**Securing TrustZone: Symbolic Execution for Side-Channel Analysis in ARM-M Binaries**

**ABSTRACT**

**Keywords:** IoT systems, Timing side-channel attacks, Binary code analysis, Symbolic execution, resource-constrained devices.

**1- INTRODUCTION**

With the rapid proliferation of Internet of Things (IoT) devices in various domains, such as smart homes, healthcare, transportation, and industrial systems, ensuring the security of these interconnected devices has become an utmost concern. IoT systems, consisting of embedded devices and networked components, handle an abundance of sensitive data, making them prime targets for malicious actors seeking to exploit vulnerabilities [1]. Among the multitude of security threats, timing side-channel attacks have emerged as a significant and pervasive challenge, leveraging timing variations to exploit vulnerabilities and compromise the confidentiality and integrity of sensitive data [2, 3, 4].

ARM family processors have emerged as the dominant choice for embedded devices, capturing a substantial market share of over 60% [5]. To enhance security, ARM has incorporated TrustZone [6, 7], a hardware-based security feature, into their processors. TrustZone ensures the isolation of security-critical software and data from the rest of the system, enabling secure execution of critical tasks and protection of sensitive information. It achieves this by dividing the processor into two separate and concurrent security realms or worlds: the 'Normal World' and the 'Secure World.' These worlds operate independently of each other, possessing distinct memory spaces and execution environments.

Here, developers often rely on the presumption that secrets are protected within the secure world due to the processor's isolation guarantees. However, extensive research [7, 8, 9, 10, 11, 12] has revealed the potential vulnerability of the TrustZone secure world to side channel attacks, which can lead to the unintended disclosure of fine-grained secrets. For instance, TruSpy [9] exploits the cache contention between normal world and secure world to implement a timing-based cache side-channel attack and then extract a full 128-bit AES encryption key stored in the trusted environment. The research demonstrated that while the contents of the processor cache are safeguarded by the hardware isolation, the access pattern to these cache lines remains unprotected. Consequently, TrustZone becomes susceptible to cache side-channel attacks, compromising its security measures. Similarly, in [11], researchers targeted Arm TrustZone in a malicious OS scenario. They leveraged the OS's capabilities to invoke interrupts and utilized the Prime+Probe technique to recover a 256-bit private key from Qualcomm's ECDSA algorithm.

Timing side-channel attacks exploit the unintentional leakage of timing information, such as execution time, cache behavior, or branch prediction, to infer sensitive data and breach system security. Early detection of side channel attacks enables proactive mitigation measures to be implemented. Over the years, researchers and practitioners have proposed various approaches to analyze binary code, or source code employing techniques such as symbolic execution [13, 14], formal verification [15, 16, 17], and machine learning [18], among others [2]. These approaches aim to identify and mitigate timing side channel vulnerabilities targeting different architectures. However, each approach carries its own limitations and strengths, necessitating a thorough exploration of the existing body of work in this field (refer to Section 3).

In this paper, we present an innovative automated approach utilizing symbolic analysis techniques for the static verification of binaries targeting the ARM Cortex-M23 microcontroller. Our objective is to ensure the absence of timing side channel attacks, interrupt-latency attacks (such as Nemesis [3]), and detect any explicit and implicit information flow, which is roughly equivalent to the concept of storage channels in later literature [19]. This is particularly relevant in the context of applications that are compartmentalized into a security critical application part (such as managing and using cryptographic credentials) and a less critical part (such as sending and receiving network packets) to make use of the ARM TrustZone Trusted Execution Environment (TEE).

