in the signature size, resulting from bug fixes and differences in the encoding and packing to address implementation vulnerabilities demonstrated by Saarinen[16] [17] make this version incompatible with the original one. All three parameter sets are able to run on our evaluation platform.

- **HAWK** [BBD+23]: The reference implementations of HAWK from the submission package have been merged into `pqm4` in #305 and all three parameter sets are running.
- **HuFu** [YJL+23]: HuFu uses public keys of 1 059 KB (security level 1) to 3 573 KB (security level 3). This exceeds the available memory available on our target platform and we, thus, do not include HuFu in `pqm4`.
- **Raccoon** [dPEK+23]: The reference implementation of Raccoon is making use of the `__int128` datatype which is not portable to our platform. The Raccoon reference implementation offers the option (`POLYR_Q32`) to switch to 32-bit NTTs (**mod** 16515073 and **mod** 33292289) rather than 64-bit NTTs (**mod** $16515073 \cdot 33292289$) for better support of 32-bit platforms. However, turning on this option does not eliminate all instances of `__int128`. We have contacted the submission team and learned that a fully portable implementation is work in progress. We cannot include Raccoon in `pqm4` at this moment.
- **SQUIRRELS** [ENST23]: SQUIRRELS requires public keys of 666 KB (`Squirrels-I`) to 2 721 KB (`Squirrels-V`) which is too large for our target platform. Furthermore, the reference implementation of SQUIRRELS depends on multiple external libraries. We do not include SQUIRRELS in `pqm4`.

3.4 MPC-in-the-Head Signatures

- **Biscuit** [BKPV23]: The reference implementation of Biscuit from the submission package has been merged into `pqm4` in #314. Of the six parameter sets, three (`biscuit128f`, `biscuit192f`, `biscuit256f`) meet the constraints of our evaluation board.
- **MIRA** [ABB+23c]: The MIRA reference implementation makes heavy use of dynamic memory that would have to be eliminated prior to merging it into `pqm4`. MIRA is re-using the same data structures (e.g., `gfqm_vec` or `gf16_mat`) for multiple sizes. This requires significant refactoring for eliminating dynamic memory allocations. We do not include MIRA in `pqm4` for now.
- **MiRitH** [ARZV+23]: The MiRitH reference implementation from the official repository[18] has been added to `pqm4` in #315. The MiRitH team also provides an implementation optimized for the Cortex-M4 in the same repository. We have merged the optimized implementation in #325. We have reported multiple small issues with those implementations to the submission team which have been fixed in the official repository by now. All `fast` parameter sets of the non-hypercube variant are functional on our testing platform. From the `hypercube` parameter sets, the `fast` and `short` sets for security level 1 and 3 and `mirith_hypercube_Va_fast` and `mirith_hypercube_Vb_fast` are running.
- **MQOM** [FR23]: The MQOM reference implementation from the submission package has been merged into `pqm4` in #322. We have eliminated a large number of dynamic memory allocations from the reference implementation. Luckily, the vast majority of dynamic memory allocations actually had a static size and could easily be replaced. A small number of variable-sized buffers have been replaced by variable-length arrays (VLAs). Those could be replaced with static buffers of worst-case length. Only the two parameter sets `mqom_cat1_gf251_fast` and `mqom_cat1_gf31_fast` of the 12 available sets are suitable for our evaluation board.
- **PERK** [ABB+23a]: The PERK team has contributed a reference implementation and a M4-optimized implementation compatible with the specification (v1.1) in #318. The M4 implementation is presented in [BBB+24]. This version is, however, incompatible with the one in the NIST submission package. Also note that compared to the official reference implementation (v1.1), the PERK team has replaced the GMP dependency with standalone arithmetic. All parameter sets are running on our evaluation board.
- **RYDE** [ABB+23b]: Similar as MIRA, RYDE uses numerous (>50) dynamic memory allocations re-using the same data-structures for differently sized buffers. Including RYDE into `pqm4` would require significant refactoring effort for removing dynamic memory allocations. We do not include it for now.
- **SDitH** [MFG+23]: The SDitH reference implementation makes light use of dynamic memory allocations, but those could be eliminated without too much effort. However, the overall memory footprint ranges from

[16] https://groups.google.com/a/list.nist.gov/g/pqc-forum/c/ImcSqGLFdoo/m/G86jtgDtBQAJ
[17] https://groups.google.com/a/list.nist.gov/g/pqc-forum/c/Hq-wRFDbIaU/m/iLZctTiLAgAJ
[18] https://github.com/Crypto-TII/mirith_nist_submission/commit/f27b540b77215dd17b10417726c6c6f7ccd41aa5

800 KB (for `SDitH-L1-gf256`) to 2.5 MB (for `SDitH-L5-gf256`) which exceeds the available memory of our target platform. Getting SDitH to work on the Cortex-M4 would require a stack-optimized implementation. We do not include SDitH in `pqm4` for now.

3.5 Multivariate Signatures

- **3WISE** [**Rod23a**]: Smith-Tone [ST23] presented a polynomial time attack on 3WISE, therefore, we did not include 3WISE in `pqm4`. Furthermore, the dependency on the FLINT library currently prevents the reference code from being included in `pqm4`.
- **DME-Sign** [**LA23**]: Briaud, Bros, Perlner, and Smith-Tone[19] presented a key recovery attack on DME-Sign, which has been acknowledged by the DME-Sign team. Thus, we currently do no consider adding DME-Sign to `pqm4`.
- **HPPC** [**Rod23b**]: The reference implementation of HPPC unfortunately depends on the external libraries FLINT and M4RI. These external dependencies prohibit a straightforward integration into `pqm4`.
- **MAYO** [**BCC+23b**]: The MAYO reference implementation from the official repository[20] as well as the M4F-optimized implementation described in [BCC+23c] have been merged into `pqm4` in #302. Note that [BCC+23c] also proposes a change to the MAYO specification by switching to a nibble representation rather than the bitsliced representation. We merged the bitsliced version that is compatible with the round-1 specification. Only the parameter set for the highest security level does not fit on the evaluation board.
- **PROV** [**GCF+23**]: Even for the smallest parameter set (`PROV-I`), PROV requires more than the available 640 KB for generating a signature. The current implementation requires 428 536 bytes for the expanded public key which together with the compressed public key (68 326 bytes) and the secret key (203 688 bytes) already exceeds the available memory.
- **QR-UOV** [**FIH+23**]: QR-UOV has not been included in `pqm4` because the reference implementation allocates huge arrays for signature computation that exceed the available memory resources of our target platform.
- **SNOVA** [**WCD+23**]: We merged the SNOVA reference implementation from the submission package into `pqm4` in #311. The current implementation of SNOVA implementation is using a pre-computed static table `S`. In the reference implementation, this table is computed dynamically and cached. The computation is done outside of the core function and, thus, not reflected in the benchmarks. For a quick integration into `pqm4`, we re-compute this table at the beginning of key generation, signing, and verification, but maintain the static array. This results in fairer benchmarks while not structurally changing the code. A better solution would be to pre-compute the constants and placing them in the code. All three parameter sets targeting security level 1 run both in the `esk` and the `ssk` variant on the evaluation board. `snova-37-8-16-4-ssk` targeting security level three is also running and leading to seven out of 18 variants. [IA24] raised concerns about the current SNOVA parameter sets reaching the claimed security levels. The SNOVA team acknowledged these concerns and have proposed updated parameters[21]. However, as of now, no updated implementation is available.
- **TUOV** [**DGG+23**]: The high memory usage of the reference implementation from the submission package of TUOV prevents it from running on our evaluation platform. The parameter set with the lowest memory consumption (`tuov-Ip`) requires around 750 KB of RAM. The implementation is making use of dynamic memory allocations. However, these can be easily eliminated. We included TUOV in `pqm4` in #327, but none of the parameter sets are functional on the target board.
- **UOV** [**BCD+23**]: The UOV implementation from the official repository as well as the M4F-optimized implementation described in [BCH+23] have been merged in #300. Note that only the `uov-Ip` parameter sets fit on our target platform. In the paper, the authors are also able to evaluate the `uov-Is` parameters on the same target by writing public and secret keys to flash memory. Writing to flash memory is not supported by the `pqm4` framework. The three security level 1 parameter set require less then the 640 KiB SRAM and are thus functional on our board, the remaining nine parameter sets require more than this.

---

[19] `https://groups.google.com/a/list.nist.gov/g/pqc-forum/c/aoXpl4TlNh4/m/Eal1YHw0BAAJ`
[20] `https://github.com/PQCMayo/MAYO-C`
[21] `https://groups.google.com/a/list.nist.gov/g/pqc-forum/c/m11kg20sTyU/m/cLkGIDaiBAAJ`

– **VOX** [**PCF+23**]: Furue and Ikematsu [FI23] raised concerns that the security of the current VOX parameters has been severely overestimated.[22]. The VOX team has acknowledged the attack and proposes new parameters in [MRPC+23]. However, at this time no reference implementations of the new parameter sets have been published and we, hence, do not include VOX in pqm4.

