Impact of Compiler Optimization Flags on Side-Channel Information Leakage of SipHash algorithm

Matúš Olekšák, Vojtěch Miškovský
Department of Digital Design
Faculty of Information Technology
CTU in Prague, Czech Republic
{oleksmat,miskovoj}@fit.cvut.cz

***Abstract*—This work presents an experimental evaluation of influence of compiler optimization flags on side-channel information leakage. SipHash was used as a reference algorithm an ARX-based pseudorandom function optimized for short inputs. ChipWhisperer CW308 with various targets was used for the evaluation using guessing entropy of CPA and Welch's t-test. The main contribution of this paper is analysis of impact of each flag and its suitability for implementations minimizing side-channel leakage.**

***Index Terms*—compiler, GCC, SipHash, ARX, side-channel, ChipWhisperer**

I. Introduction

Side-channel attacks are one of the most serious threats to the security of embedded devices as they can break cryptographically secure algorithms. These attacks exploit the fact that the physical properties of the operating cryptography device depend on the data being processed. Side channels include power consumption [8], electromagnetic radiation [4], and even sound [6]. This paper focuses on power consumption.

The leakage of sensitive information in the side channel is influenced by the processor, architecture, machine code and the environment in which the algorithm is executed. However, compiler optimizations has a significant impact on the output of the machine code, resulting in lateral channel leakage. To find out how much difference it makes, tests should be carried out on several architectures commonly used for embedded development.

SipHash [2] was used as the reference algorithm for this paper. It is an ARX-based pseudorandom function, and there was a successful CPA attack on it [10], which was used for comparison between different optimizations in this paper. This algorithm became of interest as it is used in modern automobile platforms.

II. Background

Optimization flag [5] is compiler parameter, which influence performance and/or code size at the expense of compilation time and possibly the ability to debug the program. The difference between the used instructions and the order of the binary code affects the leakage of the side channel. The main contribution of this work is the evaluation of how different optimization flags affect the leakage of the side channel.

There are optimization flags designed for debugging purposes, such as -Og or -O0, but they are not used in production, therefore they are omitted in this work. Some are used to reduce binary size (-Os and -Oz) and most embedded applications are assumed to use -Os. The remaining optimization flags (-O1, -O2, -O3 and -Ofast) are used to improve performance. However, -Ofast disregards strict standards compliance, and is therefore not intended to be used in production, but it was measured for its completeness.

Optimized binary code affects many aspects, e.g., better usage of registers and reduced data transport over the bus. However, it is not the case, the more optimized binary, the less it leaks sensitive information. But there has been no work to consider compiler optimization flags as a source of side-channel leakage or as a countermeasure. As a result, it was decided to measure how optimizations affect the side-channel leakage.

A. SipHash

SipHash is an ARX-based pseudorandom function, which can be used to generate 64-bit MACs. There are four state variables called $v0$-$v3$, the default value of which is "somepseudorandomly generatedbytes" in ASCII. However, other initial values can be used as long as $v0$ and $v1$ are different from $v2$ and $v3$. To make it clear, in this text, the subscript in the name of the variable represents the round number and represents the value of the variable at the beginning of the selected round, variable without subscript represents the initial value.

This research has been supported by the grant VJ02010010 of the Ministry of the Interior of the Czech Republic, "Tools for AI-enhanced Security Verification of Cryptographic Devices" in the program Impakt1 (2022-2025)

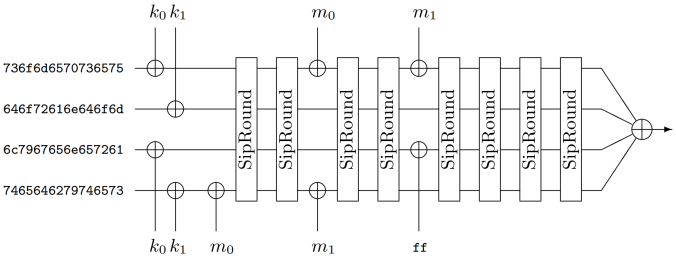


Fig. 1: SipHash Diagram [2].

The 128-bit key is half-transformed to two 64-bit values of $k0$ and $k1$. Before the first round of transformation, $v0_1 = v0 \oplus k0$, $v1_1 = v1 \oplus k1$, $v2_1 = v2 \oplus k0$, $v3_1 = v3 \oplus k1 \oplus m0$, $m0$ represents the first 64-bit text. The default number of rounds is two per 64-bit plain text and four final rounds.