The TrustZone technology employed in Armv8-M processors (such as Cortex-M23/ M33/ M35P/ M55 / M85), do not claim to protect against side channel attacks due to secret-dependent control flow with measurable timing differences or secret-dependent memory access patterns [20]. Additionally, it is important to note that TrustZone may not effectively prevent secret leakage stemming from program implementation flaws, which can arise from weaknesses in protocols or algorithms, as well as mistakes made by developers. As an example, let's consider an One-Time Password (OTP) system implemented within the TrustZone environment [22]. In a secure and well-implemented OTP system, once an OTP is utilized, it should immediately become invalid and should not be stored in any accessible location. However, if the OTPs are stored in an insecure manner, such as being logged or stored in plaintext on unprotected memory or external I/O, an unauthorized attacker who gains access to the system or the logs could retrieve the previously used OTPs.

Considering the vulnerabilities discussed above, it is crucial for developers and system designers to implement strong security measures, including secure storage and proper handling of sensitive data, robust encryption algorithms, and appropriate access controls, to complement the security features provided by TrustZone. In this paper, we leverage symbolic execution [21], a widely recognized program analysis technique that computes program behaviors using mathematical constraints based on symbolic inputs, to effectively detect potential data leakages, and provide valuable insights to programmers, enabling them to address and rectify any security vulnerabilities.

This paper introduces SCFARM, an innovative automatic tool named after [17], specifically designed for static verification of ARMv8-M binaries. The primary objective of SCFARM is to track and monitor the flow of secret information between the TrustZone's secure world and the non-secure world, detecting and reporting any potential information leakages. To the best of our knowledge, this tool represents the first of its kind in performing static analysis on ARMv8-M binaries, addressing both timing and storage channels. To validate the effectiveness of our tool, we conduct comprehensive experiments on a collection of 'x' cryptographic libraries employed in ARM Cortex-M23. Through these experiments, we demonstrate the robust capabilities of SCFARM in effectively detecting vulnerabilities and ensuring the absence of any potential information leakage.

***Contributions.*** Our contributions can be summarized as follows:

* *Innovative Approach for Binary-level Information Flow Analysis:* Our research introduces a groundbreaking approach that leverages the capabilities of symbolic execution techniques to perform information flow analysis at the binary level. Specifically designed for applications compartmentalized for ARM TrustZone, a widely adopted security feature found in commercial microcontrollers and mobile devices, aimed at protecting valuable and confidential data.
* *Development of SCFARM Tool:* To automate the process of checking ARMv8-M binaries and identifying potential information leakages, we have implemented our novel approach in a software tool called SCFARM. Written in Python, SCFARM efficiently carries out the analysis and provides detailed reports on identified vulnerabilities. We have made both SCFARM and our benchmark datasets publicly available on the GitHub repository at [https://github.com/sepidehpouyan/’x’](https://github.com/sepidehpouyan/%E2%80%99x%E2%80%99" \t "_new).
* *Detection of Various Security Threats:* We have successfully integrated static analysis techniques into SCFARM, enabling the detection of timing side channel attacks, Nemesis [3] attacks, and undesired direct and indirect information flow to accessible and unprotected locations.
* *Evaluation of SCFARM's Accuracy and Scalability:* To assess the effectiveness and scalability of SCFARM, we conducted a rigorous evaluation by applying it to a set of cryptographic libraries targeting ARM Cortex-M23. Our evaluation encompassed testing numerous scenarios to analyze the accuracy of the tool in identifying information leakages and its ability to handle larger codebases. The results demonstrated the high precision and scalability of SCFARM, validating its utility in real-world security assessments.

***Organization.*** The paper is structured as follows. We start by giving an overview of side channel attacks and explain our threat model in Section 2, followed by a discussion or related work on the detection of side-channel information leakage in programs in Section 3. In Section 4, we elaborate on our approach. We then describe the components of our SCFARM, and their tasks in Section 5. In Section 6, we present the results of our evaluation. Finally, we conclude our work and outline future directions of research.

