Efficient Cross-layer Security against Advanced Threats in Emerging Computing Systems

Zur Erlangung des akademischen Grades eines

Doktors der Ingenieurwissenschaften

von der KIT-Fakultät für Informatik des
Karlsruher Instituts für Technologie (KIT)

genehmigte

Dissertation

von

Jeferson Gonzalez Gomez

aus San Roque, Grecia (Costa Rica)

Tag der mündlichen Prüfung: 19. Dezember 2024
Referent: Prof. Dr.-Ing. Jörg Henkel, Karlsruher Institut für Technologie (KIT)
Koreferent: Prof. Dr.-Ing. Felix Freiling, Friedrich-Alexander-Universität Erlangen-Nürnberg (FAU)

Hiermit erkläre ich an Eides statt, dass ich die von mir vorgelegte Arbeit selbstständig verfasst habe, dass ich die verwendeten Quellen, Internet-Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen – die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Jeferson Gonzalez Gomez

Acknowledgements

First of all, I would like to express my sincere gratitude to Prof. Dr.-Ing. Jörg Henkel for his support and supervision over the past years. Prof. Henkel has always encouraged me to explore my own research ideas, providing valuable insights and feedback which allowed me to steer them in the right path. I would also like to thank Prof. Dr.-Ing. Felix Freiling for kindly agreeing to be my second referee and providing me with constructive comments for both the dissertation and the final presentation of my work.

I thank Prof. Dr. Thomas Bläsius for accepting the invitation to be examiner for my oral defense. Likewise, I am thankful to Prof. Dr. rer. nat. Hannes Hartenstein and Prof. Dr. Kathrin Gerling for being part of the defense as members of the doctoral committee.

I am deeply grateful to Dr.-Ing. Lars Bauer for his constant insights, questions and discussions over my ideas, even and especially when he did not have to. His support was crucial to help me shape my crazy ideas into proper scientific contributions. I also would like to thank my colleagues at CES, past and present, for their friendship and support. Special thanks to Dr.-Ing. Heba Khdr, Dr.-Ing. Hassan Nassar and Bakr Sikal for their counsel and collaboration as coauthors in joint publications.

My deepest gratitude goes to my wife, Laura, who has stood next to me with love and patience over the past years. *Lau* has always supported me, even before we decided to move to a different country so I could chase my dream. I am aware of the sacrifices she has selflessly made for me and I will be forever grateful for that. I would not be where I am if not for her.

To my family and extended family in Costa Rica, thanks for the constant support and good wishes.

Jeferson Gonzalez Gomez

Abstract

With the rapid advancement of emerging computing systems such as Internet of Things (IoT), Cloud, and Edge Computing, traditional security measures are being challenged. This modern setting requires robust and efficient security frameworks alongside innovative detection mechanisms, leading to a complex security landscape. Emerging heterogeneous computing environments are increasingly being deployed. Modern IoT and AI computing systems leverage heterogeneous resources such as Graphics Processing Units (GPUs) and custom hardware accelerators to enhance various applications, e.g., machine learning and data processing. However, security and privacy concerns persist, especially with new attack vectors such as side and covert channels. This dissertation focuses on identifying these new threats and developing efficient detection mechanisms as well as countermeasures to ensure security with minimal performance impact on these emerging systems. This dissertation proposes three main contributions to address these challenges.

As the computing capabilities of emerging systems improve, attackers continuously find new ways to compromise the system. The first contribution deals with unveiling new attacks on these emerging systems. For this, it includes four new types of covert channel attacks that leverage different hardware and software resources in a cross-layer manner. The first attack employs the temperature of the GPU to leak information. The second technique introduces an obfuscation mechanism to CPU-based thermal covert channels that defeats traditional detection techniques. The third attack leverages memory usage to establish a stealth channel from a virtual machine to a host system. The last attack exploits a benign hardware accelerator in an FPGA-Multi Processor System on Chip (MPSoC) for the first time to covertly transmit information from a Trusted Execution Environment (TEE) to a Normal World operating system, hence breaking the isolation principle of the TEE.

The first step in addressing these new threats is the detection mechanisms. Since attacks leverage new resources or actively seek to hide themselves,

detection techniques need to adapt to the improved threat. The second contribution proposes two detection mechanisms that employ Machine Learning (ML) techniques. The first detection technique introduces a supervised learning approach to detect thermal covert channels using CPU performance information over time as input. The second detection approach is an unsupervised learning-based framework for general anomaly detection used in the context of dynamic integrity verification and Remote Attestation (RA). In both approaches, the goal of detecting the threat is complemented by the need to minimize the associated overhead in the system. The trade-off between security and performance is a major theme in the contributions.

While the detection techniques for these new attacks have advanced in addressing the threats, the solution to the problem of potential attackers requires action. Straightforward solutions that seek to stop the suspicious applications might either not be feasible or could produce significant service disruption in the system. The third contribution delves into efficient-yet-effective system-level approaches to target covert and side channels. The first countermeasure involves a heuristic-based task migration technique to combat cache-based side channels. The second targets GPU-based thermal covert channels by leveraging an improved heuristic for Dynamic Voltage and Frequency Scaling (DVFS) on both embedded and general-purpose devices. Finally, the third countermeasure combines DVFS and task migration with both heuristics and ML algorithms in a holistic and system-informed manner to combat power-based covert channels efficiently from an energy and performance perspective.

Zusammenfassung

Mit dem raschen Fortschritt aufstrebender Computersysteme wie Internet of Things (IoT), Cloud und Edge Computing werden traditionelle Sicherheitsmaßnahmen zunehmend herausgefordert. Diese moderne Umgebung erfordert robuste und effiziente Sicherheitsstrukturen sowie innovative Erkennungsmechanismen, was zu einer komplexen Sicherheitslandschaft führt. Aufstrebende heterogene Rechenumgebungen werden zunehmend eingesetzt. Moderne IoT- und KI-Computersysteme nutzen heterogene Ressourcen wie Graphics Processing Units (GPUs) und benutzerdefinierte Hardware-Beschleuniger, um verschiedene Anwendungen, z.B. maschinelles Lernen und Datenverarbeitung, zu verbessern. Dennoch bestehen weiterhin Bedenken hinsichtlich der Sicherheit und des Datenschutzes, insbesondere durch neue Angriffsvektoren wie Seitenkanäle und verdeckte Kanäle. Diese Dissertation konzentriert sich darauf, diese neuen Bedrohungen zu identifizieren und effiziente Erkennungsmechanismen sowie Gegenmaßnahmen zu entwickeln, um die Sicherheit dieser aufstrebenden Systeme bei minimalen Leistungseinbußen zu gewährleisten. Diese Dissertation stellt drei wesentliche Beiträge zur Bewältigung dieser Herausforderungen vor.

Mit der Verbesserung der Rechenleistung aufstrebender Systeme finden Angreifer ständig neue Wege, das System zu kompromittieren. Der erste Beitrag befasst sich mit der Aufdeckung neuer Angriffe auf diese aufstrebenden Systeme. Dazu gehören vier neue Arten von Angriffen durch verdeckte Kanäle, die verschiedene Hardware- und Software-Ressourcen in einer schichtenübergreifenden Weise nutzen. Der erste Angriff verwendet die Temperatur der GPU, um Informationen zu leaken. Die zweite Technik führt einen Verschleierungsmechanismus für CPU-basierte thermische verdeckte Kanäle ein, der herkömmliche Erkennungstechniken überwindet. Der dritte Angriff nutzt den Speicherverbrauch, um einen verdeckten Kanal von einer virtuellen Maschine zu einem Host-System aufzubauen. Der letzte Angriff nutzt zum ersten Mal einen gutartigen Hardware-Beschleuniger in einem FPGA-MPSoC, um Informationen verdeckt von einer Trusted Execution Environment (TEE)

an ein Betriebssystem in der normalen Welt zu übertragen, wodurch das Isolationsprinzip der TEE verletzt wird.

Der erste Schritt zur Bewältigung dieser neuen Bedrohungen besteht in den Erkennungsmechanismen. Da Angriffe neue Ressourcen nutzen oder aktiv versuchen, sich zu verbergen, müssen Erkennungstechniken an die verbesserte Bedrohung angepasst werden. Der zweite Beitrag schlägt zwei Erkennungsmechanismen vor, die ML-Techniken verwenden. Die erste Erkennungstechnik führt einen überwachten Lernansatz ein, um thermische verdeckte Kanäle durch die Nutzung von CPU-Leistungsinformationen über die Zeit als Eingabe zu erkennen. Der zweite Erkennungsansatz ist ein auf unüberwachtem Lernen basierendes Framework zur allgemeinen Anomalieerkennung, das im Kontext der dynamischen Integritätsüberprüfung und der Remote Attestation (RA) verwendet wird. In beiden Ansätzen wird das Ziel der Bedrohungserkennung durch das Bestreben ergänzt, den damit verbundenen Overhead im System zu minimieren. Der Ausgleich zwischen Sicherheit und Leistung ist ein zentrales Thema in den Beiträgen.

Während die Erkennungstechniken für diese neuen Angriffe Fortschritte bei der Bewältigung der Bedrohungen gemacht haben, erfordert die Lösung des Problems potenzieller Angreifer Maßnahmen. Einfache Lösungen, die darauf abzielen, verdächtige Anwendungen zu stoppen, sind möglicherweise nicht praktikabel oder könnten erhebliche Dienstunterbrechungen im System verursachen. Der dritte Beitrag vertieft sich in effiziente und zugleich effektive systemweite Ansätze, um verdeckte und Seitenkanäle anzugehen. Die erste Gegenmaßnahme beinhaltet eine heuristikbasierte Technik zur Migration von Anwendung, um cachebasierte Seitenkanäle zu bekämpfen. Die zweite zielt auf GPU-basierte thermische verdeckte Kanäle ab, indem eine verbesserte Heuristik für Dynamic Voltage and Frequency Scaling (DVFS) sowohl auf eingebetteten als auch auf allgemeinen Geräten genutzt wird. Schließlich kombiniert die dritte Gegenmaßnahme DVFS und Migration von Anwendung mit sowohl Heuristiken als auch ML-Algorithmen in einer ganzheitlichen und systeminformierten Weise, um auf dem Leistungsverbrauch basierende verdeckte Kanäle effizient aus einer Energie- und Performance-Aspekte zu bekämpfen.

Contents

List of Figures

List of Tables

List of Publications

The following list enummerates the papers and book chapters published by the author of this dissertation during the course of his doctoral studies. Publications [1–7] make **major** contributions to this dissertation, while [8–12] bring **minor** contributions.

[1] Jeferson Gonzalez-Gomez, Mohammed Bakr Sikal, Heba Khdr, Lars Bauer, and Jörg Henkel. "Smart Detection of Obfuscated Thermal Covert Channel Attacks in Many-core Processors". In: *2023 60th ACM/IEEE Design Automation Conference (DAC)*. 2023, pp. 1–6. DOI: 10.1109/DAC56929.2023.10247844.

[2] Jeferson Gonzalez-Gomez, Kevin Cordero-Zuñiga, Lars Bauer, and Jörg Henkel. "The First Concept and Real-world Deployment of a GPU-based Thermal Covert Channel: Attack and Countermeasures". In: *2023 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. 2023, pp. 1–6. DOI: 10.23919/DATE56975.2023.10137090.

[3] Jeferson Gonzalez-Gomez, Hassan Nassar, Varun Manjunath, Lars Bauer, and Jörg Henkel. "Through Fabric: A Cross-world Thermal Covert Channel on TEE-enhanced FPGA-MPSoC Systems". In: *Asia and South Pacific Design Automation Conference (ASP-DAC)*. accepted to appear. 2025.

[4] Jeferson Gonzalez-Gomez, Jose Alejandro Ibarra-Campos, Jesus Yamir Sandoval-Morales, Lars Bauer, and Jörg Henkel. *MeMoir: A Software-Driven Covert Channel based on Memory Usage*. 2024. arXiv: 2409.13310 [cs.CR]. URL: https://arxiv.org/abs/2409.13310.

[5] Jeferson Gonzalez-Gomez, Hassan Nassar, Lars Bauer, and Jörg Henkel. "LightFAt: Mitigating Control-Flow Explosion via Lightweight PMU-Based Control-Flow Attestation". In: *2024 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*. 2024, pp. 222–226. DOI: 10.1109/HOST55342.2024.10545348.

[6] Jeferson Gonzalez-Gomez, Lars Bauer, and Jörg Henkel. "Cache-Based Side-Channel Attack Mitigation for Many-Core Distributed Systems via Dynamic Task Migration". In: *IEEE Transactions on Information Forensics and Security* 18 (2023), pp. 2440–2450. DOI: 10.1109/TIFS.2023.3266630.

[7] Jeferson González-Gómez, Mohammed Bakr Sikal, Heba Khdr, Lars Bauer, and Jörg Henkel. "Balancing Security and Efficiency: System-Informed Mitigation of Power-Based Covert Channels". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 43.11 (2024), pp. 3395–3406. DOI: 10.1109/TCAD.2024.3438999.

[8] Lars Bauer, Jörg Henkel, Timo Hönig, Wolfgang Schröder-Preikschat, Christian Eichler, Jeferson Gonzalez, Benedict Herzog, Tobias Langer, Sebastian Maier, Jonas Rabenstein, et al. "Invasive Run-Time Support System (iRTSS)". In: *Invasive Computing*. Ed. by Jürgen Teich, Jörg Henkel, and Andreas Herkersdorf. Cham: Springer International Publishing, 2022, pp. 285–305. ISBN: 978-3-96147-571-1. DOI: 10.25593/978-3-96147-571-1.

[9] Antonio González-Torres, Mónica Hernández, Jeferson González, Vetria L. Byrd, and Paul Parsons. "Information Visualization as a Method for Cybersecurity Education". In: *Innovations in Cybersecurity Education*. Ed. by Kevin Daimi and Guillermo Francia III. Cham: Springer International Publishing, 2020, pp. 55–70. ISBN: 978-3-030-50244-7. DOI: 10.1007/978-3-030-50244-7_4.

[10] Jeferson González-Gómez, Steven Ávila, Jonathan Rojas, Andres Stephen, Jorge Castro-Godínez, Carlos Salazar-García, Muhammad Shafique, and Jörg Henkel. "TailoredCore: Generating Application-Specific RISC-V-based Cores". In: *2021 IEEE 12th Latin America Symposium on Circuits and System (LASCAS)*. 2021, pp. 1–4. DOI: 10.1109/LASCAS51355.2021.9459152.

[11] Carlos Salazar-García, Jeferson González-Gómez, Kaleb Alfaro-Badilla, Ronny García-Ramírez, Renato Rímolo-Donadío, Christos Strydis, and Alfonso Chacón-Rodríguez. "PlasticNet: A low latency flexible network architecture for interconnected multi-FPGA systems". In: *2020 IEEE 3rd Conference on PhD Research in Microelectronics and Electronics in Latin America (PRIME-LA)*. 2020, pp. 1–4. DOI: 10.1109/PRIME-LA47693.2020.9062749.

[12] Carlos Salazar-García, Alfonso Chacón-Rodríguez, Renato Rímolo-Donadío, Ronny García-Ramírez, David Solórzano-Pacheco, Jeferson González-Gómez, and Christos Strydis. “A custom interconnection multi-FPGA framework for distributed processing applications”. In: *2022 35th SBC/SBMicro/IEEE/ACM Symposium on Integrated Circuits and Systems Design (SBCCI)*. 2022, pp. 1–6. DOI: 10.1109/SBCCI55532.2022.9893238.

Research at the Chair for Embedded Systems

At the Chair for Embedded Systems (CES), the current research addresses fundamental challenges in modern computing systems, focusing on non-volatile memory (NVM), resource management for multicore and embedded architectures, cross-layer security, and reconfigurable computing. The research done at CES on these areas aims to improve system performance, energy efficiency, transistor aging, adaptability, and security [14, 70].

Reconfigurable Systems

Reconfigurable computing at CES focuses on creating self-organizing adaptive systems capable of dynamic resource management. By leveraging hardware-software co-design principles, CES develops architectures that adjust resources in real time based on application needs [23, 145]. These systems enable performance optimization, energy savings, and scalability [50, 52], especially in fields such as approximate computing, where precision can be compromised in exchange for higher efficiency [24, 72]. This adaptability also contributes to improving the longevity and robustness of the hardware in heterogeneous systems.

Resource Management for Multicore and Embedded Systems

Resource management is key to enhancing the energy efficiency, performance, and reliability of multicore systems. CES researchers develop dynamic resource allocation to balance workloads and manage energy consumption across heterogeneous cores, addressing power and thermal management [47,

68], mitigating hardware aging, and optimizing power distribution [69, 98]. CES research proposes techniques to reduce heat-induced degradation and transistor wear [75, 92], ensuring sustained performance under variable workloads. They also focus on machine learning for low-resource environments such as embedded systems and IoT devices, designing lightweight ML models using model compression, approximate computing, and energy-efficient inference [48, 133]. These methods enable advanced ML algorithms on modest hardware.

Non-Volatile Memory (NVM)

NVM research at CES is centered on the development of memory technologies that retain data without power for emergin applications. CES explores novel NVM technologies that offer higher density and capacity [71] with low latency and endurance, positioning them as ideal candidates for future memory hierarchies. In the domain of machine-learning applications, CES has concetrating on leveraging the different retention times of the NVM technologies to improve the performance of complex workloads such as Convulational Neural Networks (CNNs) [147].

Cross-Layer Security

Cross-layer security is a key focus for CES, addressing vulnerabilities across the hardware-software stack. Researchers at CES develop techniques to protect against threats from physical tampering to software attacks. This includes using cryptographic methods for embedded systems, mitigating side-channel attacks [117], providing attack-resilient hardware primitives [116], and implementing attack detection mechanisms [115]. By applying security techniques across layers, CES aims to enhance system robustness and reliability in complex computing environments.

This dissertation tackles challenges in cross-layer security, focusing on new attack vectors and resource-efficient countermeasures. It also integrates relevant aspects from other CES research areas such as resource management and machine learning.

1 Introduction

In today's rapidly evolving digital landscape, robust security and data privacy are more critical than ever. As technology advances and reliance on digital systems grows, so do the risks posed by sophisticated cyber threats. These threats continually adapt, exploiting emerging vulnerabilities and challenging existing security measures. This highlights the urgent need for advanced defenses to safeguard sensitive information and maintain system integrity.

A key concern is the exploitation of side channels, where attackers leverage subtle information leaks from a system's physical implementation. Cache-based Side-channel Attack (SCA), for example, exploit timing variations in cache memory access to extract confidential information, such as secret keys during cryptographic operations [105]. As modern many-core systems, including those in autonomous vehicles [163], high-end computers, GPUs, and AI accelerators [73], become more prevalent, effective resource management techniques are crucial. Such techniques aim to optimize system e.g., from a performance or energy efficiency perspective. Moreover, in the security domain, such techniques can be leveraged to help mitigate resource-based attacks, enhancing security without requiring significant hardware or architectural changes.

In addition to side channels, covert channels represent another pressing security concern. These channels exploit covert means of communication between malicious applications, often leveraging system power consumption to transmit information surreptitiously. For example, power-based covert channels, or thermal covert channels, modulate power usage to create temperature variations that encode and transmit data. This technique involves intensive computations on processing elements such as CPUs [109], GPUs [2], or FPGA-based components [58], which can be exploited to covertly communicate between malicious applications. As the sophistication of such threats increases, addressing both side-channel and covert channel vulnerabilities in a cross-layer fashion (i.e., considering both software and hardware

mechanisms) becomes essential for maintaining robust security in modern computing systems.

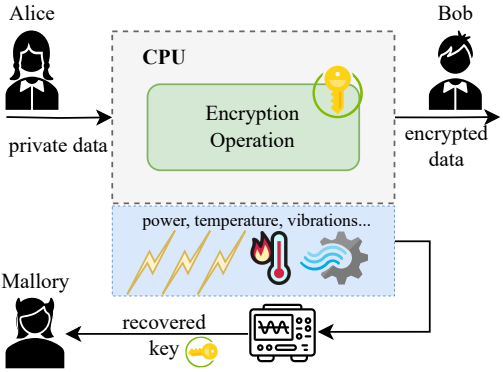
This dissertation focuses on advancing the field of security by addressing emerging threats in modern computing systems through the different computing layers. First, it explores new attack vectors that either leverage new hardware resources (e.g., GPU, memory, and accelerators) or employ sophisticated software-based techniques to evade detection such as obfuscation and complex modulation. Secondly, this dissertation studies advanced detection mechanisms designed to identify and mitigate these threats effectively by employing ML-based approaches. Finally, this dissertation studies efficient system-level countermeasures tailored to address the detected attack methods in a diverse range of computing environments. By bridging the gap between evolving threats and contemporary defense strategies, this work contributes to enhancing the resilience and security of modern computational systems.

1.1 Side and Covert Channels in Computing Systems

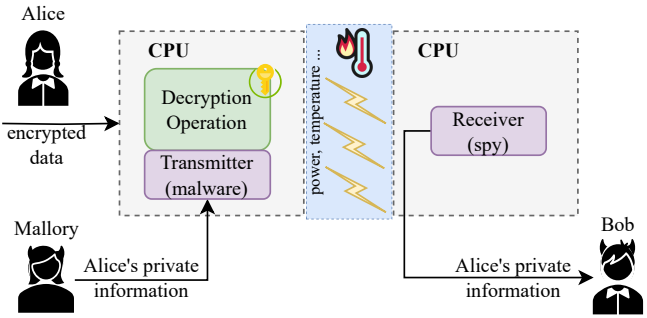
Side channels and covert channels represent significant security concerns in modern computing systems, as both exploit unintended information leakage to compromise data confidentiality. A simple generalized example of a side channel and a covert channel attack is depicted in Fig. 1.1. While side channels (Fig. 1.1a) extract sensitive information through observable physical phenomena such as timing or power consumption, covert channels (Fig. 1.1b) establish communication through an stealthy and unintended medium. The increasing sophistication of these attacks underscores the need for advanced detection and mitigation strategies, making them critical areas of focus in ensuring the security of contemporary computing environments.

Although these emerging attacks come in a diverse spectrum of implementations, the following subsections describe the specific types of attacks which are addressed in this dissertation.

Side-channel Attacks (SCAs) A SCA is a type of attack that takes advantage of implementation-specific characteristics of a system or an application. In such an attack, an adversary tries to exploit physical information leakages,



(a) Side-channel attack.



(b) Covert-channel attack.

Figure 1.1: Representation of side- and covert-channel attacks. In (a) Mallory (attacker) extract's Alice private key by analyzing a system leakage e.g., power, temperature, vibrations, etc. In (b) Mallory (attacker) communicates Alice's private information (e.g., a private key) to Bob (also attacker) using an unintended and hidden medium e.g., system power or temperature.

such as temperature [45], power consumption [114], or timing behavior [80], to obtain secret information from the victim. SCAs are often passive (require mostly observation) and non-invasive (exploit available information). Due to this nature, SCAs are hard to detect and they usually do not require special ex-

ecution privileges, thus making it difficult to eliminate them without affecting performance [67]. All these factors combined make SCAs a dangerous threat to the security of most current cryptographic hardware systems [150].

Cache-based SCAs are a particular type of attack that exploit the intrinsic timing nature of cache memories to extract security-critical information from variations of access time patterns on the target application through different techniques. In the context of cryptographic security applications (from now on *secure applications*), cache-based SCAs seek to obtain secret keys by analyzing the fine-grained timing behavior of either the secure application or the adversary itself, using their cache access times and misses patterns when performing encryption or decryption operations [105].

Covert Channel Attacks A covert channel attack is a means of communication between two applications or processes that are not permitted to communicate within a specific system [112]. Recently, covert channel communication has emerged as a significant security threat in modern computing systems [111]. These channels exploit hidden vulnerabilities in hardware and software, creating a secret medium for information transfer that is difficult for regular users to detect [141]. While covert channels can be used for legitimate purposes, such as safeguarding confidential data, they are often exploited by attackers to ex-filtrate spied information on systems or spread malicious software [138].

Among the various types of covert channels, power-based covert channels, such as thermal covert channels (TCCs), have gained attention for their stealthiness and potential for significant harm. TCCs utilize temperature variations in a system's components to establish hidden communication between malicious applications. This type of attack has been studied across various domains, including multi/many-core systems [77, 103, 109], cloud FPGA environments [58], and embedded systems [35]. In a typical TCC attack, represented in Fig. 1.2, a compromised application (transmitter) in an isolated environment encodes sensitive data as temperature fluctuations through intensive CPU usage, which are then detected by a second application, or receiver, located in a nonsecure zone (normal environment). By interpreting these temperature variations, the receiver can decode the transmitted information, thus establishing a covert communication channel [77, 78].

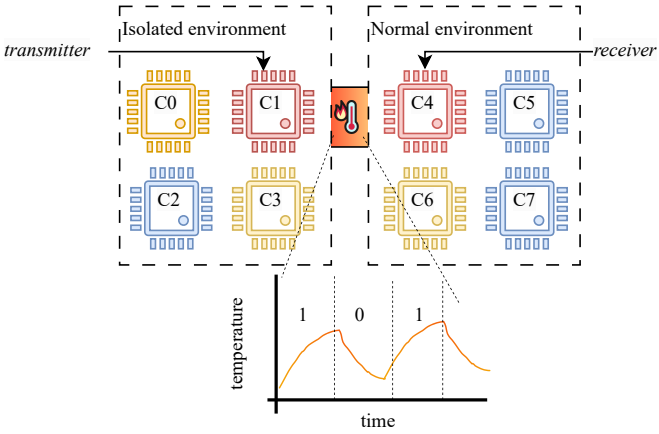


Figure 1.2: Representation of a thermal covert channel.

Challenges in attack implementations

Several challenges arise from the attacker's perspective when targeting new and emerging computing systems. Since detection and countermeasure techniques must mitigate the new threat, it is relevant to highlight potential improvements in attack mechanisms.

Challenge: Exploiting new resources With the increase of more powerful and specialized computing hardware, such as hardware accelerators for ML workloads [136], a malicious actor might take advantage of these new resources to extract private information or steal intellectual property information, for example, model features [167]. From the attacker perspective, the attack vectors must in turn become more complex to match the specialization of the new resources which might shielded by isolation or virtualization technologies [134]. The risk associated with the increase of this complexity is the increased probability of being detected due to the effects produced by the attacker in the system.

Challenge: Avoid detection Since these new and powerful resources perform potentially critical operations for the user, attack detection mechanisms must improve their capabilities to match those of the attacker. From the attacker perspective, this represents a call to further increase its stealthiness through techniques such as obfuscation [28] or unpredictable attack patterns, for example, those in time-of-check to time-to-use attacks [76, 93]. This increase in stealthiness often results in a decrease in the attack performance in terms of speed, e.g. lower transmission rates in covert channels or longer attack periods for side-channel attacks.