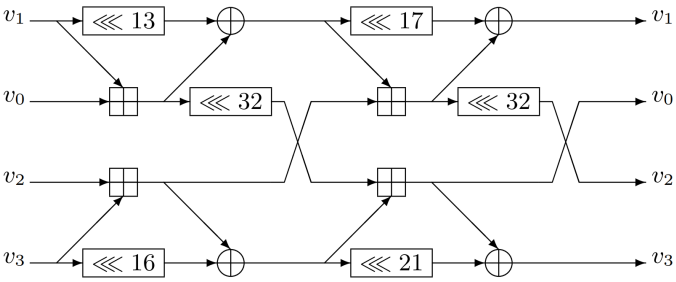


Fig. 2: SipHash Round Diagram [2].

B. CPA Attack

In 2004, Brier et al. [3] introduced the correlation power analysis (CPA), which is a non-profiled side-channel attack. At the beginning of the attack, the power consumption is measured. After this physical prerequisite, all other steps are purely computational. The leakage function must be selected according to the attacked algorithm and its implementation. Secret information (usually the encryption key) must be divided into small parts (e.g. bytes) in order to make its enumeration calculations feasible (e.g. 256 possible key candidates for one byte). Then, using the chosen leakage function, the power consumption of each key candidate is estimated. In the final step, the correlation between the measured and estimated power consumption is calculated. The most correlating key candidate is considered the correct key.

There is a CPA attack [10] on SipHash, which is used as a base for lightened attack used in this paper. In comparison with the originally proposed attack, only the first part of the attack was used in order to retrieve the fifth byte of $k1$ in this work. This approach is sufficient because it only visualizes the difference between leakage of the side channel of different optimizations. More information on the attack consists of the intermediate value $f_{k1}(m0) = v3 \oplus k1 \oplus m0$. As a leakage function, $-HW(f_{k1}(m0))$ is used where $HW(x)$ is Hamming Weight of x. The Pearson correlation between the estimated power consumption by leakage function and the measured power traces is used to distinguish the best key candidate.

III. Related work

To the best of our knowledge, there is no publication on compiler optimization flags and side-channel leakage measurement. The most related publications deal with the security vulnerabilities that are generated by a compiler [7] and the detection of specific instructions. There is research paper on the estimation of electromagnetic radiation by the execution of a particular sequence of instructions [14]. Another publication [11] refers to methods for code generators (not compilers), which preserve first-order security by taking into account specific CPU leakage characteristics. And there is also a publication, where the authors have developed a method of profiling [9], which is capable of detecting executed instructions.

IV. Experiment setup

Initially, the power traces were measured for all the optimizations (-Oz, -Os, -O1, -O2, -O3 and -Ofast). The ChipWhisperer CW308 UFO Board was used to measure three different targets: STM32F0, XMEGA, and MPC5676R to cover the most commonly used architectures for embedded development. Reference implementation [1] of SipHash algorithm was ported to ChipWhisperer and used afterwards. A CPA attack was carried out and the guessing entropy [12] was calculated in range from 2 to 2000 power traces. Values of guessing entropy in tables are calculated from 2 000 power traces. For better visualization of differences, graphs of correlation coefficients for key assumptions were generated. Blue line visualizes correct key correlation coefficient, the yellow line is for $plaintext \oplus v3$ and the red line for all other key candidates. First testing measurements were using 20 000 power traces but it was verified, that even only 2 000 power traces shows the same correlation peaks as higher number of traces, and therefore all shown measurements are generated from 2 000 power traces.

We also performed non-specific Welch's t-test [13] using fixed vs. random plain text and fixed key. It is used to test the hypothesis that two groups have equal means, which can be used to test whether device behave differently with fixed and random plaintext or cipher key.

A. STM32F0

The ChipWhisperer target used for the STM32 platform is the CW308T-STM32F. More precisely it is the STM32F0, which contains a STM32F071RBT6 processor with ARM Cortex M0 core and 128 KB of flash.

Binary files were generated using arm-none-eabi-gcc 13.2.0. The -Oz and -Os optimization flags produce the same binary file, therefore -Oz is omitted. A summary of binary size and guessing entropy for 2000 power traces is in Table I.