**2 BACKGROUND AND PROBLEM STATEMENT**

**2.1 Trusted Execution Environment**

Trusted Execution Environments (TEEs) are a fundamental component of modern security architecture, designed to provide a secure execution environment for sensitive and critical computations. TEEs are isolated and tamper-resistant processing environments within a computing system, where the confidentiality and integrity of code and data are safeguarded against a variety of threats [24]. They ensure the authenticity of executed code, verifying that it has not been tampered with. Additionally, TEEs maintain the integrity of the system's runtime states, encompassing vital components such as CPU registers, memory, and sensitive input/output operations. TEEs uphold the confidentiality of code, data, and runtime states, including their secure storage in persistent memory. This comprehensive protection against unauthorized access, code alterations, and data breaches makes TEEs a crucial component in securing sensitive computations and data within a computing system. They enforce strict access control, permitting data access only to code within the same secure execution environment, and code execution is allowed only from predefined entry points to mitigate risks like Return-Oriented Programming attacks [25]. Additionally, TEEs support remote attestation to verify their trustworthiness to third parties [23]. By providing a foundation for secure computing, TEEs have become integral in a variety of domains, from mobile devices to cloud computing. The two main TEE technologies currently available in the market are Intel SGX [26] and ARM TrustZone [7], the latter being the focus of this paper. The effectiveness of TEEs relies on a well-defined attacker model, critical for identifying potential threats and guiding the development of secure TEE-based solutions.

**2.1.1- TrustZone on ARM Cortex-M**

ARM TrustZone is a hardware-based security technology developed by ARM Holdings [7, 27]. TrustZone essentially divides the ARM processor into two distinct execution environments: the "Normal World" and the "Secure World". In fact, this system-wide approach assigns two virtual cores to each physical processor, together with the mechanism to securely switch between both realms. These environments are isolated from each other, and the Normal World is typically where the non-secure, general-purpose operating system and applications run. The Secure World, on the other hand, is a more trusted and isolated area where security-critical operations, cryptographic functions, and sensitive data can be processed and stored.

On ARM application processors (Cortex-A) [28], a separate processor mode known as the secure monitor handles secure context switching between worlds. However, on ARM microcontrollers (Cortex-M) [6, 29] lack a dedicated secure monitor software. Instead, essential mechanisms integrated into the core logic act as gatekeepers, facilitating the transition between secure and non-secure realms. These two worlds are rigidly separated at the hardware level and possess differing levels of privilege. Non-secure software is explicitly restricted from directly accessing resources in the secure world. This paper focuses exclusively on TrustZone features for Cortex-M processors.

TrustZone technology for Armv8-M devices [6, 29] is tailored for ARM microcontrollers, specifically the Cortex-M series. It's been finely tuned for swift context switching and ultra-low power embedded applications. Leveraging specialized hardware integrated into Cortex-M cores along with a dedicated secure instruction set, TrustZone facilitates the establishment of multiple software security domains. These domains enforce strict access controls, allowing trusted software exclusive access to secure memory and I/O, all while maintaining optimal system performance.

**Armv8-M Architecture** [31]typicallyfeatures a set of 32-bit general-purpose registers (R0 to R12, Link Register (LR), Program Counter (PC)) and floating-point register (D0-D15) that are shared between secure and non-secure states. TrustZone-enabled Armv8-M microcontrollers have separate stacks for each security state, with the Stack Pointer (SP) being security-banked, meaning one instance exists in each state. The CONTROL register and some other special-purpose registers are also banked, and the core automatically switches between their instances during state transitions. ARMv8-M architecture introduces a new ISA with additional instructions and features, which enhances code density, reduces interrupt latency, and improves system performance. The architecture includes a two-stage pipeline for instruction execution, providing efficient handling of instructions.

**Memory space** in the Armv8-M architecture is also partitioned into secure and non-secure memory regions. The secure memory space is further divided into two types: secure and non-secure callable (NSC). Secure addresses are exclusively allocated for memory and peripherals that can only be accessed when the core is executing in secure state. The program address, the address of the instruction currently executed, determines the security state of the processor. In contrast, non-secure addresses are designated for memory and peripherals accessible by all software running on the device, including both secure and non-secure components. NSC represents a unique class of secure memory locations that facilitates the transition of software from a non-secure to a secure state, allowing for controlled and secure state changes.

