12G | 1.20 | 1.11 | 1.10 | 1.07 | 1.07 | 1.06 | 1.06 | | 1.08 |
| Clang | 1.12 | 1.08 | 1.07 | 1.06 | 1.07 | 1.07 | 1.03 | 1.21 | 1.09 |
| GCC | 1.17 | 1.15 | 1.14 | 1.15 | 1.15 | 1.15 | 1.13 | 1.23 | 1.16 |
| Final | 1.12 | 1.08 | 1.07 | 1.06 | 1.07 | 1.07 | 1.03 | 1.21 | 1.09 |

secp256k1

| run on | 1900X | 5800X | 5950X | i7 6G | i7 10G | i9 10G | i7 11G | i9 12G | G.M. |
|---|---|---|---|---|---|---|---|---|---|
| opt on | | | | | | | | | |
| 1900X | | 1.01 | 0.99 | 1.11 | 1.10 | 1.11 | 1.11 | 1.15 | 1.07 |
| 5800X | 1.13 | | 0.98 | 1.22 | 1.22 | 1.25 | 1.18 | 1.24 | 1.15 |
| 5950X | 1.17 | 1.12 | | 1.27 | 1.27 | 1.30 | 1.26 | 1.34 | 1.21 |
| i7 6G | 1.04 | 0.95 | 0.94 | | 1.00 | 1.01 | 1.01 | 1.04 | 1.00 |
| i7 10G | 1.07 | 0.99 | 0.96 | 1.00 | | 1.01 | 1.04 | 1.05 | 1.01 |
| i9 10G | 1.05 | 1.00 | 0.98 | 1.03 | 1.04 | | 1.04 | 1.04 | 1.02 |
| i7 11G | 1.08 | 0.98 | 0.97 | 1.07 | 1.09 | 1.07 | | 1.03 | 1.03 |
| i9 12G | 1.05 | 1.00 | 0.98 | 1.08 | 1.08 | 1.12 | 1.02 | | 1.04 |
| Clang | 1.35 | 1.27 | 1.24 | 1.42 | 1.42 | 1.43 | 1.19 | 1.43 | 1.34 |
| GCC | 1.61 | 1.39 | 1.36 | 1.94 | 1.94 | 1.96 | 1.77 | 2.05 | 1.73 |
| Final | 1.35 | 1.33 | 1.33 | 1.42 | 1.42 | 1.43 | 1.19 | 1.43 | 1.36 |
| | | | | | | | | | |
| 1900X | | 1.05 | 1.04 | 1.06 | 1.13 | 1.10 | 1.13 | 1.13 | 1.08 |
| 5800X | 1.11 | | 0.98 | 1.11 | 1.15 | 1.16 | 1.15 | 1.18 | 1.10 |
| 5950X | 1.16 | 1.02 | | 1.13 | 1.22 | 1.18 | 1.21 | 1.23 | 1.14 |
| i7 6G | 1.14 | 1.04 | 1.02 | | 1.00 | 1.00 | 1.07 | 1.08 | 1.04 |
| i7 10G | 1.07 | 1.05 | 1.03 | 0.96 | | 1.00 | 1.10 | 1.06 | 1.03 |
| i9 10G | 1.05 | 1.02 | 1.00 | 1.00 | 1.02 | | 1.07 | 1.05 | 1.03 |
| i7 11G | 1.11 | 1.04 | 1.02 | 1.03 | 1.07 | 1.07 | | 1.09 | 1.05 |
| i9 12G | 1.07 | 1.02 | 1.01 | 1.02 | 1.07 | 1.07 | 1.06 | | 1.04 |
| Clang | 1.29 | 1.27 | 1.24 | 1.34 | 1.39 | 1.39 | 1.21 | 1.46 | 1.32 |
| GCC | 1.62 | 1.50 | 1.47 | 1.86 | 1.93 | 1.93 | 1.84 | 1.86 | 1.74 |
| Final | 1.29 | 1.27 | 1.26 | 1.39 | 1.39 | 1.39 | 1.21 | 1.46 | 1.33 |

Table 7. Cost of scalar multiplication (in cycles) of different implementations benchmarking on different machines.