3.6 Symmetric-based Signatures

– **AIMer** [**KCC+23**]: The AIMer reference implementation from the submission package contains a large number of dynamic memory allocations. All of those can be easily converted into stack allocations. We eliminated the dynamic memory allocations and merged AIMer into pqm4 in #323. However, for some of the parameter sets, that these buffers exceed the 4 MiB stack memory available on qemu's mps2-an386 (or even the 8 MiB default stack size on Linux). We only include parameter sets that we managed to successfully test using qemu. `aimer-l1-param1`, `aimer-l1-param2` and `aimer-l3-param1` are running on our target platform, nine further parameter sets for AIMer are not.
– **Ascon-Sign** [**SGJ+23**]: We have merged the Ascon-Sign reference implementation from the submission package into pqm4 in #263. Since pqm4 does not support Ascon, we use the Ascon version shipped in the submission package. All parameter sets are running on our evaluation board.
– **FAEST** [**BBdSG+23**]: The FAEST reference implementation in the submission package uses a large number of dynamic memory allocations (>150). All of the dynamic memory allocations in `faest_aes.c` and `vole.c` can easily be converted into VLAs. They could also be converted to fixed-sized buffers if defining parameters statically. However, some other dynamic memory allocations are harder to eliminate: The structs `vec_com_t`, `vec_com_rec_t`, `tree_t` hold pointers to buffers that have varying size even for a single parameter set. Those would have to be duplicated for each size needed, or alternatively, the worst case size needs to be used potentially increasing the memory footprint. We do not include FAEST in pqm4.
– **SPHINCS-alpha** [**YCZ23**]: The SPHINCS-alpha reference implementation from the submission package has been merged into pqm4 in #312. The implementation used static memory that contains a large (280 KB for `sphincs-a-shake-128f`) lookup table that is computed during first use (i.e., key generation) and re-used throughout the computation including signing and verification. This table computation requires significant time (around 15 million clock cycles for `sphincs-a-shake-128f`). As this leads to unfair benchmarking results in signing and verification, we instead compute the table once in the beginning of each of key generation, signing, and verification. We also move the table to the stack. We were able to make 6 out of 24 variants functional for pqm4: `128f`, `128s` and `192f` each in both the `sha2` and the `shake` version.

3.7 Other Signatures

– **ALTEQ** [**BDN+23**]: The ALTEQ implementation available in the NIST submission package makes heavy use of dynamic memory allocations. Additionally, the memory footprint is too large to fit on our target platform (`alteq-shortsig-I` requires around 1 MB, `alteq-balanced-I` requires around 2 MB). We, thus, do not include ALTEQ in pqm4.
– **eMLE-Sig 2.0** [**LZ23**]: eMLE-Sig 2.0 has been shown vulnerable by Tibouchi.[23] An implementation of the attack is available.[24] We do not include eMLE-Sig 2.0 in pqm4.
– **KAZ-SIGN** [**AAC+23**]: Bernstein demonstrated a signature forgery attack against KAZ-SIGN,[25] we therefore did not include the scheme in pqm4. The KAZ-SIGN team has published updated versions four times (with the latest iteration being KAZ-SIGN 1.4[26]), each time being broken by Bernstein within one day.
– **Preon** [**CCC+23**]: The reference implementation of the smallest parameter set of Preon (`Preon128A`) currently requires around 200 MB of memory for signing. Additionally, it has more than 250 dynamic memory allocations. We do not include Preon in pqm4.

[22] https://groups.google.com/a/list.nist.gov/g/pqc-forum/c/icHfTrzkfw4/m/Zj7GrnjMAQAJ
[23] https://groups.google.com/a/list.nist.gov/g/pqc-forum/c/zas5PLiBe6A/m/aOnAlT6cAQAJ
[24] https://github.com/mti/attack_emle
[25] https://groups.google.com/a/list.nist.gov/g/pqc-forum/c/2ljDcgtawFw/m/61PiLt6WAgAJ
[26] https://groups.google.com/a/list.nist.gov/g/pqc-forum/c/jv72ZzYwAZQ/m/ayNzr7U1GQAJ

– **Xifrat1-Sign.I** [NP23]: Xifrat1-Sign.I was practically broken by Panny[27]. We do not include it.

4 Results

In this section we summarize the benchmarking results at the time of writing. As the `pqm4` framework is under constant development, the numbers may change over time. The largest change for now is the use of a new target platform, which produces different results, due to its different memory timings. This change is accompanied by a newer compiler version. At the time of writing, we use the version 13.2 of the GNU C Compiler toolchain provided by Arm[28]. In the future, implementations for the schemes may be replaced by faster versions, or implementations with other goals (e.g., lower memory requirements) are added. The `pqm4` GitHub repository contains continually updated listing of the results.

Table 2 presents the measured execution speed of each implementation in terms of CPU cycles in thousands (i.e., kilocycles). We measured ten executions per scheme and list the average value, with the exception of the `dilithium`, `haetae`, and `hawk` schemes, which were executed 100 times, due to their significant variance in execution time. The table lists the average cycle counts of all executions, with the difference to the minimum and maximum shown in the super- and subscript. The percentage of cycles spent in symmetric primitives is shown in parentheses. As reference, we included numbers for `dilithium` implementing the third round specification.

Table 3 presents the memory requirements of each implementation. Listed are the sizes of the text (i.e., compiled code), data, and BSS sections produced by the implementations source, as well as the required stacksize (measured in KiB) of each operation, excluding the key, message, and signature. While the code/data sizes can be determined statically with the compiler tools, the stack size was determined using the QEMU simulator. As the QEMU simulator produces the same results for the memory metrics as real hardware platforms and provides more resources, it allows us to test more schemes and security levels. Some of the largest schemes, however, are still too big for the simulated platform.

Table 2: Average execution speed for key generation, signature generation, and signature verification for each scheme implementation, as measured on the Nucleo-L4R5ZI evaluation board. Execution speed is shown in thousands of cycles, with the difference to the minimum and maximum shown in the super- and subscript respectively. Cycles spent on symmetric cryptography shown in parentheses.

Scheme	impl.	keygen		sign		verify	
`dilithium2`	`clean`	1874 $^{+41}_{-35}$	(62%)	7283 $^{+13672}_{-3962}$	(37%)	2062 $^{+0}_{-0}$	(53%)
	`m4f`	1426 $^{+40}_{-47}$	(80%)	3815 $^{+7908}_{-2001}$	(67%)	1417 $^{+0}_{-0}$	(77%)
`dilithium3`	`clean`	3205 $^{+2}_{-1}$	(65%)	12893 $^{+52247}_{-7796}$	(40%)	3376 $^{+0}_{-0}$	(57%)
	`m4f`	2516 $^{+1}_{-1}$	(82%)	6374 $^{+11353}_{-3439}$	(69%)	2411 $^{+0}_{-0}$	(79%)
`dilithium5`	`clean`	5340 $^{+66}_{-53}$	(67%)	15533 $^{+35954}_{-7581}$	(45%)	5610 $^{+0}_{-0}$	(61%)
	`m4f`	4277 $^{+41}_{-46}$	(84%)	8473 $^{+16493}_{-3591}$	(74%)	4185 $^{+0}_{-0}$	(82%)
`haetae2`	`ref`	9265 $^{+49825}_{-7549}$	(25%)	32068 $^{+153018}_{-25792}$	(43%)	1154 $^{+450}_{-50}$	(45%)
	`m4f`	9184 $^{+34372}_{-7629}$	(27%)	26104 $^{+95950}_{-21385}$	(57%)	918 $^{+0}_{-0}$	(54%)
`haetae3`	`ref`	17553 $^{+59078}_{-14530}$	(30%)	44320 $^{+116183}_{-34537}$	(43%)	2097 $^{+890}_{-99}$	(50%)
	`m4f`	14630 $^{+63266}_{-11877}$	(33%)	30588 $^{+159334}_{-23135}$	(57%)	1761 $^{+0}_{-0}$	(57%)

[27] https://groups.google.com/a/list.nist.gov/g/pqc-forum/c/9FXtBZKWueA/m/DojbRt6ZAgAJ

[28] https://developer.arm.com/Tools%20and%20Software/GNU%20Toolchain

Table 2: Average execution speed for key generation, signature generation, and signature verification for each scheme implementation (cont.)