The security state assigned to each memory address are established through either the programmable Secure Attribution Unit (SAU) or by an fixed Implementation Defined Attribution Unit (IDAU). The SAU is always available in Armv8-M cores, while the IDAU is external to the core and the presence depends on the vendors implementation. In cases where both the IDAU and SAU are available within a system, the SAU's attributions take precedence, unless the IDAU specifies a higher security attribute for a particular address.The SAU can only be programmed in the secure state.

In ARM TrustZone-M, the Nested Vectored Interrupt Controller (NVIC) has been enhanced to enable secure and non-secure configuration for each interrupt. The processor seamlessly handles interrupts based on its current security state. Notably, when a non-secure interrupt occurs during secure code execution, the processor securely manages the transition, preserving secure context data and preventing information leakage.

## For Transition between two worlds, three new instructions have been introduced including secure gateway (SG), branch with exchange to non-secure state (BXNS), and branch with link and exchange to non-secure state (BLXNS). The SG instruction is employed for switching from the non-secure to the secure state. It is typically found at the start of a secure entry point's veneer, which consists of an SG instruction followed by a branch to the secure world's function. The veneers are meant to reside in memory regions attributed to the NSC by the linker. The SG instruction serves several functions, such as setting the security level to secure, banking registers, and resetting bit[0] of the LR register to 0, indicating that the return will lead to a transition back from secure to non-secure.To return from the secure world to the non-secure world, as illustrated in Fig. 1, the compiler employs the BXNS instruction. This instruction initiates a branch or return to the non-secure program.

## A diagram of a security system Description automatically generated

## Fig. 1: Secure Function Call

Conversely, secure software can invoke functions in the non-secure world. This action prompts the generation of compiler code that orchestrates the transition. It begins by preserving all registers, including the return address, within the secure stack. Subsequently, the registers are cleared. The BLXNS instruction is used to execute the branch to the non-secure world, where it sets LR to a specific value, FNC\_RETURN (0xFEFFFFFF).

Upon completion of the execution in the non-secure world, a return to the secure world is initiated using BX. When the BX instruction detects the FNC\_RETURN value in LR, it triggers a transition to the secure state. This shift is made possible by restoring all saved registers, including the return address, from the secure stack. It's also important to note that state transitions may also occur due to exceptions and interrupts.

TrustZone is not bullet-proof and has experienced successful attacks across various methods and contexts [7, 30]. The architecture, while designed to provide robust hardware-based security by isolating secure and non-secure worlds, is not immune to microarchitectural side-channel vulnerabilities [7, 9, 10, 30, 31, 32]. These vulnerabilities arise due to the shared resources and memory management between the secure and non-secure domains. Arm [20] has acknowledged that the security extensions for the Armv8-M architecture are not designed to protect against side-channel attacks resulting from control flow or memory access patterns. They argue that such attacks are not exclusive to the Armv8-M architecture and can apply to any code with secret-dependent control flow or memory access patterns. This type of attack can be mitigated by ensuring that the control flow and memory accesses patterns created by the program do not depend on secret state.

**2.2- Microarchitectural Side-Channel Attacks**

The security model proposed by TEEs is not foolproof and must be approached with caution regarding side-channel attacks. These attacks aim to uncover secret-dependent information hidden within the shared microarchitectural state during a victim's execution by exploiting observable side effects, notably timing variations. Typically, adversaries begin by initializing the shared microarchitectural elements in a predetermined state. They then proceed to measure state changes during or after the victim's execution, utilizing methods such as transactional memory aborts or performance monitoring counters. However, the most prevalent method for observing microarchitectural state changes is through timing analysis [33]. In cases where microarchitectural optimizations depend on global stateful elements like Translation Lookaside Buffers (TLB), caches, or branch predictors, any modifications to these elements during the victim's execution will result in measurable timing differences in the attacker domain.The analysis of microarchitectural state updates provides valuable insights into the victim's behavior, even in scenarios where attackers are architecturally isolated and have limited interaction with the victim, strictly through defined input and output channels.