Correlation coefficient graphs of STM32F0 are shown on Figure 3. Optimizations -Os (a) and -O1 (b) show significant correlation peak, in contrary to -O2 (c), which has no point with the highest correlation for the best key candidate. -O3 (d) has a single peak, but it is approximately the same size as other 3 peaks. The last one -Ofast (e) looks almost exactly the same like -O3.

TABLE I: A table summarizing the binary size of the different optimization flags of the STM32F0 device.

Optimization flag	Binary size	Guessing Entropy
-Os	12 564 B	10
-O1	13 596 B	9
-O2	13 592 B	159
-O3	15 708 B	22
-Ofast	15 480 B	32

Guessing entropy for all of the optimizations is shown on Figure 4. It is visible that guessing entropy is stabilized from approximately 300 power traces. The flags -O1 and -Os have the lowest guessing entropy. -O3 and -Ofast have slightly higher guessing entropy, in range from 20 to 40. The -O2 flag has made the implementation to have even higher guessing entropy value than 150.

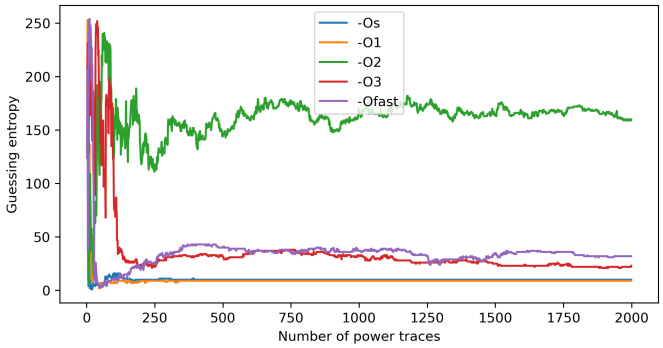


Fig. 4: Graph of STM32F0 Guessing entropy.

B. XMEGA

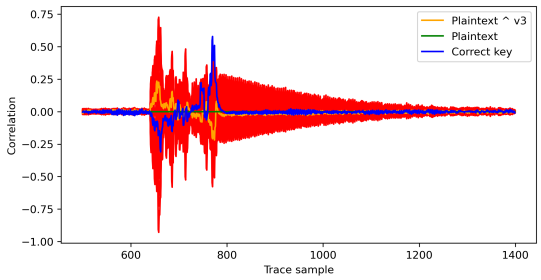
The CW308T-XMEGA target was used for the XMEGA platform. It is based on ATXmega128D4-AU processor with the AVR core architecture and 128 KB of flash.

Compiler avr-gcc 13.2.0 was used for this platform. The -Oz and -Os optimization flags generate the same binary file, therefore -Oz is omitted. A summary of binary size and guessing entropy for 2000 power traces is in Table II.

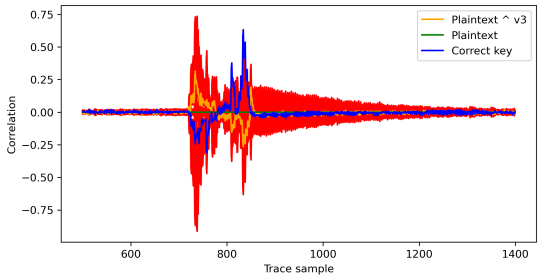
Correlation coefficient graphs of XMEGA are shown on Figure 6. All graphs (a, c, d, e) except -O1 (b) have correlation peaks of correct key, therefore they are vulnerable to the originally presented attack. Even -O1 with some postprocessing can be successfully attacked.

TABLE II: A table summarizing the binary size of the different optimization flags of the XMEGA device.

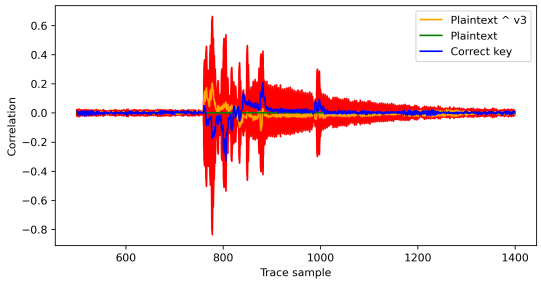
Optimization flag	Binary size	Guessing Entropy
-Os	6 912 B	2
-O1	8 296 B	22
-O2	6 622 B	1
-O3	10 674 B	1
-Ofast	10 674 B	1