Scheme	impl.	keygen		sign		verify	
haetae5	ref	19940 $^{+84658}_{-16076}$	(31%)	55087 $^{+207542}_{-43097}$	(44%)	2593 $^{+1186}_{-132}$	(54%)
	m4f	19447 $^{+92871}_{-15916}$	(34%)	42365 $^{+162129}_{-33103}$	(57%)	2324 $^{+0}_{-0}$	(58%)
hawk256	ref	16846 $^{+22553}_{-5306}$	(51%)	1116 $^{+1848}_{-161}$	(62%)	628 $^{+0}_{-0}$	(11%)
hawk512	ref	53382 $^{+48360}_{-8110}$	(12%)	1972 $^{+0}_{-0}$	(49%)	1294 $^{+1}_{-1}$	(9%)
hawk1024	ref	231721 $^{+308765}_{-47959}$	(5%)	4310 $^{+0}_{-0}$	(49%)	2782 $^{+2}_{-2}$	(8%)
biscuit128f	ref	1055 $^{+0}_{-0}$	(54%)	274072 $^{+0}_{-0}$	(10%)	254371 $^{+0}_{-0}$	(9%)
biscuit192f	ref	1886 $^{+0}_{-0}$	(54%)	765314 $^{+0}_{-0}$	(6%)	713413 $^{+0}_{-0}$	(6%)
biscuit256f	ref	3302 $^{+0}_{-0}$	(54%)	1747188 $^{+0}_{-0}$	(4%)	1678999 $^{+0}_{-0}$	(4%)
mayo1	ref	7977 $^{+0}_{-0}$	(39%)	18005 $^{+0}_{-0}$	(18%)	6294 $^{+0}_{-0}$	(50%)
	m4f	5242 $^{+0}_{-0}$	(60%)	9101 $^{+0}_{-0}$	(35%)	4953 $^{+0}_{-0}$	(63%)
mayo2	ref	18433 $^{+0}_{-0}$	(23%)	23547 $^{+0}_{-0}$	(18%)	5494 $^{+0}_{-0}$	(76%)
	m4f	11918 $^{+0}_{-0}$	(35%)	11980 $^{+0}_{-0}$	(36%)	5130 $^{+0}_{-0}$	(81%)
mayo3	m4f	18947 $^{+0}_{-0}$	(55%)	32477 $^{+0}_{-0}$	(33%)	16853 $^{+0}_{-0}$	(62%)
ov-Ip	ref	350784 $^{+0}_{-0}$	(3%)	6479 $^{+0}_{-0}$	(0%)	1301 $^{+0}_{-0}$	(1%)
	m4f	139186 $^{+0}_{-0}$	(8%)	2705 $^{+2157}_{-240}$	(1%)	994 $^{+3}_{-15}$	(1%)
ov-Ip-pkc	ref	375130 $^{+0}_{-0}$	(3%)	6924 $^{+0}_{-0}$	(0%)	11430 $^{+0}_{-0}$	(80%)
	m4fspeed	175417 $^{+0}_{-0}$	(6%)	2484 $^{+0}_{-0}$	(1%)	11200 $^{+3}_{-16}$	(82%)
	m4fstack	175417 $^{+0}_{-0}$	(6%)	2484 $^{+0}_{-0}$	(1%)	12043 $^{+6}_{-21}$	(81%)
ov-Ip-pkc-skc	ref	375130 $^{+0}_{-0}$	(3%)	241521 $^{+0}_{-0}$	(5%)	12161 $^{+0}_{-0}$	(81%)
	m4fspeed	175417 $^{+0}_{-0}$	(6%)	89193 $^{+0}_{-0}$	(13%)	11987 $^{+4}_{-13}$	(82%)
	m4fstack	175417 $^{+0}_{-0}$	(6%)	89193 $^{+0}_{-0}$	(13%)	12037 $^{+12}_{-33}$	(82%)
snova-24-5-16-4-esk	ref	24841 $^{+1}_{-1}$	(12%)	139248 $^{+9}_{-13}$	(0%)	88454 $^{+1}_{-1}$	(3%)
snova-24-5-16-4-ssk	ref	24772 $^{+1}_{-1}$	(12%)	174091 $^{+125106}_{-13926}$	(2%)	88454 $^{+1}_{-1}$	(3%)
snova-25-8-16-3-esk	ref	35281 $^{+0}_{-0}$	(10%)	65183 $^{+10}_{-10}$	(0%)	42543 $^{+0}_{-0}$	(8%)
snova-25-8-16-3-ssk	ref	35195 $^{+0}_{-0}$	(10%)	93185 $^{+13}_{-11}$	(4%)	42543 $^{+0}_{-0}$	(8%)
snova-28-17-16-2-esk	ref	51178 $^{+0}_{-0}$	(11%)	21283 $^{+11}_{-15}$	(0%)	19180 $^{+0}_{-0}$	(28%)
snova-28-17-16-2-ssk	ref	51065 $^{+0}_{-0}$	(11%)	50759 $^{+9}_{-13}$	(11%)	19180 $^{+0}_{-0}$	(28%)
snova-37-8-16-4-ssk	ref	122024 $^{+1}_{-0}$	(9%)	576130 $^{+387579}_{-43107}$	(2%)	335807 $^{+1}_{-0}$	(3%)
cross-sha2-r-sdp-1-fast	ref	5615 $^{+36}_{-25}$	(90%)	216566 $^{+830}_{-822}$	(86%)	142974 $^{+782}_{-1350}$	(91%)
cross-sha2-r-sdp-3-fast	ref	8201 $^{+7}_{-53}$	(87%)	241882 $^{+166}_{-65}$	(80%)	123737 $^{+585}_{-782}$	(82%)
cross-sha2-r-sdpg-1-fast	ref	2151 $^{+24}_{-35}$	(93%)	116163 $^{+20}_{-43}$	(91%)	87579 $^{+373}_{-365}$	(92%)
cross-sha2-r-sdpg-1-small	ref	2151 $^{+24}_{-35}$	(93%)	391735 $^{+39}_{-69}$	(91%)	368474 $^{+379}_{-658}$	(93%)

Table 2: Average execution speed for key generation, signature generation, and signature verification for each scheme implementation (cont.)

Scheme	impl.	keygen		sign		verify	
cross-sha2-r-sdpg-3-fast	ref	3049 $^{+6}_{-52}$	(90%)	136249 $^{+47}_{-13}$	(87%)	84235 $^{+426}_{-266}$	(87%)
cross-sha2-r-sdpg-5-fast	ref	4302 $^{+6}_{-52}$	(88%)	221766 $^{+56}_{-65}$	(83%)	134737 $^{+422}_{-591}$	(82%)
cross-sha3-r-sdp-1-fast	ref	968 $^{+11}_{-4}$	(67%)	58864 $^{+13}_{-11}$	(61%)	30641 $^{+15}_{-11}$	(72%)
cross-sha3-r-sdp-3-fast	ref	2195 $^{+10}_{-4}$	(68%)	90063 $^{+18}_{-32}$	(58%)	48560 $^{+9}_{-24}$	(62%)
cross-sha3-r-sdpg-1-fast	ref	290 $^{+8}_{-2}$	(72%)	29964 $^{+10}_{-4}$	(75%)	20095 $^{+7}_{-8}$	(77%)
cross-sha3-r-sdpg-1-small	ref	290 $^{+8}_{-2}$	(72%)	102854 $^{+9}_{-6}$	(75%)	75138 $^{+16}_{-8}$	(78%)
cross-sha3-r-sdpg-3-fast	ref	628 $^{+10}_{-2}$	(72%)	43573 $^{+11}_{-8}$	(68%)	27513 $^{+10}_{-7}$	(69%)
cross-sha3-r-sdpg-5-fast	ref	1146 $^{+8}_{-4}$	(71%)	93558 $^{+6}_{-9}$	(66%)	59963 $^{+81}_{-85}$	(67%)
meds13220	ref	47801 $^{+4}_{-4}$	(2%)	1773022 $^{+26}_{-23}$	(5%)	1766410 $^{+66}_{-58}$	(5%)
meds55604	ref	253604 $^{+8}_{-18}$	(2%)	8009980 $^{+43}_{-115}$	(3%)	8320807 $^{+52}_{-96}$	(3%)
aimer-l1-param1	ref	393 $^{+0}_{-0}$	(72%)	32386 $^{+0}_{-0}$	(47%)	31112 $^{+0}_{-0}$	(47%)
aimer-l1-param2	ref	393 $^{+0}_{-0}$	(72%)	79451 $^{+1}_{-0}$	(49%)	78428 $^{+7}_{-6}$	(49%)
aimer-l3-param1	ref	981 $^{+0}_{-0}$	(77%)	90954 $^{+0}_{-0}$	(42%)	88351 $^{+0}_{-0}$	(42%)
mqom_cat1_gf251_fast	ref	7790 $^{+6}_{-10}$	(74%)	149074 $^{+10}_{-6}$	(33%)	136748 $^{+44}_{-57}$	(33%)
mqom_cat1_gf31_fast	ref	8473 $^{+1}_{-1}$	(65%)	243805 $^{+31}_{-15}$	(20%)	244375 $^{+71}_{-53}$	(19%)
mirith_Ia_fast	ref	1304 $^{+0}_{-0}$	(61%)	296733 $^{+2}_{-2}$	(10%)	276068 $^{+21}_{-22}$	(10%)
mirith_Ib_fast	ref	2515 $^{+0}_{-0}$	(61%)	565780 $^{+4}_{-2}$	(7%)	528405 $^{+5}_{-4}$	(6%)
mirith_IIIa_fast	ref	3009 $^{+0}_{-0}$	(64%)	891195 $^{+3}_{-2}$	(7%)	831720 $^{+22}_{-15}$	(7%)
mirith_IIIb_fast	ref	4565 $^{+0}_{-0}$	(65%)	1298812 $^{+2}_{-3}$	(5%)	1214256 $^{+14}_{-22}$	(5%)
mirith_Va_fast	ref	6255 $^{+0}_{-0}$	(65%)	2373351 $^{+6}_{-6}$	(4%)	2233948 $^{+47}_{-55}$	(4%)
mirith_Vb_fast	ref	8808 $^{+0}_{-0}$	(65%)	3406389 $^{+4}_{-4}$	(3%)	3205121 $^{+46}_{-67}$	(3%)
mirith_hypercube_Ia_fast	ref	1304 $^{+0}_{-0}$	(61%)	116967 $^{+35}_{-16}$	(27%)	111503 $^{+14}_{-30}$	(24%)
	opt	996 $^{+0}_{-0}$	(78%)	58998 $^{+25}_{-17}$	(52%)	53603 $^{+17}_{-21}$	(49%)
mirith_hypercube_Ia_short	ref	1304 $^{+0}_{-0}$	(61%)	371003 $^{+20}_{-3}$	(67%)	364003 $^{+19}_{-5}$	(67%)
mirith_hypercube_Ib_fast	ref	2515 $^{+0}_{-0}$	(61%)	205513 $^{+25}_{-25}$	(19%)	199773 $^{+5}_{-7}$	(17%)
	opt	1880 $^{+0}_{-0}$	(79%)	83818 $^{+36}_{-19}$	(45%)	78142 $^{+9}_{-7}$	(42%)
mirith_hypercube_Ib_short	ref	2515 $^{+0}_{-0}$	(61%)	447353 $^{+8}_{-1}$	(57%)	440282 $^{+21}_{-3}$	(57%)
mirith_hypercube_IIIa_fast	ref	3009 $^{+0}_{-0}$	(64%)	323231 $^{+58}_{-42}$	(19%)	313484 $^{+16}_{-22}$	(17%)
mirith_hypercube_IIIa_short	ref	3009 $^{+0}_{-0}$	(64%)	718710 $^{+15}_{-3}$	(55%)	730959 $^{+6}_{-9}$	(56%)
mirith_hypercube_IIIb_fast	ref	4565 $^{+0}_{-0}$	(65%)	450968 $^{+24}_{-36}$	(14%)	441592 $^{+14}_{-9}$	(12%)
mirith_hypercube_IIIb_short	ref	4565 $^{+0}_{-0}$	(65%)	845099 $^{+17}_{-4}$	(47%)	860322 $^{+13}_{-9}$	(48%)
mirith_hypercube_Va_fast	ref	6255 $^{+0}_{-0}$	(65%)	808047 $^{+140}_{-99}$	(13%)	792074 $^{+59}_{-84}$	(12%)