1.2 Challenges in Attack Detection

In the context of detecting and mitigating modern security threats, several critical challenges must be addressed to ensure effective protection, especially as attackers continually evolve their methods.

Challenge: Attack knowledge One primary challenge is the need for prior knowledge of potential attacks during the design phase of a detection technique. Effective detection mechanisms often rely on distinguishing between normal and abnormal behavior [77, 159], which requires an understanding of attack patterns. However, at the design stage, it is unlikely that all possible threats are known, making it difficult to prepare for the full range of potential attacks. This limitation complicates the development of accurate and comprehensive security measures, particularly when attackers might introduce novel methods such as obfuscation or the use of previously untapped resources that the system is not initially designed to detect.

Challenge: Application monitoring Another significant challenge arises in monitoring the dynamic behavior of applications. Continuous monitoring throughout an application's execution could obscure an attack if it blends in with normal operations over an extended period. In contrast, if monitoring skips certain parts of the execution, attacks occurring in these unmeasured segments might go undetected e.g., in time-of-check to time-of-use attacks [93]. Determining the most relevant parts of the application to monitor becomes a complex task, especially when relying on dynamic behavior as the

basis for threat detection. Moreover, attackers may exploit these monitoring gaps, making it even more critical to refine detection strategies.

Challenge: System performance vs security Finally, there is a persistent challenge in balancing security and performance. While monitoring an application's behavior may impose less of a performance burden than traditional methods such as code instrumentation or control-flow disruption [13], it can also reduce the accuracy of the security measures. Unlike more deterministic approaches (e.g., those employing specialized hardware [13, 43, 169]), where it is straightforward to identify the presence of a threat, the variability in an application's behavior on existing real systems can make it difficult to definitively classify actions as either safe or malicious. The key challenge, therefore, is to develop a security solution that offers high accuracy while minimizing performance overhead, enabling the protection of complex applications without compromising their efficiency.

These challenges emphasize the need for innovative approaches capable of adapting to the ever-changing landscape of security threats in modern computing environments. As attackers continually refine their techniques, detection mechanisms must evolve simultaneously to maintain robust and reliable defense systems.

1.3 Challenges in Countermeasures

When addressing countermeasures for modern security threats, including both side-channel attacks and covert channels, several key challenges must be tackled to ensure effective protection considering the underlying platform architecture and the impact that techniques might have on it in terms of performance and energy efficiency.

Challenges in mitigating cache-based side-channel attacks

Mitigating cache-based side channels involves addressing two primary challenges: the trade-off between hardware and software solutions and managing the associated overhead. Effective countermeasure strategies must navigate

this trade-off to ensure both robust security and efficient system performance.

Challenge: Hardware specialization Balancing the need to overcome security threats while managing system overhead presents a fundamental challenge in countermeasure design. Specialized hardware approaches (e.g., cache randomization [94] or adaptive architectures [21]), can effectively reduce vulnerabilities and enhance resilience against attacks. However, these solutions typically involve substantial modifications to existing system architectures, which can be costly and complex, making them difficult to implement in many real-world systems.

Challenge: Software-based overhead In contrast to hardware approaches, software-based countermeasures (e.g., compiler modifications [39] or operating system adjustments [108]) offer the advantage of being applicable to current systems with minimal hardware changes. Yet, these solutions often come with significant performance overhead, due to factors such as execution flow alterations or frequent cache flushing.

Consequently, the solution to these challenges lies in finding a balance between the robust protection offered by hardware modifications and the more flexible, but potentially more overhead-prone, software-based solutions. Effective countermeasure strategies must navigate this trade-off to ensure both robust security and efficient system performance.

Challenges in mitigating power-based covert channels.

Mitigating covert channels, especially those leveraging power consumption and thermal variations, presents significant challenges. These challenges primarily revolve around balancing the mitigation efforts with the associated overheads and addressing the adaptability of attackers. Effective countermeasures must be carefully designed to ensure they do not introduce unacceptable performance or energy overheads, while also being robust enough to prevent attackers from circumventing them.

Balancing mitigation and overhead Techniques such as Dynamic Voltage and Frequency Scaling (DVFS) and noise generation are effective in disrupting covert communication channels by affecting power consumption and thermal variations. However, these methods often introduce performance or energy overheads, which is particularly critical for energy-constrained systems. For instance, DVFS can lead to significant performance degradation, as it has been proven to induce performance losses of up to 25% [77] for benchmark applications. Additionally, the impact on overall system energy consumption must be carefully considered to ensure the viability of these techniques in practical applications.

Attack adaptability Attackers can potentially learn to overcome mitigation strategies by adapting to the noise levels or changing transmission frequencies. To counter this, more complex solutions, such as employing machine learning, may be required. However, these advanced strategies can also incur high overheads. Therefore, it is essential to analyze and develop efficient mitigation strategies that balance complexity and overhead, ensuring that the solutions remain effective without imposing excessive burdens on the system.

Overall, the development of effective countermeasures for both side-channel and covert channel attacks requires balancing security, performance, and practical implementation considerations. Addressing these challenges involves refining existing techniques and exploring innovative solutions that can adapt to evolving threat landscapes and diverse system architectures.

1.4 Dissertation Contributions

The contributions of this dissertation are categorized into three main topics: new attacks, detection techniques, and efficient countermeasures. A high-level overview of the contributions is depicted in Fig. 1.3, where the different techniques are represented in a cross-layer manner in the hardware-software stack. These contributions aim to increase the overall system security by addressing the challenges described above. In more detailed perspective, the key contributions are the following.

- Four implementations of new covert channels have been proposed, which were previously undisclosed by the state-of-the-art. These

include an obfuscated Thermal Covert Channel (TCC), a GPU-based TCC, a MPSoC FPGA TCC, and a software-driven covert channel based on memory usage. The **new attacks** are shown to communicate information effectively between malicious applications. These new threats show how isolation-based security mechanisms, such as those found in TEEs or Virtual Machines (VMs), can be bypassed by attackers, highlighting the need for improved detection and countermeasure techniques.

- An ML-based **detection technique** for TCCs has been developed, considering traditional attacks, as well as the new obfuscated covert channel. This approach uses CPU time-series performance information

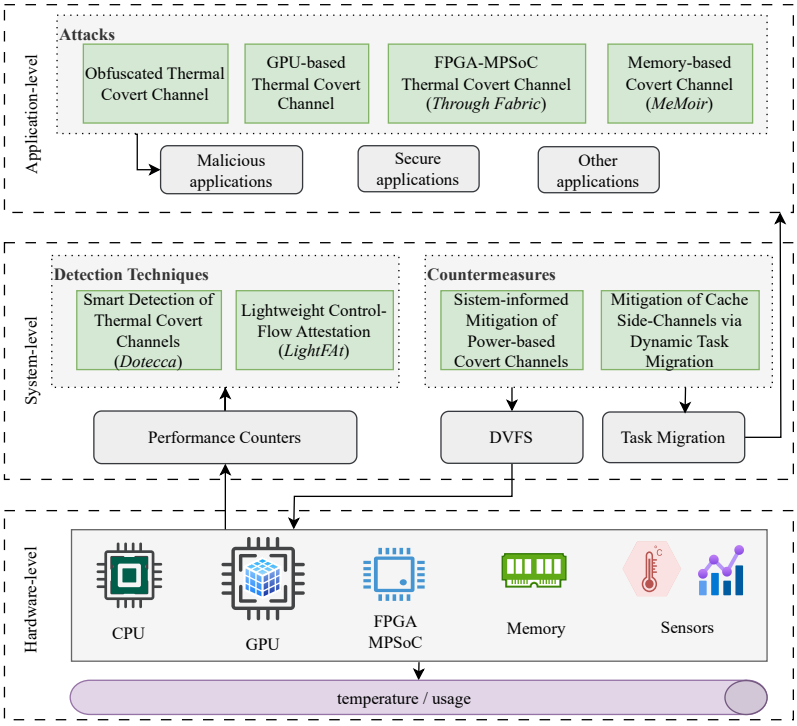


Figure 1.3: An overview of the contributions (colored in green) presented in this dissertation.

to predict whether the execution behavior can be classified as an attack or a normal trace. A lightweight Neural Network (NN) model is developed and trained following a supervised learning approach, where samples for normal executions (benchmark applications), traditional and new attacks are used. This technique enables the detection of threats with high accuracy, while inducing minimal overhead in the system.

- An unsupervised learning-based technique for detecting anomalies in application execution has been developed. This technique is applied within the context of RA, where the prover device collects its execution traces from the CPU's Performance Monitoring Unit (PMU). These traces are then sent to the verifier entity as part of the attestation process. The verifier uses one of three different ML models to classify the execution trace as either normal or malicious. By leveraging an unsupervised learning approach, this **detection technique** models normal behavior without requiring real attacker traces during training. Consequently, this approach provides a more formal and generalized method for detecting new security threats.
- A lightweight heuristic-based task migration technique has been developed to mitigate cache side-channel attacks. The approach leverages task migration to prevent cluster co-residency between the potential attacker and the victim application. This **countermeasure** relies on a migration function to trigger the migration process. The function considers both the performance degradation of the security-critical task due to the attacker and the shared execution time of the application with others on the same cluster. This technique provides a secure execution scenario for a security-critical application by migrating the application between CPU clusters while minimizing system overhead.
- A methodology has been developed to mitigate power-based covert channel attacks using heuristics and ML-based techniques in a system-informed manner. This approach relies on task migration and DVFS as the acting mechanisms. The **countermeasure** leverages system information, such as CPU and memory performance, to predict the energy efficiency of various migration scenarios during a potential attack. The most energy-efficient scenario is then enforced through task migration, while DVFS is applied to the cluster where the potential

attacker resides to disrupt the CPU power response and interfere with transmission.

1.5 Dissertation Outline

The following chapter discusses related work on covert and side-channel attacks, as well as countermeasures and threat detection techniques. Chapter 3 introduces the various frameworks used to evaluate the techniques presented in this dissertation, including both simulation setups and a range of real computing platforms. Chapters 4 to 8 preset the techniques developed in this dissertation. Chapter 4 begins by presenting four novel covert channel attacks on different computing systems. The improved attack detection techniques are next presented in Chapters 5 and 6. Chapter 5 describes a ML-based attack detection approach for TCCs, while Chapter 6 presents a generalized execution anomaly detection technique framed in the context of RA. The proposed countermeasures to the attacks are presented in Chapters 7 and 8. Chapter 7 describes an heuristic-based task migration approach to mitigate cache-based side channels by avoiding cluster-level cache co-residency between attacker and victim. In Chapter 8, a methodology is proposed using heuristics and ML techniques to efficiently combat power-based covert channels using the combination of task migration and DVFS. Finally, Chapter 9 concludes this dissertation and discusses future research directions emerging from it.

2 Related Work

In this chapter, the background and related work for side-channel and covert-channel attacks is presented, including security advances such as trusted execution environments, detection mechanisms and state-of-the-art countermeasures.

2.1 Trusted Execution Environments

A Trusted Execution Environment (TEE) is a secure and tamper-resistant processing environment, which guarantees the authenticity and integrity of the system's dynamic state as well as code and data confidentiality [137].

ARM TrustZone [18] provides hardware support for several TEEs on ARM-based systems. It ensures an isolated execution context for security-critical applications by virtually separating the hardware and software resources into two spaces: the *secure world* or TEE and the *normal world* or Rich Execution Environment (REE) [99, 113], e.g., Linux. This hardware-supported virtual separation allows computing and memory-shared resources to be private to each environment, providing the isolation mechanism needed for the TEE.

On the software side, OP-TEE is a TrustZone implementation for Cortex-A processors. OP-TEE aims at providing isolation from the normal world applications i.e., Client Applications (CAs) and operating system, as well as protecting the secure world applications i.e., Trusted Applications (TAs), from each other [154].

Although the isolation principle of TEEs serves as a good security mechanism for integrity and confidentially against a wide variety of attacks, it does not consider covert and side channels in its design [104], which makes it a valid target for such threats. In the context of Trusted Computing, where TEEs belong, side and covert channels exploit vulnerable or leaky TAs to steal and

extract private data from the secure isolated world, breaking the fundamental isolation principle of such technologies.

2.2 Side- and Covert-channel Attacks

In modern computing systems, both side and covert-channel attacks have emerged as significant threats, capable of compromising data security by exploiting unintended information leaks. These attacks pose a challenge to traditional security mechanisms, as they often bypass conventional protections and exploit subtle vulnerabilities in system architecture. This section provides a detailed description of these attacks, focusing on cache-based side-channel attacks (SCAs) and power-based covert channels, including their specific implementations in various hardware and software environments.

Cache-based Side-channel Attacks

Cache-based SCAs are a particular type of attack that exploit the intrinsic timing characteristics of cache memories to extract security-critical information from variations in access time patterns. These attacks are especially concerning in the context of cryptographic applications, where they aim to obtain secret keys by analyzing the timing behavior associated with cache accesses and misses during encryption or decryption processes [105].

This subsection delves into the fundamentals of cache-based SCAs, exploring the different types of attacks and the highlighting characteristic features employed in detection and mitigation.

Cache-based SCAs can be broadly categorized into *time*- or *access*-driven attacks, based on the source of the timing information, whether extracted from the victim or the attacker.

Time-driven Attacks In time-driven attacks, the adversary triggers a cryptographic operation and measures the victim's execution time. The motivation of the attack is that the value of the secret key influences the execution time of encryption or decryption processes [105]. By manipulating the content

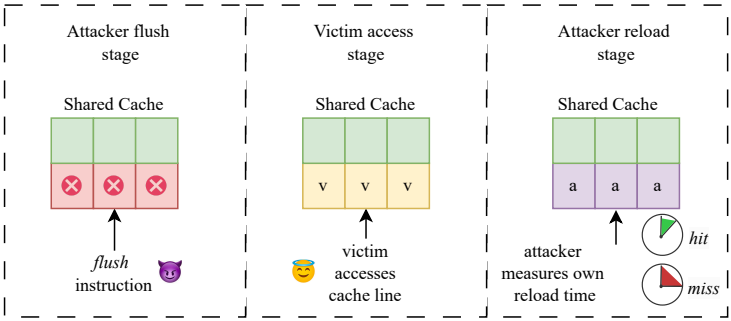


Figure 2.1: Representation of the 'Flush+Reload' [168] attack.

of a shared cache and observing the execution time, the adversary can extract partial or complete keys. A notable example of this type of attack is 'Evict+Time' [124].

Access-driven Attacks Access-driven attacks involve the adversary monitoring their own execution time to determine whether a particular cache line has been evicted by the victim [119]. The attacker continuously loads a table that fills the shared cache, and by measuring the time to reload parts of this table (see Fig. 2.1), they can infer which cache lines were evicted by the victim (i.e., where a miss happened). This timing information can then be used to deduce the secret key, as demonstrated in attacks such as 'Prime+Probe' [124] and 'Flush+Reload' [168].

Impact on Time and Performance in Cache SCAs Modern adversaries can often extract private keys from secure applications within seconds to minutes [37, 86, 87]. Previous works have shown that during a cache-based side-channel attack, the shared cache miss rate of a victim (referred from now on as *secure*) application increases due to effect of the attacker in the shared cache, leading to a decrease in performance proportional to the attack's intensity [29, 37, 157].These performance degradation indicators are crucial for detecting and mitigating such attacks.

Covert Channels

Covert channels represent another significant security concern, as they enable hidden communication between processes that are not supposed to exchange information. These channels exploit vulnerabilities in system resources, creating clandestine pathways that are challenging to detect and mitigate. This subsection explores power-based covert channels, including thermal covert channels, with a focus on their implementation in general computing systems and FPGA-based environments.

Power-based Covert Channels in General Computing Systems

In power-based covert channels, malicious applications manipulate the power consumption of a device to communicate information in a stealthy manner. Although some approaches use power directly for communication, such as memory [125], CPUs [65], or cross-device communication [62], the most prevalent power-based covert channels, from a countermeasure perspective, are Thermal Covert Channels (TCCs). These attacks leverage temperature variations caused by power changes to facilitate covert communication. As depicted in Fig. 1.2, in such channels the malicious transmitter application communicates information from an isolated environment (e.g., a TEE) via temperature variations due to controlled processing activity (e.g., by dynamically adjusting the CPU load), which are detected by the receiver application in the regular environment by measuring its own thermal sensor.

Since the early implementations of TCCs on multi-core systems [109], advances in modulation and encoding mechanisms such as Manchester [22] and Return to Zero (RZE)[103] have dramatically increased transmission rates, reaching up to 45 bps with reduced error rates of less than 5% [111].

With the advance in emerging computing systems and technologies, TCCs have extended to new resources such as 3-D multi-core systems [46], and SSDs [153]. This is an indication of the ever-increasing capabilities of threats and their ability to adapt to the new environments.

Covert Channels in Field Programmable Gate Array (FPGA) Systems

Covert channels have been extensively demonstrated in FPGA-based systems, with many attacks exploiting the power distribution network in multi-tenant FPGAs and FPGA-SoCs. These attacks typically target the device's voltage [57, 60, 63] and frequency [27, 53] to modulate power, achieving high transmission speeds and low error rates. However, these approaches often rely on malicious custom hardware modules on the transmitter side, which can be easily detected or restricted by vendors, or even disabled during runtime [115]. Additionally, the receiver modules in these approaches require extra FPGA logic, such as time-to-digital converters (TDCs), to measure power changes in the system.

While most FPGA-based covert channels modulate power, other less common covert channel approaches in FPGAs use non-conventional methods to modulate various resources, such as PCIe usage in cloud systems [59] or internal wiring in multi-tenant FPGA systems [56].

Software-based Covert Channels

Software-based covert channels rely on virtual resources to facilitate communication between malicious applications, making detection particularly challenging [146]. These channels often exploit OS-level synchronization mechanisms [146, 172], page caches [64], or TCP sockets for inter-process communication [49], among others.

Several studies have addressed memory-based covert channels. In [36], the authors estimate the current temperature of a DRAM module using cell decay rates, implementing a thermal covert channel with reliability up to 95%. Saileshwar *et al.* [139] exploits memory contention in shared resource systems, coordinating memory operation timings between transmitter and receiver applications to achieve a transfer speed of 1801kB/s.

2.3 Attack Detection Mechanisms

As threats from side-channel attacks and covert channels evolve, detection mechanisms have become a critical component in maintaining the security

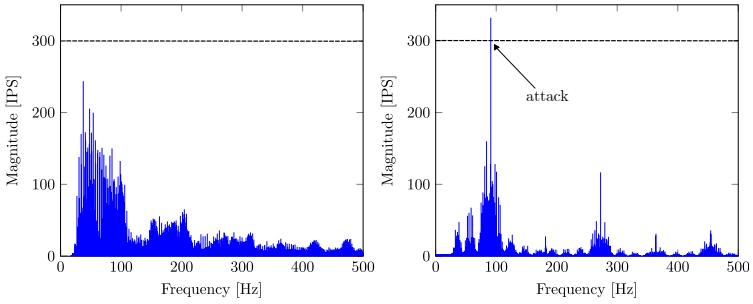
and integrity of computing systems. This section explores two primary types of detection techniques: power-based covert channel detection and generalized software anomaly detection, particularly within the context of Remote Attestation (RA). Each technique addresses different aspects of security, offering complementary strategies to safeguard systems from increasingly sophisticated attacks.

2.3.1 Detection of Power-Based Covert Channels

Power-based covert channels, such as TCCs, pose a significant threat to multi-core systems. The improvements in the attack capabilities described above have imposed challenges on the detection and countermeasure front.

Threshold-based Detection

Early detection methods employed techniques such as Discrete Fourier Transform (DFT) analysis on CPU performance metrics e.g., Instructions per Second (IPS) [78]. These methods used simple threshold-based rules to classify behavior as either normal or malicious based on the prominent component magnitudes in Discrete Fourier Transform (DFT) spectrum, as depicted in Fig. 2.2. While effective for basic attacks, these techniques fail against more sophisticated, stealthier attacks [159].



(a) Normal application (parsec-bodytrack). **(b)** TCC attack.

Figure 2.2: DFT spectrum of a normal application and a TCC attack. Following the approach from [78], a threshold (ρ = 300) is used to different the attack from the normal spectrum.

Some recent and improved attacker features, such as dynamic frequency shifts [158] and shorter data encoding times [159] have highlighted the deficiencies of heuristic-based detection and motivated the need for complex detection approaches.

Machine Learning-based Detection

To overcome the limitations of threshold-based methods, machine learning approaches have emerged. In particular, neural networks have been applied to detect covert channel activity by analyzing the shape of the DFT spectrum of core performance metrics [159]. This technique provides higher accuracy in identifying compromised cores without relying on static thresholds.

However, as demonstrated in Chapter 5, even state-of-the-art machine learning-based detection methods struggle to detect new or evolving attack vectors, highlighting the need for more robust and adaptive detection strategies.

2.3.2 Remote Attestation as Run-Time Integrity Verification

As systems become more interconnected and complex, ensuring their integrity during runtime is crucial. Remote attestation (RA) serves as a generalized mechanism (that is, it does not target specific attack types) to verify the trustworthiness of a system or application by allowing a remote *verifier* to assess the state of a *tester* system. This process is particularly important in scenarios where the system's integrity must be assured continuously, such as in critical infrastructure or high-security environments. This subsection delves into the mechanisms of RA, highlighting both traditional approaches and emerging techniques.

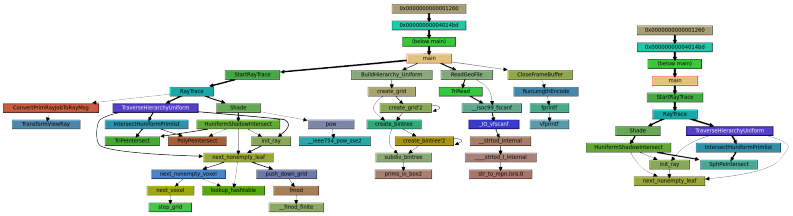
Static and Control-Flow Attestation

Static attestation, the earliest form of RA, involves the prover calculating a cryptographic hash over its code memory or a specified memory range. This method, while effective, is vulnerable to attacks that exploit the time between verification checks, leading to potential Time-of-Check to Time-of-Use (ToCToU) vulnerabilities [93]. To mitigate this, hardware-assisted

solutions have been developed [96] to reduce the attack surface and increase the reliability of the attestation process.

To address the limitations of static attestation, especially in the context of ToCToU attacks, runtime attestation has been proposed [171]. This approach continuously monitors the application, ensuring that its execution adheres to an expected behavior. The primary case of runtime attestation is Control-Flow Attestation (CFA) which verifiers that the program's execution path matches its intended Control-Flow Graph (CFG). This method traditionally calculates a sequence of hashes during runtime, with each hash incorporating the address of the next node in the CFG. However, CFA introduces significant overhead, particularly in applications with complex control flows, leading to issues such as control flow explosion [13, 43].

Control Flow Explosion



(a) Raytrace control-flow graph when processing a car

(b) Raytrace control-flow graph when processing a teapot

Figure 2.3: The executed control-flow graph for the raytrace application (a) for processing the geometry of a car (b) for processing the geometry of a teapot. The geometry of the car is far more complex which leads to a very different behavior from the teapot. The control-flow graph is extracted using the callgrind tool [118].

CFA faces significant challenges when applied to real-world applications due to the problem of control flow explosion [43, 82]. While most CFA solutions perform adequately on small, proof-of-concept applications, they struggle with the complexity and variability found in larger, real-world software. Such applications often have diverse input dependencies and numerous execution paths, making it impractical for a verifier to calculate and manage all possible hash outcomes.

Consider the *raytrace* application from the Splash3 benchmark suite [140]. Figure 2.3 shows the control-flow tracing of this application using Callgrind [118]. When processing different geometries, such as a car Fig. 2.3a and a teapot Fig. 2.3b, control flow complexity varies significantly. A traditional verifier would need to compute the cryptographic hash for every possible execution path, including all loop iterations. Handling the relatively simple control flow in Fig. 2.3b might be feasible for a traditional CFA mechanism, but the complexity in Fig. 2.3a results in an exponential increase in possible paths. The resulting computational overhead of performing a hash computation at each node in the CFG renders traditional CFA solutions impractical.

To address control flow explosion, some solutions attempt to simplify the attestation process by treating loops as single basic blocks, effectively ignoring the complexity within them [42, 123, 173]. However, this approach leaves the system vulnerable to attacks that exploit variations in loop execution, such as changing the number of iterations or inserting malicious code within loops.

Machine Learning in Remote Attestation

Machine learning (ML) has shown promise in enhancing RA, particularly in detecting anomalies and malware in complex systems. Most existing approaches focus on static code analysis [41, 166], while little exploration has been done to dynamic analysis techniques [25, 164].

ML-based RA, though relatively unexplored, offers the potential to improve detection accuracy and reduce overhead. For instance, by continuously analyzing system behavior, ML models can detect deviations indicative of an attack without relying on predefined thresholds, which are often insufficient for detecting sophisticated or novel attacks [15, 33]. As RA techniques continue to evolve, integrating ML can provide more robust and adaptive security mechanisms capable of handling a wide range of threats.

2.4 Countermeasures to Side and Covert Channels

Mitigating side- and covert-channel attacks requires a multifaceted approach that leverages detection mechanisms and various hardware and system-level

knobs to reduce the impact of these attacks while minimizing performance overhead. This section explores state-of-the-art countermeasures against cache side-channel attacks and power-based covert channels, highlighting their common principles and distinct challenges.

2.4.1 Against Cache Side Channels

This subsection reviews hardware and software-based countermeasures aimed at increasing resiliency against these attacks.

Hardware-based countermeasures introduce or modify the system architecture to mitigate side-channel vulnerabilities, particularly those targeting cache memory.

Cache Mapping Randomization An approach to improve resistance against side channel attacks is randomization of cache mappings, which dynamically and unpredictably alters the mapping of data in the cache to disrupt the attacker's ability to predict cache behavior [94].

Adaptive Architectures and Partitioning Reconfigurable and adaptive cache architectures have also been proposed to dynamically adjust cache configurations, reducing the accuracy of cache set detection by attackers [21]. Cache partitioning and coloring techniques further mitigate side channel risks by varying cache data mapping to minimize conflicts and contention [101].

While these hardware-based solutions offer robust protection with relatively low overhead, their implementation often necessitates new cache-specific designs and additional on-chip area, limiting their practicality in real-world systems.

Software-based countermeasures address side-channel attacks through implementations at the different levels of the software stack such as application, compiler, and operating system.

Application-level Approach At the application level, countermeasures have addressed the secure reimplementation of the cryptographic algorithms, which reduces the information leaked to the attacker. Although such solutions [91] might completely avoid attacks, they do not tackle attacks on existing secure applications. Moreover, the re-implementation of the algorithm is often less efficient than the original version.

Compiler-based Solutions Compiler-based techniques focus on randomizing execution paths or eliminating key-dependent control flow to obscure side-channel information [39]. Although these solutions can effectively thwart specific attacks, they often introduce significant overhead and require custom compiler development, limiting their applicability in existing systems.

