SYMLEAK: Quantifying Side Channel Leakage with Symbolic Execution

***Abstract*—The increasing deployment of data-dependent hardware optimizations to achieve performance scaling in a post-Moore's Law era renders microarchitectures increasingly vulnerable to side channel attacks. We propose using microarchitectural leakage functions to precisely model side channels that arise from proposed data-dependent hardware optimizations. Using leakage functions, we model how attacker observations depend on secret program inputs. Our tool, SYMLEAK, uses symbolic execution and ISA simulation to determine the probabilities of attacker observations. From the estimated probability distributions, we quantify the side channel leakage of real world programs. As a case study, we discuss the combined leakage of transmitters in the Poly1305 cryptographic hashing algorithm.**

***Index Terms*—Side channels, security contracts, symbolic execution**

I. INTRODUCTION

In a hardware side channel attack, a *transmitter* modulates a channel (a hardware resource) in a secret-dependent manner. Most often, a transmitter is an instruction in the victim program that causes variability in hardware resource usage as a function of its operands [15]. An attacker observes the channel modulation via its impact on certain non-deterministic aspects of program execution, like timing, resource contention, power dissipation, and more, and thereby infers the operand's value [20]. Hence, when a secret program input is passed to an unsafe transmitter operand, an attacker can gain information about the secret.

State of the art hardware side channel defenses adopt an "all or nothing" strategy, where instruction operands are either entirely safe or unsafe. Hence, unsafe operands can never process secrets. Such is the case in the well-known, constant-time programming discipline [19]. As data-dependent optimizations for performance become more pervasive, the only current mechanism to ensure secure programming is to disable the optimizations. Ultimately, this requires programmers to choose between performance and security [22, 20].

```
path ZERO_SKIP_MUL(MUL i0):
  return ite ((i0.arg0 == 0)∨(i0.arg1 == 0),
    fast_path, slow_path)
```

```
path DIGIT_SERIAL_MUL(MUL i0):
  if (i0.arg1 < 2**8): return path_1
  elif (i0.arg1 < 2**16): return path_2
  elif (i0.arg1 < 2**24): return path_3
  else: return path_4
```

Fig. 1: Zero-skip multiplication leakage function (top) and digit serial multiplication leakage function (bottom)

We use *microarchitectural leakage functions* to model leakage through transmitters. Microarchitectural leakage functions model how a transmitter's *microarchitectural execution path* (μpath) varies with respect to transmitter operands, where a μpath is the partial order of the microarchitectural state updates made by an instruction during its execution [24]. Figure 1 shows the leakage functions for two multiplier optimizations. For example, the first leakage function describes the zero-skip multiplier optimization, where a multiply instruction (`MUL`) takes a "fast" μpath if one or more operand is zero; else, it takes a "slow" μpath. Thus if an attacker observes a "fast" path, they can infer that at least one operand is zero. If a "slow" path is observed, an attacker learns that the operands are non-zero. Intuitively, a different degree of information leaks depending on the `MUL` operands. Therefore, we aim to quantify the leakage of a program using the probability distributions of μpaths conditioned on program inputs.

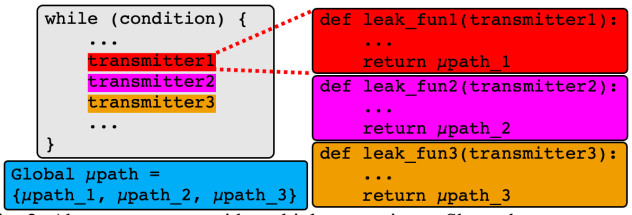

Fig. 2: Abstract program with multiple transmitters. Shows how a program's global μpath is the ordered set of individual transmitter μpaths.

II. BACKGROUND

We define random variables in our communication channel model of a side channel, similar to prior work [23]. The secret space, S, includes all possible secret values. The victim modulation space, X_V, captures the channel modulations caused by the victim program. In our model, channel modulations correspond to the variability in transmitters' μpaths. The observation space, Y, represents the attacker's observations of the channel. The observation space could be same as the victim modulation space, or it could be a projection partitioning of the victim modulation space based on an observer model (i.e. timing or contention). We assume a powerful attacker that can observe instructions' μpaths (the former scenario). Note that prior work also includes victim mitigations and attacker strategies in side channel models. While not included here, these could easily be incorporated in future work.