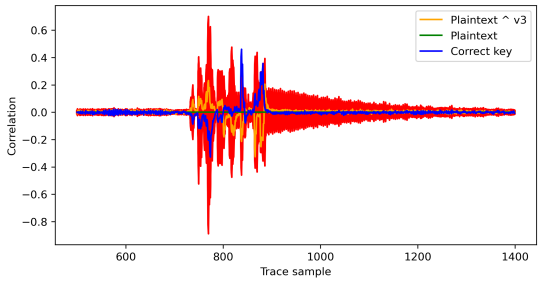
(a) -Os



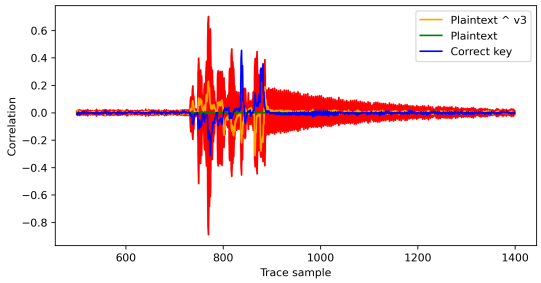
(b) -O1



(c) -O2



(d) -O3



(e) -Ofast

Fig. 3: Graphs of correlation coefficients for STM32 device.

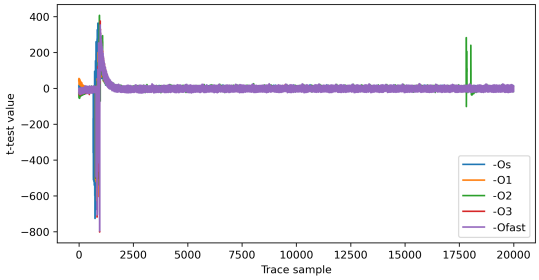


Fig. 5: Graph of Welch t-test for STM32F0 device.

Due to the correlation peaks occurring in all optimization flags with the exception of -O1, the guessing entropy graph on Figure 7 is not very diverse. -O1 optimization flag has guessing entropy around 20, while all the other flags have value of approximately 1.

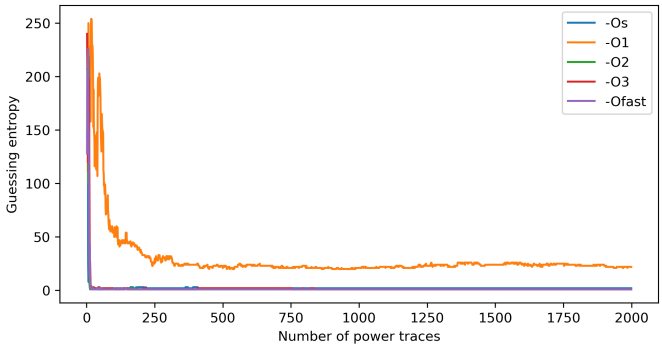


Fig. 7: Graph of XMEGA Guessing entropy.

C. MPC5676R

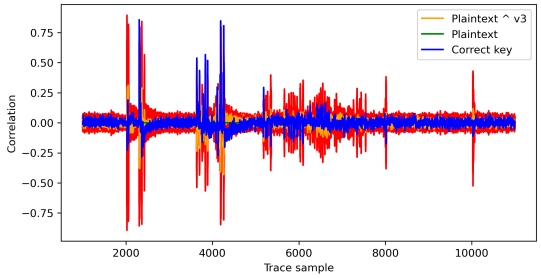
The PowerPC architecture is represented by the ChipWhisperer target CW308T-MPC5676R. It contains a SPC5676RDK3MVU1R processor with an e200z7 core architecture and 6 MB of flash.

The PowerPC architecture uses the powerpc-eabivle-gcc 4.9.4 compiler, which is embedded in the S32 Design Studio for Power Architecture v2.1. A summary of binary size and guessing entropy for 2000 power traces is in Table III.

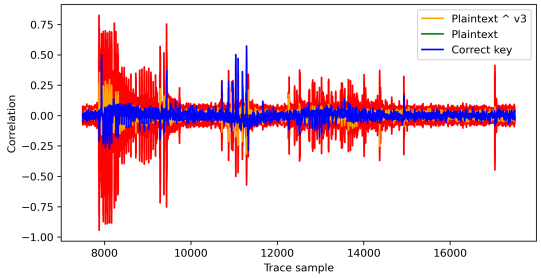
Correlation coefficient graphs of MPC5676R are shown on Figure 9. Graphs of -Os (a) and -O1 (b) optimization contain peaks of correlation of correct key, but it is not the highest point. Rest of optimization flags (c, d, e) has no peaks, and therefore it is not vulnerable to that specific attack.