Operating System-level Techniques At the operating system level, techniques such as limiting the precision of time measurements have been proposed to obfuscate timing information, making it harder for attackers to succeed [108]. However, these approaches can introduce unnecessary delays, affecting system flexibility and performance, especially for applications relying on accurate measurements for resource management purposes.

Cache flushing is another OS-level strategy used to prevent side-channel attacks by periodically clearing cache contents [61]. Despite its effectiveness, this technique can also incur in high overhead, as it disrupts normal cache operations across the entire system.

VM and Container Migration In cloud computing environments, VM and container migration strategies have been explored to mitigate side-channel attacks by physically separating the attacker and victim processes [20, 34, 135, 165]. These approaches, however, are typically designed for cloud systems with multiple computers and are less applicable to cache-based side-channel attacks in smaller, single-machine systems.

2.4.2 Against Power-based Covert Channels

This subsection examines the primary countermeasures, focusing on dynamic voltage and frequency scaling (DVFS) and noise-based approaches.

Noise-based Approaches Noise-based countermeasures introduce artificial noise into the power signal to interfere with covert channel communication [132, 158]. While these methods might produce moderate power overheads, they have not been thoroughly evaluated in terms of performance impact. Moreover, noise-based approaches require continuous processing on the core, which can hinder the performance of other applications. Additionally, they may fail against more sophisticated attacks [77] that are resistant to frequency-specific noise.

Dynamic Voltage and Frequency Scaling (DVFS) DVFS is a well-established technique used in various domains for power management and performance optimization [107, 131, 148], but it has also proven effective in mitigating power-based covert channels [77, 78]. By adjusting the frequency and voltage of processing elements, DVFS disrupts the power and temperature patterns that covert channels rely on, effectively jamming the communication medium.

However, DVFS can significantly degrade performance, particularly in environments where attacks are continuous. For instance, simulations have shown that DVFS can lead to a 25% performance loss in many-core systems [77]. In embedded systems, the performance degradation can be even more severe, reaching an average loss of 70% and exceeding 150% for certain applications (see Section 8.7.4). Despite these drawbacks, DVFS remains the reference countermeasure due to its direct impact on the power consumption medium.

3 Experimental Framework

In this section, the experimental framework used and developed for the different contributions of this dissertation is presented. In order to properly represent the variety of capabilities of modern and emerging computing devices, both simulation framework and real out-of-the-shelf hardware is used. The simulation framework employed for some of the contributions represents high-end many-core systems, while the real hardware brings up current realistic execution scenarios for embedded, FPGAs and server-like platforms.

3.1 Simulation framework - HotSniper

The main simulation framework used in this dissertation is *HotSniper* [128], an augmented version of the *Sniper* [30] simulator. HotSniper is an interval-based simulator for multi-/many-core systems that allows the modeling of modern CPU architectures by enabling the configuration of different CPU and system-level features, such as cache hierarchy and CPU cluster organization.

In addition to the simulation engine, which offers a new *open scheduler* implementation, *HotSniper* comes with *McPAT* [97] and *HotSpot* [79] integration for both power and temperature estimations, respectively. Additionally, *Hotsniper* implements a performance monitoring abstraction with a similar behavior to a Performance Monitoring Unit (PMU). This abstraction exhibits relevant CPU and memory execution metrics such as CPU IPS, cache accesses, cache miss rates, among others. These improvements allow for the implementation of dynamic resource management techniques such as task migration and DVFS on complex multithreaded applications.

For all of the simulations performed in this dissertation, the *gainestown* architecture is used, which is based on the Intel Xeon X5550 Gainestown CPU. The base architecture, depicted in Fig. 3.1, includes 16 cores with 64 kB

of L1 cache for each core, 256 kB of L2 cache per cluster and a shared 8 MB of L3.

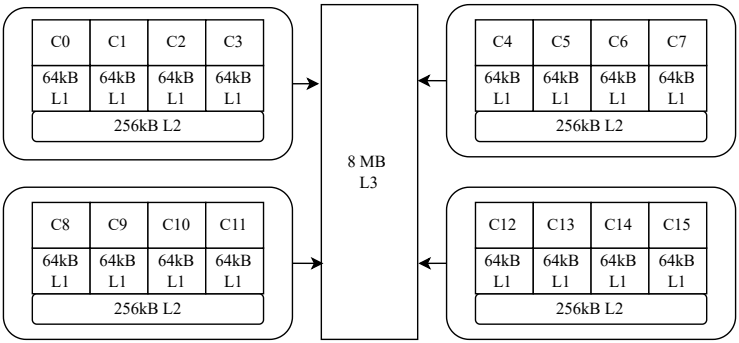


Figure 3.1: Representation of the base architecture used with the HotSniper simulator.

3.2 Real Hardware Platforms

Although the simulation setup described above might represent a flexible and configurable system, in real-world applications, the user might face different and unpredictable scenarios resulting from environmental variables such as temperature, noise, etc. In order to realistically design security solutions, both on the attacker and countermeasure front, these dynamic non-idealistic effects must be considered. To do so, in this dissertation, several real-world hardware platforms are used as the evaluation framework for the contributions. In the following subsections, these platforms are further described.

3.2.1 General Personal Computer (PC)

The first real hardware setup to be considered in this dissertation is a generic personal computer (PC), *desktop*-style platform. Although such a platform might not represent emerging systems, it does integrate different computing elements and modern software and hardware technologies existing in a common user day-to-day digital interactions, which makes it a perfect target for

potential attackers. The technical specifications of the platform are shown in Table 3.1.

Table 3.1: Characteristics of the PC evaluation platform

Feature	Details
CPU	Intel Core i9 10850K
Cores and frequency	10 cores @ 3.6GHz
Memory	2x16GB DDR4 @3.6GHz
GPU	NVIDIA GeForce RTX2070
GPU cores	2304 CUDA cores @1620 MHz
Operating systems	Windows 11 Ubuntu 22.04 (WSL)

3.2.2 Server-range CPU

To represent a multiuser execution scenario with high computing capabilities, a server-range CPU platform is employed. The features of this platform are described in Table 3.2. The setup integrates a multi-core AMD Ryzen 7 2700X processor with advanced virtualization support (AMD-V) and robust security mitigations for recent threats. It features a high-performance *adm64* architecture, capable of both 32-bit and 64-bit operation modes, making it versatile for various workloads. The multi-level cache system and NUMA architecture enhance processing efficiency. These features make the platform well-suited for handling concurrent operations in a virtualized, high-demand server environment.

As a multi-tenant computing framework, this platform effectively represents an *untrusted* system, where attackers could exploit vulnerabilities or system misconfigurations in order to compromise security-critical applications or steal user private information.

3.2.3 Embedded Devices

In the evaluation of the contributions of this dissertation, three embedded platforms are used. These platforms include modern embedded multicore CPUs alongside powerful GPUs, while still providing a constraint computing

Table 3.2: Key Features of the Server-range Computing Platform

Feature	Details
CPU Model	AMD Ryzen 7 2700X
CPU Modes	32-bit, 64-bit
Physical Address Size	43 bits
Virtual Address Size	48 bits
CPU(s)	16
Cores per Socket	8
Threads per Core	2
CPU Frequency (Max)	3700 MHz
CPU Frequency (Min)	2200 MHz
Cache (L1d)	256 KiB (8 instances)
Cache (L1i)	512 KiB (8 instances)
Cache (L2)	4 MiB (8 instances)
Cache (L3)	16 MiB (2 instances)
Operating System	CentOS7

environment for application-specific solutions. The following subsections further described each of the platforms used, as well as the reasoning for their consideration in this dissertation.

3.2.3.1 Raspberry Pi 4

The first embedded platform used in this dissertation is the Raspberry Pi 4 Model B [54], which is a compact and powerful single-board computer commonly used as a primary prototype platform for various applications, including multimedia-focused embedded systems, IoT, and educational projects. The specifications of this board are shown inTable 3.3.

The board was chosen because it represents a simple yet powerful embedded computing platform with a System on Chip (SoC) suitable for modern embedded devices e.g., those present in IoT environments. In such landscape, attackers might be able to exploit vulnerabilities in such devices to affect larger systems or gain access to users' private information.

Table 3.3: Key Features of the Raspberry Pi 4 Model B Platform

Feature	Details
CPU	Quad-core ARM Cortex-A72 (64-bit) @ 1.5 GHz
GPU	Broadcom VideoCore VI, 4K output @ 60Hz
Memory	4 GB LPDDR4 @ 3.2 GHz
GPIO	40-pin GPIO header
Storage	microSD card slot for storage
Power Supply	5V via USB-C (min 3A)
Operating System	Custom Linux (Buildroot)

3.2.3.2 Jetson TX2 and Orin Nano

In order to represent heterogeneous embedded devices with moderate to high computing capabilities, two additional boards are used as target platforms for several experiments. The selected boards exhibit a heterogeneous computing environment with a mixture of clustered CPUs, as well as a moderate integrated GPU. These platforms are NVIDIA Jetson TX2 and NVIDIA Jetson Orin Nano.

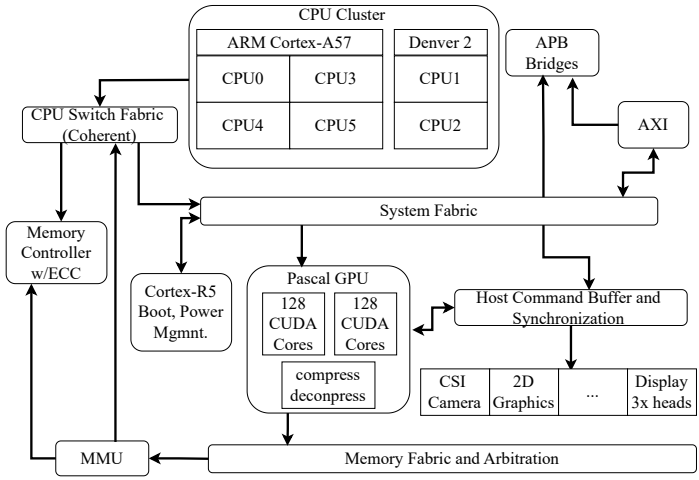


Figure 3.2: Simplified diagram of the NVIDIA Jetson TX2 architecture. Modified from [162] [55].

From a CPU point of view, the Jetson TX2 platform, shown in Fig. 3.2, is built on a heterogeneous architecture featuring two distinct clusters. One cluster houses a Quad-Core ARM Cortex-A57, while the second contains a Dual-Core NVIDIA Denver 2 64-bit CPU. The Jetson Orin Nano follows a similar dual-cluster approach, but with more advanced cores: one cluster consists of a Quad-Core ARM Cortex-A74, while the second cluster features a Dual-Core ARM Cortex-A74 processor. A key distinction between the two boards is that the Jetson Orin Nano includes an additional 4MB L3 cache, shared between both clusters, providing improved data handling and performance efficiency.

From the GPU perspective, as depicted in Fig. 3.2, the Jetson TX2 is equipped with an NVIDIA Pascal GPU, featuring 256 CUDA cores @ 1302 MHz. In contrast, the Jetson Orin Nano incorporates a more powerful NVIDIA Ampere architecture GPU, boasting 1,024 CUDA cores. Both platforms are designed to handle highly parallelized tasks, making their GPUs attractive for compute-heavy operations, which could also be a potential target for attackers seeking to exploit these capabilities.

In general, these boards present a heterogeneous computing scenario, composed of a powerful GPU and clusters of cores with different capabilities that follow the trend of modern high-end embedded devices such as those in the automotive or mobile industry.

3.2.4 FPGA-MPSoC

The final evaluation platform is FPGA-MPSoC computing device, which represents a modern, high-performance platform designed for flexible and reconfigurable embedded computing. For this, the AMD ZCU102 evaluation platform is selected [170]. The specifications and features for the framework are detailed in Table 3.4.

In this platform, the combination of a multi-core ARM processor subsystem with FPGA fabric provides a robust solution for parallel processing and hardware acceleration. This hybrid architecture allows developers to optimize for both general-purpose processing and custom logic, making it ideal for prototyping in automotive, industrial, and communication systems. In such scenarios, even under the security mechanisms such as OPTEE, adversaries

Table 3.4: Key Features of the FPGA-MPSoC Platform

Feature	Details
CPU Subsystem	
CPU Cores	Quad-core ARM Cortex-A53 @ 1.2 GHz
Real-Time Processors	Dual-core ARM Cortex-R5 @ 600 MHz
Memory	4 GB DDR4 (attached to CPU)
GPU	Mali-400 MP2
Operating System	PetaLinux, OP-TEE
FPGA Subsystem	
FPGA Fabric	XCZU9EG with 600K system logic cells
DSP Slices	2,520 DSP slices
Memory (FPGA)	512 MB DDR4 attached to FPGA fabric
I/O Pins	328 pins
High-Speed Connectivity	4x SFP+, PCIe Gen3, USB 3.0, SATA
Expansion Options	2x FMC connectors, 16 GTH transceivers

could leverage the heterogeneity and reconfigurability provided by the FPGA to mount advanced attacks and compromise such critical systems.

4 New Threats: Attacks Using System Resources

Covert channels pose a real threat to model computing systems. As motivated in chapter 1 and depicted in Fig. 1.1b, malicious actors can take advantage of system resources, both at the hardware and software level, to exfiltrate information from secure and isolated environments. In this chapter, we describe four new mechanisms through which modern attackers can implement covert channels with high transmission rates.

4.1 Shared Threat Model

To establish a common ground for the new attacks, in this section, we specified a unified threat model which describes the general capabilities of the attacker, as well as the assumptions needed for the implementation of the proposed covert channels.

The basic threat model for all covert-channel attacks presented in this chapter follows the same principles as other covert channels [38, 113], where a *malware* and a *spy* application communicate with each other in an illegitimate way.

The *malware*, or transmitter, is a malicious or *colluding* application that is triggered by an unsuspecting user to perform a set of operations on their behalf. These operations may involve handling sensitive data, such as medical records or biometric information, or performing security-critical tasks, such as encryption or decryption on sensitive data. To accomplish this, the malware

This chapter is mainly based on [1–4].

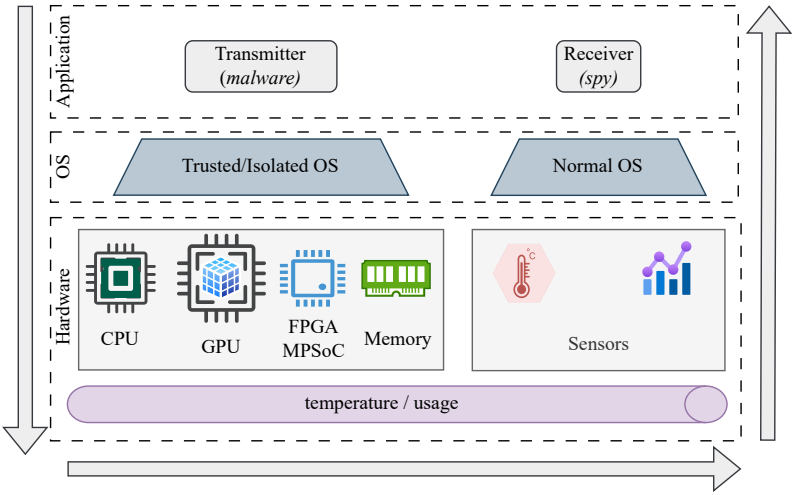


Figure 4.1: Overview of the shared threat model for the proposed covert-channel attacks.

typically runs in a trusted or isolated environment, where it gains access to private data. In practice, this isolated environment could be a virtual machine (VM) or a secure operating system, such as Trusted Execution Environment (TEE). Conversely, the *spy*, or receiver application, operates in a non-secure area of the system (e.g., the normal world) as a regular application.

Fig. 4.1 provides an overview of the shared threat model for the proposed attacks, illustrating how information flows from the transmitter to the receiver across different computing layers. Once active in the isolated environment, the transmitter application exploits system resources such as CPU, GPU, hardware accelerators, or memory to generate a modulated signal based on its usage, such as temperature variations or resource utilization. The receiver, running on the normal OS, can then decode this modulated signal by reading sensors and analyzing relevant resource utilization data accessible through the operating system.

A real-world example of such attacks is a supply chain attack, where malicious entities or dishonest vendors intentionally introduce compromised software into a system. A notable case is the recent *xz utils* vulnerability [100, 122], in which an adversary, through an elaborate supply chain attack, inserted

a backdoor into the popular open source data-compression library *xz utils*. This malicious update nearly made its way into major Linux distributions before being detected.

4.2 Novel Contributions

In this chapter, we unveil four new covert channels integrating a set of novel features previously unseen in the state of the art. As an introduction to the new threats, Table 4.1 presents a summary of covert channels and their novel characteristics.

Table 4.1: Summary of the novel attacks proposed and their features

Attack	Exploited resource	Medium	Novel features
Obfuscated TCC	CPU	Temperature	Avoids detection
GPU-based TCC	GPU	Temperature	First GPU-based attack Induces high overhead under countermeasures
Through Fabric	FPGA MPSoC	Temperature	First FPGA-MPSoC attack Breaks isolation (OPTEE)
MeMoir	Memory (RAM)	Usage	Resilient to noise Breaks isolation (Hyper-V)

Considering all the new covert-channel attacks described in this work, the integrated novel contributions presented in this chapter are the following.

- We present four new covert-channel attacks on different emerging computing platforms. We show how the data is communicated effectively through these channels with medium-to-high transmission rates and very low error rates.
- For the Obfuscated Thermal Covert Channel (TCC), we show how state-of-the-art detection techniques fail at detecting the new threat, highlighting the need for better approaches.

- For the GPU-based TCC, we highlight how the current DVFS-based countermeasure to TCCs induces a very high overhead in the system, especially on embedded GPUs.
- For *Through Fabric* - our FPGA-MPSoC-based TCC, we show for the first time how the isolation principle on a real TEE (OPTEE) is broken by the thermal covert channel using a completely benign hardware accelerator.
- Finally, for *MeMoir* - our memory-based covert channel, we present a real use case where the covert channel can be effectively employed to communicate information outside of an isolated scenario in a VM (Hyper-V) to host enviroment.

4.3 Obfuscated Short Duration Thermal Covert Channel

4.3.1 Motivation

Since traditional TCC detection techniques heavily rely on the Discrete Fourier Transform (DFT) spectrum of either the temperature or performance (IPS) signal to detect attacks, here we propose a new attack that behaves like a regular non-attacker application for most of the time, yet it is still able to transmit information at an acceptable data rate. The attack is *obfuscated* since it is intentionally designed to hide its attack nature (i.e., characteristic spectrum), and it is a *short duration* attack, because it is only transmitting information one-fourth of the time compared to a traditional attacker.

4.3.2 Attack Implementation

Figure 4.2 shows a high-level block diagram of our attack and the details are explained in the following. To encode information on temperature variations, we employ a compute-intensive kernel (i.e., busy waiting) when transmitting the bit value 1. When transmitting the bit value 0, we set the attacker core to an idle state. When transmitting information between the transmitter and receiver applications, we form *packets* consisting of a header and actual data

bits and we employ *Hamming* [66] as Error Correction Code (ECC) for our transmission. To fully transmit at least one packet on every transmission and remain undetected under DFT-based detection techniques, we propose using small packets (e.g., 10 bits).

As commonly done, we employ a return-to-zero (RZE) line encoding to avoid temperature accumulations that might interfere with the transmission. To avoid the effect of low-frequency noise, we use modulation to transmit the information at higher frequencies. For this, we implemented the on-off keying (OOK) digital modulation scheme for our attack. After a packet has been successfully encoded and serialized, the duration control module switches to the normal application kernel, hence disguising the attack until the waiting period of 3× of the packet duration has passed. After that, a new packet can be sent and the process is repeated.

On the receiver end, we first measure the temperature values on the receiver core and then apply a high-pass filter to remove low frequencies (i.e., less than 30 Hz) from the measured signal. After this, we demodulate the signal by comparing the relevant frequency components against a pre-established threshold, which also de-serializes the packet. We use the header of the packet as an identifier of the actual transmission. When the correct header is received, the packet is decoded and errors are checked via Hamming ECC. Packets with a wrong header are discarded (counted as erroneous). If a bidirectional communication channel is desired between both malicious applications, the transmitter and receiver employ different headers when sending. This is especially useful in noisy environments where an acknowledge-based protocol is needed to establish a robust communication channel.

Figure 4.3 shows a 2-second duration IPS trace of a core executing our *obfuscated short duration* attack. As depicted, the attack transmits information for the first 500 ms. After that, the attacker disguises itself as a normal application, by performing normal (i.e., non-attack) processing operations.

This similarity between attack and benign application affects the detection accuracy of state-of-the-art DFT-based detection techniques against our new attack, as we evaluate in Section 4.3.3.

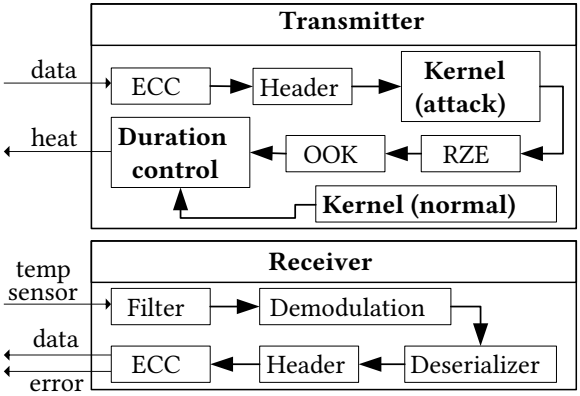


Figure 4.2: High-level block diagram of our new obfuscated short-duration TCC.

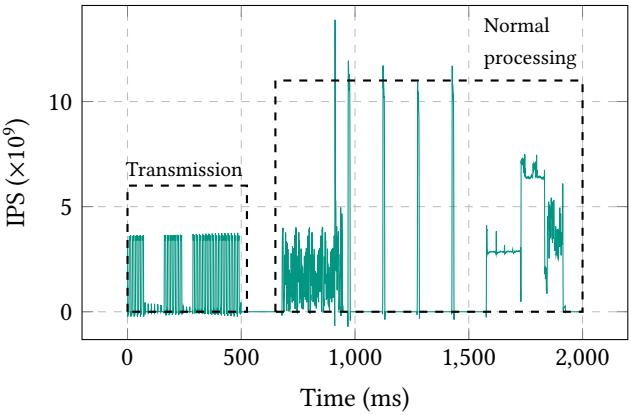


Figure 4.3: Obfuscated short-duration attack example over a 2-second window. The normal processing section belongs to the attack itself.

4.3.3 Experimental Evaluation

As evaluation framework for our obfuscated attack, we employ the *HotSniper* described in Section 3.1.

To evaluate the effectiveness of the obfuscated short-duration attack, we sent 1000 bits as 10-bit packets at a local transmission rate of 20 bps. Table 4.2 shows the results from this evaluation. After each packet, the attack does normal processing for 3× of the duration of one packet. Thus, the transmission rate gets reduced to 5 bps, with the benefit of increased stealth. When launching the receiver application at a 1-hop distance from the transmitter, the bit and packet error rate (PER) are very low (i.e., around 1 %) showing the feasibility of the attack.

Table 4.2: Evaluation of our proposed short-duration attack

Packet size	Local transmission rate (bps)	Overall transmission rate (bps)	BER (%)	PER (%)
10 bits	20	5	0.9	1

To validate the stealthiness of our new attack we implemented the main DFT-based detection solutions from the state of the art and evaluated them against the new attack, as depicted in Fig. 4.4. The DFT window length for this experiment was set to 2000 samples, which is consistent with the values reported in the state of the art [159]. First, following the three-step method proposed in [77], we utilize around 240, 000 IPS samples evenly split from our traditional attack and non-attack benchmark dataset to compute a threshold value for the DFT windows that leads to a high accuracy (around 96 %). Then, we implemented the detection solution from [159], using our training dataset consisting of 2, 140, 000 IPS samples distributed in an even split of attack and benchmark traces. The attack dataset is distributed between traditional and stealthy attacks. We implemented a 10-node model of two hidden layers (model C, from Table III in their work). We sought to perform the training until we got a high accuracy (i.e., greater than 95 %). After the target accuracy was achieved for both DFT-based approaches, we evaluated them against traces (circa 384, 000 samples) of our short-duration attack (OA). As shown, the accuracy for both DFT-based approaches gets degraded when facing our new attack, *which empirically shows the stealthiness of our novel attack as well as the limitations of DFT-based approaches when dealing with it.*

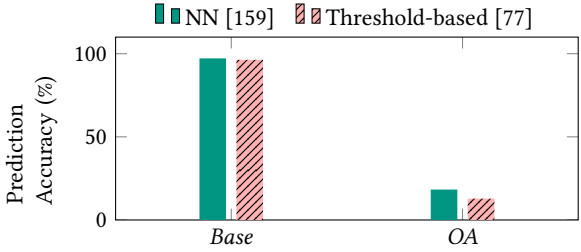


Figure 4.4: The accuracy of DFT-based solutions in the state-of-the-art (Base) significantly drops when facing our new short-duration attack (OA).

4.4 GPU-based Thermal Covert Channel

4.4.1 Motivation

In recent years, GPU computing has risen as an efficient way to accelerate computation in several domains such as high performance, cloud computing, and ML applications due to its highly parallel nature. With the surge of GPU-based solutions in these diverse domains, providing security to the GPU has become more and more critical, especially when it is used on sensitive data [89]. Given the recent utilization of GPUs in trusted environments [81, 102, 156, 174], we propose to explore the possibility of creating a GPU-based TCC attack, which has not been proposed so far and constitutes a new attack vector for which no countermeasures have been analyzed so far.

Although GPU and CPU-based attacks share a similar functioning principle, several challenges in the attack implementation and countermeasure effects make the distinction relevant.

Challenges in the attack implementation: On one hand, GPUs are comprised of hundreds or thousands of small cores, with typically one thermal sensor for the whole device. This means that raising the temperature to a noticeable degree involves a highly parallel computational effort which has to be high enough to heat the device, but also not too long so that the device can cool down in a short time. This fact affects the packet size, as temperature accumulations affect the error rates of the channel as the number of consecutive transmitted bits increases. This is not the case for CPU-based attacks, as each physical core has its thermal sensor that is leveraged individually.

As consequence, CPU-based TCCs normally reach transmission rates of up 45 bps with very high frequencies of up to 500 Hz for 64-bit packets on a single core [22, 77]. We discuss more GPU-specific TCC challenges and solutions to overcome them in Section 4.4.2.

Challenges in the countermeasures: On the other hand, Dynamic Voltage and Frequency Scaling (DVFS) techniques [78], which are considered the reference countermeasures to CPU TCCs in the state of the art, work by reducing the frequency of the CPU involved in the attack. However, due to the shared nature of the GPU, reducing the GPU frequency without any further consideration severely affects the execution of other GPU-dependent applications on the system. The same can be said of other GPU-based countermeasures (which we analyze and expand further in Section 4.4.2.3). Since the GPU is typically shared for the whole system, the performance loss on other applications when applying a countermeasure is much greater for GPU- than for CPU-based TCCs, as our evaluation shows.

