New Opportunities for Compilers in Computer Security

Junjie Shen[1], Zhi Chen[1], Nahid Farhady Ghalaty[2], Rosario Cammarota[3], Alexandru Nicolau[1], and Alexander V. Veidenbaum[1](✉)

[1] Department of Computer Science, University of California, Irvine, USA
`{junjies1,zhic2,nicolau,alexv}@ics.uci.edu`
[2] Accenture Cyber Security Technology Labs, Arlington, VA, USA
`nahid.farhady@accenture.com`
[3] Qualcomm Technologies, Inc., San Diego, USA
`rosarioc@qti.qualcomm.com`

Abstract. Compiler techniques have been deployed to prevent various security attacks. Examples include mitigating memory access corruption, control flow integrity checks, race detection, software diversity, etc. Hardware fault and side-channel attacks, however, are typically thought to require hardware protection. Attempts have been made to mitigate some timing and fault attacks via compiler techniques, but these typically adversely affected performance and often created opportunities for other types of attacks. More can and should be done in this area by the compiler community.
This paper presents such a compiler approach that simultaneously mitigates two types of attacks, namely a fault and a side-channel attacks. Continued development in this area using compiler techniques can further improve security.

1 Introduction

Compiler techniques have been successfully deployed to prevent various security attacks. Examples include mitigating memory access corruption, such as buffer overflow, in which the attacker attempts to subvert the control flow, e.g., via code reuse attack. Static analysis techniques are used to examine source code to eliminate bugs that can be exploited. Control-Flow Integrity checks the validity of the control-flow of an application. Software diversity is used to generate at compile time a unique binary layout for each compilation, limiting code reuse attacks. It is also applied at binary loading time to provide a different program memory layout for each execution, while the binary is fixed.

Hardware fault and side-channel attacks are harder to deal with and are typically thought to require hardware protection. A fault attack (FA) injects faults into the underlying microprocessor hardware to alter values in registers or memory and affect the execution of instructions. The attacker can then observe the faulty output and finally break the security of the system using systematic fault analysis models, such as Differential Fault Intensity Analysis (DFIA) [11].

A side-channel attack (SCA) may record sequences of measurements (traces), taken across cryptographic operations, such as operation counts, power consumption, execution time, etc. Statistical methods, such as Differential Power Analysis (DPA) [13] and Correlation Power Analysis (CPA) [4], on traces are then used to identify secret key dependent correlations and perform key extraction.

Mitigation strategies for FA and SCA are designed and deployed independently from each other. The former are built on redundancy [2] while the latter are based on masking and hiding [3]. Integration of both mitigation strategies is complex because of the interaction between them. The overhead usually exceeds the combined overhead of the individual mitigation strategies. A single strategy that mitigates both threats at the same time with overhead comparable to a typical mitigation strategy against FA or SCA [16] would be highly beneficial. Equally beneficial would be the integration of the combined mitigation in a compiler, because current implementations rely on manual effort by experts, which is complex and error-prone.

This work highlights one of many opportunities for compiler writers to address physical security, in collaboration with security experts. A compilation flow is developed to use vectorization to make code resistant to both fault and power/electromagnetic attacks. Vectorization is used for operation duplication and data redundancy in registers/memory, *not* for ILP. Furthermore, the duplication is performed in such a way that the Hamming weight of data in a vector register stays constant. The combined approach is referred to as *Twofer*.

To the best of our knowledge, this is the first work that exploits vector extensions to protect cryptographic algorithms against both fault and side-channel attacks within a unified framework.

2 Proposed Mitigation Technique

The compilation framework and the implementation details of the proposed mitigation technique are briefly described next. [8] proposed an approach to code vectorization for vector register value redundancy and operation duplication to mitigate fault attacks. Checking is performed at certain program points, such as stores, function calls, and branches, by comparing the equivalence of two result values in a vector register. In addition, vector gather/scatter instructions were utilized to duplicate address computations.