TABLE III: A table summarizing the binary size of the different optimization flags of the MPC5676R device.

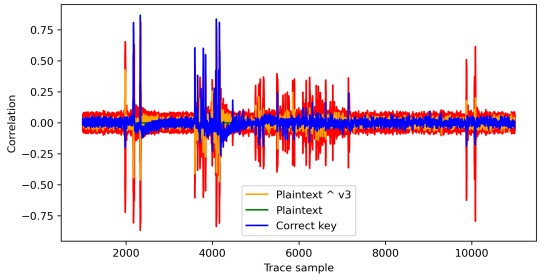
Optimization flag	Binary size	Guessing Entropy
-Os	13 550 B	51
-O1	14 066 B	16
-O2	13 938 B	200
-O3	15 730 B	167
-Ofast	15 730 B	106



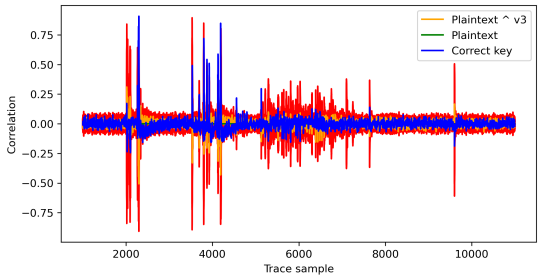
(a) -Os



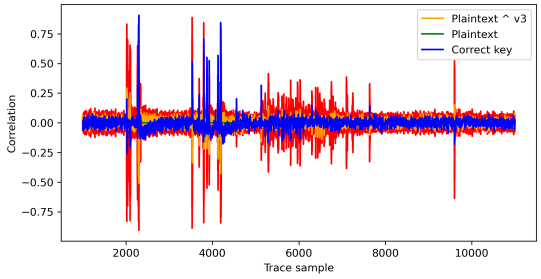
(b) -O1



(c) -O2



(d) -O3



(e) -Ofast

Fig. 6: Graphs of correlation coefficients for XMEGA device.

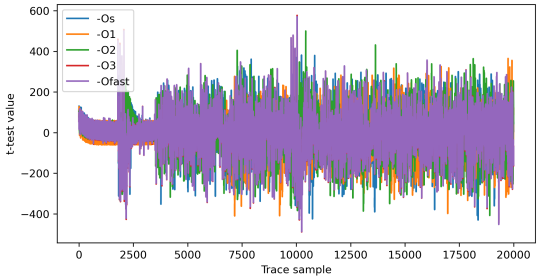


Fig. 8: Graph of Welch t-test for XMEGA device.

The guessing entropy of all optimizations is shown on Figure 10. For the first 300 power traces the guessing entropy oscillates through the entire spectrum of values. From 1300 power traces it is stabilized. -O1 and -Os are the least secure flags, while the other flags have a guessing entropy value of more than 100.

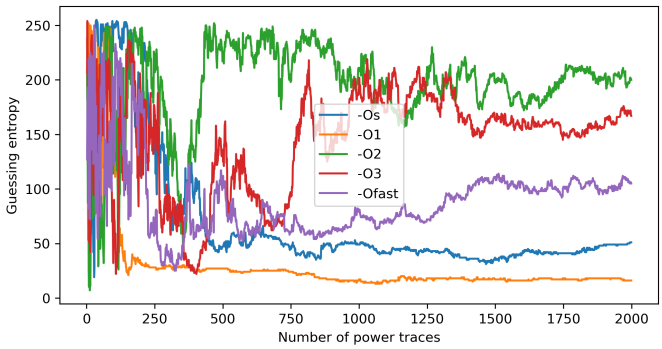


Fig. 10: Graph of MPC5676R Guessing entropy.

