



Talk: Countering Memory-Centric Side-Channel Leakage via Interleaving

Anna Pätschke* , Jan Wichelmann , and Thomas Eisenbarth 

University of Luebeck, Lübeck, Germany

{a.paetschke,j.wichelmann,thomas.eisenbarth}@uni-luebeck.de

Abstract. Constant-time code is considered the gold-standard for security-critical applications such as cryptographic libraries. However, even constant-time code is vulnerable to advanced memory-centric side-channels such as ciphertext side-channels, silent stores and leakage from data memory-dependent prefetchers. Previous work has explored masking of the plaintext and the rotation of memory locations, with mixed results.

In this talk, we discuss the strengths and weaknesses of current mitigation approaches from the recent literature and propose *interleaving* as a new building block for mitigation implementations. We then consider the design choices, requirements, and the practical complexities involved, and discuss to what extent interleaving can be used to protect against different memory-centric side-channels.

Keywords: system security · side-channel · countermeasure

1 Introduction

Memory-centric side-channel leakages endanger the secrecy of data that is written to memory without further protection, even if code adheres to standards such as constant-time programming [1,6]. Even without secret-dependent branches or memory accesses, there are various ways to inadvertently gather information about secrets. For instance, the deterministic memory encryption of whole-VM trusted execution environments (TEEs) is insufficient to protect secret keys due to ciphertext side-channels [8]. Similarly, microarchitectural optimizations like silent stores invalidate the constant-time properties of code, thus information may be disclosed through memory-centric side-channels [3]. Such disclosed information often even results in leakage of whether a certain written value equals another value already present in memory. In the worst case, this can leak, for example, secret keys bit by bit.

The common denominator of the aforementioned side-channel leakages is their root cause: Values are written to memory without any protection in place. For mitigating ciphertext side-channels and silent store leakage, ensuring freshness of the newly written values inside 16-byte blocks suffices. To ensure that read accesses to data do not leak information about the data stored in memory,

* Presenter

the activation criteria of data memory-dependent prefetchers (DMPs) need to be hindered. Thus, protecting against DMP leakage requires architecture-dependent transformations of the written value so that each 8-byte block contains a protection measure on the first write access [2,7].

2 Countering Memory-Centric Side-Channel Leakage via Interleaving

Introducing freshness of newly written values can be done in three ways [4]: By limiting the reuse of memory locations, by adding random masks to the plaintext, and by interleaving data with counters. Applying address rotations before writing data to memory has been deemed impractical for general applications [4]. To mitigate ciphertext side-channel leakage, Cipherfix [8] employs masking to ensure freshness of values written to memory. While Cipherfix can also protect against leakage from data memory-dependent prefetchers, the overheads introduced by the binary-level mitigation are high. To protect against leakage from silent stores, cio [3] adds tailored transformations to the code which ensure that there is an additional write with a proven different value in between two memory writes to the same address. In that case, the proposed mitigations against silent store leakage do not mitigate ciphertext side-channel leakage.

In this talk, we therefore discuss *interleaving* of (secret) data with counters to protect against the root cause of multiple memory-centric leakages at once. As shown in Figure 1, 16-byte blocks are split into two halves. One half contains an 8-byte counter and the other half is filled with a data chunk. Incrementing the counter on each memory write enforces constantly changing plaintexts which in turn mitigates leakage from ciphertext side-channels and silent stores. Despite some practical constraints like necessary source code adjustments, the performance and security guarantees are promising. For data memory-dependent prefetchers, interleaving must focus on architecture-specific leakage within a smaller block size, making the technical implementation much more complex and less generic than for the other memory-centric side-channel leakages.

Interleaving: Implementation & Evaluation We propose an implementation of interleaving at the middle end compiler level in order to facilitate adjustments of the memory layout. However, this compiler stage with eased memory layout adjustments does not yet have knowledge of all memory writes that are present in the resulting binary, e.g., due to high register pressure during the later register allocation stage. Therefore, we propose to complement the instrumentation with a binary level memory tracing tool that checks the protected binary for leaky memory writes. For using the proposed implementation, the developer of a cryptographic library needs to mark the outmost function of the code to be protected with a `clang` function attribute. An overview of the procedure is given in Figure 2.

Our evaluation of the interleaving implementation for mitigating ciphertext side-channels and silent stores considers practical constraints, performance, and

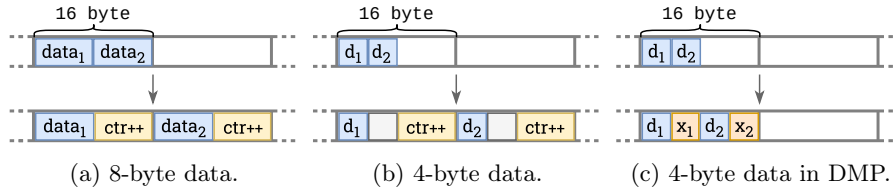


Fig. 1: Interleaving data with counter values in memory. The subfigures (taken from [5]) show different cases of application and resulting block and data chunk sizes. In 1a and 1b, each 16-byte block is split into an 8-byte counter and remaining data. The counter value is incremented on each memory write. In 1c, 8-byte blocks are split into 4-byte chunks containing data (d_i) or data memory-dependent prefetching protection (x_i).