III. SYMLEAK

A. Goal

As shown in Figure 2, programs may contain many transmitters, with the same or distinct leakage functions. An attacker observes a programs' global μpath, where a global μpath is the ordered set of all transmitters' μpaths across an entire program. Quantifying a program's leakage requires the probability distribution of global μpaths conditioned on secret program inputs. However, reasoning about global μpaths of

Optimization	Description	Total no. of global μpaths	No. of global μpaths seen in simulation	Mutual info of global μpaths
Zero-skip mult [20]	`MUL` takes fast path if one or more operands are 0, else slow path	241	7	0.1125
Digit-serial mult [6]	`MUL` takes 1 of 4 execution paths if bytes of 2nd operand are 0	1372	100	1.5079

TABLE I: Leakage of Poly1305 under two leakage funcs. Computed total no. of global μpaths via symbolic execution and mutual info via ISA simulation.

real-world programs with large secret spaces is infeasible. While model counting could exactly determine the number of secret program inputs that result in each global μpath, current model counting tools do not scale to the size of cryptographic programs' secret spaces. Instead, we use symbolic execution to determine the complete set of a program's global μpaths, and we use Monte Carlo experiments via ISA simulation to approximate the probability of each global μpath. The estimated global μpath probability distribution can be used to approximate program leakage. In this paper, we propose SYMLEAK, which uses symbolic execution and ISA simulation to conservatively, yet precisely, quantify a program's side channel leakage under a user-specified set of leakage functions.

B. Leakage Quantification Metrics

We compare two metrics to quantify program leakage: maximal leakage (L_{ML}) and mutual information (L_{MI}):

$$L_{ML} = \log_2 \sum_{y \in Y} \max_{s \in S} P(y|s)$$

$$L_{MI} = \sum_{s \in S, y \in Y} P(s)P(y|s) \log_2 \frac{P(y|s)}{\sum_{s \in S} P(s)P(y|s)}$$

Maximal leakage measures the maximum probability of each observation given any secret value. Mutual information is the amount of information that the observation space reveals about the secret space. It has been shown that maximal leakage upper bounds mutual information [18]. Thus, prior work conservatively uses maximal leakage when exact leakage functions are unknown [23]. However, maximal leakage may be overly pessimistic, because it overlooks the distribution of the secret space. Leakage functions define the probability of attacker observations conditioned on secret operands. Thus we use mutual information to quantify leakage in SYMLEAK.

C. Symbolic Execution

We use the symbolic execution engine KLEE [4] to enumerate all possible global μpaths of a program. To do so, we implemented an LLVM [2] pass which inserts the leakage function for each corresponding transmitter. For example, the leakage functions in Figure 1 would be inserted for each `MUL`. As a function of the `MUL` (or other transmitter) operands, the program updates a global μpath variable according to the individual transmitter μpaths. Thus the symbolic execution engine returns the number of possible global μpaths for the program under the user-specified leakage function(s).

D. Monte Carlo ISA Simulation

To derive the estimated probabilities of each global μpath, we run Monte Carlo simulations on a RISCV ISA simulator. The simulator is modified such that upon each transmitter, it outputs the instruction's μpath according to the leakage function. Thus by running a large number of trials with random inputs, we estimate the probability of each global μpath.

Due to the large size of the secret space, lower probability global μpaths may not be caught in simulation. Such is the case for an operand value being 0 in 32 bit multiplication. To account for the missing observations in simulation, we assume that each low probability global μpath has a maximum probability of 1 divided by the number of Monte Carlo trials. Then we can compute the mutual information using the estimated probability distribution of global μpaths.

IV. PRELIMINARY RESULTS

As a case study, we consider the cryptographic hashing algorithm, Poly1305 [1, 3], focusing on the repeated secret-dependent `MUL` instructions in the main hash function. We quantify the leakage of the program assuming the two multiplication leakage functions shown in Figure 1: zero skip multiplier, described in I, and digit serial multiplier [6], where a `MUL` can take one of 4 μpaths depending on whether the top-most bytes of the second operand are 0. We use symbolic execution to capture the total number of global μpaths, and we ran 1,000,000 Monte Carlo simulations to estimate the global μpath probability distributions. Results are shown in Table I. We see that many of the global μpaths captured by KLEE are not represented in simulation. Thus these global μpaths had estimated probability of $\frac{1}{10^6}$. We leave it to future work to investigate how to capture more global μpaths in simulation. Most significantly, the results show that there is significantly more leakage under the digit serial multiplier, approximately 1.5 bits, than for zero skip, approximately 0.1 bits.