4.4.2 Attack Implementation

In our new GPU-based TCC, the transmitter application encodes the binary data by increasing or decreasing the temperature signal of the GPU. By doing high parallel processing on the GPU for a certain amount of time t_{up}, the transmitter can raise the temperature, hence encoding a bit value of 1. When the attacker needs to transmit a bit value of 0, it sleeps the GPU for an amount of time t_{down}, allowing the temperature to decrease due to the cooling system (e.g., the fan). The receiver then reads the thermal sensor on the GPU and proceeds to decode the information.

4.4.2.1 Transmitter

The first step in the design of the transmitter is the processing mechanism to encode the binary data as temperature fluctuations.

Sending a bit value of 1 requires some sort of processing on the GPU to raise its temperature. Since GPUs are comprised of hundreds or thousands of small cores, raising the temperature of the whole device requires a high enough parallel load, which is not the case for CPU-based attacks as each physical core in a CPU comes with its thermal sensor. To perform this high

parallel processing, we chose to implement a parallel multi-threaded busy waiting kernel. In this kernel, each thread is constantly querying the state of a timer, which controls the period of a binary 1, hence heating the GPU. The number of parallel threads for each computation is extracted empirically offline for the target platform according to the number of computations it requires to produce the minimum valid temperature change at the desired channel transmission rate.

When encoding a bit value of 0 as temperature, we sleep the application for an amount of time equal to half of the period of the transmission data rate. Note that during the cooling phase, other (background) applications might execute on the GPU. This would add some noise to the channel, which is out of the control of the attacker and it is a challenge for GPU-based attacks. The interference that background GPU applications could produce on the channel is an additional design factor for the channel.

To overcome this challenge, we form small packets (e.g., from 8 to 16 bits) comprised of a header (used to identify the beginning of a packet and its sender) and data bits encoded through an error correction code (i.e., Hamming [66]) which helps to improve the reception when in noisy environments. This constitutes another difference over CPU-based TCCs, where the packet size can be extended while keeping low error rates. We experimentally demonstrate the need for a small packet size in Section 4.4.3.1. Moreover, to avoid the effect of temperature accumulation which may affect the actual reception of packets, we utilize the return-to-zero encoding (RZE) which is commonly used in TCCs exactly for this particular reason [103]. For our experiments, we additionally employ on-off-keying (OOK), as a modulation mechanism.

4.4.2.2 Receiver

The receiver application is modeled as a three-stage process, as seen in Fig. 4.5. As discussed in Section 4.4.2, the transmitted packet is encoded as the temperature variations on the GPU thermal sensor. Then, as input, the receiver module takes periodic samples from the thermal sensor of the GPU. As a general consideration, the chosen sampling frequency needs to be set to at least double the channel frequency, as per the Nyquist sampling theorem.

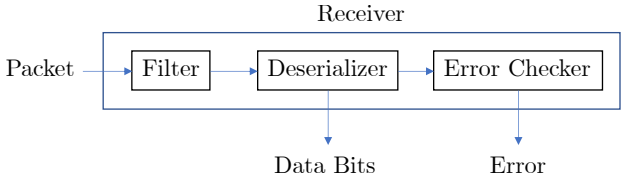


Figure 4.5: Receiver model for the time-based attack.

The first stage in the design of the receiver module is a filter, formally described in Eq. (4.1). This filter smooths the discrete temperature signal from the thermal sensor and removes the variations due to the sensor's precision δ, which may affect the accuracy of reception. If the magnitude of the difference between the current and the previous temperature sample is greater than δ, the output of the filter is assigned to the current sample. Otherwise, the output is assigned to the previous output sample. In the case of modulated attacks, the output of this filter is then used as an input to a second band-pass filter that eliminates the non-relevant frequency components.

$$y(k) = \begin{cases} x(k), & |x(k) - y(k-1)| > \delta \\ y(k-1), & |x(k) - y(k-1)| \leq \delta \end{cases} \tag{4.1}$$

The second stage of the receiver module is the de-serializer, depicted as pseudo-code in Alg. 1. This module continuously performs a comparison between the current and the previous measurement of the GPU temperature. When it detects an increase bigger than δ, it appends a bit value of 1 to the packet (line 8). If this difference is not detected within the transmission period (i.e, the inverse of the transmission rate), a *timeout* signal occurs on a parallel timer thread (not shown in Alg. 1). When this happens (line 12), the receiver interprets the sent bit as a 0, appends it to the packet (shown in line 13) and then restarts the timer thread. This process is repeated until the packet is successfully de-serialized. After de-serialization, the header of a packet is checked. Packets composed with a wrong header are discarded. Once a packet with a correct header has been de-serialized, we decode the data bits by using the Hamming ECC, which reports whether there were errors in the received packed.

By using a header of at least two bits (one to indicate the beginning of the packet and one to identify the sender), both malicious applications can act

as transmitter and receiver in an acknowledge-based protocol. By doing so, the communication becomes more robust, since the receiver can ask the transmitter to re-send an erroneous packet.

Algorithm 1: Receiver de-serializer for the GPU-based TCC

Input : $currentGPUTemp$, $timeout$, $packetBits$
Output: $packet$, $resetTimer$

```
1  packet ← 0;
2  N ← 0;
3  prevTemp ← currentGPUTemp;
4  while N ≤ packetBits do
5  |   nextTemp ← currentGPUTemp;
6  |   deltaTemp ← nextTemp − prevTemp;
7  |   if deltaTemp > δ then
8  |   |   packet ← (packet << 1) | 1;
9  |   |   N ← N + 1;
10 |   |   resetTimer ← 1;
11 |   end
12 |   else if timeout then
13 |   |   packet ← (packet << 1);
14 |   |   N ← N + 1;
15 |   |   resetTimer ← 1;
16 |   end
17 |   prevTemp ← nextTemp;
18 |   resetTimer ← 0;
19 end
```

4.4.2.3 Analysis of Current Countermeasures

GPU Thermal Noise One of the simplest solutions to thermal covert channels is the generation of baseband thermal noise. In the context of CPUs, the baseband thermal noise technique comes from the execution of background workloads on the CPUs that are suspected to be involved in the channel. By adding extra processing on those cores, the temperature response deviates from the expected attack response, making it harder for the receiver to correctly interpret data. When moving to a GPU domain, this naïve approach requires further considerations. The first one is the computing required to raise the temperature for the GPU. As discussed, GPUs consist of hundreds or thousands of smaller cores, which need to be heated up. This means that the otherwise innocuous noise translates into a high computing workload

on the GPU. This noise will be present as constant background affecting the power of the system and the performance of all other GPU-accelerated applications. The latter would not be the case for CPUs, as only the cores involved in the attack are affected. To improve efficiency, approaches in the state of the art for CPU-based TCCs have suggested frequency-specific (or *narrow-band*) noise [158]. By applying background processing at a specific rate, the computational power and performance impact is reduced over constant noise.

Similarly to the baseband noise, the GPU-based narrow-band noise requires a highly parallel computational workload, but it can be executed at a much higher rate. For high-frequency communication channels in GPU devices, its efficient implementation becomes a challenge itself, as it is already a challenge for the CPU-based countermeasure [132]. However, since the GPU is still a shared resource for all GPU-accelerated applications, the contention for it creates performance degradation on all other GPU application.

DVFS Dynamically changing the voltage and frequency levels of the CPU has been shown before as a way to jam CPU-based TCCs as an improvement over noise-based solutions. In the reference state-of-the-art approach for CPU-based TCCs [78], authors suggest a periodic DVFS policy that switches the frequency of the compromised CPU from the highest level to a lower level every channel period T. To establish the frequency values, they employ a down up rate β, computed as the ratio of the time the CPU stays at a low frequency (t_{down}) and the time that it stays at a high frequency (t_{up}). Although this state-of-the-art approach seems to successfully block the attack with a slight performance degradation on other applications that run on the same core, when translating this effect to GPU, the performance loss is much bigger (as it is discussed in Section 4.4.3.2).

4.4.3 Experimental Evaluation

To evaluate the new GPU-based TCC, we perform a series of experiments on two real-world platforms: the PC described in Section 3.2.1 and the embedded Jetson TX2 board (Section 3.2.3.2).

The PC represents an execution scenario with a dedicated GPU, while the Jetson TX2 deploys an embedded GPU. In both platforms, we employ a

Table 4.3: Description of the GPU-based attack settings on the effectiveness experiment

Platform	Sent bits	Encoding	Transmission rates (bps)
PC Jetson TX2	24,064	RZE, RZE + OOK	0.68, 1.38 4.375, 8.75

sampling rate of 500 Hz. Using these platforms, we proceed to evaluate our new attack, as well as the expanded countermeasures introduced in Section 4.4.2.3.

4.4.3.1 Effectiveness of the New GPU-based Attack

To verify the feasibility and effectiveness of our new GPU-based attack, we launched attacks with different packet sizes and encoding mechanisms, for both platforms. In this setup, we did not limit the GPU to any application. This means that the underlying OS and applications use it at any time if needed. In fact, hardware-accelerated GPU scheduling is enabled for graphical interface display in the OS. In this experiment, we run attacks with different encoding and packet sizes, as summarized in Table 4.3. We used an even split of RZE and OOK encoding. With our new attack we achieved maximum transmission rates of 1.38 and 8.75 bps for the general computing and embedded platform respectively. As seen in Table 4.4, this error rate comes with very low BER and PER for small packets (e.g., less than 2 % for our recommended packet size of 12 bits). Note that as previously discussed, increasing the packet size (more than 16 bits in our case), decreases the quality of the channel due to temperature accumulations and background noise.

As a comparison, the first implementation of a CPU-based thermal covert channel [109] achieved a lower transmission rate of 1.33 bps, with a much higher BER of 11 %. This is a fair comparison, as our GPU-based attack is also the first implementation of a TCC of its kind.

4.4.3.2 Evaluation of Countermeasures

As a second evaluation, we perform an experiment where we submit our new GPU-based attack to the extended countermeasures discussed in Section 4.4.2.3. First, we evaluate the proposed extended countermeasures them-

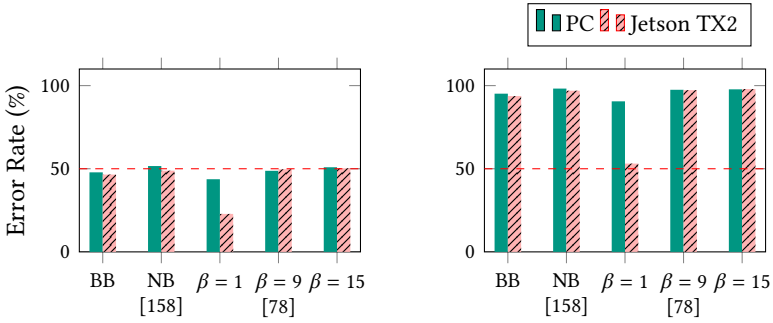
Table 4.4: BER and PER for different packet sizes for the GPU-based attack on the evaluation platforms

Packet size (bits)	BER (%)		PER (%)	
	PC	Jetson TX2	PC	Jetson TX2
8	0.09	0.25	0.34	0.34
12	0.14	0.29	1.00	1.50
16	0.30	1.15	1.33	4.00
24	1.28	1.73	6.00	9.00
32	4.25	4.82	9.21	14.48

selves in terms of their capacity to block the attack at the maximum transmission frequency. The evaluated countermeasures are baseband (BB) noise, narrow-band (NB) noise [158], the DVFS approach from [78] using $\beta = 1$ (low β, 50 % duty cycle), $\beta = 9$ (suggested for most of their experiments), $\beta = 15.67$ (high β), For the BB noise, we employed the Gaussian kernel from the Rodinia benchmark [32] as the generator application. For NB noise, we implemented a parallel Single-Precision A·X Plus Y (SAXPY) operation as the noise application due to its highly parallel nature and the fact that it can be toggled fast enough, as discussed in Section 4.4.2.3. As the minimum frequency, we employed a value of 300 MHz. The maximum frequency is the same value as shown for each platform (Sections 3.2.1 and 3.2.3.2). The error rate results for running the attack alongside the countermeasures on the both platforms are depicted in Fig. 4.6.

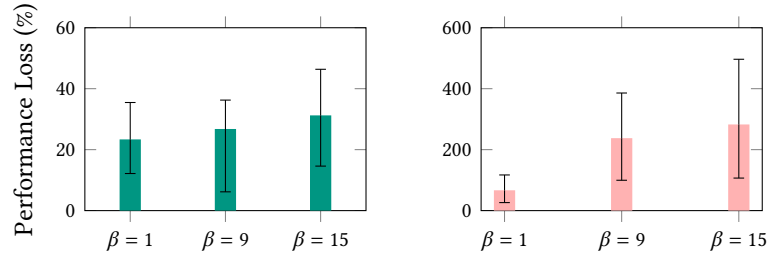
As it can be seen from the figure, the channel gets severely degraded with the majority of the extended DVFS and noised-based techniques, as the PER rises as high as 98 % for the high β scenario. Note that on the Jetson TX2 platform, for the low β scenario ($\beta = 1$), both BER and PER are lower that any other solution, resulting in about 50 % of the packets being unaffected by the countermeasure.

Finally, to evaluate the overhead of the reference DVFS countermeasure from the state of the art [78], as it has proven to be more efficient than the previous noise-based countermeasures. We evaluate the state-of-the-art DVFS approach for the β values mentioned above. To keep consistency with their work, we use the same metric (i.e., performance loss). To this end, we used four applications from the Rodinia benchmark as application set: *streamcluster*, *particlefilter*, *guassian*, and *myocyte*.



(a) BER. **(b)** PER.

Figure 4.6: Average Bit Error Rate (BER) (a) and Packet Error Rate (PER) (b) for the GPU-based attack under the extended countermeasures on the PC platform and the Jetson TX2 board.



(a) PC. **(b)** Jetson TX2.

Figure 4.7: Performance loss on benchmark applications due to β-based DVFS countermeasure from [78] on the PC platform (a) and the Jetson TX2 board (b).

In the experiment, we measure the average execution time of the application set without any DVFS, and then we measure again applying each one of the evaluated approaches individually to compute the performance loss due to the countermeasure. We assume the attacker is present at all times ($\tau = 0$ in [78]). The results for the performance loss evaluation are depicted in Fig. 4.7. As it can be seen from the figure, the state-of-the-art technique produces high performance loss i.e., more than 25% and 200% performance loss on the PC and Jetson platform accrodingly, using the recommended $\beta = 9$ value.

An interesting outcome of this experiment is the performance loss on the embedded Jetson TX2 platform. Firstly, on this platform, the performance

loss from all the countermeasures increases drastically when compared to the PC platform, reaching almost 300 % for the high β scenario and around 70 % for the lowest β. Secondly, although the lowest β scenario produces less performance, when analyzing the bit error rates depicted in Fig. 4.6 (b), the same countermeasure performs poorly affecting only half of the packets being sent.

Finally, since most of the attacks and countermeasures on TCCs found in state of the art have tackled mostly high-end multi-/many-core systems, we believe more attention should be put into disclosing new TCC embedded attacks, and (more importantly) proposing new countermeasures tailored for embedded devices. We see our new GPU-based attack and countermeasure implementation as a starting point in this direction.

4.5 Through Fabric: A Thermal Covert Channel on FPGA-MPSoC Systems

4.5.1 Motivation

FPGAs are a prominent component of the computing landscape. With the ability to change the implemented hardware at run-time, they offer an interesting option to accelerate a variety of applications. Several systems now include FPGAs as part of the compute infrastructure alongside the multiprocessor system-on-a-chip (MPSoC), e.g., Zynq-MPSoCs [170].

Previous works tackled the establishment of covert channels on FPGAs [27, 60, 63]. These works, assume (i) an FPGA-MPSoC running bare metal without any TEE mechanism existing and (ii) that they are able to implement malicious hardware for the transmitter and receiver, e.g., ring oscillators. These two assumptions are, however, not very realistic, as TEE can be easily established on FPGA-MPSoCs, e.g., OP-TEE [154]. Furthermore, malicious hardware is easily detected and banned on FPGA-MPSoCs [115]. In contrast to them, our work establishes a thermal covert channel on FPGA-MPSoCs without these two assumptions.

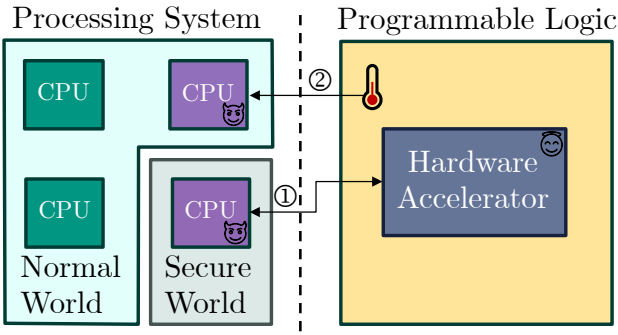


Figure 4.8: Overview of our new cross-device thermal covert channel on a TEE-enhanced FPGA-MPSoC.

4.5.2 Attack Implementation

In a simplified view, the system we target is depicted in Fig. 4.8. A colluding application (e.g., from a dishonest vendor) running in the secure world uses a benign AES accelerator from the FPGA, the so-called Programmable Logic (PL), to transmit messages by modulating the temperature of the PL (1). The malicious receiver running in the normal world reads the temperature sensor of the PL (2), which is accessible from the normal world, to decode the messages. In this way, we are able to communicate information between CPUs executing applications in a cross-world fashion, employing the temperature of the PL in the FPGA-MPSoC as the means for communication.

4.5.2.1 Software

A simplified overview of the software components of the system is depicted as a flow in Fig. 4.9. In the normal world, the innocent CA intends to perform a hardware-accelerated AES decryption on a ciphertext through the decryption TA, which resides in the secure world. To do so, the innocent CA uses the OP-TEE client API (1) to invoke the AES decryption TA, unaware of its malicious nature. In turn, the OP-TEE client API routes the request to the OP-TEE driver (2) in the Linuxkernel. On the Linux kernel side of the normal world, the OP-TEE driver then directs the request to the secure monitor (3) on the secure world, which itself handles the communication between worlds

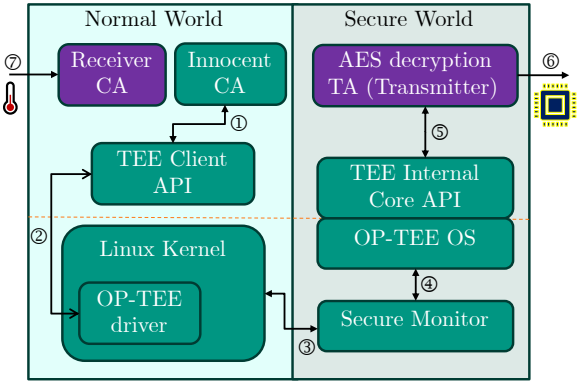


Figure 4.9: Overview of the software components of the system

by routing the request to the OP-TEE Trusted Operating System (4). Through the internal API, the OP-TEE OS framework determines the malicious AES decryption TA as the one being invoked and passes the control to it to handle the request (5). Finally, the TA uses the hardware accelerator API (6) to perform the decryption. In a normal (benign) operation, at this point, the execution control would return in a reversed path back to the CA with the ciphertext being decrypted. However, because of its malicious nature, before returning control, the TA leverages the hardware accelerator **again** (6) to encode and leak secret data (e.g., the key or plaintext) by modulating the temperature of the programmable logic on the FPGA.

Notably, the TA is configured to keep the execution context (instance) after the sessions ends, using the `TA_FLAG_INSTANCE_KEEP_ALIVE` flag [113]. This allows the TA to store, and further leak, private data after the transaction has finished.

In the normal world, another malicious application (i.e., the receiver CA, potentially owned by a different user) continually reads the temperature sensor of the FPGA (7) to detect the beginning of the transmission and decode the secret being sent by the TA, hence establishing an illegitimate communication channel between the secure and the normal world.

4.5.2.2 Hardware

To demonstrate our attack, we employ an existing hardware accelerator as the heating mechanism. As Fig. 4.8 shows, the processor residing in the trusted world is connected to one accelerator on the PL. The connection is done via an AXI crossbar. We use the benign AES 128 bit decryption engine from [106] as the accelerator, as it is available as open source and as it is highly parallelized. The AES engine receives the cipher text and decryption key as input from the processor and returns the plain text in one clock cycle.

Notably, while a custom and more power hungry hardware accelerator (e.g., ring oscillators) would benefit the transmission by heating faster, depending on the attacker to compromise the hardware or implement their own logic could be either be impractical or easy to detect. We decided to employ a completely benign module, with a realistic use case. The selected AES engine performs a security-related operation, which justifies its use from the secure world, while also being a tested device provided by an honest vendor.

4.5.2.3 Transmitter

The transmitter module of the attack is implemented within the malicious decryption TA. Algorithm 2 shows the implementation logic for the transmission. Upon being invoked (5) in Fig. 4.9), the TA performs the normal (benign) decryption of the ciphertext. However, as a malicious application, the TA proceeds to leak the newly decrypted plaintext by modulating the temperature of the FPGA. To do so, it first computes the number of decryptions needed to encode a bit of '1', and the time it requires to wait to encode a bit of '0', using the desired bit rate and the latency accelerator. Then for each bit in the secret, the TA performs the extra decryptions for each bit value that is a '1', or waits for the corresponding time for each bit of '0'. Finally, the application returns the plaintext normally to the calling CA.

In the case of a long secret, in order to avoid suspiciously long decryption delays, the transmitter module can leverage the `TA_FLAG_INSTANCE_KEEP_ALIVE` flag, as described in Section 4.5.2.1, to save it, while only leaking a few bytes at a time per call, especially with short ciphertext decryptions. On further calls, the attacker can obfuscate the transmission of the rest of the message by leveraging the decryption of longer ciphertexts.

Algorithm 2: TA transmitter for the TCC

```
1  Input: ciphertext: encrypted text from innocent CA
   Result: plaintext: decrypted text
2  plaintext ← HwAESDecrypt(ciphertext);          /* Performs normal
     decryption */
3  N ← 1/(bit_rate * latency_acc) ; /* Calculate the required number
     of decryptions to heat up enough to encode a '1' */
4  tdown ← 1/(2 * bit_rate);
5  for bit in secret do
6      if bit is 1 then
7          for i = 0 to N-1 do
8              HwAESDecrypt(randominput) ;          /* Perform extra
                 decryptions to increase temperature */
9      else
10         TEE_Wait(tdown) ; /* Sleeps to cool down the hardware */
11 return plaintext;
```

4.5.2.4 Receiver

The receiver is implemented as an application running in the normal world. It performs three simple steps which are shown as the three loops in Alg. 3. The first step (line 2 to line 5) is to continuously collect the data from the PL temperature sensor and store it in an array of size *total_samples*. This step as involves only reading data from a register. Once it collected the samples, it filters out the data to the desired frequency of communication (line 6 to line 9). The receiver performs this by calculating the Fast Fourier Transform (FFT) over a moving window and keeping only the bins that correspond to the frequency of communication.

The final step is to decode the filtered data into the corresponding sent bits (line 10 to line 21). To achieve this, the receiver applies two criteria: an absolute value and a gradient. During the moving window corresponding to each bit, if a value higher than a pre-computed high threshold (ρ_1) is achieved then a bit value '1' is interpreted. Similarly, if the maximum value is lower than the pre-computed low threshold (ρ_2) then a bit value '0' is interpreted. However, when multiple bits of the same value are sent sequentially, then

Algorithm 3: Normal world receiver for the TCC

Result: *msg*: demodulated message

```
1  s ← 0;
2  while s < total_samples do
3      temps[s] ← readTemp() ;    /* Get new temperature reading */
4      sleep(sampling_time);
5      i++;
6  N ← total_samples − win_size ;        /* Compute number of moving
       windows to decode */
7  for i in N do
8      X ← FFT(temps[i : i + win_size]);       /* Computes FFT of the
         moving window */
9      filtered[i] ← X[ω_k] ;     /* Filter the data at the frequency
         index ω_k */
10 for j in N do
11     bit_w ← filtered[(j ∗ win_size : j ∗ win_size + win_size] ;   /* get
         the demodulated bit window */
12     if max(bit_w) > ρ_1 then
13         msg[j] ← 1 ;  /* if high absolute change then bit is '1'
             */
14     else
15         if max(bit_w) < ρ_2 then
16             msg[j] ← 0 ;    /* if low absolute change then bit is
                 '0' */
17         else
18             if |slope(bit_w)| > δ_H then
19                 msg[j] ← not(msg[j − 1]) ;      /* if high absolute
                     change in slope then bit flipped */
20             else
21                 msg[j] ← msg[j − 1] ; /* if low absolute change in
                     slope then bit stayed the same */
22 return msg;
```

the temperature saturates to a value in between. To solve this, we compute the gradient between two consecutive samples as a moving slope. If the slope

of the readings from the moving window is greater than the pre-compute threshold for a high slope δ_H, it means that a rapid change of temperature occurred. Consequently, a bit with value opposite to the previously received bit is transmitted, so we toggle the value based on the last received bit. Otherwise, if the slope is low, it means that the same value is received and no toggling occurs.

To set the thresholds ρ_1, ρ_2, and δ_H we use a subset of the sent data and analyze it. Each bit is sent over a period T and an interval of temperature samples t is recorded. We collect the temperature recorded for the '1' bits in dataset t_1. Similarly, the temperatures for '0' bits are collected in dataset t_0. The high threshold (ρ_1) is set as $\rho_1 = \mu(max(t_1)) - \sigma(max(t_1))$ with μ being average and σ being standard deviation. For the low threshold (ρ_2), we set it as $\rho_2 = \mu(max(t_0)) + \sigma(max(t_0))$. Finally, for the slope threshold δ_H, we collect t_g which is the dataset of any bits that are of opposite value than the previous bit. We then calculate δ_H as $\delta_H = \mu(max(t_g) - min(t_g))/T$.

4.5.3 Experimental Evaluation

We run our experiments on the ZCU102 evaluation platform decribed in Section 3.2.4. The design is implemented using Vivado 2018. The AES accelerator is open source from [106]. It uses 11k LUTs and the PS to PL AXI interface uses 3.1k LUTs, at a clock frequency of 100MHz. The normal world operating system is a custom Linux distribution built using the PetaLinux SDK from Xilinx.