Table 2: Average execution speed for key generation, signature generation, and signature verification for each scheme implementation (cont.)

Scheme	impl.	keygen		sign		verify	
mirith_hypercube_Vb_fast	ref	$8808\ ^{+0}_{-0}$	(65%)	$1134255\ ^{+113}_{-75}$	(11%)	$1117091\ ^{+19}_{-34}$	(10%)
perk-128-fast-3	ref	$698\ ^{+7}_{-4}$	(59%)	$217643\ ^{+409}_{-309}$	(28%)	$96371\ ^{+1308}_{-400}$	(62%)
	m4	$595\ ^{+4}_{-7}$	(69%)	$175927\ ^{+4141}_{-4014}$	(67%)	$81048\ ^{+511}_{-272}$	(72%)
perk-128-fast-5	ref	$911\ ^{+43}_{-2}$	(51%)	$215520\ ^{+719}_{-481}$	(27%)	$93954\ ^{+1388}_{-483}$	(60%)
	m4	$733\ ^{+28}_{-2}$	(63%)	$169043\ ^{+6630}_{-4737}$	(66%)	$78118\ ^{+509}_{-438}$	(71%)
perk-128-short-3	m4	$595\ ^{+4}_{-7}$	(69%)	$976147\ ^{+41536}_{-26439}$	(64%)	$477798\ ^{+18823}_{-19046}$	(65%)
perk-128-short-5	m4	$733\ ^{+28}_{-2}$	(63%)	$902935\ ^{+44104}_{-39106}$	(63%)	$445552\ ^{+18846}_{-19329}$	(64%)
perk-192-fast-3	m4	$1422\ ^{+43}_{-5}$	(73%)	$420038\ ^{+7735}_{-7607}$	(68%)	$194755\ ^{+1390}_{-1202}$	(72%)
perk-192-fast-5	m4	$1679\ ^{+46}_{-14}$	(68%)	$400224\ ^{+9757}_{-11504}$	(67%)	$187024\ ^{+993}_{-768}$	(71%)
perk-192-short-3	m4	$1422\ ^{+43}_{-5}$	(73%)	$2405700\ ^{+92909}_{-108003}$	(63%)	$1203963\ ^{+29542}_{-29447}$	(63%)
perk-192-short-5	m4	$1686\ ^{+39}_{-22}$	(68%)	$2222672\ ^{+90220}_{-70349}$	(62%)	$1120718\ ^{+23752}_{-24792}$	(62%)
perk-256-fast-3	m4	$2482\ ^{+50}_{-9}$	(74%)	$872567\ ^{+15040}_{-14028}$	(70%)	$417918\ ^{+2037}_{-2912}$	(75%)
perk-256-fast-5	m4	$2888\ ^{+52}_{-19}$	(68%)	$834202\ ^{+18719}_{-15315}$	(70%)	$398767\ ^{+1968}_{-1190}$	(74%)
perk-256-short-3	m4	$2482\ ^{+50}_{-9}$	(74%)	$5076941\ ^{+110828}_{-61663}$	(65%)	$2650317\ ^{+15593}_{-20683}$	(65%)
perk-256-short-5	m4	$2882\ ^{+57}_{-3}$	(68%)	$4682541\ ^{+89928}_{-119145}$	(64%)	$2454136\ ^{+8774}_{-8640}$	(64%)
ascon-sign-128f-robust	ref	$122506\ ^{+0}_{-0}$	(0%)	$2855798\ ^{+1}_{-0}$	(0%)	$177864\ ^{+4568}_{-3738}$	(0%)
ascon-sign-128f-simple	ref	$69377\ ^{+0}_{-0}$	(0%)	$1629111\ ^{+0}_{-0}$	(0%)	$96768\ ^{+5151}_{-3091}$	(0%)
ascon-sign-128s-robust	ref	$7842367\ ^{+0}_{-0}$	(0%)	$59267553\ ^{+14}_{-16}$	(0%)	$61063\ ^{+2616}_{-4028}$	(0%)
ascon-sign-128s-simple	ref	$4441129\ ^{+0}_{-0}$	(0%)	$33877716\ ^{+16}_{-22}$	(0%)	$34009\ ^{+1442}_{-1534}$	(0%)
ascon-sign-192f-robust	ref	$222614\ ^{+0}_{-0}$	(0%)	$5712961\ ^{+3}_{-2}$	(0%)	$320251\ ^{+4393}_{-4910}$	(0%)
ascon-sign-192f-simple	ref	$128167\ ^{+0}_{-0}$	(0%)	$3345488\ ^{+7}_{-4}$	(0%)	$178458\ ^{+10555}_{-5104}$	(0%)
ascon-sign-192s-robust	ref	$14249839\ ^{+1}_{-2}$	(0%)	$126646611\ ^{+24}_{-19}$	(0%)	$109281\ ^{+6202}_{-7236}$	(0%)
ascon-sign-192s-simple	ref	$8204011\ ^{+0}_{-0}$	(0%)	$74760764\ ^{+23}_{-38}$	(0%)	$61709\ ^{+2204}_{-1276}$	(0%)
sphincs-a-sha2-128f	ref	$30279\ ^{+0}_{-0}$	(43%)	$382271\ ^{+1}_{-1}$	(87%)	$35696\ ^{+2}_{-1}$	(48%)
sphincs-a-sha2-128s	ref	$814837\ ^{+1}_{-1}$	(73%)	$6981930\ ^{+5}_{-6}$	(88%)	$187091\ ^{+2}_{-3}$	(12%)
sphincs-a-sha2-192f	ref	$45931\ ^{+0}_{-0}$	(58%)	$634374\ ^{+2}_{-1}$	(88%)	$35146\ ^{+1}_{-1}$	(46%)
sphincs-a-shake-128f	ref	$61578\ ^{+0}_{-0}$	(72%)	$1188147\ ^{+1}_{-1}$	(96%)	$76330\ ^{+2}_{-2}$	(76%)
sphincs-a-shake-128s	ref	$2342299\ ^{+1}_{-2}$	(91%)	$22926755\ ^{+4}_{-3}$	(97%)	$241835\ ^{+2}_{-3}$	(32%)
sphincs-a-shake-192f	ref	$110028\ ^{+0}_{-0}$	(83%)	$1814954\ ^{+1}_{-1}$	(97%)	$64022\ ^{+1}_{-1}$	(71%)

Table 3: Memory requirements for each scheme implemenation. Code, data and BSS size listed are in bytes, stack usage in 2^{10} byte (i.e., KiB).