Single-purpose embedded processors typically emphasize simplicity, power efficiency, and cost-effectiveness over advanced microarchitectural features like caches, pipelining, and speculative execution. This focus results in predictable instruction timings, reducing the risk of side-channel attacks. However, research [34, 17] has demonstrated that secrets can still be revealed through start-to-end timing side channels, by measuring the overall execution time of secret-dependent branches, even on processors with entirely deterministic instruction timing behavior.

In addition, Nemesis-type interrupt timing attacks [3] can exploit highly precise, instruction- granular timing measurements, which can even compromise secrets from branches with balanced start-to-end timings. These side-channel attacks abuse the CPU's interrupt mechanism to reveal microarchitectural instruction timings within TEEs. The attack leverages the fact that hardware interrupts are only processed upon instruction retirement, afterthe currently executing instruction has completed, resulting in variable CPU cycles for different instruction types and processor states. Consequently, an untrusted operating system can precisely measure interrupt handling time, to retrieve the execution length of interrupted instruction and distinguish between secret-dependent program branches. In essence, for a successful Nemesis attack on processors with constant-time interrupt latency and multi-cycle instruction sets, where each instruction is uninterruptible, an attacker just requires a different execution time for at least one instruction in the if/else branch.

Side-channel attacks also can exploit information leaked through covert storage channels [19, 35]. These channels, whether explicit or implicit, provide pathways for the unauthorized transfer of sensitive data, which can compromise system security. Implicit covert storage channels, in particular, introduce challenges, as data can inadvertently traverse indirect or unintended pathways resulting from the program's control structure. This contrasts with explicit information flows, which occur when confidential data is directly assigned to public variables. To illustrate, let us consider a scenario with two security levels: “high” and “low,” denoting varying degrees of confidentiality. We can examine the code presented in Figure 2, which exhibits a data flow from the high variable *h* to the low variable *l*. The insecurity within this code stems from the *l = 1* assignment in a control context conditioned upon the confidential variable *h*.

A white paper with black text

Description automatically generated

**Fig. 2: An implicit information flow**

In recent findings, researchers have identified DMA-based side-channel attacks specifically aimed at embedded TEEs [31, 36]. These attacks exploit nuanced timing variations arising from contention between a DMA device and the CPU as they access the shared memory bus. This exploitation enables the attacker to construct a cycle-accurate memory access trace of a victim program. Notably, at the Black Hat Asia conference, Cristiano Rodrigues and team [36] presented a side-channel attack that leverages DMA to target ARM TrustZone. This sophisticated attack effectively bypasses hardware-enforced isolation primitives, providing unauthorized access to trusted applications (TAs) and resulting in the unauthorized leakage of sensitive information.

**2.2.1- BUSted: Software-based Microarchitectural Timing Side-Channels Attacks on TrustZone-M MCUs**

BUSted [36] is a novel class of microarchitectural side-channel attacks that leverage the timing differences exposed in the internal bus interconnect arbitration logic that decides which bus master (e.g., CPU, DMA) controls the internal data bus. It's evident that concurrent access by multiple bus master to a shared bus slave (e.g., memory controller) leads to time delays for at least one, causing subtle variations in timing. By observing the timing drifts on memory transactions, an attacker can extract information regarding the victim’s memory access pattern. Consequently, without breaking security isolation boundaries, a malicious bus master can spy on bus activity and determine when another bus master accessed a specific slave.