Memory contents was not duplicated in [8], however, in line with much prior work that relied on memory/cache ECC. This is no longer sufficient due to the recent proliferation of “Rowhammer” based memory attacks. This work, therefore duplicates all variables of the cryptographic primitives and checks the memory contents integrity using the duplicated values.

The proposed mitigation uses 1’s complement to compensate the Hamming weight variance. The compiler can insert additional instructions to invert the result of a memory load before packing the original and inverted values in a vector register to form a Hamming weight-constant vector. However, this will affect the performance significantly since memory loads are very common in block ciphers. More importantly, key-dependent scalar loads are vulnerable to DPA. The size of key-dependent storage, such as *S-box* and cipher state arrays in AES, is therefor increased by 4x. In *Twofer*, the new array duplicates and interleaves the value of each element *val* in the original array with its 1's complement: (val, $\sim val$, val, $\sim val$). The array indices need to be multiplied by four to reflect the correct location in the new array (i.e., stride-indexing). Arrays are also aligned to the cache line size so that every (val, $\sim val$) pair fits in a cache line.

A vector memory load reads four consecutive values from the original memory address simultaneously using the masked vector load primitive. Two data items are used to counter fault attacks and the other two are used to prevent side-channel leakage. Stores are handled similar to loads, but are protected using masked stores. However, vector load and store instructions are more expensive than their scalar counterparts.

Most ALU instructions can be effectively protected against both FA and SCA using the proposed 1's complement approach, but the *xor* operation can cancel the masking effect and increase the Hamming weight. This is because both operand registers for *xor* always contain the original values and their inverted values. Hence, the two related lanes of a result vector register will be identical. An *xor* operation will produce a vector result with two values where one is the inverted value of the other if we invert one of the vector lanes in a vector operand before a *xor* operation. For example, if the two operands are $\langle a, \neg a\rangle$ and $\langle b, \neg b\rangle$, either $\neg a$ or $\neg b$ needs to be inverted. This pre-processing requires an additional *xor* operation, but the performance cost is very low.

3 Evaluation

The evaluation of *Twofer* was performed by comparing it with Scalar in defending against side-channel and fault attacks. It used the cryptographic library GNU Libgcrypt. Pin [14] was used as the base to build a binary instrumentation tool to collect traces, by recording Hamming weights corresponding to a discrete instant in time when data is written into registers during the execution of cryptographic software. This tool provides an accurate representation of the Hamming weight leakage model for cryptographic implementations in software. The evaluation of fault attack resistance was performed by injecting 1,000 single-bit faults at random positions in a cryptographic algorithm execution.

3.1 Resistance Against Side-Channel Attacks

Attacking with DPA. The cumulative differential for each key guess is calculated. DPA will generate 256 differential traces corresponding to each key byte guess. The key byte guess with the highest cumulative differential will be the speculation for the correct byte. The differential analysis is applied on all 16 key byte positions. For brevity, we only include a single key byte in the analysis.

The differential analysis results of a key byte are presented in Fig. 1. DPA is applied on a set of randomly generated keys. The unprotected Scalar is completely broken in the DPA attack with 20,000 traces. On the other hand, *Twofer* remains invulnerable after the differential analysis to the limits of compliance–FIPS 140 [10] and BSI AIS [12] series of recommendations for side-channel resistance of cryptographic software modules. The correct byte is indistinguishable as its cumulative differential is perfectly blended into other guesses. More importantly, the correct byte in *Twofer* shows no trend of standing out as more traces are added in the differential attack.

Attacking with CPA. A more powerful attack, correlation power analysis, is applied on both *Scalar* and *Twofer*. For a given key guess, CPA calculates the Pearson's correlation coefficient between the power hypothesis for the Hamming weight of `SubBytes` output and the actual power usage (traces).

Fig. 2 shows the CPA results on a key byte position. The correlation of the correct byte in *Scalar* remains 1 throughout the experiment. In fact, *Scalar* is completely broken with merely 5 traces. However, *Twofer* shows full mitigation against CPA, benefiting from the constant Hamming weight of key-dependent instructions.