4.5.3.1 Channel metrics

In order to evaluate the effectiveness of the TCC, we run the compromised Trusted Application (TA), which performs several consecutive descriptions using the AES accelerator, in order to send 8, 000 bits encoded in 8-bit packets on the FPGA-MPSoC board. Table 4.5 shows the result metrics for our new cross-world thermal covert channel from this experiment including bit error rate (BER), packet error rate (PER), and transmission rate. As it can be seen, the channel is effective under the tested scenario, achieving a transmission rate of 2 bps, which is on par with similar TCCs on non-CPU devices [2, 83]. Moreover, our attack was able to produce very low error rates, in a

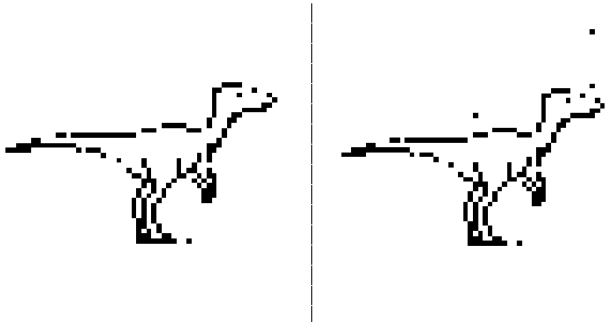


Figure 4.10: Representation of the channel performance when transmitting a binary image. The image on the left was sent, the image on the right was received.

similar range to other state-of-the-art approaches for thermal covert channels (1-11%) [111]. Notably, since the attack is performed within the OP-TEE environment, its effectiveness shows how the isolation and data confidentially principles of the TEE have been effectively broken by our attack.

Table 4.5: Thermal covert channel evaluation metrics

Bits	Packets	Transmission rate (bps)	BER (%)	PER (%)
8000	1000	2	1.9	4.3

To visually represent the performance of the covert communication, we sent a $64x64$ pixel binary image with our thermal cover channel. Figure 4.10 shows the sent and received images from this test. In this experiment, the obtained BER was less than 2%, which further shows the applicability of the channel.

4.5.3.2 Comparison to state of the art

As mentioned in Section 2.2, other works exploited covert channels on FPGAs before. However, as Table 4.6 shows, our work is distinct from them in several ways. First, our work is the first to exploit a temperature-based covert channel between CPUs using the FPGA. The second distinction is that our work neither requires special malicious hardware for the transmitter nor for

Table 4.6: Comparison to related works. Our work does not need any malicious hardware on the transmitter or receiver side.

Work	Requires Mal. (HW) Transmitter	Requires Mal. (HW) Receiver	Covert Channel	Break TEE
Our work	✗	✗	temperature	✓
Ref. [60]	✓	✓	voltage	✗
Ref. [27]	✓	✓	frequency	✗
Ref. [63]	✗	✓	voltage	✗
Ref. [57]	✓	✓	voltage	✗
Ref. [53]	✗	✓	frequency	✗
Ref. [59]	✗	✗	PCIe	✗
Ref. [56]	✓	✓	inter. wiring	✗

the receiver. Finally, none of the related works showed that they were able to break TEE on FPGAs.

4.6 MeMoir: A Covert Channel Based on Memory Usage

4.6.1 Motivation

Several types of covert channels have been implemented in the literature, with notable recent works on thermal covert channels [2, 78, 159] for hardware-supported attacks, and OS synchronization mechanisms [49, 146, 172] for those driven by software. With an increase in covert channel research in recent years, covert channels have become a threat to emerging computing systems [111].

In order to unveil a new type of covert channel, this section deals with the implementationof memory-usage-based threats in a multi-tenant and a VM to host scenarios. To introduce our new software-based covert channel, Fig. 4.11 shows an overview of the mechanism and actors involved in the attack form the perspective of a multi-tenant server. In such an attack, the malicious

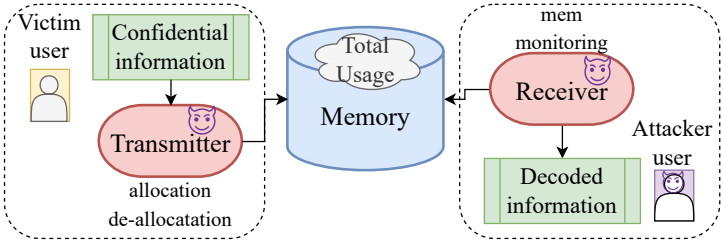


Figure 4.11: Overview of the new software-controlled memory-usage-based covert channel

transmitter application, which executes in a target victim user (private) context, has gained access to the unaware victim's confidential information, and it seeks to communicate the secret to other users in the system avoiding obvious direct mechanisms (e.g., shared memory, files, sockets, etc.) which are normally monitored [113] and hence easy to detect. In order to leak the secrets in a stealthy manner, as part of our new attack, the transmitter modulates the memory usage in the system, creating periodic patterns or memory allocations and de-allocations to encode the '1's and '0's of the message. Under a second user's context (or any other nonvictim zone), a second malicious application —the receiver— reads the system's memory usage and decodes the message being sent. Because the modulated signal (memory usage) is a virtual resource, it does not necessarily have a physical effect on the system which makes it hard to detect, especially if its existence is not yet unveiled.

4.6.2 Attack Implementation

4.6.2.1 Transmitter

This module's role is to encode and send the target data through the covert channel. In this context, it is seen as a malicious application with access to confidential information related to private data.

As seen in Fig. 4.12, the first step consists of converting these data to a binary, as well as separating them into blocks of 4 bits each. Then, the Hamming error correction code (ECC) is used in its 4-7 form —taking a 4-bit input and converting it to a 7-bit ECC encoded output— to minimize information loss

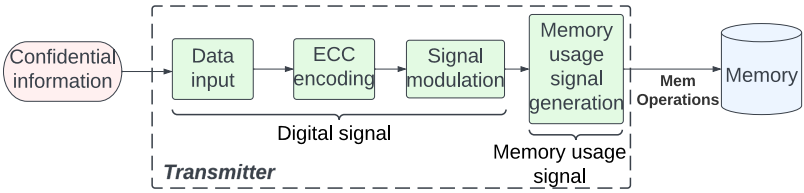


Figure 4.12: Overview of the transmitter module

due to the effects of temporal offsets caused by delays in the execution of memory operations, either by OS processes or other applications.

Furthermore, the data are also separated into packages of a fixed length to maintain integrity. This way, if a package has more errors than the ECC can correct (a total of one bit flip in the 4-7 form), it is considered to be incorrect but may not affect the next packages to be transmitted. We introduce a control sequence in front of each package in the form of a binary 1 appended to its most significant bit as a header, which indicates to the receiver that what follows is the package itself. Therefore, the total length per package is 8 bits.

Each bit is modulated by allocating, writing and freeing memory, using OOK (on-off keying), where the amplitude of the signal indicates the transmission of a '1' or a '0'. A high amplitude value (i.e., high memory usage), which represents a '1', can be seen as a rise, and a 'low' constant flat value (i.e., no memory used) represents a logical '0' in the signal. This behavior corresponds to the communication line code RZ (return to zero). Equation (4.2) shows how a logical '1' is represented, where t_p is the time it takes to send a pulse, t_h is the time when the signal rises (i.e., the total memory usage in the system goes up) and t_l is the time where the signal returns to its low value (memory usage is similar to the one before). On the other hand, a '0' would be a constant low signal of duration t_p ($t_p = t_l$).

$$t_p = t_h + t_l, \tag{4.2}$$

To build a pulse, we first need to increase the total system memory usage by copying a block of data to reserve it in memory and hold it for t_h, equivalent to $T/2$ where T is the total pulse time. Then, the space is freed to lower the

signal value once again. A sleep function is used to wait for the next pulse (low value in the remaining $T/2$), depending on whether a '1' or a '0' was sent. This is done via software through a program written in C++. The pseudocode algorithm for the transmitter module is shown in Alg. 4.

Algorithm 4: Send Data through the Memory Covert Channel

Input: binaryString
Output: Transmitted Data

```
1  Initialization:
2      BIT_PULSE_COUNT ← 2
3      PACKAGE_BIT_COUNT ← 4
4      SIZE_BYTES ← 20 × 1024 × 1024
5      REST_MS ← 10 ;                                  /* Assuming t_h = t_l = 10ms */
6  if binaryString % PACKAGE_BIT_COUNT ≠ 0 then
7      Append zeros to make binaryString divisible by PACKAGE_BIT_COUNT;
8  encodedData = "" ;
9  foreach package of PACKAGE_BIT_COUNT from binaryString do
10     encodedPackage = hamming_codec::encode(package, PACKAGE_BIT_COUNT);
11     encodedPackage = "1" + encodedPackage;
12     encodedData += encodedPackage;
13 Sending the encoded data:
14 foreach bit in encodedData do
15     if bit = 1 then
16         for i = 1 to BIT_PULSE_COUNT do
17             for j = 1 to SIZE_BYTES do
18                 Allocate pBigArray[j] ← 0xA ; /* Write dummy data to the array
                    */
19                 Copy pBigArray to another array pDestArray;
20             Free memory of pBigArray and pDestArray;
21             Sleep for REST_MS milliseconds;
22     else
23         for i = 1 to BIT_PULSE_COUNT do
24             Sleep for REST_MS × 2 milliseconds;
25 End of Transmission.
```

4.6.2.2 Receiver

The receiver keeps track of the memory usage values throughout a period of time and with these data extracts the information that is being transmitted

through the channel. The process implemented as the receiver module is depicted in Fig. 4.13. First, it samples the total memory usage by reading and

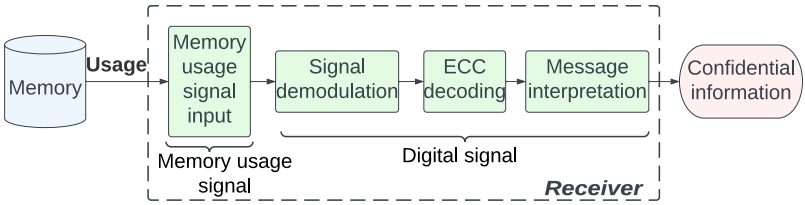


Figure 4.13: Overview of the receiver module

parsing the '/proc/meminfo' file from the Linux OS, which by default does not require administrator privileges. This value is calculated using Eq. (4.3), reading the corresponding fields of the file. The sample rate is determined by the Nyquist theorem, which is set to at least double the channel frequency.

$$mem_{used} = mem_{total} - mem_{free} - buffers - cache \tag{4.3}$$

Given that the modulated memory usage signal has a periodic behavior in which the pulses have a similar duration, we can group the receiver samples into sets of size N. Each group would contain the representation of either a '1' or a '0'.

We implement a hybrid demodulation technique in this covert channel, working with the net total memory usage values over a period of time, as well as a frequency analysis to process and convert the channel signal to binary data. Because we know that the first bit (header) in a package is always '1', the receiver starts by looking for a pattern in the differences between the signal values that may correspond to the rise and fall of '1' (in the time domain). Then it is very likely that from the first sample of the control bit, the next $8 \times N$ samples contain the transmitted package.

The receiver then takes each set of samples and calculates its DFT spectrum. To minimize the effects of low-frequency interference in the signal, we include a 5th order Butterworth high-pass filter, to be applied before the binary translation.

If the set of samples housed a bit of '1', there will be a peak in amplitude tied to the frequency where the channel is transmitting, which is related to the frequency of the pulse used in the modulation process. Whereas, in a DFT spectrum of a bit of '0', there will be no appreciable peaks in amplitude. This classification is formalized in Eq. (4.4), where a '1' is tagged as such if its amplitude A_i is greater than or equal to a defined threshold $A_0(f_t)$, or as a '0' otherwise.

$$S_i = \begin{cases} 1 & A_i \geq A_0(f_t) \\ 0 & A_i < A_0(f_t), \end{cases} \tag{4.4}$$

Once the package is demodulated in binary form, what is left is to apply the reverse Hamming code to decode the message.

In practice, this method achieves bit transmission errors less than 5% in our transmission time, as shown in Section 4.6.3.2, making it very reliable to send data across the two modules.

4.6.3 Experimental Evaluation

4.6.3.1 Evaluation platforms

Our experiments were carried out on two different platforms: the general-purpose desktop PC and an embedded Raspberry Pi 4, both described in Sections 3.2.1 and 3.2.3.1 accordingly.

Both platforms intend to show the feasibility of the attack under different architectures while also using different transmission speeds. Both devices implement the transmitter and receiver modules (covert channel) as well as the countermeasure. The transmitter and countermeasure are C++ applications, while the receiver module has a sampler monitor stage written in C++ and a data post-processing stage written in Python.

4.6.3.2 Channel metrics

To evaluate the effectiveness of the attack, we devised two experiments using the evaluation platforms described above. The summary of these experiments

Table 4.7: Covert channel evaluation results

Platform	Bits sent	Transmission speed	BER (%)	PER (%)
PC	337,952	6.5	0.32	0.71
RPi 4	4096	125	2.23	15.7

can be seen in Table 4.7. First, as the main evaluation for our channel, we employ the general computing PC platform. In this experiment, we send a total of 337,952 bits through the channel. This corresponds to 42,244 packages, each with a length of 8 bits, which include a mixture of ASCII-encoded messages, random bit strings, and images with different types of encoding schemes. Our new software-driven covert-channel attack operates at a net speed of 6.25 bits per second (bps) with a frequency of 25Hz on this platform. The memory block that the transmitter reserves and releases to build the pulse is 20MB in size. Samples of total memory used by the computer, calculating it like shown in Eq. (4.3), are logged every 1 ms.

As depicted, the average bit error rate (BER) and the packet error rate (PER) were below 0. 5% and 1%, respectively, with many individual messages achieving perfect transmission (0% error rates). This indicates a high reliability of this new covert attack for the leaking of confidential user data in a real setting at a reasonable transmission speed.

To visually demonstrate how secret information can be sent through the covert channel, we tested encoded images in various formats, as seen in Fig. 4.14. This shows that even if the error rate may not be 0% in all cases, what these images represent and their details can be easily interpreted by a potential malicious agent.

As a second experiment, to demonstrate how the attack can be implemented in a hardware-independent way, we use the embedded Raspberry Pi 4 as the target. Here, we transmit 4096 bits at a much faster channel transmission frequency to show the versatility of the attack. Although the packet error rate is higher due to increased speed, the bit error rate remains very low. More importantly, no changes were required from the original PC implementation, which shows how this new software-based covert channel does not depend on specific hardware components.

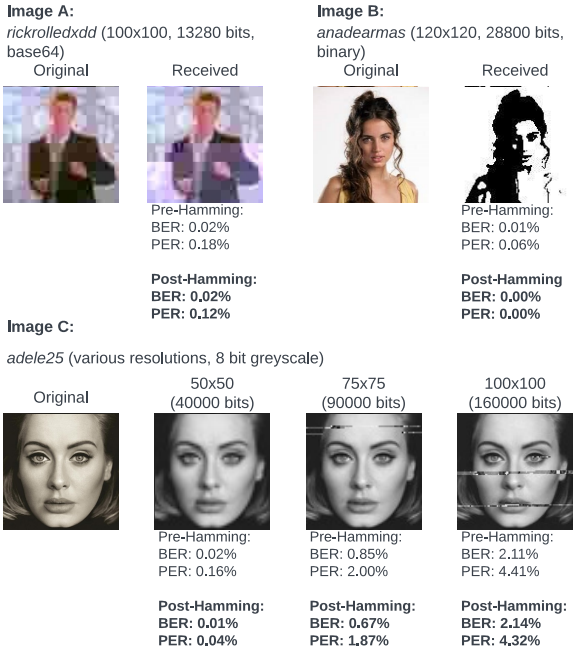


Figure 4.14: Visual demonstration of the effectiveness of the software-driven memory usage covert channel

4.6.3.3 Covert channel under background noise

In order to evaluate the effect of other background applications (and O.S) in the channel, we devise an experiment where we run applications benchmark from the Phoronix Suite alongside our covert channel in the PC platform. The results of this experiment can be seen in Table Table 4.8. As shown, even when the different applications execute producing multiple background memory accesses, the covert channel is able to effectively communicate all the packets with a very low error rate of less than 5%. These results confirm the robustness of the channel even when transmitted under a realistic noise background from the operating system and other applications.

Table 4.8: Effect of background application noise in the memory-based covert channel

Test application	Packets sent	BER (%)	PER (%)
lighthouse_chorus_cachebench	222	0.11	0.90
lighthouse_vp (1080p video)	56	0.00	0.00
lighthouse_vp4k (4k video)	56	2.68	3.57
lighthouse_vp4k_2	56	0.45	3.57
lighthouse_vp4k_3	56	0.00	0.00
sg_game (Left 4 Dead 2)	76	0.00	0.00
lh_game (Left 4 Dead 2)	280	0.80	2.86
lh_br_game (Left 4 Dead 2)	316	1.11	4.43
TOTAL / AVERAGE %	**1118**	**0.64**	**1.92**

4.6.3.4 Real use case: VM to host communication through MeMoir in a Hyper-V environment

To evaluate the effectiveness of our memory-based covert-channel attack in a real scenario, we conducted an experiment utilizing a virtualized environment with Windows Subsystem for Linux 2 (WSL 2) running on a Windows 11 host, on the PC platform described in Section 3.2.1

WSL 2, a compatibility layer for running Linux binaries on Windows, leverages a full Linux kernel within a lightweight virtual machine managed by Microsoft's Hyper-V [110]. Hyper-V's robust virtualization infrastructure enables WSL 2 to dynamically allocate and deallocate memory based on the workload within the virtual machine. This dynamic memory management feature is crucial to our covert channel, as it allows the memory usage of the WSL 2 VM to fluctuate in response to specific actions taken by processes inside the Linux environment. By monitoring these fluctuations in memory usage from the Windows 11 host, we can infer the transmitted information.

In our experimental setup for this real use case, the memory-based covert channel operated by carefully orchestrating memory usage patterns inside the WSL 2 instance, employing a nonreturn-to-zero encoding, where an increase (Δ) in the memory usage corresponds to the enconding of a bit of a '1', while a constant memory usage corresponds to an enconding of a '0'. These patterns were designed to create detectable changes in the memory metrics observed from the host system. The memory usage of the virtualized environment is

done via the *vmmemWSL* process in the host system, which is responsible for managing the WSL 2 virtual machine. This monitoring was performed at a high frequency (that is, 50 Hz) to capture subtle changes in memory allocation and deallocation. To better visualize the applied encoding, Fig. 4.15 shows the memory usage of the VM for three packets at a transmission rate of 5 bits per second (bps). As it can be further extracted from the figure, a threshold (δ) of about 25MB on the Δ memory usage signal, is sufficient to differentiate between '1' and '0' on the receiver's side.

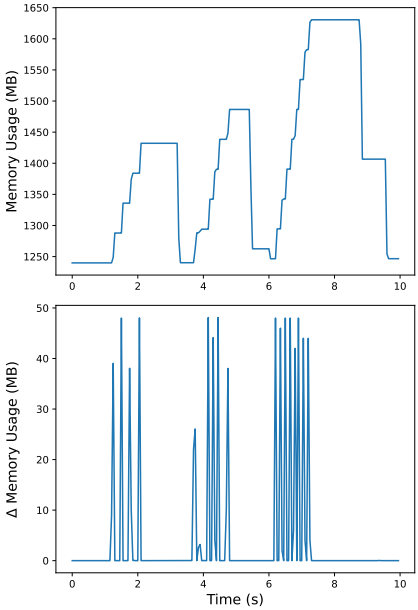


Figure 4.15: Memory usage signal (top) and processed memory Δ (bottom) measured from the host machine's *vmmemWSL* process when transmitting three 8-bit packets: b'10101010 (0xAA), b'10011101 (0x9D) and b'11111111 (0xFF), at a transmission rate of 5 bps.

Table 4.9: Metrics for VM-to-host covert channel communication

Bits sent	Packets	Bit rate (bps)	BER (%)	PER (%)
8,000	1,000	5	1.05	3.70

In order to further validate the VM-to-host covert channel, we performed an experiment where we sent 1,000 8-bit packets from the VM to the host

environment, at a local transmission rate of 5 bits per second (bps). The results of this experiment can be seen in Table 4.9. As depicted, the very low bit and error rates from this experiment (i.e., less than 5%) highlight the feasibility of our memory-based covert channel under a realistic use case, by leveraging dynamic memory allocation features inherent in WSL 2 and Microsotf's Hyper-V to facilitate covert communication channels.

4.7 Summary

In this chapter, we have presented four new implementations of covert-channel attacks on emerging systems that exploit new computer resources and obfuscation techniques to communicate information with medium to high transmission rates in unintended ways from a cross-layer perspective.

On modern CPUs, we have shown how obfuscated short-duration attacks can effectively transmit information through a TCC while bypassing the state-of-the-art DFT-based detection mechanisms, which implies the need for better detection techniques. This outcome is the motivation for the improved detection approach proposed in chapter 5.

In GPU-based computing systems, both in general and in embedded computing, we presented a TCC for the first time that leverages the parallel capacities of the device to mount a covert channel. Moreover, in such systems, we have shown how current countermeasures, especially those based on DVFS produce a significant performance overhead when applied to the GPU domain.

For FPGA-MPSoC systems, we unveil a TCC that for the first time exploits the reconfigurability capacity of the FPGA fabric to extract sensitive information from inside a TEE (i.e.,OPTEE) through the modulation of the temperature of a benign hardware accelerator. This attack shows how the isolation principle of OPTEE can by broken with such an attack.

Finally, for MeMoir - our memory-based covert channel, we have demonstrated for the first time how information from an isolated and virtualized environment (i.e., Hyper-V WSL2) can be communicated to the host system by leveraging memory allocation and de-allocation patterns.

5 Smart Detection Of Thermal Covert Channels

Since the first study on TCC for multi-core systems, back in [109], analyzing the thermal behavior of a suspecting core has been used in one way or another to identify attackers. Solutions for TCC detection first employed the thermal sensor information to identify an attack [78], using the DFT combined with a threshold-based frequency scanning on each core. Though effective, the thermal sensor measurements were not sufficient to pinpoint the exact attacking cores, since the temperature variations would also be noticeable in nearby processing cores, due to heat dissipation. To solve this problem, in [77], the authors proposed to employ the core's IPS performance information as a clear indication for the attacking physical core. This approach, however, still employed threshold-based detection, which exhibits several limitations when dealing with so-called *stealthy* attacks [159]. Unlike traditional attacks, these *stealthy attacks* are able to circumvent threshold-based detection by reducing the heat-up time when encoding a bit of 1, similar to pulse-width modulation.

To overcome the limitations of threshold-based approaches, a ML solution [159] was then proposed. In particular, a NN model is trained on collected DFT windows of IPS traces from different attacks and benchmark applications. Using the *shape* of the DFT data as input to the model, this solution solves the problem of threshold dependency in previous approaches. However, as we explain in the following motivational example and as we show in our evaluation in Section 5.5, relying upon the DFT technique as input to detection mechanisms is not always sufficient to detect the obfuscated *short duration*

This chapter is mainly based on [1].

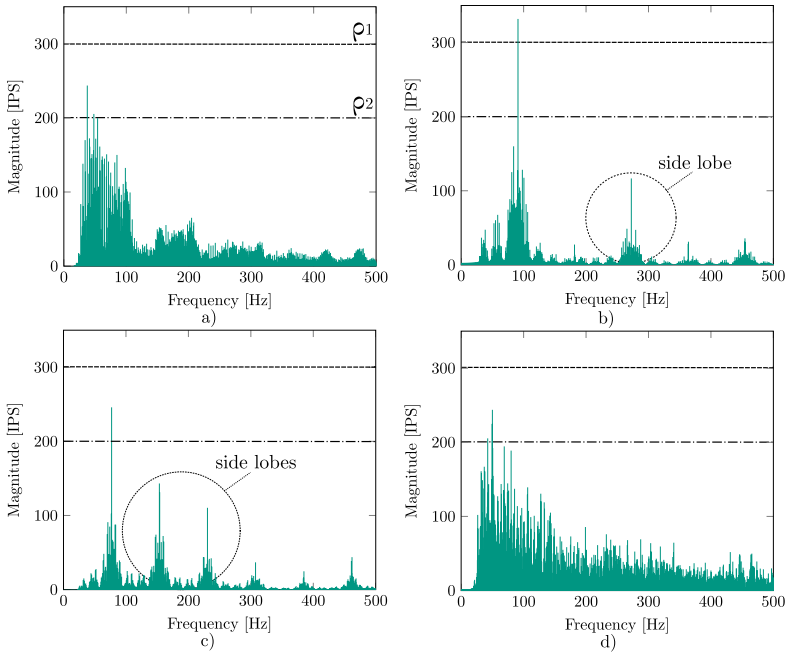


Figure 5.1: DFT spectrum of a two second long Instructions per Second (IPS) trace from a) parsec-bodytrack benchmark application, b) traditional attack, c) stealthy attack from [159] and d) our novel short duration attack.

attacks (described in Section 4.3), where the attack transmits information for a short period and then stops for another time interval.

5.1 Motivational example

To highlight the limitation of state-of-the-art DFT-based detection techniques, Fig. 5.1 shows the frequency spectrum of IPS traces from cores executing different attacks and a benchmark application from PARSEC [26]. In Fig. 5.1 a), we depict the spectrum of the benchmark application, while in Fig. 5.1 b) we show the spectrum from a *traditional attack*. Here, as the majority of previous works in the state-of-the-art have done, a threshold of $\rho_1 = 300$ can be used to

distinguish the benchmark application from the attack. However, as shown in Fig. 5.1 c) and d), this threshold does not work for *stealthy* attacks [159] and our obfuscated short-duration attacks described in Section 4.3, as the relevant frequency components are below ρ_1. A naive solution to this problem would be to lower the threshold to $\rho_2 = 200$ to capture those attacks. However, the benign benchmark application in Fig. 5.1 a) would be misclassified as an attack. From the example, we can see how a simple threshold over the DFT window, which is employed by the majority of state-of-the-art solutions, cannot distinguish the diverse TCC attacks types from benign applications.

5.2 Problem Definition

As just described, traditional threshold-based approaches for detecting TCCs fail when facing new threats. This problem has been noticed in a recent work [159], where Wang et al. proposed a machine learning solution that leverages the *shape* of the DFT (side-lobes depicted in Fig. 5.1 b) and c)) as features of an attack to improve the detection accuracy over the threshold-based solutions in the state of the art. However, as shown in Fig. 5.1 d), these side-lobes are not distinguishable in the DFT spectrum of our proposed obfuscated short-duration attacks. Moreover, the spectrum of the short-duration attack is more similar in its shape to the benchmark application (Fig. 5.1 a) than to any of the other attacks. This similarity leads to low detection accuracy levels for DFT-based solutions, as we demonstrate in Section 4.3.3.

This example brings the observation that the frequency-based detection techniques are not sufficient to detect the new threat. Hence, an improved approach should smartly leverage the behavior of the offending application over time to detect the new threats.