		library size			stack usage		
Scheme	impl.	code	data	bss	keygen	sign	verify
dilithium2	clean	7996	0	0	37.4	50.7	35.3
	m4f	18516	0	0	37.4	48.2	35.3
dilithium3	clean	7496	0	0	59.4	77.7	56.3
	m4f	20004	0	0	59.4	67.2	56.3
dilithium5	clean	7784	0	0	95.4	119.7	90.6
	m4f	18312	0	0	95.4	113.2	90.6
haetae2	ref	25568	0	0	25.5	53.1	29.0
	m4f	35708	0	0	19.3	54.3	22.8
haetae3	ref	25980	0	0	42.5	78.5	47.3
	m4f	35936	0	0	28.8	81.4	31.0
haetae5	ref	25688	0	0	53.5	97.8	60.6
	m4f	35692	0	0	33.3	101.4	36.3
hawk256	ref	102015	0	0	7.7	3.2	3.6
hawk512	ref	102027	0	0	14.2	4.7	6.1
hawk1024	ref	102031	0	0	27.2	7.8	11.2
biscuit128f	ref	7580	0	0	0.6	134.0	14.2
biscuit128s	ref	7696	0	0	0.6	1067.6	81.7
biscuit192f	ref	7780	0	0	0.6	259.6	20.2
biscuit192s	ref	7904	0	0	0.6	2193.0	104.7
biscuit256f	ref	8216	0	0	0.7	466.2	32.2
biscuit256s	ref	8248	0	0	0.7	3889.6	144.7
mayo1	ref	26436	8	0	72.7	213.5	390.0
	m4f	19300	8	0	72.7	110.8	430.3
mayo2	ref	24404	8	0	108.9	232.7	263.9
	m4f	17292	8	0	108.9	121.7	271.8
mayo3	ref	30828	8	0	239.0	699.2	1110.7
	m4f	23612	8	0	239.0	332.7	458.8
ov-Ip	ref	29215	0	0	15.2	12.4	6.0
	m4f	118939	0	0	15.2	5.1	2.5
ov-Ip-pkc	ref	29423	0	0	15.2	12.4	277.9
	m4fspeed	119131	0	0	138.8	5.1	274.4
	m4fstack	119059	0	0	138.8	5.1	6.3

Table 3: Memory requirements for each scheme implemenation. (cont.)

Scheme	impl.	code	data	bss	keygen	sign	verify
ov-Ip-pkc-skc	ref	29467	0	0	247.6	247.5	277.9
	m4fspeed	119175	0	0	371.1	237.4	274.4
	m4fstack	119103	0	0	371.1	237.4	6.3
tuov_ip	ref	92760	0	0	3201.4	3517.0	1764.9
tuov_ip_pkc	ref	88400	0	0	502.6	15.7	401.6
tuov_ip_pkc_skc	ref	88448	0	0	736.4	800.5	401.6
tuov_is	ref	49316	0	0	337.6	10.6	1.6
tuov_is_pkc	ref	49504	0	0	740.1	10.6	595.6
tuov_is_pkc_skc	ref	49548	0	0	1082.1	1179.2	595.6
tuov_iii	ref	92576	0	0	981.0	24.0	5.8
tuov_iii_pkc	ref	92712	0	0	2177.7	24.0	1764.9
tuov_iii_pkc_skc	ref	92760	0	0	3201.4	3517.0	1764.9
tuov_v_pkc	ref	86304	0	0	unable to test		
tuov_v_pkc_skc	ref	86352	0	0	unable to test		
snova-24-5-16-4-esk	ref	52132	0	336	165.0	87.9	115.4
snova-24-5-16-4-ssk	ref	52132	0	336	165.0	165.1	115.4
snova-25-8-16-3-esk	ref	28240	0	299	186.2	85.4	119.8
snova-25-8-16-3-ssk	ref	28240	0	299	186.2	186.3	119.8
snova-28-17-16-2-esk	ref	16436	0	280	302.0	124.2	195.0
snova-28-17-16-2-ssk	ref	16436	0	280	302.0	302.1	195.0
snova-37-8-16-4-esk	ref	52440	0	336	625.3	289.1	401.8
snova-37-8-16-4-ssk	ref	52440	0	336	625.3	625.4	401.8
snova-43-25-16-2-esk	ref	15060	0	280	1015.1	407.9	650.6
snova-43-25-16-2-ssk	ref	15060	0	280	1015.1	1015.2	650.6
snova-49-11-16-3-esk	ref	26912	0	299	852.6	361.6	528.0
snova-49-11-16-3-ssk	ref	26912	0	299	852.6	852.7	528.0
snova-60-10-16-4-esk	ref	52588	0	336	1897.8	820.3	1179.7
snova-60-10-16-4-ssk	ref	52588	0	336	1897.8	1897.9	1179.7
snova-61-33-16-2-esk	ref	15020	0	280	2581.5	1027.9	1643.0
snova-61-33-16-2-ssk	ref	15020	0	280	2581.5	2581.6	1643.0
snova-66-15-16-3-esk	ref	27624	0	299	2117.3	876.1	1297.9
snova-66-15-16-3-ssk	ref	27624	0	299	2117.3	2117.4	1297.9

Table 3: Memory requirements for each scheme implemenation. (cont.)

Scheme	impl.	code	data	bss	keygen	sign	verify
cross-sha2-r-sdp-1-fast	ref	14244	0	104	5.1	213.2	103.1
cross-sha2-r-sdp-1-small	ref	15285	0	104	5.1	691.1	314.9
cross-sha2-r-sdp-3-fast	ref	14744	0	128	9.7	317.2	154.4
cross-sha2-r-sdp-3-small	ref	14921	0	128	9.7	1238.9	584.9
cross-sha2-r-sdp-5-fast	ref	14580	0	152	16.4	839.3	401.1
cross-sha2-r-sdp-5-small	ref	14657	0	152	16.4	1661.8	784.1
cross-sha2-r-sdpg-1-fast	ref	18409	0	104	2.7	127.8	67.9
cross-sha2-r-sdpg-1-small	ref	18674	0	104	2.7	455.4	239.7
cross-sha2-r-sdpg-3-fast	ref	19609	0	128	4.4	200.3	105.9
cross-sha2-r-sdpg-3-small	ref	19994	0	128	4.4	737.1	383.9
cross-sha2-r-sdpg-5-fast	ref	18669	0	152	7.2	389.2	208.7
cross-sha2-r-sdpg-5-small	ref	18818	0	152	7.2	1004.5	521.6
cross-sha3-r-sdp-1-fast	ref	14472	0	208	4.6	213.3	103.1
cross-sha3-r-sdp-1-small	ref	15353	0	208	4.6	691.2	314.9
cross-sha3-r-sdp-3-fast	ref	14884	0	208	9.1	317.1	154.1
cross-sha3-r-sdp-3-small	ref	14825	0	208	9.1	1238.9	584.6
cross-sha3-r-sdp-5-fast	ref	14576	0	208	15.8	839.3	400.7
cross-sha3-r-sdp-5-small	ref	14629	0	208	15.8	1661.9	783.7
cross-sha3-r-sdpg-1-fast	ref	18605	0	208	2.3	127.9	67.9
cross-sha3-r-sdpg-1-small	ref	18846	0	208	2.3	455.5	239.8
cross-sha3-r-sdpg-3-fast	ref	19689	0	208	3.9	200.3	105.6
cross-sha3-r-sdpg-3-small	ref	19846	0	208	3.9	737.2	383.6
cross-sha3-r-sdpg-5-fast	ref	18593	0	208	6.7	389.3	208.3
cross-sha3-r-sdpg-5-small	ref	18762	0	208	6.7	1004.5	521.2
meds9923	ref	16720	0	0	36.8	973.8	98.1
meds13220	ref	16844	0	0	43.0	176.1	46.6
meds134180	ref	9180	0	0	392.5	853.4	375.7
meds167717	ref	9152	0	0	448.7	567.0	420.3
meds41711	ref	8948	0	0	135.6	1268.2	172.2
meds55604	ref	9012	0	0	158.3	383.6	156.5
aimer-l1-param1	ref	19302	468	0	10.7	183.2	192.1
aimer-l1-param2	ref	19894	468	0	10.7	432.2	441.1

Table 3: Memory requirements for each scheme implemenation. (cont.)