ARM adopts a load-store memory access model, restricting memory interaction solely to load/store (LDR and STR) instructions. Thus, an attacker can exploit the MCU's load-store architecture to observe timing variations. Specifically, if the victim code employs secret-dependent control flow where loads/stores execute at different clock offsets on conditional paths, resulting in distinct memory access patterns. An attacker could exploit this to deduce and illicitly obtain a particular secret.

To elaborate further, let's examine a code snippet (compiled for the Arm Cortex-M23) that includes a balanced conditional statement dependent on a secret variable, as visualized in Figure 3. Since both execution paths have an identical execution time of 5 clock cycles, starting from t + 1 after the *cmp* instruction, an attacker wouldn't observe any difference in execution time. Consequently, distinguishing between these execution paths and subsequently extracting the secret becomes unfeasible. Yet, an observer could still detect divergent memory access patterns between these execution paths. When the branch (bne) isn't taken, it completes within a single clock cycle, causing the *str* instruction to occur at clock cycle t + 3. Conversely, if taken, the branch incurs a two-clock-cycle process, resulting in the *str* instruction taking place at clock cycle t + 4. By monitoring either of these specific clock cycles, an attacker can potentially deduce the secret by detecting any contention on the data memory bus.

A screenshot of a computer

Description automatically generated

**Fig. 3: (a) secret-dependent branch, (b) compiled code CFG for Arm Cortex-M23, (c) memory access pattern and monitoring of clock t+3.**

The attack was conducted on a security mechanism employing ARM TrustZone's 'safety deposit box' alongside a trusted keypad to safeguard a user's PIN during unlocking. The successful demonstration of this attack revealed the potential to infer the PIN code without compromising ARM TrustZone's security measures. To put this into perspective, consider a scenario where a smart home operates on an ARM processor. If an attacker gains physical access, they could exploit this side-channel attack, potentially bypassing security measures. This breach could grant them control over the entire system, enabling alterations to thermostat settings, disabling security cameras, or unauthorized unlocking of doors.

Given the nature of this attack, targeting on microarchitectural design issues, comparisons have arisen likening its impact to that of renowned exploits like 'Spectre' and 'Meltdown' which affected more complex architectures in recent history. Embedded projects relying on hardware-assisted privilege separation via TrustZone-M must now factor in the potential for information leakage from secure components operating within the secure world. According to the researchers, there are software-based countermeasures available to mitigate the impact of this microarchitectural design flaw. The crucial consideration in addressing time-based attacks involves minimizing the presence of secret-dependent code in security operation implementations. Essentially, the time required for a security procedure should remain independent of the success or any secret of the operation.

**2.3- Taint and Value Set Analysis**

Angr: taint, VSA, symbolic execution

**3- RELATED WORK**

* **Timing side channels on embedded devices**
* **Timing side channel on ARM/TZ**
* **Detection by different approaches (formal, machine learning, symbolic execution)**
* **(SCFMSP, BinSec, PitchFork, Static analysis tools, symbolic execution-based tools for detecting side-channel)**

**4- Design**

**4.2- Attacker Model**

**5- Implementation**

**6- Evaluation**

**7- CONCLUSIONS AND FUTURE DIRECTIONS**

**REFERENCES**

[1] Papp, D., Ma, Z., Buttyan, L., 2015. Embedded systems security: Threats, vulnerabilities, and attack taxonomy, in: 2015 13th Annual Conference on Privacy, Security and Trust (PST), pp. 145–152. doi:10.1109/PST.2015.7232966.

[2] Qian Ge, Yuval Yarom, David Cock, and Gernot Heiser. 2018. A survey of microarchitectural timing attacks and countermeasures on contemporary hardware. Journal of Cryptographic Engineering 8, 1(2018), 1–27. https://doi.org/10.1007/ s13389- 016- 0141- 6

[3] Jo Van Bulck, Frank Piessens, and Raoul Strackx. 2018. Nemesis: Studying Microarchitectural Timing Leaks in Rudimentary CPU Interrupt Logic. In CCS ’18.

