APPENDIX A
PATH-FINDING ALGORITHM

When the type of the branch condition is `secret`, SecDivCon performs an analysis to discover all paths starting from the branch condition (source) to a common node (sink). We assume that the program is split into basic blocks, pieces of code with at most one branch (apart from function calls) at the end of the block. To identify all possible paths, we generate the Control-Flow Graph (CFG) between the basic blocks of the program.

Listing 1 shows the algorithm for extracting the paths that start from a basic block n (the secret-dependent branch), given the CFG (`BCFG`). We use two data structures, a priority queue, `P`, which contains all paths under analysis, and a queue, `t` that represents the current path and starts with the first basic block, n (line 4). The priority queue uses the block order as the priority, with smaller numbers having priority. At line 5, `P` is initialized with `t`. We store the final results in `W` (line 6). At line 7, we start a loop that terminates when there are no paths left to analyze in `P` or when we find a cycle. At lines 8 and 9, we get the top element of the top path from `P`. Subsequently, the algorithm finds all successor nodes in the CFG, which correspond to possible basic blocks that follow the current basic block (line 10). Then, the algorithm performs different actions depending on the successor nodes. First, if the current node, `h` does not have any successors, it means that `h` is an exit node, thus, `h` is the last node in the current path. Lines 12 and 13 add the path to `W` and remove it from the paths under analysis. If `h` has one successor, `s`, then we push the successor to the path and update `P` (lines 14-16). Here, we need to check if the new node leads to the current paths having a sink, i.e. the same final node (line 17). The last case is when the branch is conditional and there are two possible destinations. Here, we need to generate two paths `p1` and `p2` for each of the two destinations and insert them to `P` for further analysis (lines 20-27). When the analysis finishes and the algorithm exits the loop, then it returns `W`.

Listing 1: Path extraction

```
1  GET_PATHS(n, BCFG):
2    t.empty() # Queue - First path
3    P.empty() # Priority queue - Paths
4    t.insert(n)
5    P.insert(t)
6    W.empty() # final paths
7    while (¬P.isempty() and ¬P.hasCycle()):
8      p ← P.top() # Top path
9      h ← p.pop() # Last element of path
10     succ ← BCFG.successors(h)
11     if (succ = ∅): # exit node
12       W.push(p)
13       P.remove(p)
14     elif (succ = {s}):
15       p.push(s)
16       P.replace(p)
17       # if this is a sink, we terminate
18       if (W.extend(P).hasSink()):
19         return W.extend(P)
20     elif (succ = {s1,s2}):
21       p1 ← p.copy()
22       p2 ← p.copy()
23       p1.push(s1)
24       p2.push(s2)
25       P.remove(p)
26       P.insert(p1)
27       P.insert(p2)
28   return W
```

This analysis does not support loops.

APPENDIX B
RELATED WORK

Table VIII shows a representative subset of compiler-based or binary-rewrite contributions against CRAs and SCAs in the literature. For each of these works, we show the publication citation reference (Pub.), the attack it is mitigating (Attack), the type of mitigation the work is proposing (Mitigation), the performance overhead it introduces (PO), and the target language/architecture (Target).

In the literature, multiple approaches use automatic software diversification/randomization against CRAs [51, 13, 60, 59, 14, 12]. These works target mostly x86 systems [51, 13, 59], with the exception of AVRAND [60] and HARM [14], and DivCon [12], which target embedded processors. The main characteristic of these approaches is that they lead to relatively low performance overhead. Recent approaches focus on re-randomization to protect against advanced CRAs such as JIT-ROP and BROP [14, 60, 59]. These approaches may introduce additional performance overhead, which may be low, such as HARM, which introduces up to 6% additional overhead.

Another approach against code-reuse attacks is CFI, which ensures that the dynamic program execution satisfies the intended program control flow [63, 21, 64]. Software-based CFI systems [63, 21] typically result in high overhead, whereas hardware-assisted methods may lead to reduced overhead (24%) [64]. Microguard [7] combines code diversification and CFI and leads to an overhead of up to 70%. Compared to diversification approaches, CFI approaches against CRAs lead, in general, to higher overhead.