Scheme	impl.	code	data	bss	keygen	sign	verify
aimer-l1-param3	ref	19658	468	0	10.7	1390.4	1399.3
aimer-l3-param1	ref	23398	468	0	23.3	404.1	425.4
aimer-l3-param2	ref	23570	468	0	23.3	1027.7	1049.0
aimer-l5-param1	ref	28142	468	0	57.0	821.4	876.1
aimer-l5-param2	ref	28490	468	0	57.0	2035.7	2090.4
mqom_cat1_gf251_fast	ref	16865	0	0	180.0	390.0	246.7
mqom_cat1_gf251_short	ref	18193	0	0	180.0	649.8	371.0
mqom_cat1_gf31_fast	ref	23718	0	0	256.7	598.5	411.6
mqom_cat1_gf31_short	ref	23510	0	0	256.7	847.8	541.1
mqom_cat3_gf251_fast	ref	17074	0	0	665.0	1254.4	803.1
mqom_cat3_gf251_short	ref	18710	0	0	665.0	1840.4	1116.9
mqom_cat3_gf31_fast	ref	23881	0	0	952.0	2099.4	1500.2
mqom_cat3_gf31_short	ref	24025	0	0	952.0	2606.9	1732.4
mqom_cat5_gf251_fast	ref	20401	0	0	1662.4	3146.7	2115.1
mqom_cat5_gf251_short	ref	20137	0	0	1662.4	4019.1	2477.9
mirith_Ia_fast	ref	7610	256	0	10.0	119.6	22.2
mirith_Ia_short	ref	7635	256	0	10.0	986.7	92.1
mirith_Ib_fast	ref	7602	256	0	18.7	147.5	32.9
mirith_Ib_short	ref	7675	256	0	18.7	1156.7	117.1
mirith_IIIa_fast	ref	7720	256	0	21.4	260.2	45.0
mirith_IIIa_short	ref	7889	256	0	21.4	2130.0	139.8
mirith_IIIb_fast	ref	7760	256	0	32.1	291.2	57.7
mirith_IIIb_short	ref	7901	256	0	32.1	2314.5	162.3
mirith_Va_fast	ref	7890	256	0	44.0	473.1	83.2
mirith_Va_short	ref	7987	256	0	44.0	3701.9	202.5
mirith_Vb_fast	ref	7914	256	0	61.4	522.6	103.5
mirith_Vb_short	ref	7991	256	0	61.4	3994.6	234.9
mirith_hypercube_Ia_fast	ref	8844	256	0	10.0	75.1	20.4
	opt	10932	0	0	10.0	75.1	20.4
mirith_hypercube_Ia_short	ref	8781	256	0	10.0	212.5	30.4
mirith_hypercube_Ia_shorter	ref	8843	256	0	10.0	1728.9	211.2

Table 3: Memory requirements for each scheme implemenation. (cont.)

Scheme	impl.	code	data	bss	keygen	sign	verify
mirith_hypercube_Ib_fast	ref	8820	256	0	18.7	94.7	30.5
	opt	10908	0	0	18.7	94.7	30.5
mirith_hypercube_Ib_short	ref	8825	256	0	18.7	231.8	40.2
mirith_hypercube_Ib_shorter	ref	8819	256	0	18.7	1748.7	221.0
mirith_hypercube_IIIa_fast	ref	8966	256	0	21.4	163.1	41.6
mirith_hypercube_IIIa_short	ref	9139	256	0	21.4	475.0	55.2
mirith_hypercube_IIIa_shorter	ref	9109	256	0	21.4	3788.1	325.9
mirith_hypercube_IIIb_fast	ref	9002	256	0	32.1	185.1	53.8
mirith_hypercube_IIIb_short	ref	9159	256	0	32.1	497.1	67.0
mirith_hypercube_IIIb_shorter	ref	9093	256	0	32.1	3809.8	337.6
mirith_hypercube_Va_fast	ref	9140	256	0	44.0	301.2	79.2
mirith_hypercube_Va_short	ref	9241	256	0	44.0	832.8	93.8
mirith_hypercube_Va_shorter	ref	9203	256	0	unable to test		
mirith_hypercube_Vb_fast	ref	9160	256	0	61.4	336.8	98.8
mirith_hypercube_Vb_short	ref	9245	256	0	61.4	868.6	112.8
mirith_hypercube_Vb_shorter	ref	9247	256	0	unable to test		
perk-128-fast-3	ref	11053	4	0	7.5	306.0	305.8
	m4	13421	4	0	7.5	23.5	20.2
perk-128-fast-5	ref	11129	4	0	8.8	298.5	298.3
	m4	13493	4	0	8.8	24.6	21.2
perk-128-short-3	ref	31757	4	0	7.5	1524.5	1524.3
	m4	26313	4	0	7.5	27.1	24.6
perk-128-short-5	ref	31905	4	0	8.8	1428.5	1428.3
	m4	26461	4	0	8.8	27.9	25.4
perk-192-fast-3	ref	11377	4	0	14.6	670.5	670.2
	m4	12253	4	0	14.6	46.6	40.4
perk-192-fast-5	ref	11365	4	0	16.5	646.7	646.4
	m4	12277	4	0	16.5	47.6	41.4
perk-192-short-3	ref	31789	4	0	14.6	3388.5	3388.2
	m4	24189	4	0	14.6	50.1	45.6
perk-192-short-5	ref	31813	4	0	16.5	3148.5	3148.2
	m4	24913	4	0	16.5	50.7	46.1

Table 3: Memory requirements for each scheme implemenation. (cont.)

Scheme	impl.	code	data	bss	keygen	sign	verify
perk-256-fast-3	ref	11313	4	0	24.9	1163.9	1163.7
	m4	12337	4	0	24.9	78.4	68.3
perk-256-fast-5	ref	11325	4	0	27.4	1114.5	1114.1
	m4	12337	4	0	27.4	79.0	69.0
perk-256-short-3	ref	31961	4	0	unable to test		
	m4	31905	4	0	24.9	80.3	73.0
perk-256-short-5	ref	31973	4	0	unable to test		
	m4	32989	4	0	27.4	80.1	73.1
ascon-sign-128f-robust	ref	17664	0	0	3.1	2.7	2.9
ascon-sign-128f-simple	ref	17596	0	0	2.6	2.1	2.4
ascon-sign-128s-robust	ref	17972	0	0	3.3	2.7	2.2
ascon-sign-128s-simple	ref	17904	0	0	2.8	2.2	1.7
ascon-sign-192f-robust	ref	17960	0	0	5.9	4.8	4.4
ascon-sign-192f-simple	ref	17912	0	0	4.7	3.6	3.6
ascon-sign-192s-robust	ref	18472	0	0	6.2	4.9	4.2
ascon-sign-192s-simple	ref	18424	0	0	5.0	3.7	3.0
sphincs-a-sha2-128f	ref	6343	0	0	274.6	274.1	274.1
sphincs-a-sha2-128s	ref	6899	0	0	572.2	571.9	571.6
sphincs-a-sha2-192f	ref	6867	0	0	492.3	490.7	490.6
sphincs-a-sha2-192s	ref	7423	0	0	1259.2	1258.4	1258.0
sphincs-a-sha2-256f	ref	7203	0	0	1046.7	1044.6	1044.3
sphincs-a-sha2-256s	ref	7511	0	0	2208.4	2207.1	2206.7
sphincs-a-shake-128f	ref	5552	0	0	274.5	274.0	274.0
sphincs-a-shake-128s	ref	6108	0	0	572.2	571.8	571.5
sphincs-a-shake-192f	ref	5916	0	0	491.7	490.2	490.1
sphincs-a-shake-192s	ref	6428	0	0	1258.6	1257.8	1257.5
sphincs-a-shake-256f	ref	6188	0	0	1046.2	1044.1	1043.8
sphincs-a-shake-256s	ref	6456	0	0	2207.9	2206.5	2206.2

References

AAC+23. Muhammad Rezal Kamel Ariffin, Nur Azman Abu, Terry Lau Shue Chien, Zahari Mahad, Amir Hamzah Abd Ghafar, and Nurul Amiera Sakinah Abdul Jamal. KAZ-SIGN, 2023. Submission to the NIST Additional Digital Signature Scheme Project [NIS23a]. 5, 9

AAS+19. Gorjan Alagic, , Jacob Alperin-Sheriff, Daniel Apon, David Cooper, Quynh Dang, Yi-Kai Liu, Carl Miller, Dustin Moody, Rene Peralta, Ray Perlner, Angela Robinson, and Daniel Smith-Tone. NISTIR8240 – status report on the first round of the nist post-quantum cryptography standardization process. January 2019. https://doi.org/10.6028/NIST.IR.8240. 1

AASA+20. Gorjan Alagic, Jacob Alperin-Sheriff, Daniel Apon, David Cooper, Quynh Dang, John Kelsey, Yi-Kai Liu, Carl Miller, Dustin Moody, Rene Peralta, Ray Perlner, Angela Robinson, and Daniel Smith-Tone. NISTIR8309 – status report on the second round of the nist post-quantum cryptography standardization process, July 2020. https://doi.org/10.6028/NIST.IR.8309. 2

ABB+23a. Najwa Aaraj, Slim Bettaieb, Loïc Bidoux, Alessandro Budroni, Victor Dyseryn, Andre Esser, Philippe Gaborit, Mukul Kulkarni, Victor Mateu, Marco Palumbi, Lucas Perin, and Jean-Pierre Tillich. PERK, 2023. Submission to the NIST Additional Digital Signature Scheme Project [NIS23a]. 5, 7

ABB+23b. Nicolas Aragon, Magali Bardet, Loïc Bidoux, Jesús-Javier Chi-Domínguez, Victor Dyseryn, Thibauld Feneuil, Philippe Gaborit, Antoine Joux, Matthieu Rivain, Jean-Pierre Tillich, and Adrien Vinçotte. RYDE, 2023. Submission to the NIST Additional Digital Signature Scheme Project [NIS23a]. 4, 5, 7