V. Flag evaluation

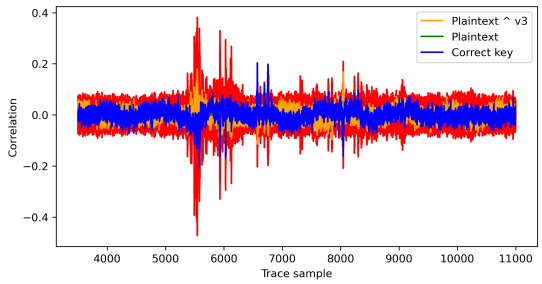
To find out which flags caused difference between -O1 and -O2, the -O2 flag was replaced by all optimization flags it includes and sequentially it was measured with all but one.

The difference of guessing entropy between -O1 and -O2 for all tested architectures was caused by flags -fexpensive-optimizations, -falign-labels, -fcrossjumping, -finline-small-functions, -freorder-blocks-algorithm=stc, -fgcse, -fschedule-insns2, -ftree-loop-vectorize, -ftree-vrp, -fstrict-aliasing. The reduction in STM32F0's guessing entropy of the -O3 optimization flag was caused by -fpeel-loops, -funswitch-loops and -fversion-loops-for-strides.

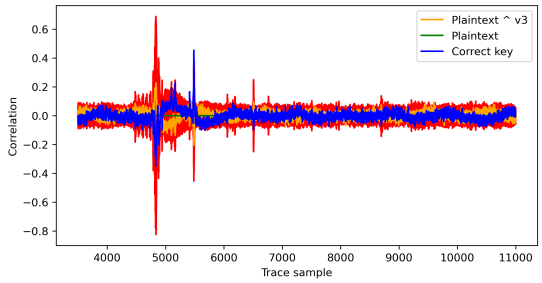
However, there is no noticeable difference in t-value amplitude between the different optimization flags, only slightly different graph courses caused by different binary codes visible in Figure 8 and 11. The only exception is the -O2 optimization flag on the STM32F0 platform, it can be seen in Figure 5, where the t-value peak is visible around sample 18000.

VI. Conclusion

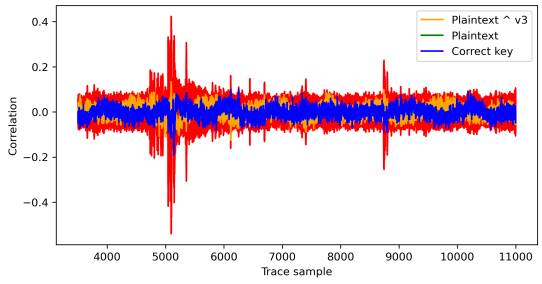
The measurements clearly showed that there is a significant difference between the optimization flags in terms of



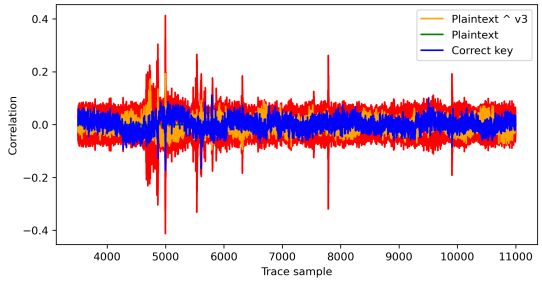
(a) -Os



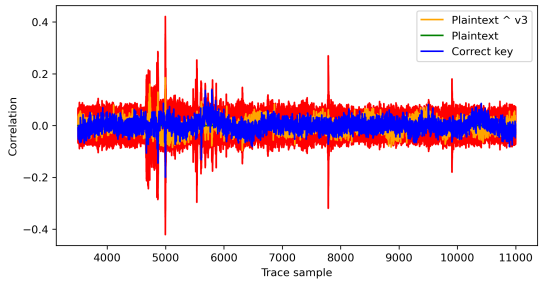
(b) -O1



(c) -O2



(d) -O3



(e) -Ofast

Fig. 9: Graphs of correlation coefficients for MPC5676R device.

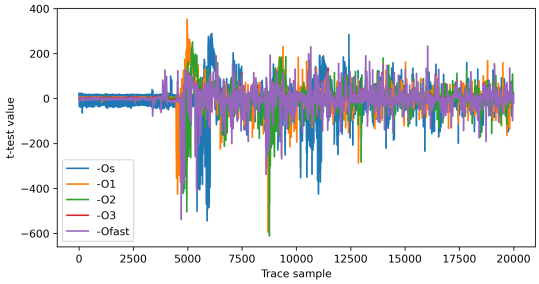


Fig. 11: Graph of Welch t-test for MPC5676R device.

side-channel leakage. That implies developers of embedded systems using not only SipHash but also other cryptographic algorithms must be aware of this phenomenon. Particularly for SipHash on the STM32F0 platform, the flags that make implementation more vulnerable are -fpeel-loops, -funswitch-loops and -fversion-loops-for-strides. It is most evident in the guessing entropy. However, the results of Welch's t-test did not distinguish the various optimizations. This implies that there is a difference in the distribution of the leakage rather than in its quantity. Therefore, the side-channel attacks, including the one which is used in this case study, must adapt to different optimization flags. This should be the subject of further research.