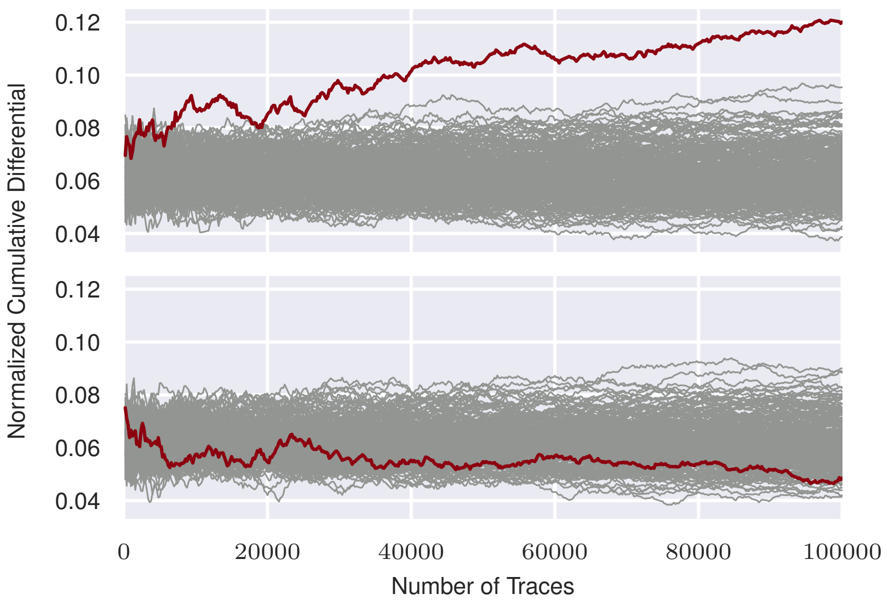

Fig. 1. DPA results of *Scalar* (top) and *Twofer* (bottom) on a key byte.

3.2 Resistance to Fault Attacks

1,000 single-bit faults were injected to each of the ciphers, similar to the approach in [8]. Fig. 3 presents the results, where **detected** are the faults detected by our error checking code; **incomplete** are faults causing segmentation faults or causing the cipher to enter an infinite loop; **masked** are the faults with no effect on the cipher result (i.e. was masked); and **corrupted** shows the faults for which the cipher finishes and generates an incorrect result. The unprotected ciphers have a 24.34% corruption rate, on average, while Twofer reduces it significantly – down to 0.53%.

