Classic McEliece on the ARM Cortex-M4 (extended abstract of ia.cr/2021/492)

Ming-Shing Chen

mschen@crypto.tw

Ruhr University Bochum, Bochum, Germany

Tung Chou

blueprint@crypto.tw

Academia Sinica, Taipei, Taiwan

April 23, 2021

This paper presents a constant-time implementation of Classic McEliece tailored for `stm32f4-Discovery`. The implementation follows the 3rd-round specification. There is no data cache on `stm32f4-Discovery`, but our implementation does not take advantage of this: our implementation does not use secret-dependent memory indices, so it is constant-time even on M4 devices with data caches.

As shown in Table 1, for the level-1 parameter sets `mceliece348864*` (which means `mceliece348864` and `mceliece348864f`), our implementation takes 582 199 cycles for encapsulation and 2 706 681 cycles for decapsulation. The encapsulation time is more than 80 times faster, and our decapsulation time is more than 17 times faster than the corresponding numbers of FrodoKEM, which is often considered as the most conservative lattice-based scheme submitted to the NIST Post-quantum Cryptography Standardization Process (while Classic McEliece is considered the most convervative code-based scheme). For the level-3 parameter sets `mceliece460896*`, our implementation takes 1 081 335 cycles for encapsulation and 6 535 186 cycles for decapsulation. We note that the cycle counts in Table 1, along with all other cycles counts for our implementation are measured at the maximum frequency 168 Mhz of the device unless specified otherwise.

In addition to encapsulation and decapsulation for all level-1 and level-3 parameter sets, our implementation is also able to carry out key generation for `mceliece348864` and `mceliece348864f` on the device, which takes 2 146 932 033 cycles (12.8 seconds) and 1 430 811 294 cycles (8.5 seconds), respectively. Our implementation is also able to carry out decapsulation for the level-5 parameter sets `mceliece6688128*` and

parameter set	level	decapsulation	encapsulation	key generation
`mceliece348864f`	1	2 706 681	582 199	1 430 811 294
`mceliece348864`	1			2 146 932 033
`mceliece460896*`	3	6 535 186	1 081 335	
`mceliece6688128*`	5	7 412 111		
`mceliece8192128*`	5	7 481 747		

Table 1: Cycle counts for encapsulation and decapsulation in our implementation. We use `*` to mean both the "non-f" parameter set (simply removing `*`) and the corresponding "f" parameter set (replacing `*` by f). Note that a non-f parameter set and the corresponding f parameter set share the same encapsulation and decapsulation algorithms. The cycle counts for encapsulation and key generation are average numbers of 100 measurements. All the cycle counts are measured at the maximum frequency 168 MHz.

`mceliece8192128*` on the development board. We have not implemented decapsulation for `mceliece6960119*`, but we do not see any reason why it cannot run on the device.

The reason why we are able to perform encapsulation and key generation (for some parameter sets) is because we are able store the public key in the 1MB of flash memory on the device, and the amount of SRAM required to perform the operations turns out to be much smaller than the size of the public key.

Our current implementation is not able to carry out encapsulation for the level-5 parameter sets and key generation for the level-3 and level-5 parameter sets. As one might have expected, this is due to the size limit of SRAM and flash memory on `stm32f4-Discovery`. However, the reader should be aware that this does not mean that the operations cannot be carried out on all M4 platforms. For example, we believe that all the three operations of all the 10 parameter sets can be carried out on some of the "Giant Gecko" microcontrollers manufactured by Silicon Labs, where 512KB of SRAM and 2MB of flash memory are available. The optimization techniques presented in this paper are expected to be useful for implementing Classic McEliece on such larger devices.