ABB+23c. Nicolas Aragon, Magali Bardet, Loïc Bidoux, Jesús-Javier Chi-Domínguez, Victor Dyseryn, Thibauld Feneuil, Philippe Gaborit, Romaric Neveu, Matthieu Rivain, and Jean-Pierre Tillich. MIRA, 2023. Submission to the NIST Additional Digital Signature Scheme Project [NIS23a]. 4, 5, 7

AP20. Alexandre Adomnicai and Thomas Peyrin. Fixslicing AES-like ciphers: New bitsliced AES speed records on ARM-Cortex M and RISC-V. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2021(1):402–425, Dec. 2020. https://tches.iacr.org/index.php/TCHES/article/view/8739. 3

ARZV+23. Gora Adj, Luis Rivera-Zamarripa, Javier Verbel, Emanuele Bellini, Stefano Barbero, Andre Esser, Carlo Sanna, and Floyd Zweydinger. MiRitH, 2023. Submission to the NIST Additional Digital Signature Scheme Project [NIS23a]. 5, 7

BBB+23a. Marco Baldi, Alessandro Barenghi, Luke Beckwith, Jean-François Biasse, Andre Esser, Kris Gaj, Kamyar Mohajerani, Gerardo Pelosi, Edoardo Persichetti, Markku-Juhani O. Saarinen, Paolo Santini, and Robert Wallace. LESS, 2023. Submission to the NIST Additional Digital Signature Scheme Project [NIS23a]. 5, 6

BBB+23b. Marco Baldi, Alessandro Barenghi, Sebastian Bitzer, Patrick Karl, Felice Manganiello, Alessio Pavoni, Gerardo Pelosi, Paolo Santini, Jonas Schupp, Freeman Slaughter, Antonia Wachter-Zeh, and Violetta Weger. CROSS, 2023. Submission to the NIST Additional Digital Signature Scheme Project [NIS23a]. 4, 5

BBB+24. Slim Bettaieb, Loïc Bidoux, Alessandro Budroni, Marco Palumbi, and Lucas Pandolfo Perin. Enabling perk on resource-constrained devices. Cryptology ePrint Archive, Paper 2024/088, 2024. https://eprint.iacr.org/2024/088. 7

BBD+23. Joppe W. Bos, Olivier Bronchain, Léo Ducas, Serge Fehr, Yu-Hsuan Huang, Thomas Pornin, Eamonn W. Postlethwaite, Thomas Prest, Ludo N. Pulles, and Wessel van Woerden. HAWK, 2023. Submission to the NIST Additional Digital Signature Scheme Project [NIS23a]. 5, 7

BBdSG+23. Carsten Baum, Lennart Braun, Cyprien Delpech de Saint Guilhem, Michael Klooß, Christian Majenz, Shibam Mukherjee, Emmanuela Orsini, Sebastian Ramacher, Christian Rechberger, Lawrence Roy, and Peter Scholl. FAEST, 2023. Submission to the NIST Additional Digital Signature Scheme Project [NIS23a]. 4, 5, 9

BCC+22. Daniel J. Bernstein, Tung Chou, Carlos Cid, Jan Gilcher, Tanja Lange, Varun Maram, Ingo von Maurich, Rafael Misoczki, Ruben Niederhagen, Edoardo Persichetti, Christiane Peters, Nicolas Sendrier, Jakub Szefer, Cen Jung Tjhai, Martin Tomlinson, and Wen Wang. Classic McEliece. Technical report, National Institute of Standards and Technology, 2022. available at https://csrc.nist.gov/projects/post-quantum-cryptography/round-4-submissions. 1

BCC+23a. Gustavo Banegas, Kévin Carrier, André Chailloux, Alain Couvreur, Thomas Debris-Alazard, Philippe Gaborit, Pierre Karpman, Johanna Loyer, Ruben Niederhagen, Nicolas Sendrier, Benjamin Smith, and Jean-Pierre Tillich. Wave, 2023. Submission to the NIST Additional Digital Signature Scheme Project [NIS23a]. 5, 6

BCC+23b. Ward Beullens, Fabio Campos, Sofía Celi, Basil Hess, and Matthias J. Kannwischer. MAYO, 2023. Submission to the NIST Additional Digital Signature Scheme Project [NIS23a]. 5, 8

BCC+23c. Ward Beullens, Fabio Campos, Sofía Celi, Basil Hess, and Matthias J. Kannwischer. Nibbling MAYO: Optimized implementations for AVX2 and Cortex-M4. Cryptology ePrint Archive, Paper 2023/1683, 2023. https://eprint.iacr.org/2023/1683. 8

BCD+23. Ward Beullens, Ming-Shing Chen, Jintai Ding, Boru Gong, Matthias J. Kannwischer, Jacques Patarin, Bo-Yuan Peng, Dieter Schmidt, Cheng-Jhih Shih, Chengdong Tao, and Bo-Yin Yang. UOV, 2023. Submission to the NIST Additional Digital Signature Scheme Project [NIS23a]. 5, 8

BCH+23. Ward Beullens, Ming-Shing Chen, Shih-Hao Hung, Matthias J. Kannwischer, Bo-Yuan Peng, Cheng-Jhih Shih, and Bo-Yin Yang. Oil and vinegar: Modern parameters and implementations. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2023(3):321–365, Jun. 2023. 8

BDN+23. Markus Bläser, Dung Hoang Duong, Anand Kumar Narayanan, Thomas Plantard, Youming Qiao, Arnaud Sipasseuth, and Gang Tang. ALTEQ, 2023. Submission to the NIST Additional Digital Signature Scheme Project [NIS23a]. 5, 9

BKPV23. Luk Bettale, Delaram Kahrobaei, Ludovic Perret, and Javier Verbel. Biscuit, 2023. Submission to the NIST Additional Digital Signature Scheme Project [NIS23a]. 5, 7

CCC+23. Ming-Shing Chen, Yu-Shian Chen, Chen-Mou Cheng, Shiuan Fu, Wei-Chih Hong, Jen-Hsuan Hsiang, Sheng-Te Hu, Po-Chun Kuo, Wei-Bin Lee, Feng-Hao Liu, and Justin Thaler. Preon, 2023. Submission to the NIST Additional Digital Signature Scheme Project [NIS23a]. 5, 9

CCD+23a. Jung Hee Cheon, Hyeongmin Choe, Julien Devevey, Tim Güneysu, Dongyeon Hong, Markus Krausz, Georg Land, Marc Möller, Damien Stehlé, and MinJune Yi. Haetae: Shorter lattice-based fiat-shamir signatures. Cryptology ePrint Archive, Paper 2023/624, 2023. https://eprint.iacr.org/2023/624. 6

CCD+23b. Jung Hee Cheon, Hyeongmin Choe, Julien Devevey, Tim Güneysu, Dongyeon Hong, Markus Krausz, Georg Land, Junbum Shin, Damien Stehlé, and MinJune Yi. HAETAE, 2023. Submission to the NIST Additional Digital Signature Scheme Project [NIS23a]. 5, 6

CNL+23. Jinkyu Cho, Jong-Seon No, Yongwoo Lee, Young-Sik Kim, and Zahyun Koo. Enhanced pqsigRM, 2023. Submission to the NIST Additional Digital Signature Scheme Project [NIS23a]. 5, 6

CNP+23. Tung Chou, Ruben Niederhagen, Edoardo Persichetti, Lars Ran, Tovohery Hajatiana Randrianarisoa, Krijn Reijnders, Simona Samardjiska, and Monika Trimoska. MEDS, 2023. Submission to the NIST Additional Digital Signature Scheme Project [NIS23a]. 5, 6

CSSF+23. Jorge Chavez-Saab, Maria Corte-Real Santos, Luca De Feo, Jonathan Komada Eriksen, Basil Hess, David Kohel, Antonin Leroux, Patrick Longa, Michael Meyer, Lorenz Panny, Sikhar Patranabis, Christophe Petit, Francisco Rodríguez Henríquez, Sina Schaeffler, and Benjamin Wesolowski. SQIsign, 2023. Submission to the NIST Additional Digital Signature Scheme Project [NIS23a]. 5, 6

DGG+23. Jintai Ding, Boru Gong, Hao Guo, Xiaoou He, Yi Jin, Yuansheng Pan, Dieter Schmidt, Chengdong Tao, Danli Xie, Bo-Yin Yang, and Ziyu Zhao. TUOV, 2023. Submission to the NIST Additional Digital Signature Scheme Project [NIS23a]. 5, 8

dPEK+23. Rafael del Pino, Thomas Espitau, Shuichi Katsumata, Mary Maller, Fabrice Mouhartem, Thomas Prest, Mélissa Rossi, and Markku-Juhani Saarinen. Raccoon, 2023. Submission to the NIST Additional Digital Signature Scheme Project [NIS23a]. 5, 7

ENST23. Thomas Espitau, Guilhem Niot, Chao Sun, and Mehdi Tibouchi. SQUIRRELS, 2023. Submission to the NIST Additional Digital Signature Scheme Project [NIS23a]. 5, 7

FI23. Hiroki Furue and Yasuhiko Ikematsu. A new security analysis against MAYO and QR-UOV using rectangular minrank attack. In Junji Shikata and Hiroki Kuzuno, editors, *Advances in Information and Computer Security - 18th International Workshop on Security, IWSEC 2023, Yokohama, Japan, August 29-31, 2023, Proceedings*, volume 14128 of *Lecture Notes in Computer Science*, pages 101–116. Springer, 2023. 9