Table VIII presents also approaches against SCAs [71, 69, 72, 65, 47]. Rosita [65] targets PSC attacks in small ARM processors and has an overhead of up to 64%. A recent approach, SecCG [31], optimizes masked implementations with a small maximum overhead of 13% compared to non-secure code optimization. The rest of the approaches target mostly TSC attacks [67, 71, 69, 72, 47]. Constantine [67] is an approach to automatically linearize code in order to make it constant time. However, in some cases, the generated code has high performance overhead (5x). Winderix et al. [47] present an approach against TSC and IL (Interrupt Latency) SCAs. Our strategy to mitigate TSC attacks is similar to their approach, while SecDivCon also attempts to minimize the introduced performance overhead. The maximum performance overhead in an MSP430 microcontroller is 60%. Raccoon [69] uses control-flow obfuscation to mitigate TSC attacks. While this is an original approach, Joshi et al. [70] have shown that code obfuscation may increase the code-reuse gadgets in the code.

Pub.	Attack	Mitigation	PO	Target
[51]	CRA	Div	25%	x86
[13]	CRA	Div	0%	x86
[71]	TSC	Div	8x	x86
[69]	TSC	Obf	16x	x86
[60]	CRA	Div, RR	-	AVR
[63]	CRA	CFI	∼80%	ARM
[21]	CRA	CFI	5x	ARM
[72]	TSC, MS	CT	-	C
[59]	CRA	Div, RR	7%	x86
[7]	CRA	Div,CFI	70%	ARM
[12]	CRA	Div	0-20%	Mips
[65]	PSC	SM	64%	ARM
[47]	TSC, IL	BB	60%	MSP430
[14]	CRA	Div, RR	6%	ARM
[67]	TSC	CT	5x	x86
[31]	PSC	SM	13%	Mips, ARM
[64]	CRA	HWCFI	24%	ARM
SDC	TSC,PSC,CRA	Div, SM/BB	70%-80%	MIPS, ARM

TABLE VIII: Related work; CRA stands for code-reuse attacks; TSC stands for timing side-channel attacks; MS stands for memory safety; PSC stands for power side-channel attacks; IL stands for interrupt-latency SCA; Div stands for diversification; Obf stands for obfuscation; CFI stands for control-flow integrity; CT stands for constant-time discipline; SM stands for software masking; BB stands for basic-block balance; RR stands for re-randomization; HWCFI stands for hardware-assisted CFI; PO corresponds to the upper bound of the performance overhead; SDC stands for SecDivCon.

Hence, Raccoon may increase the attacker surface for CRAs, while mitigating SCAs. Moreover, Raccoon introduces high overhead (16x), which is prohibiting for resource-constrained devices. Crane et al. [71] present an original approach against cache-based TSC attacks based on code diversification. The main drawback of this approach is the introduced overhead of up to eight times, which is prohibiting for resource-constrained devices. Finally, HACL* [72] presents an approach to generate cryptographic implementations in C that are constant time (against TSCs) and memory safe (MS). However, HACL* does not prohibit memory vulnerabilities in other parts of the code base, which may lead to CRAs.

In summary, there are different approaches against CRAs and SCAs, but none of them evaluates the effect of combining mitigations against these attacks or ensures that the combination of mitigations against CRAs and SCAs remains effective against both attacks. Our proposed approach, SecDivCon, preserves both code diversification and SCA mitigations with a relatively small performance overhead.

REFERENCES

[1] D. Papp, Z. Ma, and L. Buttyan, "Embedded systems security: Threats, vulnerabilities, and attack taxonomy," in *2015 13th Annual Conference on Privacy, Security and Trust (PST)*, Jul. 2015, pp. 145–152.

[2] H. Shacham, "The Geometry of Innocent Flesh on the Bone: Return-into-libc Without Function Calls (on the x86)," in *Proceedings of the 14th ACM Conference on Computer and Communications Security*, ser. CCS '07. ACM, 2007, pp. 552–561.

[3] R. Roemer, E. Buchanan, H. Shacham, and S. Savage, "Return-Oriented Programming: Systems, Languages, and Applications," *ACM Transactions on Information and System Security*, vol. 15, no. 1, pp. 2:1–2:34, Mar. 2012.

[4] S. Pastrana, J. Tapiador, G. Suarez-Tangil, and P. Peris-López, "AVRAND: A Software-Based Defense Against Code Reuse Attacks for AVR Embedded Devices," in *Detection of Intrusions and Malware, and Vulnerability Assessment*. Springer International Publishing, 2016, vol. 9721, pp. 58–77, series Title: Lecture Notes in Computer Science.