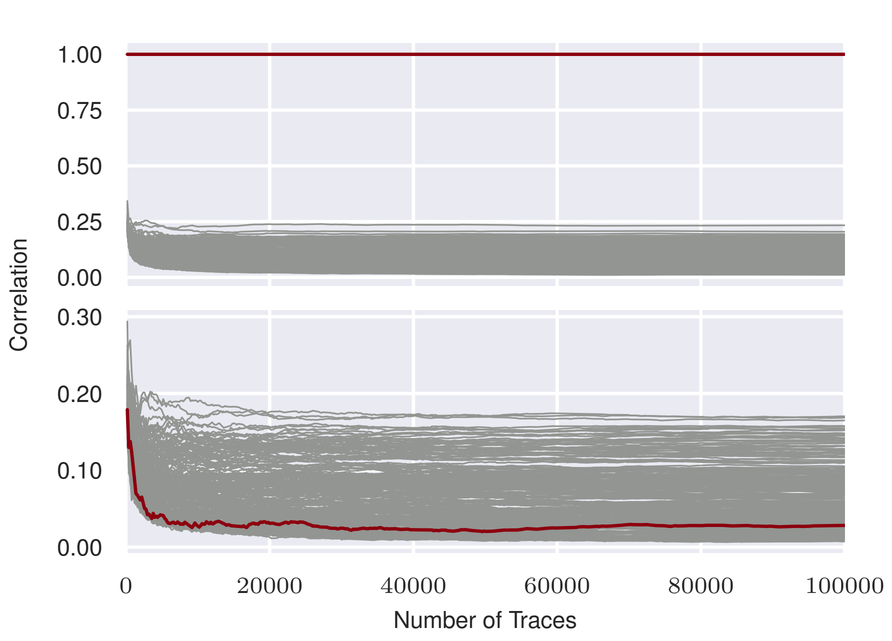

Fig. 2. CPA results of *Scalar* (top) and *Twofer* (bottom) on a key byte. The attacks were performed using 200 to 100,000 traces with a step size of 200.

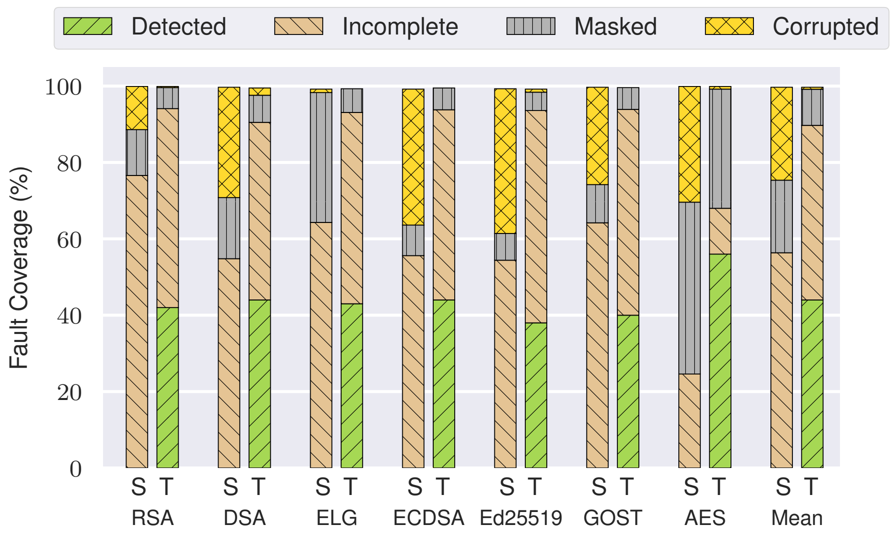

Fig. 3. Fault injection results. *S* for *Scalar* and *T* for *Twofer*.

3.3 Overhead Evaluation

The run time and energy overheads of *Twofer* were collected over 1,000 consecutive cipher operations (private and public for public key ciphers, and encryption and decryption for AES). *Twofer* incurs a reasonable 2.38x slowdown in performance. The overhead is primarily due to (a) extra instructions required for error checking, and (b) the high latency of AVX-512 vector instructions.

The energy consumption was measured with Likwid 4.2.1 [17] using hardware performance counters. The results show that applying *Twofer* imposes a 2.32x energy overhead in comparison to *Scalar*.

Twofer memory overhead is also negligible because cipher states and *S-boxes* only occupy a small fraction of the overall memory footprint in execution. Public ciphers generally have a small cipher state and are dominated by arithmetic calculations.

4 Related Work

Prior work for the mitigation of both fault and side-channel attacks exists. For instance, Wiretap codes that provide resistance against SCA can be used for fault detection up to a certain level of injection [6]. Bringer et al. proposed a smart card friendly Orthogonal Direct Sum Masking technique to protect the AES algorithm against both SCA and FA [5]. This technique is not fully protected against fault attacks because of the author's assumption that generating a fault with a higher Hamming distance is more difficult. An example of such an attack can be DFA attacks that require random byte fault injection [18]. Similar ideas have been proposed in [7]. These proposed techniques also have not been automated, however, unlike the work presented here. The need for combined countermeasures is also increasing due to recent advances towards combined attacks. Examples of such attacks are discussed in [1,9,15].