5.3 Novel Contributions

As we show throughout this chapter, by employing a smart time-domain-based detection scheme, we overcome the limitation of the DFT-based detection, outperforming the state-of-the-art approaches. Our novel contributions are the following:

- We show how DFT-based state-of-the-art TCC detection solutions are vulnerable to the obfuscated short-duration attacks described in Section 4.3.
- We present *Dotecca*: the first smart detection technique that can detect both, our new obfuscated short-duration attacks and state-of-the-art attacks. It is smart in the sense that it is an ML-based approach that uses short windows of time-domain measurements to quickly detect a series of diverse attacks (i.e., traditional, stealthy, and obfuscated short duration attacks) with higher accuracy and lower overhead compared to the state of the art.

5.4 *Dotecca*: Smart Detection of Thermal Covert-channel Attacks

As motivated in Section 5.1 and further demonstrated in Section 4.3.3, the DFT-based approaches can be inaccurate when dealing with short-duration TCCs. From this analysis, we have identified several challenges, which guide the design of our smart detection of TCC attacks: *Dotecca*. As discussed in Section 5.1, using frequency analysis to classify attackers is not sufficient to accurately detect all the diversity of attacks, especially those of short duration. To overcome this challenge, ***Dotecca* switches to the time domain**, by utilizing windows of time-domain measurements, where the periodic nature of the attack is perceptible and distinguishable from normal applications.

Moreover, in short-duration attacks, the adversary can communicate information by employing shorter-size packets. If a detection mechanism employs a large window size, the time that it takes to sample the long window could be large enough to allow the communication of several short packets. This means that, even if the attack is detected, by employing a large window on the detector side, an attacker might still be successful in leaking critical information. To overcome this challenge, ***Dotecca* employs short-duration windows**, which allows to detect the attacker faster while reducing the amount of possible information leakage before its detection.

Any solution that tackles many-core systems should be able to identify the core that executes the attack. Therefore, sensor data (IPS, temperature etc.) needs to be sampled from every core. Since a centralized mechanism, where

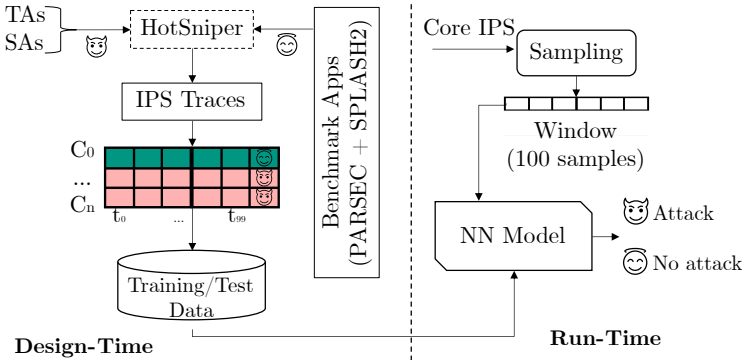


Figure 5.2: Overview of *Dotecca* including the design-time training process, as well as the run-time inference.

all required processing is performed on a single core, would not scale with an increasing number of cores, ***Dotecca* uses a distributed core-level detection mechanism**, where every core samples, stores, and analyses its own sensor data to quickly detect attackers without losing efficiency when scaling to many-cores.

Figure 5.2 shows an overview of *Dotecca*. At design time, we train a classification NN model to effectively predict if a set of time-domain performance readings (window) contain a TCC attack, using traces from benchmark applications as well as from traditional attacks (TAs) and stealthy attacks (SAs). At runtime, our technique invokes the NN model periodically (i.e., every 100 ms) to predict whether a core is running a malicious transmitter application.

Dotecca is designed as a distributed core-level detection policy that can be used to detect not only the presence of an attack in the system but also the location of the attacker, i.e., the core on which the transmitter is running. This enables the design of less invasive and more localized countermeasures that can act at the level of the core hosting the malicious application, hence minimizing the accumulated overhead on the system.

Table 5.1: Different datasets are used to train and test our model

Model	Description
BA	Non-attack benchmark applications
TA	Traditional TCC attacks
SA	Stealthy attacks [159]
OA	Our obfuscated short-duration attacks

5.4.1 Training Data Generation

To generate training data for our model, we first run multiple simulations[1] of TCC attacks and non-attack benchmark applications from *PARSEC* [26] and *SPLASH-2* [161], and we record their IPS over time. From these simulation traces, we extract multiple windows of 100 ms sampled at a 1 kHz frequency, i.e., 100 samples. Since a window of 100 samples at 1 kHz would hold at least two periods of a very low-frequency attack (i.e., 30 Hz), we chose 100 ms as our *detection epoch*. Such a short window allows us to quickly detect and react to potential attacks, including our new obfuscated short-duration attack from Section 4.3. Windows are then labeled either with 1 or 0, depending on whether they represent an attack or not. Thus, the features of our classification model are the 100 IPS values, representing the sample window, and the prediction label determines whether or not the window corresponds to an attack. This window extraction process is repeated for non-attack benchmark applications (*BA*) and for three types of attacks: traditional (*TA*), stealthy (*SA*) and our new obfuscated short-duration attacks (*OA*), as shown in Table 5.1. All of them employ a variety of transmission frequencies ranging from 30 Hz to 400 Hz.

5.4.2 Model Topology Selection

Following the supervised learning methodology, we use the training data generated in the previous step to train and test our model at design time. We start by building a preliminary NN model with 5 hidden layers (32 neurons each) with ReLU activation and one output layer (1 neuron) with sigmoid activation. We combine the four datasets generated previously into one

[1] The simulation setup is detailed in Section 5.5.

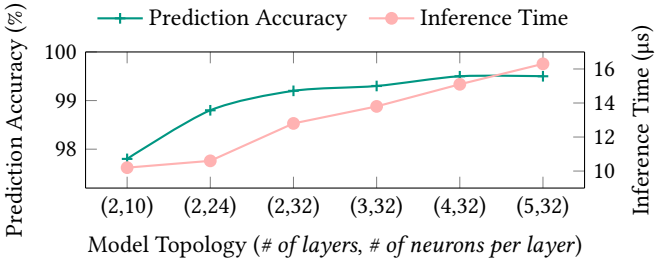


Figure 5.3: Different NN architectures achieve different prediction accuracy and inference times.

balanced dataset, where both attacks and non-attacks are represented equally i.e., 50/50 split. We randomly select 75 % of the dataset for training and 25 % for testing the model. The training of the NN is monitored with early stopping to halt the training once the model converges and once the accuracy stops improving on a validation set, representing 10 % of the training dataset. This preliminary test shows that the NN can achieve a high accuracy of more than 99 % and an inference time of 16.3 µs. Since this model will be used at runtime, it is crucial that it is as lightweight as possible. Therefore, we perform a Neural Architecture Search (NAS) to explore the impact of using different network topologies on prediction accuracy and inference overhead. The results of this experiment are shown in Fig. 5.3. We observe that the prediction accuracy only marginally improves with topologies of more than 2 hidden layers while incurring an extra inference overhead of more than 35 %. Therefore, we select a NN model consisting of 2 hidden dense layers (32 neurons each) that can achieve a prediction accuracy of 99 % while maintaining a low overhead of 12.8 µs.

5.5 Evaluation

We run simulations on the *HotSniper* described in Section 3.1. Cores follow the Xeon X5550 Gainestown model with 64 kB of L1 cache and 256 kB of L2 cache. The default *HotSpot* cooling parameters are used with an ambient temperature of 45 °C. For the non-attack simulations, we use 15 multi-threaded applications from two benchmark suites *PARSEC* and *SPLASH-2*, with *simmedium* and *large* input sizes. These applications are: *blackscholes*, *bodytrack*, *stream-*

cluster, *swaptions*, *x264*, *cholesky*, *fft*, *fmm*, *lu.cont*, *lu.ncont*, *radiosity*, *radix*, *raytrace*, *water.sqn* and *water.sp*. They are run with different numbers of threads and at different Voltage/frequency (V/f) levels. In HotSniper, we sample the IPS (from an internal performance counter structure) and the temperature signal at a rate of 1 kHz. The resolution of the thermal sensor in the simulator is 0.1 °C, which is similar to modern implementations [126]. As for the different attacker types (i.e., traditional, stealthy, and obfuscated short duration) we collect their traces from multiple simulations at different attack transmission frequencies from 30 Hz to 350 Hz, with core frequencies varying from 1 GHz to 4 GHz.

5.5.1 Evaluating the Effectiveness of *Dotecca*

To further evaluate the effectiveness of our time-based detection model by itself and in comparison with the DFT-based approach, we conduct the following experiments (summarized in Figs. 5.4 and 5.5) while keeping the same architectures for both our NN and the model from [159] and keeping the same DFT window size of 2000 samples.

Experiments 1–3: We train our model, *Dotecca*, and our implementation of the state-of-the-art solution using the traditional attacks (TA) and non-attack benchmark applications (BA) datasets. When using a split of 75 % of the data for training, and 25 % for validation, both models obtained a relatively high accuracy of more than 92 % (Exp1). Then, we evaluated the same models against traces from the stealthy attack (SA), which were unseen during training (Exp2). Here the our model maintains a high accuracy of 96.33 %, while the DFT-based solution had a significant accuracy drop. Finally, we employed the same models to evaluate them against unseen traces from our new short duration attack (SA) exclusively (Exp3). Once again our solution kept a high accuracy of 95.73 %, while the DFT-based approach shows a reduced accuracy again. Altogether, *our time-based* Dotecca *model is capable of generalizing to unseen attacks*, when trained only with the traditional attacks, while the DFT-based solution fails to do so.

Experiment 4: We combine the *TA*, *SA* and *BA* datasets, train both models with 75 % of the data, and use the remaining 25 % for testing. The results of this experiment show that *Dotecca* and our implementation of the model from [159] are both able to predict traditional and stealthy attacks with a very similar accuracy of more than 97 %. *This means that* Dotecca *is as effective as*

state-of-the-art DFT-based detection techniques when dealing with traditional attacks.

Experiment 5: We combine the *TA*, *SA* and *BA* datasets, use them to train the two models, and then test them on the new obfuscated short-duration *OA* dataset that were unseen during training. The results of this experiment show an interesting trend. Our model detects the new obfuscated short-duration attacks with an accuracy of 97.5 %, while the accuracy of the model from [159] drastically dropped to only 18 %, as it was shown in Fig. 4.4. *Our model was therefore able to effectively generalize to the unseen new attack variant, while the DFT-based approach failed to identify it.*

Experiment 6: We combine the *TA*, *SA*, *BA*, and 50 % of *OA* datasets, use them to train the two models, and then test them on the remaining 50 % of the *OA* dataset. The results of this experiment show that *Dotecca* detects the short-duration attacks with an accuracy as high as 99 %. Although the new attacks have been seen during training this time, the DFT-based approach struggles to reach the previously achieved high accuracy of 97.2 % and rather stagnates at 80.7 %.

Experiment 7: We evaluate the state-of-the-art model as in Experiment 6, but using different window sizes for the DFT. We tested window sizes of 3000, 1000 and 500 samples. Our results show that the detection accuracy of the model actually degrades (60.32 %, 73,82 %, and 55.82 %, respectively). This shows that *the size of the DFT window is not the factor why the frequency-domain approach has a reduced accuracy* and it confirms that the literature-recommended window size of 2000 samples[159] is a good choice.

As it can be seen from our comprehensive experiments, when dealing with obfuscated short duration attacks, the state-of-the-art approaches exhibit a reduced detection accuracy. Although the attack is still present in the windows used for the DFT, its periodic behavior is overshadowed by the normal processing part. Since state-of-the-art solutions employ the shape of the DFT (i.e., peaks and side-lobes) as indication of the attack, it is expected that this non-traditional spectrum is miss-classified as a normal application, because for the most part, the attacker behaves like one. However, since the transmission and the normal processing segments of the attack occur at a different time, the periodic behavior of the attack can not be hidden in the time domain. Moreover, since the transmission segment of the attack is essentially similar to traditional and stealthy attacks, our time-domain model is able to easily generalize to such attacks even when they are unseen during training.

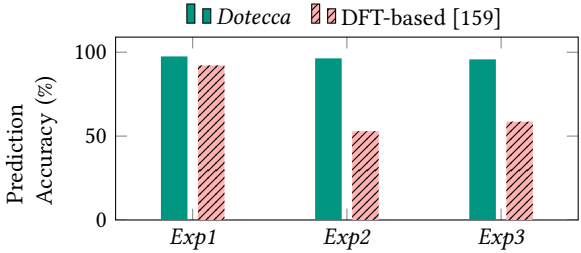


Figure 5.4: Exp 1 - Both models are trained and evaluated with trad. attacks (TA) and benchmark applications (BA). Exp 2 - Models trained with BA and TA, evaluated against stealthy attacks. Exp 3 - Models trained with BA and TA, evaluated against our obfs. attack (OA). *Dotecca* is able to generalize with high accuracy to unseen attacks, when trained only with TA and BA, while the DFT-based approach fails to do so.

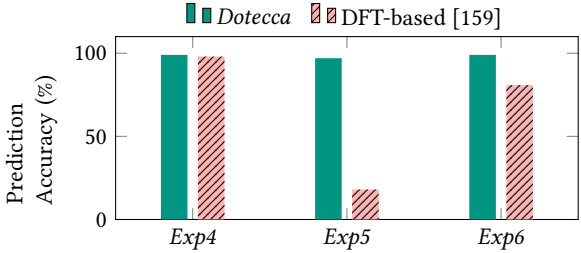


Figure 5.5: Exp 4 - Both models are trained and evaluated with trad. attacks (TA), stealthy attacks (SA) and benchmark applications (BA). Exp 5 - Models are trained with TA, SA and BA, evaluated against our obsf. attack (OA). Exp 6 - Models are trained and evaluated with TA, SA, BA and OA. The time domain analysis allows a highly accurate prediction of all types of TCC attacks, while the DFT-based approach struggles to maintain this standard when exposed to our novel obfuscated short-duration attack.

This suggests that new potential attacks with periodic behavior are very likely to be detected by *Dotecca*. The combination of these factors contribute to the high detection accuracy achieved by *Dotecca* and its improvement over the DFT-based state-of-the-art solutions.

5.5.2 Runtime Overhead

To estimate the overhead produced by *Dotecca*, both in the data collection and inference processes, we implemented our NN model as an application. As

detailed in Section 5.4, *Dotecca* first records the IPS of the core over the 100 ms window duration and then passes this data to the NN to predict whether the core is exhibiting an attack or not. This two-step process requires a total overhead of 12.8 µs on our target platform running at 2 GHz, which corresponds to an overhead of 0.013 % of the execution time of normal applications. As a comparison, the DFT-based approach from [159] requires about 450 µs to perform the DFT plus an inference time of 50 µs for a CPU running at 2 GHz. Note that although our model has more neurons in its hidden layers, we are able to outperform the inference time of the state-of-the-art solution by using a much smaller input to the NN. Moreover, their reported total overhead corresponds to 0.187 % of the execution time of normal applications. This means that by avoiding the DFT computation, *Dotecca* reduces the overhead by more than 14× compared to the state-of-the-art solution.

5.6 Summary

In this chapter, we have shown how conventional and more recent DFT-based techniques for TCC attack detection struggle when facing new varieties of attacks. To this end, we further show how the new attack described in Section 4.3 remain undetected even under advanced DFT-based state-of-the-art detection approaches, reducing their detection accuracy to about 18 %. To overcome the limitations of DFT-based detection schemes found in the state of the art, we proposed *Dotecca*: a new smart machine-learning-based detection technique. By switching to short windows of time-domain measurements as input for our NN model, we were able to detect traditional, stealthy, and obfuscated attacks with an accuracy of 99% , while inducing a minimal overhead in the system of 0.013 % of the execution time of normal applications. This overhead is 14× smaller than the one produced by the state-of-the-art counterpart. Overall, we have shown how our advanced detection technique remains very effective at identifying new threats at a very low cost in terms of system overhead.

6 Lightweight Control Flow Attestation

In today's computing environment, trust is a fundamental element. As devices become more interconnected to collaborate and communicate across various computing applications, including artificial intelligence, edge processing, and industrial automation [96, 155], they share sensitive data. Moreover, the operation of one device may be influenced by the computational results of others. Therefore, to prevent any harmful activities from remote computing entities, it is crucial to establish trust.

Remote Attestation (RA) facilitates the creation of trust between two computing parties. For example, in a cloud computing context, users delegate their data computation and processing to the cloud. In such a scenario, RA ensures that the computation is not compromised and operates as expected [40, 96, 144, 151]. In this situation, RA operates between a verifier (the user) and a prover (the remote computing device). Typically, RA is initiated by a request from the verifier to the prover, following a challenge-response protocol. The verifier issues a challenge to the prover, who then computes a cryptographic hash over a specific system aspect in response to the challenge. This is done to verify its normal operation using its known information about the prover (e.g., expected memory contents, valid control flow paths, etc.).

As a security technique, RA can be classified into two types: static and dynamic. In the context of static RA, the primary objective is to ascertain the integrity of the code memory and to verify that the binaries being executed have not been tampered with. This is typically achieved by employing state-of-the-art methods that compute a cryptographic hash over a specific range of code memory [96]. To counteract replay attacks, where an attacker might

This chapter is mainly based on [5].

record the value of a previous attestation and resend it, an initial value is concatenated with the memory range. This ensures that two separate attestation requests for the same memory range yield two distinct responses.

However, static attestation is not equipped to tackle all potential attacks. Therefore, dynamic attestation of the control flow, also known as Control-Flow Attestation (CFA) is implemented [13, 88, 96]. CFA is designed to thwart various control-flow hijack attacks where an attacker may attempt to manipulate the control flow to expose sensitive data or execute malicious code. The standard method to perform CFA involves calculating a sequence of cryptography hash values over the different nodes executed in the Control-Flow Graph (CFG) [13]. While this sequence of hashes would effectively capture any deviation from the control flow, the process of calculating a hash value for each executed node of the code can lead to a significant overhead. This overhead can result in an increase in the execution time by hundreds to thousands of times for applications with complex control flow [127]. Consequently, there is a critical need for a lightweight solution for enabling an efficient CFA. To address this need, we introduce our solution, referred to as *LightFAt*: a novel lightweight control flow attestation scheme powered by unsupervised machine learning.

6.1 Motivational Example

To introduce the key idea behind our novel attestation scheme, we devised an experiment to analyze how monitoring an executing application on the prover side, through the CPU's PMU, can be used as an attestation response to classify normal from abnormal behavior on the remote verifier side.

First, we created an IoT-based application on the prover that monitors user-sensitive data (e.g., blood pressure, heart beat, electrocardiograms, etc.), depicted as an overview in Fig. 6.1. This application works as follows: After an initialization stage (N_1), the application enters a sampling stage (N_2) reading samples from local sensors. In this stage, the application first reads a configuration file ("config.txt") containing the user-defined monitoring parameters. Then, the application writes the content of the file to a buffer and sets the monitoring parameters accordingly, before sampling data from the sensors. After the sampling, the application enters a low-pass filtering stage (N_3) to reduce the noise in the samples. When the filtering is done, the user's private

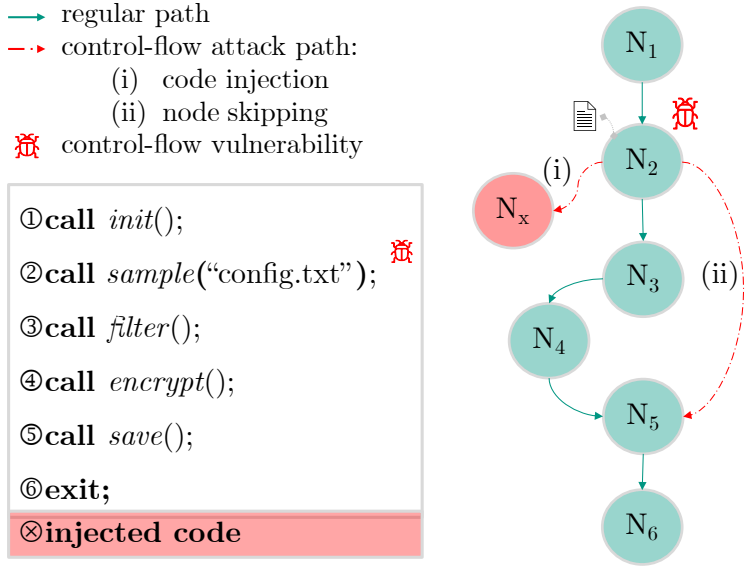


Figure 6.1: Overview of the control-flow attack paths on a vulnerable custom application.

data samples are encrypted (N_4) and then saved to a file (N_5), before exiting (N_6).

However, N_2 of the application exhibits a buffer overflow vulnerability, because the content of the input file is written directly into the buffer without any size checking. This can be exploited by an attacker through a crafted payload written in the file to (*i*) deviate the control flow of the application and execute arbitrary code or (*ii*) perform node-skipping (e.g., to leak sensitive information by skipping an encryption step).

In order to characterize the execution behavior of the application under normal and attack scenarios, we sought run-time features that represent both computation and memory intensity during an execution interval. For this, we selected the average Instructions per Cycle (IPC) as the measurement of computation, and the average number of memory accesses (collected at the L1 cache) as a measurement for memory intensity. We measure the average IPC and L1 cache accesses from multiple runs of the application under normal and attack executions, both for node skipping and control flow deviations. Figure 6.2 shows the average IPC and cache accesses, as a

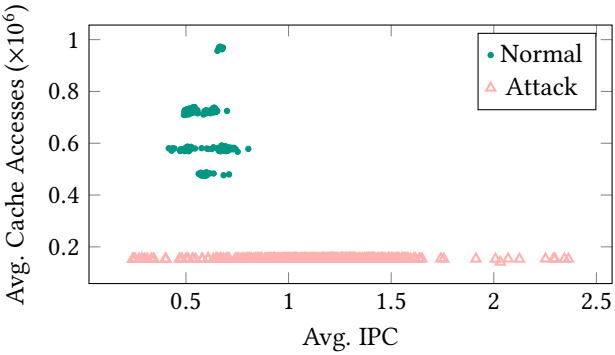


Figure 6.2: Normal runs and attack runs for the example application are differentiable in the feature space.

feature space, extracted from the processor's PMU for the different runs. As shown, the normal execution and the attack executions are differentiable. This means that **these features** of the IoT-based application as a prover **can potentially be used by a remote verifier to identify abnormal executions caused by control-flow deviations** in RA protocol. The validity of these characteristics as normality indicators are further confirmed with multiple real-world applications in Section 6.4.2.

6.2 Problem Definition

Several challenges arise when integrating the application's dynamic behavior into an attestation context. First, such a solution might require knowledge of the attacks at design time of the application in order to produce a method to differentiate between normal and abnormal traces. However, at design time, the designer of the application would not know about the possible attacks or vulnerabilities (otherwise, the attestation tasks would become trivial). This means that producing enough abnormal traces to represent the diversity of possible attacks might not be feasible at all. Moreover, using a simple threshold heuristic to separate normal from abnormal behavior might not be possible without properly describing all attacks.

The second challenge is how to monitor the application. If monitoring is done throughout the execution of the application, the attack part of the dynamic execution might be hidden within the normal part. On the other side, if some parts are skipped, attacks might occur on the unmeasured part, and the attack would go undetected. This means that choosing relevant regions of attestation becomes a bigger challenge when using the execution information as the attestation mechanism.

The third challenge is the trade-off between security and performance. While monitoring the application (instead of computing hash chains) might have a much lower performance impact on the application, the security is potentially reduced. The hash-based solutions are 100% accurate since the attestation response either matches with the expected computed value or does not match. However, the dynamic behavior of an application running on a real system can never be exactly the same, so there might be cases where it is hard to classify whether an observed behavior is or is not due to an attack. The challenge then resides in how to provide a good security mechanism (i.e., achieve a high accuracy) while achieving a very low and feasible overhead, which can allow the CFA of complex applications in a real-world scenario. Notably, the state-of-the-art hash based solutions without specialized hardware support are impractical in such complex applications due to these high overheads [127].

We focused on these challenges in our design process for *LightFAt*: a novel unsupervised ML-based attestation solution using the dynamic behavior of the application as an indicator of abnormal or malicious execution due to control-flow deviations.

6.3 Novel Contributions

In this chapter, we significantly extend the initial observations from our previous work [5]. As such, here we present the **following contributions**:

- We are the first to propose employing execution readings from the CPU's PMU, such as IPC and L1 cache accesses, as indicators of normality in a CFA scheme. We present an analysis and quantitative validation utilizing the execution features as indicators for normality in the execution of an application set in Section 6.4.2.

- We devise a general methodology to identify relevant Regions of Attestation (RoAs) to be used in our approach (Section 6.4.6), especially in cases of complex applications, where a single RoA might not be sufficient to properly assert the integrity of the application.
- We propose a novel unsupervised ML-based solution to control-flow attestation, which greatly reduces the overhead on the prover's side when compared to traditional solutions. We evaluated three unsupervised learning models: One-class Support Vector Machine (OSVM), Local Outlier Factor (LOF), and Isolation Forest (IFO). For the latter, as an outlier detection mechanism, we implemented a synthetic anomaly generation (Section 6.4.5), which helps to represent abnormal behavior in the models without the need for real abnormal traces.
- We present an exhaustive evaluation of our lightweight attestation scheme, by including both benchmark and real-world vulnerable applications. Moreover, we perform a quantitative comparison against three other state-of-the-art approaches, which shows the benefits of our lightweight attestation technique (Section 6.5.4.3).

6.4 *LightFAt*: Lightweight Control-flow Attestation

6.4.1 Target System, Threat Model, and Assumptions

Current devices, even in the cyberphysical domain, have become increasingly advanced and do not usually run bare metal applications anymore. Modern systems such as Raspberry Pi and MPSoC FPGAs are capable of running a fully functional operating system (OS), which adds further complexity to the system and the computation running on it. *LightFAt* targets such kinds of modern OS-based platforms. Our solution leverages the existing CPU's PMU, commonly available in modern architectures for such systems [149], in order to gather the execution metrics of the applications running in the system.

The targeted threat model for our approach has the same assumptions used by the state-of-the-art for CFA [13, 42, 43, 173]. We assume that the attacker wants to hijack the control flow through code injection or reuse, by exploiting a software vulnerability. Moreover, the attacker will not change the code memory content, as we assume that *LightFAt* will run orthogonal to static

attestation, which would catch any tampering with the memory code. We also assume the OS on the prover side is trusted and therefore our deployed attestation service is trusted as well. Additionally, we assume a remote verifier to be trusted. The verifier starts the attestation process and checks the validity of the attestation response.

Unlike other hardware-assisted attestation schemes, *LightFAt* does not require or assume any additional special hardware, i.e., it can be deployed on existing systems without further hardware modifications. Note that since we focus on the attestation scheme itself for a single device, IoT Swarm attestation [17, 31], physical attacks [19, 85], and restoring the state of the verifier [84] are all out of scope for this work.

6.4.2 Execution Behavior as Normality Indication

As we showed in our motivational example in Section 6.1, the IPC and L1 cache access features could serve as indications of an abnormal execution of the exemplary application. Two critical questions may arise from the motivation example.

Why did we choose these features?