[5] G.-A. Jaloyan, K. Markantonakis, R. N. Akram, D. Robin, K. Mayes, and D. Naccache, "Return-Oriented Programming on RISC-V," in *Proceedings of the 15th ACM Asia Conference on Computer and Communications Security*, ser. ASIA CCS '20, Oct. 2020, pp. 471–480.

[6] T. Bletsch, X. Jiang, V. W. Freeh, and Z. Liang, "Jump-oriented Programming: A New Class of Code-reuse Attack," in *Proceedings of the 6th ACM Symposium on Information, Computer and Communications Security*, ser. ASIACCS '11. ACM, 2011, pp. 30–40.

[7] M. Salehi, D. Hughes, and B. Crispo, "MicroGuard: Securing Bare-Metal Microcontrollers against Code-Reuse Attacks," in *2019 IEEE Conference on Dependable and Secure Computing (DSC)*, Nov. 2019, pp. 1–8.

[8] T. S. Messerges, E. A. Dabbish, and R. H. Sloan, "Investigations of power analysis attacks on smartcards." *Smartcard*, vol. 99, pp. 151–161, 1999.

[9] J. Deogirikar and A. Vidhate, "Security attacks in iot: A survey," in *2017 International Conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud)(I-SMAC)*. IEEE, 2017, pp. 32–37.

[10] M. Devi and A. Majumder, "Side-Channel Attack in Internet of Things: A Survey," in *Applications of Internet of Things*, ser. Lecture Notes in Networks and Systems, J. K. Mandal, S. Mukhopadhyay, and A. Roy, Eds. Springer, 2021, pp. 213–222.

[11] S. T. Vu, A. Cohen, A. De Grandmaison, C. Guillon, and K. Heydemann, "Reconciling optimization with secure compilation," *Proceedings of the ACM on Programming Languages*, vol. 5, no. OOPSLA, pp. 1–30, 2021.

[12] R. M. Tsoupidi, R. Castaneda Lozano, and B. Baudry, "Constraint-based diversification of jop gadgets," *Journal of Artificial Intelligence Research*, vol. 72, pp. 1471–1505, 2021.

[13] V. Pappas, M. Polychronakis, and A. D. Keromytis, "Smashing the Gadgets: Hindering Return-Oriented Programming Using In-place Code Randomization," in *2012 IEEE Symposium on Security and Privacy*, May 2012, pp. 601–615, iSSN: 1081-6011.

[14] J. Shi, L. Guan, W. Li, D. Zhang, P. Chen, and P. Chen, "HARM: Hardware-assisted continuous re-randomization for microcontrollers," in *2022 IEEE european symposium on security and privacy (EuroS P)*,

2022.

[15] A. Bittau, A. Belay, A. Mashtizadeh, D. Mazières, and D. Boneh, "Hacking Blind," in *2014 IEEE Symposium on Security and Privacy*, May 2014, pp. 227–242, iSSN: 2375-1207.

[16] S. Checkoway, L. Davi, A. Dmitrienko, A.-R. Sadeghi, H. Shacham, and M. Winandy, "Return-oriented Programming Without Returns," in *Proceedings of the 17th ACM Conference on Computer and Communications Security*, ser. CCS '10. ACM, 2010, pp. 559–572.

[17] O. Gilles, F. Viguier, N. Kosmatov, and D. G. Pérez, "Control-flow integrity at risc: Attacking risc-v by jump-oriented programming," 2022. [Online]. Available: https://arxiv.org/abs/2211.16212

[18] J. Salwan, "ROPgadget Tool," 2020. [Online]. Available: http://shell-storm.org/project/ROPgadget/

[19] T. Abera, N. Asokan, L. Davi, J.-E. Ekberg, T. Nyman, A. Paverd, A.-R. Sadeghi, and G. Tsudik, "C-FLAT: Control-Flow Attestation for Embedded Systems Software," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '16, Oct. 2016, pp. 743–754.

[20] P. Larsen, A. Homescu, S. Brunthaler, and M. Franz, "SoK: Automated Software Diversity," in *2014 IEEE Symposium on Security and Privacy*, May 2014, pp. 276–291.

[21] T. Nyman, J.-E. Ekberg, L. Davi, and N. Asokan, "CFI CaRE: Hardware-Supported Call and Return Enforcement for Commercial Microcontrollers," in *Research in Attacks, Intrusions, and Defenses*, ser. Lecture Notes in Computer Science, M. Dacier, M. Bailey, M. Polychronakis, and M. Antonakakis, Eds. Springer International Publishing, 2017, pp. 259–284.