5 Conclusion

Compiler techniques have not been fully examined in the context of physical attacks and especially in mitigating multiple types of attacks simultaneously. For instance, countermeasures against fault and side-channel attacks rely on different techniques. Integrating both countermeasures is a nontrivial task and often imposes overhead that exceeds the sum of individual countermeasures.

This work demonstrated that a unified compiler-based approach is possible to tackle both fault and side-channel attacks by leveraging redundancy through vectorization along with masking.

References

1. Amiel, F., Villegas, K., Feix, B., Marcel, L.: Passive and active combined attacks: Combining fault attacks and side channel analysis. In: FDTC'07. pp. 92–102. IEEE (2007)

2. Bar-El, H., Choukri, H., Naccache, D., Tunstall, M., Whelan, C.: The sorcerer's apprentice guide to fault attacks. Proceedings of the IEEE **94**(2), 370–382 (2006)
3. Bayrak, A.G., Velickovic, N., Regazzoni, F., Novo, D., Brisk, P., Ienne, P.: An EDA-friendly protection scheme against side-channel attacks. In: DATE'13. pp. 410–415. EDA Consortium (2013)
4. Brier, E., Clavier, C., Olivier, F.: Correlation power analysis with a leakage model. In: CHES'04. pp. 16–29. Springer (2004)
5. Bringer, J., Carlet, C., Chabanne, H., Guilley, S., Maghrebi, H.: Orthogonal direct sum masking. In: WISTP'14. pp. 40–56. Springer (2014)
6. Bringer, J., Chabanne, H., Le, T.H.: Protecting AES against side-channel analysis using wire-tap codes. Journal of Cryptographic Engineering pp. 1–13 (2012)
7. Carlet, C., Guilley, S.: Complementary dual codes for counter-measures to side-channel attacks. Advances in Mathematics of Communications **10**(1) (2016)
8. Chen, Z., Shen, J., Nicolau, A., Veidenbaum, A., Farhady, N.: CAMFAS: A compiler approach to mitigate fault attacks via enhanced simdization. In: FDTC'17. pp. 57–64. IEEE (2017)
9. Clavier, C., Feix, B., Gagnerot, G., Roussellet, M.: Passive and active combined attacks on AES combining fault attacks and side channel analysis. In: FDTC'10. pp. 10–19. IEEE (2010)
10. FIPS, P.: 140-2. Security Requirements for Cryptographic Modules **25** (2001)
11. Ghalaty, N.F., Yuce, B., Taha, M., Schaumont, P.: Differential fault intensity analysis. In: FDTC'14. pp. 49–58. IEEE (2014)
12. Killmann, W., Lange, T., Lochter, M., Thumser, W., Wicke, G.: Minimum requirements for evaluating side-channel attack resistance of elliptic curve implementations. Downloadable via http://www.bsi.bund.de (2011)
13. Kocher, P., Jaffe, J., Jun, B.: Differential power analysis. In: CRYPTO'99. pp. 789–789 (1999)
14. Luk, C.K., Cohn, R., Muth, R., Patil, H., Klauser, A., Lowney, G., Wallace, S., Reddi, V.J., Hazelwood, K.: Pin: building customized program analysis tools with dynamic instrumentation. In: ACM SIGPLAN Notices. vol. 40, pp. 190–200 (2005)
15. Roche, T., Lomné, V., Khalfallah, K.: Combined fault and side-channel attack on protected implementations of AES. CARDIS'11 pp. 65–83 (2011)
16. Schneider, T., Moradi, A., Güneysu, T.: ParTI–towards combined hardware countermeasures against side-channel and fault-injection attacks. In: Annual Cryptology Conference. pp. 302–332 (2016)
17. Treibig, J., Hager, G., Wellein, G.: Likwid: A lightweight performance-oriented tool suite for x86 multicore environments. In: ICPPW'10. pp. 207–216. IEEE (2010)
18. Tunstall, M., Mukhopadhyay, D., Ali, S.: Differential fault analysis of the advanced encryption standard using a single fault. In: WISTP'11. pp. 224–233. Springer (2011)