Firstly, it is commonly used in application benchmarking and profiling to categorize applications into compute-bound and memory-bound [142]. Since we want our solution to be effective for *any* application, we should consider both types. To represent memory intensiveness, we selected the most memory-related general metric we could measure in the system, which is the total number of memory accesses over a period of time. Since all memory accesses are L1 accesses, we measure and use this value as a proxy to characterize the memory-bound behavior. For the compute-bound behavior, naturally, we chose the IPC.

Another reason we chose these two metrics is that IPC alone or memory accesses alone might not accurately describe the behavior of an application. This is because although IPC metric could also implicitly contain partial effects of the memory accesses, i.e., IPC might be lowered down by many memory accesses, this is not exclusively the case, as some compute instructions are more complex and take more time than others. Thus, the combination of both features is needed in order to retrieve enough information for such an application task.

Table 6.1: MI score for the average IPC and average cache accesses on different applications

App	MI: μIPC (μCache)	App	MI: μIPC (μCache)
custom	0.570 (0.694)	tinyexr	0.693 (0.693)
pcf2bdf	0.691 (0.693)	radiosity	0.640 (0.640)
raytrace	0.601 (0.605)	water-nsq	0.592 (0.657)
fmm	0.636 (0.614)		

How do we know they can be used as normality indicators?

While the motivation example shown in Section 6.1 serves as an initial confirmation that these features are good indicators of normality, we further explore this observation for a bigger application set, using the Mutual Information (MI) score as a quantitative indicator of the validity of these two features.

For this purpose, we tested seven applications including the exemplary application described in Section 6.1, two real-world vulnerable applications, and four benchmark applications. The real-world applications used are *pcf2bdf* and *tinyexr*, which contain a confirmed vulnerability listed by NIST [120, 121]. As an attack for these applications, we employ the provided proof of concept payloads, found in their corresponding issue report in their code repositories. For the four benchmark applications, we also employ *fmm*, *water-nsq*, *raytrace*, and *radiosity* applications from the Splash3 benchmark suite [140]. Although the latter does not present any known vulnerabilities, we create emulated attacks on them, similar to the attacks from the other real-world applications, which divert the control flow by either skipping nodes in the CFG or executing external code. For normal executions, we use several different input files and run the applications with diverse system loads.

To evaluate the validity of the features, we compute the MI score for each one of them, using normal and attack traces. In this case, the MI score gives an estimation of how much information about the normality of the set can be obtained from any individual feature. As we have two classes in our data (i.e., normal and abnormal) that can be represented by one bit, the highest MI score between one feature and the classified data is '1' according to the Shannon entropy of probability distributions [90]. As depicted in Table 6.1, the MI score remains higher than 0.5 for all the applications, which means

that the features are indeed relevant to the classification problem since each feature brings more than 50% of the information about the normality of the data. Moreover, the scores are consistently high for all the applications, which suggests robustness and generality. Overall, the obtained MI scores show strong evidence supporting the relevance and validity of the features as indicators of normality in our technique.

6.4.3 Attestation Flow

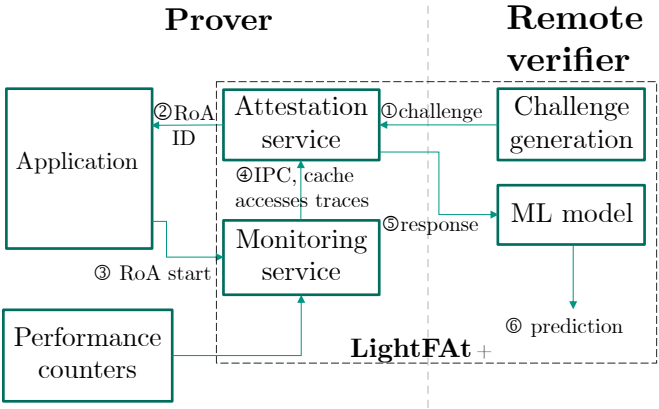


Figure 6.3: *LightFAt* working procedure overview

LightFAt follows the steps of a regular RA scheme. As depicted in Fig. 6.3, our attestation flow starts with the challenge (1) that is generated on the verifier side. The challenge is received on the prover side by a trusted attestation service, which translates the challenge into a Region of Attestation (RoA). The RoA specifies the starting point and length of the monitoring for the performance (IPC) and cache accesses metrics. Since complex applications might include several RoAs, the attestation service specifies an RoA identifier (ID) (2), so the application knows which specific region is under attestation. After receiving the RoA ID, the application executes normally until it reaches the RoA. When the RoA is reached, the application indicates the start of the RoA by generating a signal (3) to a trusted monitoring service. This service reads the performance counter information from the application process and all its corresponding threads. When the length of the RoA is reached,

the monitoring service returns the IPC and cache access traces (4) for the duration of the RoA to the attestation service. Then, the attestation service signs the collected traces and sends them back to the verifier as the attestation response (5). The response signature is checked by the verifier to ensure it is untainted, and then the IPC and cache access information is passed to an unsupervised ML-based model, which emits the final verification as a prediction result (6).

Note that *LightFAt* requires the code to be instrumented to add the triggers for the RoAs. This instrumentation can be done either by the application designer on the code level or the binary level by the verifier using a similar solution to [13, 123]. For the purposes of implementation and further evaluation, we build our solution to work on a Linux-based system.

In the following sections, we detail the implementation of the main components of *LightFAt* both on the prover and the verifier side.

6.4.4 Prover Implementation

On the prover side, *LightFAt* mainly consists of three components: the attestation service, the monitoring service, and the RoA instrumentation of the application.

Attestation Service

As previously described, the attestation service starts the attestation process on the prover side by generating the RoA identifier. We implemented this service as a C++ application. Depending on the challenge received from the verifier entity, the attestation service issues a POSIX signal to the application. The signal *number* directly corresponds to a specific RoA. Note that this gives us up to 32 different possible RoAs, which should be enough for most applications since the regions should cover large sections of execution. If more regions are required, the indication can be easily implemented through one single signal synchronization combined with a shared medium (e.g., a file), where the region ID is stored.

After the traces have been collected, the attestation service packs the data and signs the response with a key only known by the verifier. For all our evaluations, the collected data from a single attestation procedure consists of

no more than 1,000 single-precision floating-point samples (i.e., a maximum of 4 kB of actual data).

Application Instrumentation

In order to carry out our execution behavioral attestation, we instrument the application code. First, we add a small asynchronous signal handler module that gets triggered upon the reception of the RoA ID from the attestation service. Within this signal handler, we enable a flag corresponding to the region that must be attested and return control to the application. When the application enters the designated RoA, we trigger the start of RoA by issuing a SIGUSR1 POSIX signal to the monitoring process. As far as the application goes, no additional modifications are required. Notably, the added instrumentation is small and mostly focused on synchronization and signal issuing.

Monitoring Service

The monitoring service is a background running application that reads the performance counter information from the target application. Upon receiving the 'start of RoA' signal from the application code, the service measures IPC and L1 cache accesses periodically every 1ms. To this end, we employ the Linux *perf* tool [44] to gather the samples from the target's process id and all its child processes, using a sampling duration of a maximum of 500ms per RoA. When the sampling finishes, the monitoring service sends the data to the attestation service through a private secure file.

6.4.5 ML-based Remote Verifier

On the remote verifier side, the main component of our solution is the ML-based classifier that decides whether the attestation response data (i.e., the IPC and cache accesses information) corresponds to a normal or an abnormal execution.

When designing an ML-based solution to classify between two or more categories, one of the main challenges is the possible lack of adversarial data for the model training process. In our case, collecting the normal data traces is rather straightforward, as it only requires measuring the executions in different RoAs. However, abnormal traces are harder to get.

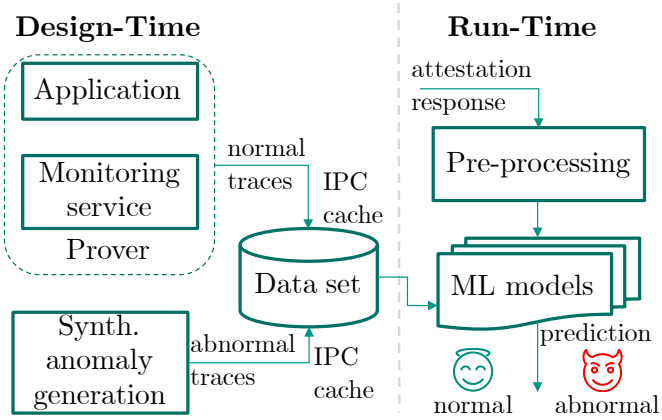


Figure 6.4: Overview of the ML-based verifier in *LightFAt*

First, since application designers often struggle to identify vulnerabilities in their own code (otherwise, they would have fixed them), modeling an attack on an potential and identified vulnerability to gather abnormal samples on their own code is not an easy task. Second, even if an attack is modeled, collecting abnormal samples on a particular application can hardly reflect all possible abnormal executions, since the dynamic behaviors of an attacker or abnormality are unpredictable and attack strategies are vast. To deal with this challenge, we chose to use the unsupervised learning technique, where models are fitted with mostly normal (unlabeled) execution traces, avoiding the need for an attack model or implementation.

For comparison purposes, we use two types of unsupervised learning classifiers: novelty detection and outlier detection. For the novelty detection technique, we propose to use the OSVM and LOF approaches, **which do not require training samples from abnormal traces**. This means that the models only learn the normal behaviors, rather than relying on the knowledge of any particular attack beforehand. As for the outlier detection, we use the IFO model. This outlier detection technique requires a small percentage of abnormal data for training purposes. In order to apply the method, similar to other anomaly detection approaches [143, 160], we augment the normal data with semi-random synthetic anomalies ($\hat{X}$), which correspond to 10% of the normal samples. These synthetic anomalies represent variations from the

original application. They are randomly generated within a region of up to five standard deviations (σ) outside of the normal traces (X), that is:

$$\hat{X} \in \left[\min(X) - 5\sigma,\ \min(X)\right] \cup \left[\max(X),\ \max(X) + 5\sigma\right]$$

The proposed ML-based remote attestation model from the verifier perspective can be seen as a two-phase process, as depicted in Fig. 6.4. At design time, we compute the average IPC and cache access window traces from runs of the application's RoA and create a training dataset of normal execution traces extracted from the monitoring service on the prover for a specific RoA. For the outlier detection models, this dataset is expanded with 10% of synthetic anomalies as previously described. Note that for the design of the verifier, no real attack traces are needed. However, to evaluate the model, as we show in more detail in Section 6.5.3, we created real attacks on vulnerable applications to validate the effectiveness of our solution.

At runtime, once the model has been trained, *LightFAt* gets the attestation response, computes the average for IPC and cache accesses within the RoA, and feeds both features to the model to obtain the final prediction.

6.4.6 Choosing the Regions of Attestation

LightFAt works upon triggers of RoAs. One application can have multiple RoAs for which the verifier might send the challenge. Choosing the beginning of a RoA is not as trivial as it might first seem. To better understand this, an example is shown in Fig. 6.5. It shows a part of a program's CFG where a trigger for RoA has to be included. N_2 is a node where a condition is evaluated. Based on the decision it would take either *Path A* or *Path B*. *Path A* will execute N_4 then N_3 then N_5, while *Path B* will execute N_3 before N_4, and lastly N_5. Suppose that N_3 is an infected node where an attack is executed. If the trigger for the RoA is added at N_4, but *Path B* is taken, it will go undetected. However, if the trigger is added at the latest to N_2, any attack on N_3 will be detected. Hence, when specifying the RoAs, special attention needs to be put to where the triggers are exactly added.

In a general case, we propose the following **methodology for selecting RoA points** in our solution:

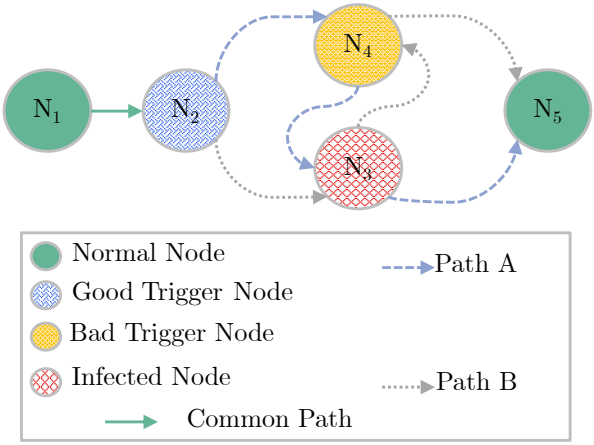


Figure 6.5: Example of bad trigger placement for a region of attestation.

- Notice the normal execution time of the application. This could mean the whole execution or a single iteration for periodic applications. If the execution time is relatively short (i.e., at most in the order of seconds) then one single RoA triggered at the beginning of the program works for our attestation scheme.
- If the execution time is not short, then multiple RoAs should be placed. From our observations and experiments, one RoA at the initialization, and then one additional RoA per complex function of a group of functions with a similar duration as the previous step should be placed. In that case, the verifier would need to issue a different challenge for each RoA, as we described in Section 6.4.3.
- To select the specific trigger point in a multi-RoA scenario, we suggest computing a call graph of the application for different inputs, using available tools such as Valgrind [118]. Based on the observation of the graph, the common functions or *nodes* where the application most frequently goes through in its executions before producing major diversions (e.g., N_2 in the above example) are good candidates for RoA trigger points.

Although multiple RoAs may require frequent attestations, the resulting overhead in our solution is negligible, as will be demonstrated in Section 6.5.4. We further show the effect of proper RoA placement in Section 6.5.2.

6.5 Experimental Evaluation

6.5.1 Experimental Setup and Data Collection

To evaluate *LightFAt*, we use monitor traces from running applications on the platform described in Section 3.2.2. Overall, we use the same application set described in Section 6.4.2.

First, we use the vulnerable custom application explained in Section 6.1, which reflects an IoT node behavior that reads sensor data, encrypts it, and saves it for further processing. We exploit the buffer overflow vulnerability to perform both control-flow hijack attacks shown in Fig. 6.1.

Second, we use two applications (*pcf2bdf* and *tinyexr*) with real vulnerabilities listed as described in Section 6.4.2.

Finally, we use four applications (*Raytrace*, *Radiosity*, *FMM*, *Water-nsq*) from the Splash3 benchmark. While these four applications do not have real vulnerabilities, in order to show that our solution works on a diversity of workloads, we emulate attacks on them similar to the attacks from the other vulnerable applications. We chose these benchmark applications with a variety of workloads to emulate the behavior of applications with dependency on inputs and have the potential of showing control-flow explosion as discussed in Section 2.3.2. To further increase the diversity in the executions we change around 30 different parameters and inputs for each of them. We instrument each application on the code level with triggers for the RoAs. The triggers for the RoAs are placed based on the explanation from Section 6.4.6. The monitoring for each RoA runs for 300 ms with a monitoring rate of 1 ms. We run each of the seven applications 1500 times with different inputs and parameters to get the different possible behaviors and all the possible variants of the control-flow explosion. We then run them another 1500 times but with attacks implemented. We collect the IPC and cache access traces for the chosen RoAs to feed them to the unsupervised classifiers.

For the training of the models, we use the scikit-learn Python library [129]. For execution time and overhead evaluation, we use a laptop machine with a 1.60 GHz Intel i5-8250U CPU with 4 GB RAM.

6.5.2 Effect of RoA placement

To further show the importance of the RoA placement, we test the effect of the definition of RoAs on the *raytrace* application from the Splash3 benchmark. We define two RoAs, one with a good placement of the trigger, following the methodology as stated in Section 6.4.6, and the other with a bad placement of the trigger, i.e., at an input-dependent node in the CFG as depicted in Fig. 6.5. Figure 6.6 shows the results of this test. For good placement, it can be seen from the top figure that the attack traces are indeed separated from the normal traces in the feature space. In contrast, the bottom part of Fig. 6.6 shows that, while some samples are separated in the feature space when choosing a bad placement there is a large overlap between the attack traces and the normal traces in the feature space. This overlapping would make it impossible for any classifier to perform correctly.

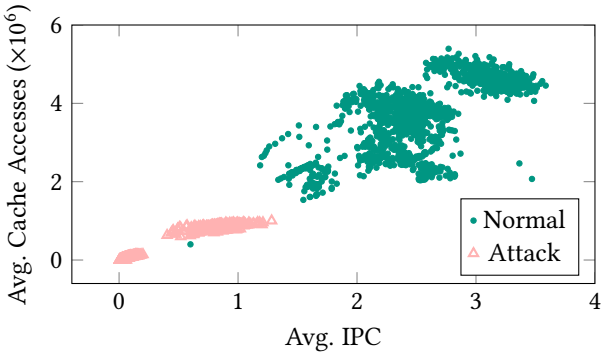
6.5.3 ML Models Training and Evaluation

6.5.3.1 Training

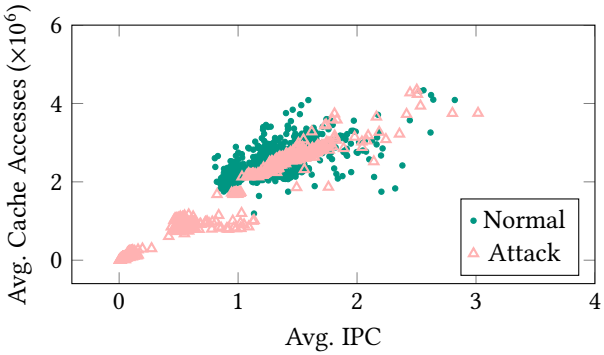
Table 6.2: Prediction accuracy for the different applications under the evaluated unsupervised learning models using normal, attack, and benchmark traces

Application	Models					
	One-Class SVM		Local Outlier Factor		Isolation Forest	
	Accuracy attack (%)	Accuracy benchmark (%)	Accuracy attack (%)	Accuracy benchmark (%)	Accuracy attack (%)	Accuracy benchmark (%)
custom	96.82	99.02	95.62	98.23	98.81	98.97
pcf2bdf	98.27	95.59	99.49	95.10	96.96	91.64
tinyexr	98.05	97.20	98.58	98.69	99.67	98.65
radiosity	96.04	96.02	98.59	96.36	96.68	96.62
raytrace	95.10	90.49	96.54	92.85	98.85	89.37
water-nsq	96.53	91.86	97.36	92.16	93.98	96.70
fmm	94.40	91.67	92.97	96.27	93.56	98.08

For the purposes of training the unsupervised learning classifiers, we employ a dataset consisting of normal traces from our application set. As described



(a) Good placement of RoA.



(b) Bad placement of RoA.

Figure 6.6: Normal and attack runs for *raytrace* are differentiable in the feature space for a good RoA (a), but overlap when a bad RoA is chosen (b)

Table 6.3: Average false negative rate (FNR), false positive rate (FPR), recall, precision and F1 scores for the different models when evaluating against normal, attack, and benchmark traces.

Score (%)	Model					
	OSVM		LOF		IFO	
	Attack	Benchmark	Attack	Benchmark	Attack	Benchmark
FNR	1.10	4.89	1.18	4.28	1.74	3.96
FPR	6.85	6.28	5.61	5.49	4.90	6.18
Recall	98.90	95.11	98.82	95.72	98.26	96.04
Precision	95.30	98.49	96.40	99.13	96.84	98.58
F1	97.03	96.71	97.57	97.36	97.52	97.26

above, each application is run 1500 times with a variety of inputs. For each run, we collect over 100 individual samples of both IPC and L1 cache accesses from the PMU. Collectively, the total dataset of normal execution traces contains over 2000000 samples.

The training of the ML classifiers is done differently according to the approach as described in Section 6.4.5. For the novelty detection technique (i.e., OSVM), we solely employ a randomly selected set of 50% of the applications' normal execution traces from our dataset, as the model does not require a representation of abnormal samples. On the other hand, for the outlier detection techniques (i.e., LOF, IFO), we employ the same set of normal execution traces as used on the OSVM model, plus a set of synthetic anomalies corresponding to a 10% of the normal traces, in order to represent abnormal samples as required by these techniques.

6.5.3.2 Model Validation

To validate our models, we use **two different test sets**; the first is a mix between the remaining 50% of the traces of the normal runs from the dataset and an equivalent number of attack traces chosen randomly, and the second is a mix of the remaining traces of the normal runs and an equivalent number of other benchmark traces chosen randomly. The first test set ensures that the attacks are differentiable from normal execution. The second test set reflects the fact that application designers will not necessarily have an idea about which attacks would be used against the application and they would only be able to evaluate the trained model against random runs of other applications.

Table 6.2 shows the accuracy of the different trained models when evaluating against normal, attack, and abnormal benchmark traces. In general, all of the models achieve high accuracy of more than 92% when dealing with attacks. When evaluating against benchmark traces, the models exhibit lower accuracy, due to the diversity of the traces and possible random similarities with the normal behavior. Nonetheless, the accuracy remains higher than 90% for OSVM and LOF for all the applications.

Although the accuracy gives a general idea of how well a model would perform, we do not rely solely on it. To better evaluate the effectiveness of the models, we measure the false positive rate (FPR), false negative rate (FNR),

Table 6.4: Monitoring overhead on the different applications due to the attestation scheme.

Application	Overhead (%)	Application	Overhead (%)
custom	1.265	pcf2bdf	9.781
tinyexr	2.374	raytrace	5.957
radiosity	2.100	water-nsq	8.221
fmm	1.940		

recall, precision, and F1 scores. Table 6.3 shows the average scores for the three models when using our test sets. As can be seen from the table, the FNR (i.e., the ratio of attack traces being misclassified as normal) is very low for all models. Although the FPR is higher than the FNR, the ratio is still less than 7% on average for all models. Moreover, as a security mechanism, *LightFAt* prioritizes minimizing the number of attacks that are misclassified as normal, as this could have potentially grave consequences for the systems, which is not the case for false positives. Notably, once again, due to the variability of the benchmark traces, the FNR is higher for the benchmark traces than for the attack traces. Despite this, the three models perform well, as confirmed by the high precision, recall, and F1 scores.

6.5.4 Overhead

We perform the overhead evaluation of *LightFAt* both on the prover and on the verifier side. The prover overhead is dominated by the monitoring service. On the verifier's side, the overhead is the result of the model execution. In the following, we evaluate the overhead of both sides of the attestation process.

6.5.4.1 Prover-side Overhead

Most of the lightweight RA solutions focus on reducing the overhead on the prover side as it may affect the normal operation of the target system. Table 6.4 shows the overhead on the prover side for *LightFAt*. Our overhead is always below 10% with an average of approximately 5%, which is minimal, especially compared to state-of-the-art solutions (see Section 6.5.4.3).

Table 6.5: Execution time for the different unsupervised learning models.

Model	Pre-processing time (μs)	Prediction time (μs)	Total (μs)
One-Class SVM	0.991	1.828	2.819
Local Outlier Factor	0.991	2.335	3.326
Isolation Forest	0.991	3.107	4.098

6.5.4.2 Execution Time of the ML models

Table 6.5 shows the execution time for each model on the verifier side. This execution time has two components. The first is pre-processing, where the features are extracted from the raw IPC and L1 cache accesses data. This has the same time of around 1 μs for all three classifiers as we are using the same feature computation for all of them. The other component is inference time, which is the time needed by the model to classify the traces as normal or abnormal. The three classifiers require minimal overhead in the range of 2 μs to 3 μs. This is especially noticeable when compared with other control-flow attestation schemes in the state-of-the-art, where the verifier is expected to emulate all possible hash calculation sequences from all possible control-flow paths [13, 95]. Instead, *LightFAt* is lightweight on the verifier side and it does not require any pre-computed CFGs.

6.5.4.3 State-of-the-art Comparison

Table 6.6 shows the average overhead on the prover side for *LightFAt* compared to the solutions from the state-of-the-art. Notably, both C-FLAT [13] and Tiny-CFA [123] evaluate their solution using simple applications without complex control flow. The solution from [127] evaluates against applications with complex control flow that might suffer from control flow explosion and hence produces much higher overheads. As shown, *LightFAt* has the lowest overhead, even when compared to the approaches dealing with simple applications ($\approx 50\times$ less). Moreover, other low-overhead solutions such as Lo-FAt [43] or LiteHAX [42] require special hardware for hash computations and tracking loop information. In contrast, our solution does not require additional hardware, as it relies on already existing performance counter registers commonly found in modern architectures [149].

Table 6.6: Prover-side overhead compared to state of the art. The overheads for Tiny-CFA and C-FLAT are based on the experiments reported in [123], which test the overhead for a simple application with execution time in the range of milliseconds. The overhead for *LightFAt* and ref. [127] is based on a variety of applications some with more complex control flow (CF).

Overhead	**Tiny-CFA** [123]	**C-FLAT** [13]	**Ref.** [127]	***LightFAt***
Simple CF	1.50×	1.76×	3.2×	1.01×
Complex CF	N.A.	N.A.	1000×	1.1×

To the best of our knowledge, no state-of-the-art solution reports the overhead on the verifier side. Traditional solutions assume a verifier with unconstrained computing power that is capable of the pre-computation or run-time emulation of a possibly large CFG and its corresponding hash chain computation for a specific challenge. We assume this computation/emulation to be rather long, especially for very large CFGs. Our solution, in contrast, takes less than 10μs to produce a verdict even on a not-high-end system, regardless of the complexity of the CFG. We believe this fact could be leveraged to produce real-time actions to a failed attestation, which could potentially avoid system damage or information leakage.

6.6 Summary

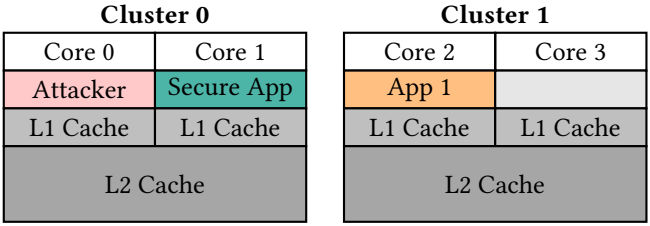
In this chapter, we presented *LightFAt*: an unsupervised ML-based attestation scheme that leverages IPC and cache access execution traces from an attestation region to predict whether the attestation response corresponds to normal or abnormal execution. Our unsupervised learning approach manages to separate between normal and abnormal traces at run-time, without the need for actual malicious samples in the training process. On the prover's side *LightFAt* requires small code instrumentation mainly to trigger the monitoring of the performance counter information while producing a minimal overhead of less than 10% for the evaluated application set. This overhead is approximately 50× less than existing approaches using simple control-flow applications, and at least 100× less than other control-flow attestation schemes for simple workloads in the state of the art. On the verifier side, we have shown how three different unsupervised ML models present a high prediction accuracy of over 90%, with low false positive and false negative

rates. Moreover, the inference time is low, with a maximum of less than 5μs, allowing the detection of abnormalities and their possible countermeasure in real-time without the need for big pre-computed control-flow graphs or extensive hash computation chains, as is the case for other state-of-the-art solutions.