[22] P. C. Kocher, "Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems," in *Advances in Cryptology — CRYPTO '96*, ser. Lecture Notes in Computer Science. Springer, 1996, pp. 104–113.

[23] E. Brier, C. Clavier, and F. Olivier, "Correlation Power Analysis with a Leakage Model," in *Cryptographic Hardware and Embedded Systems - CHES 2004*, ser. Lecture Notes in Computer Science. Springer, 2004, pp. 16–29.

[24] B. B. Brumley and N. Tuveri, "Remote Timing Attacks Are Still Practical," in *Computer Security – ESORICS 2011*, ser. Lecture Notes in Computer Science, V. Atluri and C. Diaz, Eds. Springer, 2011, pp. 355–371.

[25] M. Randolph and W. Diehl, "Power Side-Channel Attack Analysis: A Review of 20 Years of Study for the Layman," *Cryptography*, vol. 4, no. 2, p. 15, Jun. 2020, number: 2 Publisher: Multidisciplinary Digital Publishing Institute. [Online]. Available: https://www.mdpi.com/2410-387X/4/2/15

[26] R. Xu, L. Zhu, A. Wang, X. Du, K.-K. R. Choo, G. Zhang, and K. Gai, "Side-Channel Attack on a Protected RFID Card," *IEEE Access*, vol. 6, pp. 58 395–58 404, 2018, conference Name: IEEE Access.

[27] K. Papagiannopoulos and N. Veshchikov, "Mind the Gap: Towards Secure 1st-Order Masking in Software," in *International Workshop on Constructive Side-Channel Analysis and Secure Design*. Springer, 2017, pp. 282–297.

[28] V. C. Ngo, M. Dehesa-Azuara, M. Fredrikson, and J. Hoffmann, "Verifying and Synthesizing Constant-Resource Implementations with Types," in *2017 IEEE Symposium on Security and Privacy (SP)*, May 2017, pp. 710–728, iSSN: 2375-1207.

[29] S. Ahmed, Y. Xiao, K. Z. Snow, G. Tan, F. Monrose, and D. D. Yao, "Methodologies for Quantifying (Re-)randomization Security and Timing under JIT-ROP," in *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '20, Oct. 2020, pp. 1803–1820.

[30] P. Kocher, J. Jaffe, and B. Jun, "Differential Power Analysis," in *Advances in Cryptology — CRYPTO' 99*, ser. Lecture Notes in Computer Science. Springer, 1999, pp. 388–397.

[31] R. M. Tsoupidi, R. C. Lozano, E. Troubitsyna, and P. Papadimitratos, "Securing optimized code against power side channels," 2022. [Online]. Available: https://arxiv.org/abs/2207.02614

[32] G. Barthe, S. Blazy, R. Hutin, and D. Pichardie, "Secure Compilation of Constant-Resource Programs," in *CSF 2021 - 34th IEEE Computer Security Foundations Symposium*. IEEE, Jun. 2021, pp. 1–12.

[33] P. Gao, J. Zhang, F. Song, and C. Wang, "Verifying and Quantifying Side-channel Resistance of Masked Software Implementations," *ACM Transactions on Software Engineering and Methodology*, vol. 28, no. 3, pp. 16:1–16:32, Jul. 2019.

[34] J. Wang, C. Sung, and C. Wang, "Mitigating power side channels during compilation," in *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ser. ESEC/FSE 2019, Aug. 2019, pp. 590–601.

[35] V. D'Silva, M. Payer, and D. Song, "The Correctness-Security Gap in Compiler Optimization," in *2015 IEEE Security and Privacy Workshops*, May 2015, pp. 73–87.

[36] K. Athanasiou, T. Wahl, A. A. Ding, and Y. Fei, "Automatic detection and repair of transition-based leakage in software binaries," in *Software Verification*. Springer, 2020, pp. 50–67.

[37] R. Castañeda Lozano and C. Schulte, "Survey on Combinatorial Register Allocation and Instruction Scheduling," *ACM Computing Surveys*, vol. 52, no. 3, pp. 62:1–62:50, Jun. 2019.

[38] R. Castañeda Lozano, M. Carlsson, G. H. Blindell, and C. Schulte, "Combinatorial Register Allocation and Instruction Scheduling," *ACM Trans. Program. Lang. Syst.*, vol. 41, no. 3, pp. 17:1–17:53, Jul. 2019.