FIH+23. Hiroki Furue, Yasuhiko Ikematsu, Fumitaka Hoshino, Tsuyoshi Takagi, Kan Yasuda, Toshiyuki Miyazawa, Tsunekazu Saito, and Akira Nagai. QR-UOV, 2023. Submission to the NIST Additional Digital Signature Scheme Project [NIS23a]. 5, 8

FR23. Thibauld Feneuil and Matthieu Rivain. MQOM, 2023. Submission to the NIST Additional Digital Signature Scheme Project [NIS23a]. 5, 7

GCF+23. Louis Goubin, Benoît Cogliati, Jean-Charles Faugère, Pierre-Alain Fouque, Robin Larrieu, Gilles Macario-Rat, Brice Minaud, and Jacques Patarin. PROV, 2023. Submission to the NIST Additional Digital Signature Scheme Project [NIS23a]. 5, 8

HBD+19. Andreas Hulsing, Daniel J. Bernstein, Christoph Dobraunig, Maria Eichlseder, Scott Fluhrer, Stefan-Lukas Gazdag, Panos Kampanakis, Stefan Kolbl, Tanja Lange, Martin M Lauridsen, Florian Mendel, Ruben Niederhagen, Christian Rechberger, Joost Rijneveld, Peter Schwabe, and Jean-Philippe Aumasson. SPHINCS+. Technical report, National Institute of Standards and Technology, 2019. available at https://csrc.nist.gov/projects/post-quantum-cryptography/round-2-submissions. 1

IA24. Yasuhiko Ikematsu and Rika Akiyama. Revisiting the security analysis of snova. Cryptology ePrint Archive, Paper 2024/096, 2024. https://eprint.iacr.org/2024/096. 8

KCC+23. Seongkwang Kim, Jihoon Cho, Mingyu Cho, Jincheol Ha, Jihoon Kwon, Byeonghak Lee, Joohee Lee, Jooyoung Lee, Sangyub Lee, Dukjae Moon, Mincheol Son, and Hyojin Yoon. AIMer, 2023. Submission to the NIST Additional Digital Signature Scheme Project [NIS23a]. 5, 9

KRSS19. Matthias J. Kannwischer, Joost Rijneveld, Peter Schwabe, and Ko Stoffelen. pqm4: Testing and benchmarking nist pqc on arm cortex-m4. Cryptology ePrint Archive, Paper 2019/844, 2019. https://eprint.iacr.org/2019/844. 2

KSSW22. Matthias J. Kannwischer, Peter Schwabe, Douglas Stebila, and Thom Wiggers. Improving software quality in cryptography standardization projects. In *IEEE European Symposium on Security and Privacy, EuroS&P 2022 - Workshops, Genoa, Italy, June 6-10, 2022*, pages 19–30, Los Alamitos, CA, USA, 2022. IEEE Computer Society. 2

LA23. Ignacio Luengo and Martín Avendaño. DME-Sign, 2023. Submission to the NIST Additional Digital Signature Scheme Project [NIS23a]. 5, 8

LDK+19. Vadim Lyubashevsky, Léo Ducas, Eike Kiltz, Tancrède Lepoint, Peter Schwabe, Gregor Seiler, and Damien Stehlé. CRYSTALS-DILITHIUM. Technical report, National Institute of Standards and Technology, 2019. available at `https://csrc.nist.gov/projects/post-quantum-cryptography/round-2-submissions`. 1

LZ23. Dongxi Liu and Raymond K. Zhao. eMLE-Sig 2.0, 2023. Submission to the NIST Additional Digital Signature Scheme Project [NIS23a]. 5, 9

MFG+23. Carlos Aguilar Melchor, Thibauld Feneuil, Nicolas Gama, Shay Gueron, James Howe, David Joseph, Antoine Joux, Edoardo Persichetti, Tovohery H. Randrianarisoa, Matthieu Rivain, and Dongze Yue. SDitH, 2023. Submission to the NIST Additional Digital Signature Scheme Project [NIS23a]. 5, 7

MRPC+23. Gilles Macario-Rat, Jacques Patarin, Benoit Cogliati, Jean-Charles Faugère, Pierre-Alain Fouque, Louis Gouin, Robin Larrieu, and Brice Minaud. Rectangular attack on vox. Cryptology ePrint Archive, Paper 2023/1822, 2023. `https://eprint.iacr.org/2023/1822`. 9

NIS23a. Post-quantum cryptography: Digital signature schemes, 2023. Available at `https://csrc.nist.gov/Projects/pqc-dig-sig/round-1-additional-signatures`. 1, 19, 20, 21, 22

NIS23b. Post-quantum cryptography: Round 4 submissions, 2023. Available at `https://csrc.nist.gov/Projects/post-quantum-cryptography/round-4-submissions`. 1

NP23. Jianfang "Danny" Niu and Daniel Enrique Náger Piazuelo. Xifrat1-Sign.I, 2023. Submission to the NIST Additional Digital Signature Scheme Project [NIS23a]. 5, 10

PCF+23. Jacques Patarin, Benoît Cogliati, Jean-Charles Faugère, Pierre-Alain Fouque, Louis Goubin, Robin Larrieu, Gilles Macario-Rat, and Brice Minaud. VOX, 2023. Submission to the NIST Additional Digital Signature Scheme Project [NIS23a]. 5, 9

PFH+19. Thomas Prest, Pierre-Alain Fouque, Jeffrey Hoffstein, Paul Kirchner, Vadim Lyubashevsky, Thomas Pornin, Thomas Ricosset, Gregor Seiler, William Whyte, and Zhenfei Zhang. Falcon. Technical report, National Institute of Standards and Technology, 2019. available at `https://csrc.nist.gov/projects/post-quantum-cryptography/round-2-submissions`. 1

RBK+23. Stefan Ritterhoff, Sebastian Bitzer, Patrick Karl, Georg Maringer, Thomas Schamberger, Jonas Schupp, Georg Sigl, Antonia Wachter-Zeh, and Violetta Weger. FuLeeca, 2023. Submission to the NIST Additional Digital Signature Scheme Project [NIS23a]. 5, 6

Rod23a. Borja Gómez Rodríguez. 3WISE, 2023. Submission to the NIST Additional Digital Signature Scheme Project [NIS23a]. 5, 8

Rod23b. Borja Gómez Rodríguez. HPPC, 2023. Submission to the NIST Additional Digital Signature Scheme Project [NIS23a]. 5, 8

SAB+19. Peter Schwabe, Roberto Avanzi, Joppe Bos, Léo Ducas, Eike Kiltz, Tancrède Lepoint, Vadim Lyubashevsky, John M. Schanck, Gregor Seiler, and Damien Stehlé. CRYSTALS-KYBER. Technical report, National Institute of Standards and Technology, 2019. available at `https://csrc.nist.gov/projects/post-quantum-cryptography/round-2-submissions`. 1

SF23. Igor Semaev and Martin Feussner. EHTv3 and EHTv4, 2023. Submission to the NIST Additional Digital Signature Scheme Project [NIS23a]. 5, 6

SGJ+23. Vikas Srivastava, Naina Gupta, Arpan Jati, Anubhab Baksi, Jakub Breier, Anupam Chattopadhyay, Sumit Kumar Debnath, and Xiaolu Hou. Ascon-Sign, 2023. Submission to the NIST Additional Digital Signature Scheme Project [NIS23a]. 5, 9

SHDS23. Djiby Sow, Abiodoun Clement Hounkpevi, Sidoine Djimnaibeye, and Michel Seck. EagleSign, 2023. Submission to the NIST Additional Digital Signature Scheme Project [NIS23a]. 5, 6

SS17. Peter Schwabe and Ko Stoffelen. All the AES you need on Cortex-M3 and M4. In *Selected Areas in Cryptology – SAC 2016*, pages 180–194, 2017. `https://eprint.iacr.org/2016/714`. 3

ST23. Daniel Smith-Tone. A total break of the 3wise digital signature scheme. Cryptology ePrint Archive, Paper 2023/1535, 2023. `https://eprint.iacr.org/2023/1535`. 8

WCD+23. Lih-Chung Wang, Chun-Yen Chou, Jintai Ding, Yen-Liang Kuan, Ming-Siou Li, Bo-Shu Tseng, Po-En Tseng, and Chia-Chun Wang. SNOVA, 2023. Submission to the NIST Additional Digital Signature Scheme Project [NIS23a]. 5, 8

YCZ23. Yu Yu, Hongrui Cui, and Kaiyi Zhang. SPHINCS-alpha, 2023. Submission to the NIST Additional Digital Signature Scheme Project [NIS23a]. 5, 9

YJL+23. Yang Yu, Huiwen Jia, Leibo Li, Delong Ran, Zhiyuan Qiu, Shiduo Zhang, Xiuhan Lin, and Xiaoyun Wang. HuFu, 2023. Submission to the NIST Additional Digital Signature Scheme Project [NIS23a]. 5, 7