Conclusion: Leakage functions allow us to be more precise about the risk of side channel leakage. Thus, programs can be designed to protect secrets despite using unsafe instructions.

V. RELATED WORK

Leakage quantification: Several works have evaluated different quantitative metrics for side channel leakage [7, 5, 8, 14, 11, 18, 9]. A few works look at metrics for specific types of side channels [10] or specific mitigation strategies [23]. Our work evaluates how more precise metrics can leveraged by using microarchitectural leakage functions.

Leakage contracts: Hardware-software leakage contracts have been proposed in prior work [17, 21, 20]. These contracts often broadly classify operands of transmitters as safe or unsafe. Microarchitectural leakage functions, on the other hand, return exact μpaths, and therefore allow us to consider the conditions under which said operands are safe or unsafe.

Symbolic execution for leakage analysis: Prior work uses symbolic execution in the realm of side channel discovery and mitigation [13, 16, 12]. However, these approaches often target a single type of side channel, such as cache-based side channel attacks, or they are purely focused on analyzing software. We use symbolic execution to model attacker observations (i.e. μpaths) that result from secrets passed to transmitter operands.

References

[1] The OpenSSL Project. "OpenSSL: The Open Source toolkit for SSL/TLS". www.openssl.org. Apr. 2003.

[2] Chris Lattner and Vikram Adve. "LLVM: A Compilation Framework for Lifelong Program Analysis & Transformation". In: *Proceedings of the International Symposium on Code Generation and Optimization: Feedback-Directed and Runtime Optimization*. CGO '04. Palo Alto, California: IEEE Computer Society, 2004, p. 75. ISBN: 0769521029.

[3] Daniel J. Bernstein. "The Poly1305-AES Message-Authentication Code". In: *Fast Software Encryption*. Ed. by Henri Gilbert and Helena Handschuh. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 32–49. ISBN: 978-3-540-31669-5.

[4] Cristian Cadar, Daniel Dunbar, and Dawson Engler. "KLEE: Unassisted and Automatic Generation of High-Coverage Tests for Complex Systems Programs". In: *Proceedings of the 8th USENIX Conference on Operating Systems Design and Implementation*. OSDI'08. San Diego, California: USENIX Association, 2008, pp. 209–224.

[5] Christelle Braun, Konstantinos Chatzikokolakis, and Catuscia Palamidessi. "Quantitative Notions of Leakage for One-try Attacks". In: *Electronic Notes in Theoretical Computer Science* 249 (2009). Proceedings of the 25th Conference on Mathematical Foundations of Programming Semantics (MFPS 2009), pp. 75–91. ISSN: 1571-0661. DOI: https://doi.org/10.1016/j.entcs.2009.07.085. URL: https://www.sciencedirect.com/science/article/pii/S1571066109003077.

[6] Johann Großschädl et al. *Side-Channel Analysis of Cryptographic Software via Early-Terminating Multiplications*. Cryptology ePrint Archive, Paper 2009/538. https://eprint.iacr.org/2009/538. 2009. URL: https://eprint.iacr.org/2009/538.

[7] Màrio S. Alvim et al. "Measuring Information Leakage Using Generalized Gain Functions". In: *2012 IEEE 25th Computer Security Foundations Symposium*. 2012, pp. 265–279. DOI: 10.1109/CSF.2012.26.

[8] John Demme et al. "Side-Channel Vulnerability Factor: A Metric for Measuring Information Leakage". In: *Proceedings of the 39th Annual International Symposium on Computer Architecture*. ISCA '12. Portland, Oregon: IEEE Computer Society, 2012, pp. 106–117. ISBN: 9781450316422.