[39] E. Hebrard, B. O'Sullivan, and T. Walsh, "Distance Constraints in Constraint Satisfaction," in *International Joint Conference on Artificial Intelligence - IJCAI 2007*,

2007, p. 6.
[40] L. Ingmar, M. G. de la Banda, P. J. Stuckey, and G. Tack, "Modelling diversity of solutions," in *Proceedings of the thirty-fourth AAAI conference on artificial intelligence*, 2020.
[41] Gecode Team, "Gecode: Generic constraint development environment," 2022. [Online]. Available: https://www.gecode.org
[42] ARM, "Cortex-M0 - Technical Reference Manual," accessed: November 2022. [Online]. Available: https://developer.arm.com/documentation/ddi0432/c/
[43] A. G. Bayrak, F. Regazzoni, D. Novo, and P. Ienne, "Sleuth: Automated Verification of Software Power Analysis Countermeasures," in *Cryptographic Hardware and Embedded Systems - CHES 2013*, ser. Lecture Notes in Computer Science. Springer, 2013, pp. 293–310.
[44] M. Rivain and E. Prouff, "Provably Secure Higher-Order Masking of AES," in *Cryptographic Hardware and Embedded Systems, CHES 2010*, ser. Lecture Notes in Computer Science. Springer, 2010, pp. 413–427.
[45] J.-S. Coron, E. Prouff, M. Rivain, and T. Roche, "Higher-Order Side Channel Security and Mask Refreshing," in *Fast Software Encryption*, ser. Lecture Notes in Computer Science. Springer, 2014, pp. 410–424.
[46] H. Mantel and A. Starostin, "Transforming Out Timing Leaks, More or Less," in *Computer Security – ESORICS 2015*, ser. Lecture Notes in Computer Science. Springer International Publishing, 2015, pp. 447–467.
[47] H. Winderix, J. T. Mühlberg, and F. Piessens, "Compiler-Assisted Hardening of Embedded Software Against Interrupt Latency Side-Channel Attacks," in *2021 IEEE European Symposium on Security and Privacy (EuroS P)*, Sep. 2021, pp. 667–682.
[48] D. Broman, "A Brief Overview of the KTA WCET Tool," Dec. 2017, number: arXiv:1712.05264 arXiv:1712.05264 [cs].
[49] R. M. Tsoupidi, "Two-phase WCET analysis for cache-based symmetric multiprocessor systems," Master's thesis, Royal Institute of Technology KTH, 2017.
[50] A. Lindner, R. Guanciale, and M. Dam, "Technical Report on Proof-Producing Symbolic Execution for Binary Code Verification," accessed: November 2022. [Online]. Available: https://people.kth.se/~lindnera/bin_se_wcet.pdf
[51] A. Homescu, S. Neisius, P. Larsen, S. Brunthaler, and M. Franz, "Profile-guided Automated Software Diversity," in *Proceedings of the 2013 IEEE/ACM International Symposium on Code Generation and Optimization (CGO)*, ser. CGO '13. IEEE Computer Society, 2013, pp. 1–11.
[52] S. Crane, C. Liebchen, A. Homescu, L. Davi, P. Larsen, A. Sadeghi, S. Brunthaler, and M. Franz, "Readactor: Practical Code Randomization Resilient to Memory Disclosure," in *2015 IEEE Symposium on Security and Privacy*, May 2015, pp. 763–780.
[53] L. Foundation, "Tool interface standard (tis) portable formats specification version 1.1," accessed February 2023. [Online]. Available: https://refspecs.linuxfoundation.org/elf/TIS1.1.pdf
[54] M. Tran, M. Etheridge, T. Bletsch, X. Jiang, V. Freeh, and P. Ning, "On the Expressiveness of Return-into-libc Attacks," in *Recent Advances in Intrusion Detection*, ser. Lecture Notes in Computer Science, R. Sommer, D. Balzarotti, and G. Maier, Eds. Springer, 2011, pp. 121–141.
[55] J. Cabrera Arteaga, P. Laperdrix, M. Monperrus, and B. Baudry, "Multi-variant Execution at the Edge," in *Proceedings of the 9th ACM Workshop on Moving Target Defense*, ser. MTD'22, Nov. 2022, pp. 11–22.
[56] K. Ngo, E. Dubrova, and T. Johansson, "Breaking Masked and Shuffled CCA Secure Saber KEM by Power Analysis," in *Proceedings of the 5th Workshop on Attacks and Solutions in Hardware Security*, Nov. 2021, pp. 51–61.
[57] K. Ngo, E. Dubrova, Q. Guo, and T. Johansson, "A Side-Channel Attack on a Masked IND-CCA Secure Saber KEM Implementation," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pp. 676–707, Aug. 2021.
[58] Anonymous, "Supplemental material: Secure-by-design code diversfication against code-reuse and side-channel attacks in embedded systems," 2022. [Online]. Available: https://anonymous.4open.science/r/secdivcon_experiments-0BEB/main_appendix.pdf
[59] H. Koo, Y. Chen, L. Lu, V. P. Kemerlis, and M. Polychronakis, "Compiler-Assisted Code Randomization," in *2018 IEEE Symposium on Security and Privacy (SP)*, May 2018, pp. 461–477.
[60] S. Pastrana, J. Tapiador, G. Suarez-Tangil, and P. Peris-López, "AVRAND: A Software-Based Defense Against Code Reuse Attacks for AVR Embedded Devices," in *Detection of Intrusions and Malware, and Vulnerability Assessment*, ser. Lecture Notes in Computer Science, 2016, pp. 58–77.
[61] K. Z. Snow, F. Monrose, L. Davi, A. Dmitrienko, C. Liebchen, and A. Sadeghi, "Just-In-Time Code Reuse: On the Effectiveness of Fine-Grained Address Space Layout Randomization," in *2013 IEEE Symposium on Security and Privacy*, May 2013, pp. 574–588.
[62] J. Seibert, H. Okhravi, and E. Söderström, "Information Leaks Without Memory Disclosures: Remote Side Channel Attacks on Diversified Code," in *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '14, Nov. 2014, pp. 54–65.
[63] T. Abera, N. Asokan, L. Davi, J.-E. Ekberg, T. Nyman, A. Paverd, A.-R. Sadeghi, and G. Tsudik, "C-FLAT: Control-Flow Attestation for Embedded Systems Software," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '16, Oct. 2016, pp. 743–754.
[64] A. Fu, W. Ding, B. Kuang, Q. Li, W. Susilo, and

