Benchmarking class groups

Class groups, for which finding their order is considered hard (under standard assumptions), are important building blocks to many cryptographic primitives including RSA accumulators, time lock puzzles, polynomial commitments, etc.
For evaluating our implementation of class group we attempted to compare to three other class group implementations written in rust (which are the only public libraries we found), where two of which, Cambrian's accumulator class group , and classy groups , failed to run and threw the (exact same) exception:

| ^^^^^^^^^^^ expected struct NonNull, found *-ptr

Hence, we benchmarked our class group and the VDF's class group using our machine (2.2 GHz Intel Core i7-4770HQ). Specifically we tested exponentiation in this group. Benchmarking VDF's class group gave odd results; on the one hand, for the very specific discriminant (that they chose), generated by a 2048 bit number, their implementation outperform ours (as can be seen in the table), on the other hand, for **any other** discriminant value that we checked, our implementation outperforms theirs. In fact their running time is extremely slow, even for very small exponents, and hence we omitted the corresponding benchmarking results.
The following table presents the statistics of 10 samples of exponentiation in this group by comparing the running time of computing the term b^e for various sizes of the exponents e (e is sampled randomly, although its size is fixed) where the base b is a fixed element in the group, of size 2048 bits. The description of the benchmark output can be found in the Criterion documentation.

exponent size (bits)	**ZenGo-X's class group**	**VDF's class group**
1000	time: [200.22 ms **202.34 ms** 204.66 ms] mean [200.22 ms 204.66 ms] std. dev. [2.2018 ms 4.4583 ms]	time: [146.75 ms **147.10 ms** 147.66 ms] mean [146.12 ms 147.89 ms] std. dev. [601.23 us 2.0205 ms]
2000	time: [399.75 ms **403.66 ms** 408.00 ms] mean [399.75 ms 408.00 ms] std. dev. [2.5260 ms 8.2210 ms]	time: [306.68 ms **311.01 ms** 315.04 ms] mean [306.68 ms 315.04 ms] std. dev. [4.0035 ms 9.2049 ms]
3000	time: [786.19 ms **795.08 ms** 804.42 ms] mean [786.19 ms 804.42 ms] std. dev. [9.5712 ms 18.502 ms]	time: [470.08 ms **479.96 ms** 490.66 ms] mean [470.08 ms 490.66 ms] std. dev. [9.8650 ms 22.112 ms]
4000	time: [786.19 ms **795.08 ms** 804.42 ms] mean [786.19 ms 804.42 ms]	time: [609.69 ms **616.68 ms** 624.55 ms] mean [609.69 ms 624.55 ms]

	std. dev. [9.5712 ms 18.502 ms]	std. dev. [5.6596 ms 15.579 ms]
5000	time: [1.0139 s **1.0388 s** 1.0682 s] mean [1.0139 s 1.0682 s] std. dev. [16.667 ms 57.918 ms]	time: [749.73 ms **762.09 ms** 776.36 ms] mean [749.73 ms 776.36 ms] std. dev. [9.4668 ms 29.220 ms]
6000	time: [1.2000 s **1.2096 s** 1.2194 s] mean [1.2000 s 1.2194 s] std. dev. [10.714 ms 20.099 ms]	time: [903.48 ms **931.34 ms** 963.35 ms] mean [903.48 ms 963.35 ms] std. dev. [14.973 ms 61.293 ms]