Acknowledgment

This work was supported by the Czech Technical University (CTU) grant No. SGS23/208/OHK3/3T/18.

References

[1] Jean-Philippe Aumasson. Github - veorq/siphash: High-speed secure pseudorandom function for short messages. https://github.com/veorq/SipHash. Accessed: 2024-01-15.

[2] Jean-Philippe Aumasson and Daniel J Bernstein. Siphash: a fast short-input prf. In *International Conference on Cryptology in India*, pages 489–508. Springer, 2012.

[3] Eric Brier, Christophe Clavier, and Francis Olivier. Correlation power analysis with a leakage model. In *Cryptographic Hardware and Embedded Systems-CHES 2004: 6th International Workshop Cambridge, MA, USA, August 11-13, 2004. Proceedings 6*, pages 16–29. Springer, 2004.

[4] Anh Do, S Thet Ko, A Thu Htet, Thomas Eisenbarth, and Berk Sunar. Electromagnetic side-channel analysis on intel atom processor. *Worcester Polytechnic Institute*, 2013.

[5] Inc. Free Software Foundation. Optimize options (using the gnu compiler collection (gcc)). https://gcc.gnu.org/onlinedocs/gcc/Optimize-Options.html. Accessed: 2024-01-15.

[6] Daniel Genkin, Adi Shamir, and Eran Tromer. Rsa key extraction via low-bandwidth acoustic cryptanalysis. In *Advances in Cryptology–CRYPTO 2014: 34th Annual Cryptology Conference, Santa Barbara, CA, USA, August 17-21, 2014, Proceedings, Part I 34*, pages 444–461. Springer, 2014.

[7] Michael J Hohnka, Jodi A Miller, Kenrick M Dacumos, Timothy J Fritton, Julia D Erdley, and Lyle N Long. Evaluation of compiler-induced vulnerabilities. *Journal of Aerospace Information Systems*, 16(10):409–426, 2019.

[8] Paul Kocher, Joshua Jaffe, and Benjamin Jun. Differential power analysis. In *Advances in Cryptology—CRYPTO'99: 19th Annual International Cryptology Conference Santa Barbara, California, USA, August 15–19, 1999 Proceedings 19*, pages 388–397. Springer, 1999.

[9] Mehari Msgna, Konstantinos Markantonakis, and Keith Mayes. Precise instruction-level side channel profiling of embedded processors. In *Information Security Practice and Experience: 10th International Conference, ISPEC 2014, Fuzhou, China, May 5-8, 2014. Proceedings 10*, pages 129–143. Springer, 2014.

[10] Matúš Olekšák and Vojtěch Miškovskỳ. Correlation power analysis of siphash. In *2022 25th International Symposium on Design and Diagnostics of Electronic Circuits and Systems (DDECS)*, pages 84–87. IEEE, 2022.

[11] Hermann Seuschek, Fabrizio De Santis, and Oscar M Guillen. Side-channel leakage aware instruction scheduling. In *Proceedings of the fourth workshop on cryptography and security in computing systems*, pages 7–12, 2017.

[12] François-Xavier Standaert, Tal G Malkin, and Moti Yung. A unified framework for the analysis of side-channel key recovery attacks. In *Advances in Cryptology-EUROCRYPT 2009: 28th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Cologne, Germany, April 26-30, 2009. Proceedings 28*, pages 443–461. Springer, 2009.

[13] Bernard L Welch. The generalization of 'student's'problem when several different population varlances are involved. *Biometrika*, 34(1-2):28–35, 1947.

[14] Baki Berkay Yilmaz, Robert L Callan, Milos Prvulovic, and Alenka Zajić. Capacity of the em covert/side-channel created by the execution of instructions in a processor. *IEEE Transactions on Information Forensics and Security*, 13(3):605–620, 2017.