| | Implementation | Lang. | 1900X | 5800X | 5950X | i7 6G | i7 10G | i9 10G | i7 11G | i9 12G | G.M. |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Curve25519 | sandy2x [Chou 2015] | asm-v | 603k (1.06x) | **427k (1.00x)** | **426k (1.00x)** | 521k (1.15x) | 521k (1.15x) | 521k (1.15x) | 492k (1.16x) | 480k (1.31x) | 496k (1.08x) |
| | amd64-64 [Chen et al. 2014] | asm | 622k (1.09x) | 573k (1.34x) | 573k (1.34x) | 569k (1.26x) | 571k (1.26x) | 572k (1.27x) | 533k (1.26x) | 453k (1.24x) | 556k (1.21x) |
| | amd64-51 [Chen et al. 2014] | asm | 771k (1.35x) | 584k (1.37x) | 586k (1.37x) | 568k (1.25x) | 567k (1.25x) | 567k (1.26x) | 496k (1.17x) | 424k (1.16x) | 563k (1.22x) |
| | donna [Langley 2022] | asm-v | 1092k (1.92x) | 970k (2.27x) | 970k (2.27x) | 1013k (2.24x) | 1015k (2.25x) | 1015k (2.25x) | 955k (2.26x) | 887k (2.43x) | 988k (2.14x) |
| | donna-c64 [Langley 2022] | C | 806k (1.42x) | 607k (1.42x) | 602k (1.41x) | 581k (1.28x) | 582k (1.29x) | 583k (1.29x) | 543k (1.28x) | 452k (1.24x) | 588k (1.27x) |
| | OSSL ots [OpenSSL 2022] | C | 700k (1.23x) | 538k (1.26x) | 540k (1.27x) | 569k (1.26x) | 567k (1.26x) | 568k (1.26x) | 492k (1.16x) | 420k (1.15x) | 544k (1.18x) |
| | OSSL fe-51 ots [OpenSSL 2022] | asm | 695k (1.22x) | 540k (1.27x) | 539k (1.26x) | 568k (1.25x) | 567k (1.26x) | 568k (1.26x) | 491k (1.16x) | 419k (1.15x) | 543k (1.18x) |
| | OSSL fe-51+**CryptOpt** | asm | 715k (1.26x) | 531k (1.24x) | 537k (1.26x) | 561k (1.24x) | 558k (1.24x) | 559k (1.24x) | 497k (1.17x) | 402k (1.10x) | 539k (1.17x) |
| | OSSL fe-64 ots [OpenSSL 2022] | asm | **570k (1.00x)** | 444k (1.04x) | 444k (1.04x) | 462k (1.02x) | 460k (1.02x) | 464k (1.03x) | 461k (1.09x) | 412k (1.13x) | 463k (1.00x) |
| | OSSL fe-64+**CryptOpt** | asm | 590k (1.04x) | 459k (1.08x) | 464k (1.09x) | 494k (1.09x) | 494k (1.09x) | 497k (1.10x) | 451k (1.06x) | 394k (1.08x) | 478k (1.04x) |
| | HACL* fe-64 [HACL 2022] | asm | 588k (1.03x) | 494k (1.16x) | 494k (1.16x) | **452k (1.00x)** | **452k (1.00x)** | **451k (1.00x)** | **423k (1.00x)** | **365k (1.00x)** | **461k (1.00x)** |
| secp256k1 | libsecp256k1 [Bitcoin Core 2022] | asm | 718k (1.06x) | 568k (1.06x) | 566k (1.05x) | 564k (1.03x) | 566k (1.05x) | 564k (1.04x) | 540k (1.08x) | 437k (1.04x) | 561k (1.04x) |
| | libsecp256k1 [Bitcoin Core 2022] | C | **676k (1.00x)** | 543k (1.01x) | 540k (1.00x) | 563k (1.03x) | 563k (1.05x) | 562k (1.04x) | 503k (1.00x) | 438k (1.04x) | 545k (1.01x) |
| | libsecp256k1+**CryptOpt** | asm | 727k (1.08x) | 547k (1.02x) | 546k (1.01x) | 552k (1.01x) | 561k (1.04x) | 551k (1.02x) | **501k (1.00x)** | **420k (1.00x)** | 545k (1.01x) |
| | libsecp256k1+**CryptOpt**(CS2) | asm | 715k (1.06x) | **535k (1.00x)** | **539k (1.00x)** | **548k (1.00x)** | **538k (1.00x)** | **541k (1.00x)** | 506k (1.01x) | 420k (1.00x) | **538k (1.00x)** |

as do AMD Zen 3 processors in the 2x2 sub-matrices have values in the small interval $[0.98, 1.01]$ (mul), and equal for square.

This specialization to platforms is a double-edged sword. On the one hand, it allows CryptOpt to produce high-performing code, as previously shown. On the other hand, it also means that the code is potentially less generic, and it might be less performant if executed on platforms other than originally intended, as observed in our a-posteriori analysis.

## D DETAILED PERFORMANCE INFORMATION

Table 7 shows the detailed performance information of the scalar multiplication experiments. The table is divided in two sections, one for Curve25519 and one for secp256k1. In each of those sections we compare different implementations of the scalar multiplication for respective curve against each other. The language, in which the core operations are implemented, is written in the column *Lang*. The following columns show the elapsed cycles for the scalar multiplication. The fastest implementation per section and per platform is highlighted in (bold text) and notes a (1.00x). All other implementations are then compared against this fastest in the form of the ratio, which is written in parenthesis. E.g. 1.05x means that it takes 1.05 times as many cycles than the fastest. The last column, G.M., is the geometric mean of the cycles across all platforms, the ratio is then recalculated on that G.M. as well.

For secp256k1, we base all evaluation on the implementation of `libsecp256k1`. We compare its asm and C implementations (first two rows) against our optimized version, when we plug in verified and optimized field operations (third row) and when we optimize its own implementation (in our Case Study 2).

Note: "ots" stands for off-the-shelf; "asm" means assembly; "-v" indicates the use of vector instructions. The HACL* implementation [HACL 2022] uses parallelized field arithmetic. For Curve25519, "-51" and "-64" indicates whether the representation for the field elements is unsaturated or saturated.