[9] Tianwei Zhang et al. "Side Channel Vulnerability Metrics: The Promise and the Pitfalls". In: *Proceedings of the 2nd International Workshop on Hardware and Architectural Support for Security and Privacy*. HASP '13. Tel-Aviv, Israel: Association for Computing Machinery, 2013. ISBN: 9781450321181. DOI: 10.1145/2487726.2487728. URL: https://doi.org/10.1145/2487726.2487728.

[10] Casen Hunger et al. "Understanding contention-based channels and using them for defense". In: *2015 IEEE 21st International Symposium on High Performance Computer Architecture, HPCA 2015* (Mar. 2015), pp. 639–650. DOI: 10.1109/HPCA.2015.7056069.

[11] Isabel Wagner and David Eckhoff. "Technical Privacy Metrics: a Systematic Survey". In: *CoRR* abs/1512.00327 (2015). arXiv: 1512.00327. URL: http://arxiv.org/abs/1512.00327.

[12] Sudipta Chattopadhyay et al. "Quantifying the Information Leak in Cache Attacks through Symbolic Execution". In: *CoRR* abs/1611.04426 (2016). arXiv: 1611.04426. URL: http://arxiv.org/abs/1611.04426.

[13] Tegan Brennan, Seemanta Saha, and Tevfik Bultan. "Symbolic path cost analysis for side-channel detection". In: May 2018, pp. 424–425. DOI: 10.1145/3183440.3195039.

[14] Ibrahim Issa, Aaron B. Wagner, and Sudeep Kamath. "An Operational Approach to Information Leakage". In: *CoRR* abs/1807.07878 (2018). arXiv: 1807.07878. URL: http://arxiv.org/abs/1807.07878.

[15] Vladimir Kiriansky et al. "DAWG: A Defense Against Cache Timing Attacks in Speculative Execution Processors". In: *2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 2018, pp. 974–987. DOI: 10.1109/MICRO.2018.00083.

[16] Robert Brotzman et al. "CaSym: Cache Aware Symbolic Execution for Side Channel Detection and Mitigation". In: *2019 IEEE Symposium on Security and Privacy (SP)*. 2019, pp. 505–521. DOI: 10.1109/SP.2019.00022.

[17] Marco Guarnieri et al. *Hardware-Software Contracts for Secure Speculation*. 2020. arXiv: 2006.03841 [cs.CR].

[18] Benjamin Wu, Aaron B. Wagner, and G. Edward Suh. "A Case for Maximal Leakage as a Side Channel Leakage Metric". In: *CoRR* abs/2004.08035 (2020). arXiv: 2004.08035. URL: https://arxiv.org/abs/2004.08035.

[19] Sunjay Cauligi et al. "SoK: Practical Foundations for Spectre Defenses". In: *CoRR* abs/2105.05801 (2021). arXiv: 2105.05801. URL: https://arxiv.org/abs/2105.05801.

[20] Jose Rodrigo Sanchez Vicarte et al. "Opening Pandora's Box: A Systematic Study of New Ways Microarchitecture Can Leak Private Data". In: *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*. 2021, pp. 347–360. DOI: 10.1109/ISCA52012.2021.00035.

[21] Nicholas Mosier et al. "Axiomatic Hardware-Software Contracts for Security". In: *Proceedings of the 49th Annual International Symposium on Computer Architecture*. ISCA '22. New York, New York: Association for Computing Machinery, 2022, pp. 72–86. ISBN: 9781450386104. DOI: 10.1145/3470496.3527412. URL: https://doi.org/10.1145/3470496.3527412.

[22] Jonathan Corbet. *Constant-time instructions and processor optimizations*. Last accessed 21 July 2024. 2023. URL: https://lwn.net/Articles/921511/.

[23] Peter W. Deutsch et al. "Metior: A Comprehensive Model to Evaluate Obfuscating Side-Channel Defense Schemes". In: *Proceedings of the 50th Annual International Symposium on Computer Architecture*. ISCA '23. Orlando, FL, USA: Association for Computing Machinery, 2023. ISBN: 9798400700958. DOI: 10.1145/3579371.3589073. URL: https://doi.org/10.1145/3579371.3589073.

[24] Yao Hsiao et al. "RTL2MPATH: Multi-PATH Synthesis with Applications to Hardware Security Verification". In: *2024 57th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 2024.