From: D. J. Bernstein <djb@cr.yp.to>
To: pqc-comments@nist.gov
CC: pqc-forum@list.nist.gov
Subject: Re: [pqc-forum] ROUND 3 OFFICIAL COMMENT: Classic McEliece
Date: Monday, July 04, 2022 05:58:54 PM ET
Attachments: smime.p7m

D. J. Bernstein, T. Lange, and C. Peters wrote:
> Our PQCrypto 2008 ISD algorithm is faster than the Eurocrypt 2022 ISD
> algorithm, on the CPUs selected in the new paper, for the challenges
> selected in the new paper, according to a direct comparison of (1) our
> measurements of the 2008 software (the 2+2 case of the 2008 algorithm)
> and (2) the speeds reported in the new paper for that paper's software.

As further confirmation, I used the 2008 software to run full attacks on two EPYC 7742 CPUs (the 2022 paper selected these CPUs and used four of them) against all three dimension-1223 Goppa challenges available from https://decodingchallenge.org. (The 2022 paper covered one of the dimension-1223 challenges and a dimension-1284 challenge.) All three runs completed successfully without any surprises.

Thanks to the Fast Crypto Lab cluster at the Institute of Information Science at Academia Sinica, Taiwan, for providing the dual EPYC 7742.

The exact time for any particular attack run is well known to have considerable randomness, just like searching randomly for an AES key. Because of the variance in run times, individual run times are not statistically meaningful and should never be used to compare algorithms of this type. However, I monitored wall-clock times, CPU times, cycle counts (using RDTSC, which runs at 2.245GHz on these CPUs), and iteration counts to check for any discrepancies across these numbers.

All three of the runs happened to be lucky, using, respectively, just 83%, 1.7%, and 83% of the average number of iterations (184 billion): more precisely, 153022933873, 3111365348, 153926141900 iterations. Each iteration took 1.68 million cycles (average; there was some variance, as expected from the cache structure). The wall-clock times using 128 cores

were about 248 hours, 5 hours, and 250 hours. Here are the CPU times and wall-clock times in more detail:

```
   114454917.40user 497.65system 248:23:46elapsed 12799%CPU (0avgtext+0avgdata
15380maxresident)k
   2327597.87user 21.17system 5:03:05elapsed 12799%CPU (0avgtext+0avgdata
15488maxresident)k
   115167358.18user 200.05system 249:56:30elapsed 12799%CPU (0avgtext+0avgdata
15496maxresident)k
```

The output vectors for the three runs appear below, in the same order as the "providers" on https://decodingchallenge.org.

It is natural to wonder whether lucky runs are actually an indication that average iteration counts have been miscalculated. Here are three independent ways to check the calculations.

The first is to run more experiments, meaning smaller runs for any given CPU budget. For example, here are the iteration counts observed in running the same code (with the same l=22, m=1, c=8) against a range of smaller challenges from https://decodingchallenge.org:

```
   431 3.8% 154 4060 4033
   431 52.8% 2144 4060 4033
   431 60.0% 2435 4060 4033
   482 131.1% 18482 14101 14029
   482 220.7% 31120 14101 14029
   482 16.6% 2338 14101 14029
   534 45.2% 5341 11820 11748
   534 47.5% 5616 11820 11748
   534 2.6% 302 11820 11748
   587 33.2% 13600 40969 40766
   587 73.6% 30168 40969 40766
   587 9.6% 3937 40969 40766
   640 154.1% 225989 146688 146077
   640 51.2% 75132 146688 146077
   640 137.4% 201604 146688 146077
```

```
695 131.9% 730398 553830 551850
695 43.9% 243069 553830 551850
695 1.1% 6030 553830 551850
751 64.3% 6500820 10111752 10087366
751 67.3% 6803082 10111752 10087366
751 134.4% 13586235 10111752 10087366
808 133.9% 55699927 41583562 41492753
808 11.4% 4748656 41583562 41492753
808 3.5% 1460929 41583562 41492753
865 60.7% 96347036 158848155 158526591
865 53.9% 85624754 158848155 158526591
865 11.6% 18492422 158848155 158526591
923 269.7% 1847405002 684909178 683634590
923 174.9% 1197860601 684909178 683634590
923 6.3% 43134954 684909178 683634590
982 35.1% 974255720 2776326652 2771487282
982 128.7% 3572455007 2776326652 2771487282
982 24.1% 668935675 2776326652 2771487282
1041 47.4% 236327577 498978858 497804860
1041 248.1% 1237981333 498978858 497804860
1041 170.7% 851782527 498978858 497804860
```

The first column is the dimension. The last two columns are the average iteration counts calculated for type-1 and type-3 iterations, always very close together. (The attack code uses type-2 iterations, which are intermediate.) The third column is the observed iteration count for one attack. The second column is the observed iteration count as a percentage of the calculated average type-1 iteration count.