[4] Dag Arne Osvik, Adi Shamir, and Eran Tromer. 2005. Cache Attacks and Countermeasures: The Case of AES. In Topics in Cryptology - CT-RSA 2006.

[5] “Arm holdings and qualcomm: The winners in mobile.” http://www.forbes.com/sites/darcytravlos/2013/02/28/ arm- holdings- and- qualcomm- the- winners- in- mobile/.

[6] Arm Ltd. 2017. TrustZone technology for ARMv8-M Architecture. Version 2.0.

[7] Pinto, Sandro & Santos, Nuno. (2019). Demystifying Arm TrustZone: A Comprehensive Survey. ACM Computing Surveys. 51. 1-36. 10.1145/3291047.

[8] Z. Kou, W. He, S. Sinha and W. Zhang, "Load-Step: A Precise TrustZone Execution Control Framework for Exploring New Side-channel Attacks Like Flush+Evict," *2021 58th ACM/IEEE Design Automation Conference (DAC)*, San Francisco, CA, USA, 2021, pp. 979-984, doi: 10.1109/DAC18074.2021.9586226.

[9] N.Zhangetal.,“TruSpy: Cache Side-channel Information Leakage from the Secure World on Arm Devices.” Trans. on IACR Cryptol, 2016.

[10] Bukasa, Sebanjila & Lashermes, Ronan & Bouder, Hélène & Lanet, Jean-Louis & Legay, Axel. (2018). How TrustZone Could Be Bypassed: Side-Channel Attacks on a Modern System-on-Chip. 10.1007/978-3-319-93524-9\_6.

[11] K. Ryan, “Hardware-Backed Heist: Extracting ECDSA Keys from Qualcomm’s TrustZone,” in Proc. of ACM CCS, 2019.

[12] Saß, Marvin & Mitev, Richard & Sadeghi, Ahmad-Reza. (2023). Oops..! I Glitched It Again! How to Multi-Glitch the Glitching-Protections on ARM TrustZone-M.

[13] Lesly-Ann Daniel, Sébastien Bardin, and Tamara Rezk. Binsec/Rel: Efficient relational symbolic execution for constant-time at binary-level, 2019.

[14] Disselkoen, Craig. “Finding and Eliminating Timing Side-Channels in Crypto Code with Pitchfork.” (2021).

[15] G. Barthe, G. Betarte, J. D. Campo, C. D. Luna, and D. Pichardie, “System-level non-interference for constant- time cryptography”, in CCS, 2014.

[16] J. B. Almeida, M. Barbosa, J. S. Pinto, and B. Vieira, “Formal verification of side-channel counter- measures using self-composition”, Sci. Comput. Pro- gram., vol. 78, no. 7, 2013.

[17] Pouyanrad, Sepideh et al. “SCFMSP: static detection of side channels in MSP430 programs.” Proceedings of the 15th International Conference on Availability, Reliability and Security (2020): n. pag.

[18] M. Mushtaq et al., "Machine Learning For Security: The Case of Side-Channel Attack Detection at Run-time," 2018 25th IEEE International Conference on Electronics, Circuits and Systems (ICECS), Bordeaux, France, 2018, pp. 485-488, doi: 10.1109/ICECS.2018.8617994.

[19] Toby Murray, Daniel Matichuk, Matthew Brassil, Peter Gammie, Timothy Bourke, Sean Seefried, Corey Lewis, Xin Gao, and Gerwin Klein. seL4: from general purpose to a proof of information flow enforcement. In IEEE Symposium on Security and Privacy, pages 415–429, San Francisco, CA, May 2013.

[20] Arm Developer, “Clarification of Timing Side Channel Attacks on TrustZone enabled Cortex”, <https://developer.arm.com/documentation/ka005578/latest>

[21] C. Cadar and K. Sen, “Symbolic execution for software testing: Three decades later”, Communications of the ACM, vol. 56, no. 2, 2013.