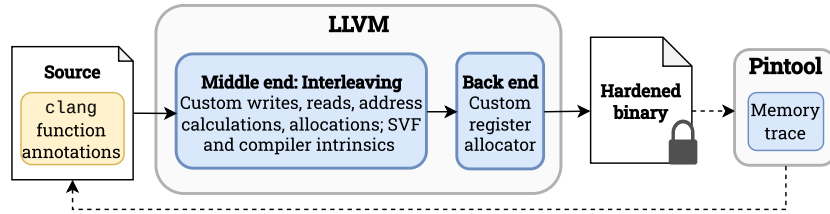


Fig. 2: High-level overview of the proposed *interleaving* implementation.

security. The amount of necessary source code adjustments to integrate the mitigation into cryptographic libraries is higher compared to applying masking. However, despite these practical constraints, interleaving offers advantages in both the performance and security guarantees. We conclude that interleaving introduces a trade-off between practical applicability and versatility on the one hand, and improved performance and security results on the other. For data memory-dependent prefetchers, interleaving must focus on leakage within 8-byte blocks that can potentially be dereferenced as pointers. The technical realization is in this case even more complex than for ciphertext side-channel or silent store leakages as data chunks larger than 4 byte have to be split and recombined. We leave the implementation of this necessary adaption for future work.

Further reading: Some of the findings presented in this talk are also described in a preprint available at [5].

References

1. Barthe, G., Böhme, M., Cauligi, S., Chuengsatiansup, C., Genkin, D., Guarnieri, M., Romero, D.M., Schwabe, P., Wu, D., Yarom, Y.: Testing Side-Channel Security of Cryptographic Implementations Against Future Microarchitectures. In: 2024 ACM

- SIGSAC Conference on Computer and Communications Security (CCS) (2024), <https://doi.org/10.1145/3658644.3670319>
2. Chen, B., Wang, Y., Shome, P., Fletcher, C.W., Kohlbrenner, D., Paccagnella, R., Genkin, D.: GoFetch: Breaking Constant-Time Cryptographic Implementations Using Data Memory-Dependent Prefetchers. In: 33rd USENIX Security Symposium. <https://www.usenix.org/conference/usenixsecurity24/presentation/chen-boru>
 3. Flanders, M., Sharma, R.K., Michael, A.E., Grossman, D., Kohlbrenner, D.: Avoiding Instruction-Centric Microarchitectural Timing Channels Via Binary-Code Transformations. In: 2024 Architectural Support for Programming Languages and Operating Systems (ASPLOS). <https://doi.org/10.1145/3620665.3640400>
 4. Li, M., Wilke, L., Wichelmann, J., Eisenbarth, T., Teodorescu, R., Zhang, Y.: A Systematic Look at Ciphertext Side Channels on AMD SEV-SNP. In: 2022 IEEE Symposium on Security and Privacy (S&P). <https://doi.org/10.1109/SP46214.2022.9833768>
 5. Pättschke, A., Wichelmann, J., Eisenbarth, T.: Zebrafix: Mitigating Memory-Centric Side-Channel Leakage via Interleaving (2025), <https://arxiv.org/abs/2502.09139>
 6. Vicarte, J.R.S., Shome, P., Nayak, N., Trippel, C., Morrison, A., Kohlbrenner, D., Fletcher, C.W.: Opening Pandora’s Box: A Systematic Study of New Ways Microarchitecture Can Leak Private Data. In: 48th ACM/IEEE Annual International Symposium on Computer Architecture (ISCA). <https://doi.org/10.1109/ISCA52012.2021.00035>
 7. Wang, A., Chen, B., Wang, Y., Fletcher, C., Genkin, D., Kohlbrenner, D., Paccagnella, R.: Peek-a-Walk: Leaking Secrets via Page Walk Side Channels. In: 2025 IEEE Symposium on Security and Privacy (S&P). IEEE, <https://www.computer.org/csdl/proceedings-article/sp/2025/223600a023/21B7QepK7Fm>
 8. Wichelmann, J., Pättschke, A., Wilke, L., Eisenbarth, T.: Cipherfix: Mitigating Ciphertext Side-Channel Attacks in Software. In: 32nd USENIX Security Symposium. <https://www.usenix.org/conference/usenixsecurity23/presentation/wichelmann>