A graph of this distribution of percentages shows no evident anomalies compared to graphs of percentages sampled from the calculated distribution. The usual statistics do not show surprising p-values. One can, of course, carry out many more experiments to pin down the experimental average to within, e.g., 1% and check for a match with the calculated average. The 2008 paper already reported doing this across millions of experiments for small sizes.

The second way to check the calculations is to review the calculation software, including (1) the formulas and (2) examples checked by hand. This work was already done in 2008 for the C calculator available from https://github.com/christianepeters/isdf2. I've further checked output of the C calculator against a new Sage calculator for both "type 1" and "type 3" iterations, and checked the Sage calculator against the Markov chains described in the paper.

The third way to check the calculations is to do a much simpler calculation of the number of _independent_ iterations, equivalent to taking the c parameter much larger. Structurally, it's obvious that this will be smaller than the actual number of iterations for (say) c=8, but not _much_ smaller, since randomizing 8 columns has a considerable chance of changing the error weight in the information set. For dimension 1223, this calculation is an unsurprising 7% smaller than the 185 billion calculated for c=8.

The average 185 billion iterations at 1.68 million cycles/iteration for the 2008 software match the 6.28 days reported in README in

   https://cr.yp.to/software/lowweight-20220616.tar.gz

for the dimension-1223 challenges on four EPYC 7742 CPUs. This is faster than the 8.22 days reported at the bottom of page 12 of the 2022 paper for the calculated average time for that paper's AVX2-specific software attacking those challenges on those CPUs.

The 2022 paper claims to "demonstrate that these algorithms lead to significant speedups for practical cryptanalysis on medium-sized instances (around 60 bit)"; concretely, it says "12.46 and 17.85 times faster on the McEliece-1284 challenge and 9.56 and 20.36 times faster on the McEliece-1223 instance than [14] and [24]". But [14] and [24] are missing various speedups from the 2008 paper. The 2022 paper failed to compare the speed of its algorithm to the speed of the 2008 algorithm.

The README from https://cr.yp.to/software/lowweight-20220616.tar.gz also runs through known speedups not included in the 2008 software, and

concludes as follows:

> Overall it would not be surprising if at least half of the attack cycles can be removed on current CPUs compared to the 2008 code running on current CPUs. A much larger reduction in attack cost from 2008 to 2022 has come from changes in hardware: computers do more per cycle and cost less per cycle. Accounting for continued improvements in technology is an important part of selecting cryptosystem parameters.

——D. J. Bernstein

positions 606 518 167 959 461 163 190 302 1204 206 812 668 947 240 368 837 187 1117 829 723 1160 1014 913
sorted 163 167 187 190 206 240 302 368 461 518 606 668 723 812 829 837 913 947 959 1014 1117 1160 1204
vector
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000001000100
0000000000000000010010000000000000001000000000000000000000000000000000100000000000000
0000000000000000000000000000000000000000000000010000000000000000000000000000000000000
0000000000000000000000000000100000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000001000000000000000000000000000000000000000000000000
0000000010000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000010000000000000000000000000000000000000000000000000000000000000100000000000
0000000000000000000000000000000000000000000100000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000010000000000000000100000001000000000000
0000000000000000000000000000000000000000000000000000000000000001000000000000000000000
0000000000001000000000001000000000000000000000000000000000000000000000000000000100000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000001000000000000000000000000000000000000000000001000000000000000000000000000
0000000000000010000000000000000000
positions 101 1073 350 529 519 827 1210 410 1038 797 479 906 418 403 342 575 146 397 805 226 716 704 102
sorted 101 102 146 226 342 350 397 403 410 418 479 519 529 575 704 716 797 805 827 906 1038 1073 1210

vector
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000001100000000000000000000000000000000000000000000000000000100000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000100000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0010000000100000000000000000000000000000000000000000000000000000000000100000100000001000000010000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000100000000000000000000000000000000000000000
0000000001000000000100000000000000000000000000000000000000000000000000100000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000010000000000010000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000100000001000000000000000000000000000000100000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000100000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000010000000000000000000000000000000000000000000000000000100000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000001000000000000
positions 797 450 705 1107 1005 658 1068 345 866 952 335 849 134 987 943 900 553 16
577 407 904 631 1162
sorted 16 134 335 345 407 450 553 577 631 658 705 797 849 866 900 904 943 952 987
1005 1068 1107 1162
vector
0000000000000000100000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000100000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000010000
0000010000000000000000000000000000000000000000000000000000000000000000000000000000000000000000100000000000000000
0000000000000000000000000100000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000100000000000000000000000000000100000000000000000
0000000000000000000000000000000000000000000000000000000000000000100000000000000000000000000000100000000000000000
0000000000000000000000000000000000000000000000000000000000000000100000000000000000000000000000000000000000000001
0000000000000000100000000000000000000000000000000000000000000000100010000000000000000000000000000000000000000000
0000000010000000010000000000000000000000000000000000000000000000100000000000000000000000000000100000000000000000
0000000000000000000000000000000000000000000000000000000000000000100000000000000000000000000000000000000000000000
0010000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000100000000000000000
0000000000000000000000000000000000