[22] H. Sun, K. Sun, Y. Wang, and J. Jing. 2015. Trust OTP: Transforming smartphones into secure one-time password tokens. In Proceedings of the ACM SIGSAC Conference on Computer and Communications Security. ACM, 976–988. DOI:https://doi.org/10.1145/2810103.2813692

[23] SEPIDEH POUYANRAD, JOB NOORMAN, FRITZ ALDER, BAUMANN CHRISTOPH, FRANK PIESSENS, JAN TOBIAS MÜHLBERG. End-to-End Security for Distributed Event-Driven Enclave Applications on Heterogeneous TEEs. ACM Transactions on Privacy and Security, [Volume 26](https://dl.acm.org/toc/tops/2023/26/3), [Issue 3](https://dl.acm.org/toc/tops/2023/26/3), Article No.: 39, pp 1–46, <https://doi.org/10.1145/3592607>

[24] P. Maene, J. Götzfried, R. De Clercq, T. Müller, F. Freiling, and I. Verbauwhede. Hardware-based trusted computing architectures for isolation and attestation. IEEE Transactions on Computers, 67(3):361–374, 2017.

[25] R. Roemer, E. Buchanan, H. Shacham, and S. Savage. Return-oriented program- ming: Systems, languages, and applications. ACM Transactions on Information and System Security (TISSEC), 15(1):1–34, 2012.

[26] V. Costan and S. Devadas. Intel sgx explained. IACR Cryptology ePrint Archive, 2016(086):1–118, 2016.

[27] Bernard Ngabonziza, Daniel Martin, Anna Bailey, Haehyun Cho, and Sarah Martin. 2016. TrustZone Explained: Architectural Features and Use Cases. 2016 IEEE 2nd Int. Conf. Collab. and Internet Computing (CIC) (2016), 445–451.

[28] Arm Ltd. 2009. ARM Security Technology: Building a Secure System using TrustZone Technology.

[29] J. Taylor. 2016. Security for the next generation of safe real-time systems. In Proceedings of Embedded World Conference.

[30] Muñoz, Antonio & Rios, Ruben & Roman, Rodrigo & Lopez, Javier. (2023). A survey on the (in)security of Trusted Execution Environments. Computers & Security. 129. 103180. 10.1016/j.cose.2023.103180.

[31] Cristiano Rodrigues, Daniel Oliveira, and Sandro Pinto, "BUSted!!! Microarchitectural Side-Channel Attacks on the MCU Bus Interconnect",  2024 IEEE Symposium on Security and Privacy (SP), San Francisco, CA, USA, 2024.

[32] Z. Ma, X. Tan, L. Ziarek, N. Zhang, H. Hu and Z. Zhao, "Return-to-Non-Secure Vulnerabilities on ARM Cortex-M TrustZone: Attack and Defense," *2023 60th ACM/IEEE Design Automation Conference (DAC)*, San Francisco, CA, USA, 2023, pp. 1-6, doi: 10.1109/DAC56929.2023.10247972.

[33]: Van Bulck, J. (2020). Microarchitectural Side-Channel Attacks for Privileged Software Adversaries (Doctoral dissertation). KU Leuven, Belgium.

[34] Travis Goodspeed. Practical attacks against the MSP430 BSL. In 25th Chaos Communications Congress., 2008.

[35] A. Sabelfeld and A. C. Myers, "Language-based information-flow security," in *IEEE Journal on Selected Areas in Communications*, vol. 21, no. 1, pp. 5-19, Jan. 2003, doi: 10.1109/JSAC.2002.806121.

[36] Marton Bognar, Jo Van Bulck, and Frank Piessens. Mind the gap: Studying the insecurity of provably secure embedded trusted execution architectures. In 43rd IEEE Symposium on Security and Privacy (S&P), pages 1638–1655. IEEE, 2022.