Y. Zhang, "FH-CFI: Fine-grained hardware-assisted control flow integrity for ARM-based IoT devices," *Computers & Security*, vol. 116, p. 102666, May 2022.

[65] M. A. Shelton, N. Samwel, L. Batina, F. Regazzoni, M. Wagner, and Y. Yarom, "Rosita: Towards Automatic Elimination of Power-Analysis Leakage in Ciphers," *Proceedings 2021 Network and Distributed System Security Symposium*, 2021, appears in NDSS 2022.

[66] D. Molnar, M. Piotrowski, D. Schultz, and D. A. Wagner, "The program counter security model: Automatic detection and removal of control-flow side channel attacks," in *Information Security and Cryptology - ICISC 2005, 8th International Conference, Seoul, Korea, December 1-2, 2005, Revised Selected Papers*, 2005, pp. 156–168.

[67] P. Borrello, D. C. D'Elia, L. Querzoni, and C. Giuffrida, "Constantine: Automatic Side-Channel Resistance Using Efficient Control and Data Flow Linearization," *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, pp. 715–733, Nov. 2021.

[68] C. Brown, A. D. Barwell, Y. Marquer, O. Zendra, T. Richmond, and C. Gu, "Semi-automatic ladderisation: improving code security through rewriting and dependent types," in *Proceedings of the 2022 ACM SIGPLAN International Workshop on Partial Evaluation and Program Manipulation*, ser. PEPM 2022, Jan. 2022, pp. 14–27.

[69] A. Rane, C. Lin, and M. Tiwari, "Raccoon: Closing Digital {Side-Channels} through Obfuscated Execution," in *26th USENIX Security Symposium (USENIX Security 15)*, 2015, pp. 431–446.

[70] H. P. Joshi, A. Dhanasekaran, and R. Dutta, "Trading off a vulnerability: does software obfuscation increase the risk of rop attacks," *Journal of Cyber Security and Mobility*, pp. 305–324, 2015.

[71] S. Crane, A. Homescu, S. Brunthaler, P. Larsen, and M. Franz, "Thwarting Cache Side-Channel Attacks Through Dynamic Software Diversity," in *Proceedings 2015 Network and Distributed System Security Symposium*. Internet Society, 2015.

[72] J.-K. Zinzindohoué, K. Bhargavan, J. Protzenko, and B. Beurdouche, "HACL*: A Verified Modern Cryptographic Library," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, 2017, pp. 1789–1806.