From: Johann Großschädl <johann.groszschaedl@gmail.com>
Sent: Thursday, July 25, 2019 3:23 PM
To: lightweight-crypto
Cc: lwc-forum@list.nist.gov
Subject: OFFICIAL COMMENT: SPARKLE

Dear LWC-Forum,

The SPARKLE team has developed new Assembler implementations of the SPARKLE256/384/512 permutation family for the ARMv7-M architecture, e.g. Cortex-M3 and M4 microcontrollers. These new implementations are between 2.7 and 3.2 times faster than the timings given in Section 5.4 of the specification document that we submitted in March. The following table summarizes our results and compares them with some other ARX-based and ARX-like permutations for which the designers provided optimized Assembler code for ARMv7-M.

Permutation	Execution time	Code size
Gimli	1041 cycles	3950 bytes
SNEIK 1.1 fast (6r)	1227 cycles	568 bytes
SNEIK 1.1 fast (7r)	1427 cycles	568 bytes
SNEIK 1.1 fast (8r)	1627 cycles	568 bytes
SPARKLE256 (slim)	632 cycles	348 bytes
SPARKLE384 (slim)	1016 cycles	476 bytes
SPARKLE512 (slim)	1528 cycles	624 bytes
Xoodoo	657 cycles	2376 bytes

All execution times include the full function-call overhead and were determined with the cycle-accurate instruction-set simulator of Keil MicroVision v5.24.2.0 using a generic Cortex-M3 model as target device.
However, as mentioned on http://www2.keil.com/mdk5/simulation, the simulator assumes ideal conditions for memory accesses and does not simulate wait states for data or code fetches. Therefore, the timings in the table above should be seen as *lower bounds* of the actual execution times one will get on a real Cortex-M3 device. The fact that the Keil simulator does not take Flash wait-states into account may also explain why our simulated execution time for the Gimli permutation (1041 cycles) differs slightly from the 1047 cycles specified in Section 5.5 of http://gimli.cr.yp.to/gimli-20170627.pdf.

We took the source code of Gimli, SNEIK, and Xoodoo from [1], [2], and [3], respectively, whereby we converted the former two from the GNU syntax to the Keil syntax. The permutations of Gimli and Xoodoo are fully unrolled, which explains their relatively large code size. On the other hand, the main loop of the SNEIK and SPARKLE permutations is "rolled," and the number of loop iterations can be passed as an argument to the function that implements the permutation. The execution time of SNEIK and SPARKLE could be further improved by full loop unrolling.

The source code of the three SPARKLE permutations is available online at https://www.cryptolux.org/index.php/Sparkle

The current version of the Assembler code can only be used with the Keil tools, but we plan to provide versions for the GNU Assembler within the next few days.

Kind regards,

Johann (for the SPARKLE team)

[1] https://www.cryptolux.org/index.php/Sparkle, function gimli() in the the arm-m3 sub-directory.
[2] http://github.com/PQShield/sneik/tree/master/common/sneik_f512 , function sneik_f512() in the file sneik_f512_armv7_fast.S.
[3] http://github.com/XKCP/XKCP/tree/master/lib/low/Xoodoo , function Xoodoo_Permute_12rounds() in the file OptimizedAsmARM/Xoodoo-uf-armv7m-le-armcc.s