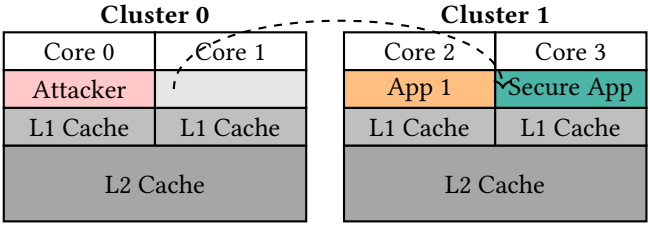
7 Mitigation of Cache Side Channels via Task Migration

Cache-based Side-channel Attacks (SCAs) are a particular type of attack that exploit the intrinsic timing nature of cache memories to extract security-critical information from variations of access time patterns on the target application through different techniques. In the context of cryptographic security applications (from now on *secure applications*), cache-based SCAs seek to obtain the secret key by analyzing the timing behavior of either the adversary or the secure application, using their cache access times and misses patterns when performing encryption or decryption operations [105].

With the rise of modern many-core-based domain-specific system-on-chip solutions in fields such as autonomous vehicles [163], high-end heterogeneous computers, graphic processing units, or AI accelerators [73], resource management techniques have become a key factor to accomplish high efficiency and performance. Due to the scalability degree of these systems and the variety of shared resources, secure applications, such as AES or RSA implementations, running on many-core systems, face potentially bigger threats from resource-based attacks, such as cache-based side-channel attacks. Resource management techniques are a powerful tool that can help to detect, mitigate, and overcome those threats without the need for hardware or architectural-dependent alterations.



(a) Ongoing cache-based SCA due to co-residency.



(b) Co-residency avoided via task migration.

Figure 7.1: Representation of task migration as a countermeasure for cache-based SCA on a distributed memory architecture.

7.1 Motivational example

As an example of how a resource management mechanism can act as a countermeasure to cache-based SCAs, consider the distributed memory system depicted in Fig. 7.1. The basic processing element is a *core* that contains its private instruction and data L1 cache. A *cluster* is a group of cores that share one or more cache levels (e.g., L2/L3) with each other. Clusters do not share cache memory with each other. This model is similar to the AMD "Zen 3" micro-architecture implemented in many commercial devices [16, 51]. The system shown in Fig. 7.1 represents a four-core system with two clusters, two cores per cluster, and two individual L2 cache memories; one shared between cores 0 and 1, and the other shared between cores 2 and 3.

In the modeled system in Fig. 7.1a, a secure application is executing on core 1. On core 0, an adversary is performing a cache-based SCA against the secure

This chapter is mainly based on [6].

application exploting the cache co-residency between the two applications. In this type of system, the operating system/resource manager can avoid the attack by applying dynamic task migration. In Fig. 7.1b, we depict a scenario where the secure application is migrated to core 3, which is no longer sharing an L2 with the attacker, hence making it impossible for the adversary to ever succeed.

7.2 Problem Definition

Although simplistic, the above example demonstrates the principle of dynamic task migration as a countermeasure to SCAs. By eliminating the co-residency between potential attacker and the security application, the attack is completely avoided.

However, since the constant migrations can produce adverse effects on the system, a more complex solution should address the following questions:

- When should the resource manager apply a migration to a secure application?
- To which core/cluster can the secure application be safely migrated?
- How to avoid an attack even if migration is not possible due to full utilization?
- What is the potential impact on the system and other applications due to the countermeasure?

In this chapter, we address the challenges associated with these questions, in order to produce an efficient countermeasure to cache side channels by leveraging task migration as the acting mechanism to mitigate the attack.

7.3 Novel Contributions

In this chapter, we present a novel approach to cache-based side-channel-attack mitigation on many-core systems. With the aid of a smart dynamic task migration heuristic, we show how to ensure a secure execution scenario

for security-critical applications residing with potential attackers, as well as non-secure applications. Our main contributions are:

- We present the first use of dynamic task migration as an operating system level resource-management approach to specifically mitigate cache-based side-channel attacks on distributed systems. One of the main advantages of our approach, besides being the first of its kind, is that it does not require additional hardware, architectural, or compiler support, while keeping a very low overhead in the system, compared to other software-based approaches from the state of the art.
- We offer a resource-management-based mechanism to provide a secure execution for an application when migrations are not possible (e.g., under a full utilization scenario).

7.4 Threat Model

Our threat model assumes that the attacker has the capacity of measuring either its own or the secure application's execution time, which is the norm for cache-based side-channel attacks. Similarly, for a successfully attack, the adversary needs to be executing in the same cluster as the secure application, sharing a cache level. Hence, it may have the ability to trigger the secure application execution, but this is not a requirement for our solution. The adversary does not have or require any administrator privileges.

As a model for the attacker in all our evaluations, we base our implementation on the 'Prime+Probe' and 'Evict+Time' access-driven attacks from [124]. Specifically, we implement the 'eviction' step, where the attacker fills the shared cache with its own data, producing misses on the victim application on its subsequent execution. Since we focus on the effect that that attacker has on the victim application's performance and not on the effectiveness of the attack, we do not implement the key extraction step. Moreover, since the key extraction happens after the secure application has finished its execution (i.e., offline for the purposes of the secure application), it is not relevant to our solution.

Note that a real-world attack requires application-specific knowledge (e.g., T-table address for AES, or a concrete address of a library's function), as well OS support, which is not necessarily available in a simulation environment.

Since the attacker implementation may vary significantly from one victim and platform to another, we extend our model to represent this diversity of attacks considering the impact that the attack has on the dynamic behavior (i.e., cache misses) of the victim due to the eviction. To this end, we present the following scenarios that will be used both in the development of the solution and its evaluation:

- **Fast**: A fast and strong attack that focuses on all cache lines simultaneously (i.e., 100 % of the Last-level Cache (LLC)), causing many misses on the target application, proportionally affecting its performance. On this scenario, a possible detection measure could easily notice the change in the secure application's performance.
- **Slow**: A slow attack that focuses on much less cache lines at the same time (i.e., 25 % of the LLC), causing low, or almost undetectable, performance degradation on a secure application. Here, because the degradation is not detectable, the countermeasure requires to consider another factor: time. Since the adversary is slower, by tracking the time that both the attacker and the secure application share on the same cache, the countermeasure could act before an attack is successful.
- **Intermediate**: An intermediate-strength adversary is one that focuses on attacking a moderate number of cache lines (i.e., 50 % of the LLC), making it faster than the slow attack, but slower and harder to detect than a fast attack. Due to the intermediate nature, a threshold-based performance monitoring technique might not be able to notice the attack. Similarly, a maximum shared time condition might also fail to avoid the attack, since it could succeed before the time limit is reached. On this mixed scenario, a combination of both, performance degradation and shared time, could help to avoid such adversary.

7.5 Migration Decision

To determine the moment for migration due to a possible attack, we propose a *migration function* m_f that ponders two *factors*: performance degradation and time. As described in Section 2.2, a secure application under a cache-based SCA experiences performance degradation, which is especially noticeable under a high-intensity fast attack, as described on Section 7.4.

We refer to performance as Instructions per Cycle (IPC), measured from the CPU-internal performance counters. The performance degradation then is computed in Eq. (7.1) as the difference between the original application's performance running in isolation (IPC_{isol}) and its current performance. This difference is then amplified by a constant factor α. Since the values are normalized, the migration function is designed to reach a value of 1 to trigger a migration. As shown in Eq. (7.2), α can therefore be obtained as the inverse of the difference between the application's performance in isolation and its minimum accepted value.

$$m_f = \alpha * (IPC_{isol} - IPC) + \frac{t_{shared}}{t_{smax}} \tag{7.1}$$

$$\alpha = \frac{1}{(IPC_{isol} - IPC_{min})} \tag{7.2}$$

On the other hand, a less intensive slower attack might not produce a detectable effect on the secure application's performance. To cover against such attacks, we consider an additional factor in the migration function: time. As described in Section 2.2, successful cache-based SCAs require a certain amount of time (i.e., from seconds to minutes) in which the adversary shares the victim's cache. In Eq. (7.1), this shared time is computed as the fraction of the current shared time between secure and non-secure applications (t_{shared}) and the maximum tolerated shared time (t_{smax}). Based on the observation from that most existing attack implementations require seconds to extract the key [86, 87], a t_{smax} in the order of tens or hundreds of milliseconds would suffice to tackle a fast adversary. It is worth noticing that a smaller t_{smax} would increase the resiliency against fast attackers, but this would lead to more migrations, which would negatively impact secure and non-secure application's performance.

Because the migration function is composed of two main factors (which have been normalized to reach a maximum value of 1), it is designed to trigger the migration when either of those factors reaches its maximum. Additionally, by adding both factors, we can trigger a migration when their combination also reaches 1. This is meant as a way to catch an intermediate strength attacker, which may not ever reach 1 in the IPC factor of the migration function, but when adding the second factor (time) the migration will happen sooner because of the combination. When adding both factors this way, attacks

with intermediate intensity can be covered. Although none of the individual factors might be enough to trigger a migration, their sum will add up to trigger a migration early enough to prevent a successful attack.

7.6 Dynamic Task Migration Heuristic

Once the migration decision has been made for the secure application, the resource manager must ensure a new secure execution scenario for the application. Our proposed algorithm, which is shown as pseudo-code in Alg. 5, seeks to meet this requirement. As seen in the figure, at the beginning of the periodic scheduling call, we evaluate the migration function (m_f, see Eq. (7.1)). If its output its greater than one, we must apply the migration heuristic.

To do this, the algorithm first tries to find a free core on an empty cluster to which the secure application should be moved. If that happens, as depicted in the figure, we select the first core from that cluster and migrate the secure application to it. If there are no empty clusters, the next step is to find a cluster with at least one free core and whose running threads have not exceeded the maximum allowed shared time restriction between the secure application and other non-secure applications (t_{smax} in the flowchart). Since there could be multiple attackers, even on different clusters, this condition is checked to avoid migrating into a cluster with a possible attacker on it. If we find a core this way, then we migrate the secure application to it. Otherwise, the algorithm tries to produce a *forced isolation execution* (lines 19 to 40 from Alg. 5) for the secure application, by creating an empty cluster or ensuring a cluster without non-secure applications on it, to which to migrate. To do so, our heuristic first checks that there are enough free resources for all other non-secure applications from the cluster executing the least number of threads. If there are enough resources, each one of those non-secure applications is migrated and the secure application is migrated to that cluster. This is done to minimize the number of migrations and reduce the impact on performance. If it is not possible to produce a secure execution via migration (e.g., there are no available resources to reallocate applications), then an alternative mechanism is invoked, as seen in line 42. Section 7.6 shows a detailed description of this countermeasure.

Algorithm 5: High-level pseudo-code representation of migration algorithm

```
input  : SecureThreads, NonSecureThreads, AvailableCores
forall i ∈ SecureThreads do
    ncluster ⟵ find_cluster(EMPTY);
    if ncluster ≠ ∅ then
        new_core ⟵ core 0 of ncluster;
        done ⟵ true;
    else
        ncluster ← find_cluster(ts_j < t_smax, ∀ j);
        if ncluster ≠ ∅ & cluster ≠ cluster(i) then
            new_core ⟵ find_core(ncluster);
            if new_core ≠ ∅ then
                done ⟵ true;
            end
        end
    end
    if done = true then
        migrate i ⟶ new_core;
    else
        done ⟵ true;
        lcluster ⟵ find_cluster(LEAST_THREADS);
        forall j ∈ NonSecureThreads on lcluster do
            if AvailableCores ≠ ∅ then
                done ⟵ false;
                for c ∈ AvailableCores do
                    if cluster(c) ≠ cluster(i) & cluster(c) ≠ lcluster then
                        C_j ⟵ c;
                        done ⟵ true;
                    end
                end
            else
                done ⟵ false;
            end
        end
        if done = true then
            forall j ∈ NonSecureThreads on lcluster do
                migrate j ⟶ C_j;
            end
            if lcluster ≠ cluster(i) then
                new_core ⟵ core 0 of lcluster;
                migrate i ⟶ new_core;
            end
        else
            apply_alternative_mechanism() ;
        end
    end
end
```

Secure Execution on Full Utilization

When no secure scenario can be found due to full utilization, then an additional countermeasure is invoked to prevent a possible attack. The goal of this countermeasure is to create an isolation scenario for the secure application. Since there are no available resources left, the only way of ensuring this isolation is by choosing a cluster for the secure application and move all other applications on that cluster. A problem arises here, since there are no resources left where to move these applications. To solve this problem we implemented a *sleep queue*, which holds applications temporally while no resources can be found for them.

To ensure a secure execution, the resource manager first finds the cluster with the least amount of threads and puts each of its non-secure threads on the sleep queue, before moving the secure application to that cluster. With the sleep queue, additional logic is added to our migration heuristic. In each scheduling interval, as shown in Alg. 6, the resource manager checks the sleep queue to assign sleeping threads to free cores. As shown from lines 2 to 8, if free cores are found on a cluster where no secure application is running, a sleeping thread is assigned to that core. Otherwise, a thread from a different cluster is put to the end of the sleep queue (line 9), and the original thread takes its place (lines 10 to 12). Thus, the secure application's execution is never affected and the potential attack is avoided. As a consequence of this alternative, the performance of other non-secure applications might be affected, but fairness is ensured by applying a round-robin scheduling on which thread to send to sleep.

7.7 Experimental Evaluation

The experimental evaluation uses the setup described for the HotSniper Simulator in Section 3.1 with minor modifications. We extended the open scheduler from to support the cluster concept and to implement our migration function and main algorithm. As for the open scheduler, we use a scheduling interval of 10 ms. For all the experiments described below, we use a modified version of the Xeon X5550 Gainestown model. To simulate a distributed system, we modified the *gainestown* model in Sniper to include a distributed L2 and L3 cache for each cluster on the system, while keeping the default

Algorithm 6: High-level pseudo-code representation of the additional countermeasure

```
input  : AvailableCores, SleepQueue
for j ∈ SleepQueue do
    if AvailableCores ≠ ∅ then
        for c ∈ AvailableCores do
            if cluster(c) ≠ cluster(i) then
                new_core ⟵ c;
            end
        end
    else
        t ⟵ next_nonsecure_thread_round_robin();
        new_core ⟵ core(t);
        sleep (t);
        assign j ⟶ new_core;
    end
end
```

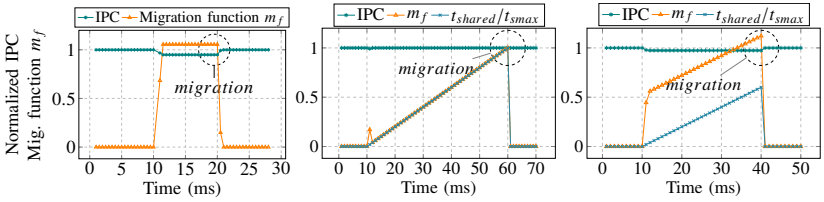
private L1 cache per core. The configuration of the cache hierarchy is shown in Table 7.1. Unless otherwise specified, we work with a baseline platform of 64 cores, distributed evenly as 4 cores per cluster.

Table 7.1: Cache hierarchy configuration.

Parameter	L1	L2	L3
associativity	8-way	8-way	16-way
size	32 kB	256 kB	8 MB
block size	64 B	64 B	64 B
data access time	4 cycles	8 cycles	30 cycles
tag access time	1	3	10 cycles
shared cores	1	4	4
write back time	0	0	50 cycles

For all the tests, we use an α-value of 8.5 in the migration function, which is computed according to Eq. (7.2), using an minimum accepted IPC value of approximately 85% of the isolation performance.

As discussed in Section 7.5, a higher α might work better against faster attacks, but it also may produce a larger number of migrations, affecting the overall performance. Unless otherwise specified, we settle to a t_{smax} base value of 120 ms, which is less than it takes for most modern attacks to extract the key without any previous interaction with the application, as described in



(a) Fast attack. **(b)** Slow attack. **(c)** Intermediate attack.

Figure 7.2: Effect on the normalized performance, shared time ratio, and migration function output for AES under the fast, slow and intermediate attack scenarios, launched at t =10 ms.

Section 2.2. We discuss this trade-off between security and performance further down in Section 7.7.3.

7.7.1 Security analysis

The first part of our evaluation method consists of the security analysis of our proposed solution. We construct several cache-based side-channel attack scenarios against a secure application. The secure application used for all scenarios is the OpenSSL 1.1.1k [152] implementation of AES with a 128-bit key. Throughout all our experiments, we launch the secure application and the attacker sequentially, to highlight the impact of the attack itself on the secure application's performance and shared time before and after the attack is launched. Although they are launched sequentially, both applications run concurrently, since in practice both applications must execute at the same time for the attack to succeed.

Migration due to performance loss

In the first scenario, we model a fast attacker, described in Section 7.5, which fills the L2 cache with its data to force a higher miss rate on the secure application. To measure the performance degradation and its migration function's impact (see Eq. (7.1)), we start executing the AES application on cluster 1 at t=0 ms. Then, at t=10 ms we launch the aggressive attacker on the same cluster. As shown in Fig. 7.2a, right after the attacker is launched, the normalized performance (i.e., current performance divided by the maximum achievable performance) of our secure application decreases. The output from the migration function (m_f) increases, reacting to the reduced performance.

When m_f is greater or equal to 1, the migration of the secure application is triggered. After the secure application has been migrated to a different cluster, its performance returns to its original value. When the migration happens, the attack is avoided since the attacker and the secure application are on a different cluster and no longer share the cache.

Migration due to maximum shared time violation

In a second scenario, we model a slower attack, which seeks to avoid detection by reducing its intensity. An attacker with this behavior, although requiring more time to extract the sensitive information, would not affect the secure application's performance significantly. Since our solution contemplates the shared time between secure and non-secure applications, our migration function will trigger a migration when the maximum allowed shared time (t_{smax}) is reached. For this experiment, we set t_{smax} to 50 ms, which is within the range discussed in Section 7.6. Fig. 7.2b shows the results of an example of this attack type. As before, we launch the attack at t=10 ms, but due to its low intensity, the effect on the secure application's performance over time is almost undetectable, besides from the slight decrease right after its launch, which is not sufficient ($m_f \approx 0.1$) to trigger a migration. Nevertheless, the output from the migration function keeps increasing because of the accumulated shared time. At t=60 ms, when t_{shared}/t_{smax} becomes 1, the secure application is migrated to a new cluster thus avoiding the potential attacker.

From these results we can extract that under our solution, the secure application will never share a cluster with a potential attacker for more than the allowed shared time, which means that the slow attacks are avoided.

Migration due to mixed factors

For a third scenario, we considered the intermediate-strength attacker described in Section 7.5, which is moderately fast but not aggressive. If our migration function would consider performance and time as separated conditions, then the intermediate-strength attacker might extract the secure key in less time than t_{smax}. While doing so, the secure application's performance is not affected enough to trigger a simple threshold condition. Therefore, by adding both factors we cover such attacker. In Fig. 7.2c, we show an example of this type of attack. For this experiment, we kept t_{smax} to 50 ms. Similar to the other scenarios, the attacker is launched at t=10 ms, but it is not until t=40 ms when the combined results of performance loss and shared time

fraction trigger the migration condition. Otherwise, this would just happen at t=60 ms, when the t_{smax} limit would be reached as shown in Fig. 7.2b. After 40 ms, the performance returns to a normal level, because the application is now running on a different cluster.

As the attacker and the victim are running on a different cluster, the attack has been avoided.

Forced Isolation

The fourth scenario corresponds to the forced isolation, described in Section 7.6, which happens when the secure application cannot be migrated to any cluster. This scenario is depicted in Fig. 7.3. In this experiment, we run a total of 12 applications alongside a secure application (AES128). Any of these 12 applications can be considered a potential attacker. In the figure, applications that are not relevant to the isolation process (i.e., those who are never migrated) are marked as background. Here, these background applications are used to force migration of the secure application when the maximum shared time is expired. In this experiment, we launch our secure application at t=10 ms on cluster 0, core 3, with a t_{smax} of 10 ms. We chose this small value to trigger a force isolation scenario quicker, in order to illustrate the principle. After 10 ms, due to expired time shared with the background applications, the secure application is migrated to a different cluster. This process repeats every 10 ms until there are no available clusters to which to migrate. This happens at t=50 ms, on cluster 1. At this time, our algorithm enforces isolation by migrating all other applications from this cluster (apps 5, 6, and 7) to cores on other clusters. As the secure application is now on isolation, any potential attack has been avoided.

Detection policy and false-positives

The proposed migration function serves as a detection-like mechanism for fast, slow, and intermediate adversaries, as described above. As such, could it be susceptible to false positives? Would a non-attacker memory-intensive application have a big enough degradation on the secure application's performance to trigger a false-positive migration? In our solution that would certainly be the case. Nonetheless, this scenario is much less critical in security terms than a false-negative (i.e., a real attacker that does not trigger the migration function by performance degradation means), which we fully avoid. A migration due to false-positive mainly affects the secure application's performance, which will recover quickly after the migration occurs, preserving

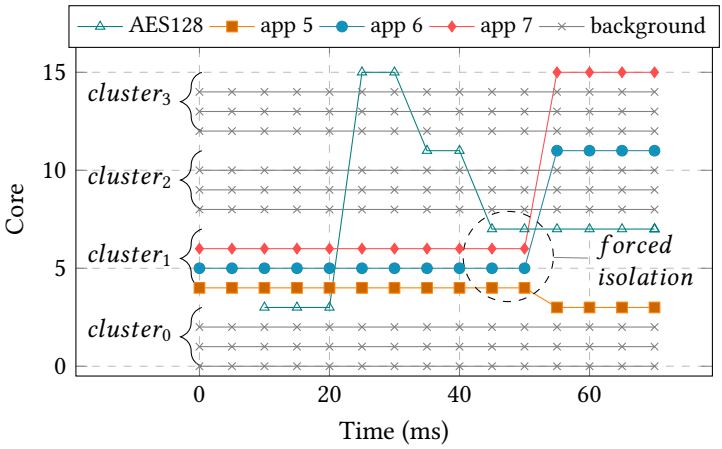


Figure 7.3: Core assignment over time on forced isolation scenario.

the overall system performance. On the other hand, false-negatives, that do exist on other detection solutions, would allow the attack to happen. Since our approach is a security-first solution, we prioritize avoiding all false-negatives at the cost of allowing some false-positives.

7.7.2 Application Overhead Analysis

In this chapter, we analyze the effect of the migration policy on the performance of both secure and benchmark applications.

Migration Overhead

For secure application, we perform an experiment where we use the OpenSSL 1.1.1k implementation of AES and apply a one time migration to it. The migration moves the AES application from core 0 on cluster 0, to core 4 on cluster 1, so that it gets a different L1/L2 cache on the new cluster. Fig. 7.4 shows the result of this one time migration. For the purposes of this experiment, we perform an unconditional migration at $t = 10ms$. As it can be seen from the figure, the performance decreases as expected for the next sampling interval, but it quickly recovers its original average value after less than $100\mu s$. Con-

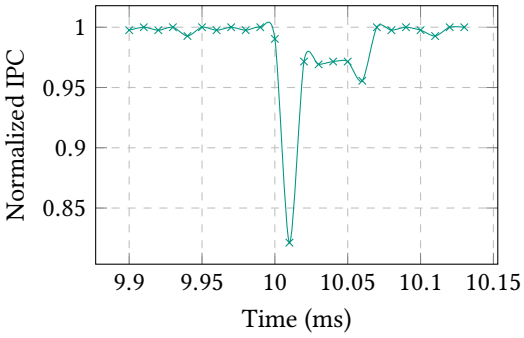


Figure 7.4: Effect of one-time migration on AES performance at $t = 10ms$.

sidering that our scheduling interval is 10 ms, this shows that the migration has no significant effect on the secure application's performance.

In order to measure the migration effect of non-secure applications, we devised an experiment where we apply a one-time migration to 6 randomly selected applications from PARSEC2 and SPLASH2 suites. For this purpose, we configure a 64 core platform, with 4 cores per cluster. Each application is launched on isolation at $t = 0ms$ on a different cluster. Results from this experiment are shown in Fig. 7.5. In this test, we apply the one time migration at $t = 50ms$, which moves each application to a different cluster, replicating our forced isolation scenario described on Section 7.7.1. As expected, the normalized performance of all applications decreases due to the migration. Although the recovery is not as fast as the AES example, all the applications get their original average performance after $400\mu s$. Considering that this value is much smaller than our scheduling interval, and the fact this scenario does not happen as frequently, it can be determined that the migration has no overall significant effect on non-secure applications' performance.

Realistic execution

To evaluate the performance impact of our solution in a realistic execution, we devised an experiment where we run 12 applications from the PARSEC2 [26] and SPLASH2 [161] benchmarks, alongside the AES128 secure application. We configured the Sniper simulator with two different systems. The *high utilization scenario* uses 16 cores and is configured such that the total number

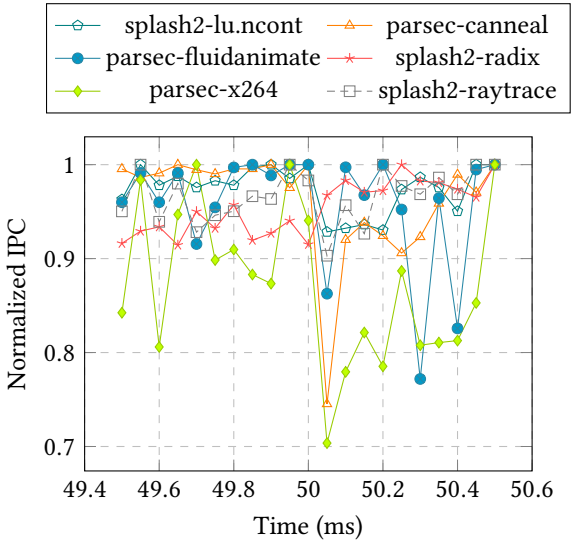


Figure 7.5: Effect of one-time migration on applications from benchmark suite, performed at $t = 50$ms.

of threads exceeds the number of cores. In the *medium utilization scenario*, we use 32 cores and the number of threads is lower than that. We set the arrival time for each benchmark application to a random time between 0 and 100 ms. This window is shorter than the required execution time of any individual application. This means that applications run simultaneously, while also experiencing the dynamic effect of other applications arriving in the system. The slowdown of the applications due to our migration heuristic is shown in Fig. 7.6. As it is depicted, the performance loss is less than 10 % (1.09× of execution time) on all tested applications, with an average of 1.6 %.

As it was mentioned on Section 7.6, when the system is at full utilization, non-secure applications are migrated or put on a sleep queue. For the high utilization scenario depicted in Fig. 7.6, the maximum number of migrations and sleep time for each application is shown in Table 7.2. As it can be seen, all applications experience migrations and/or sleeps during its execution. Additionally, no non-secure application gets migrated or slept significantly more times than other, thus showing how fairness is ensured. The parsec-

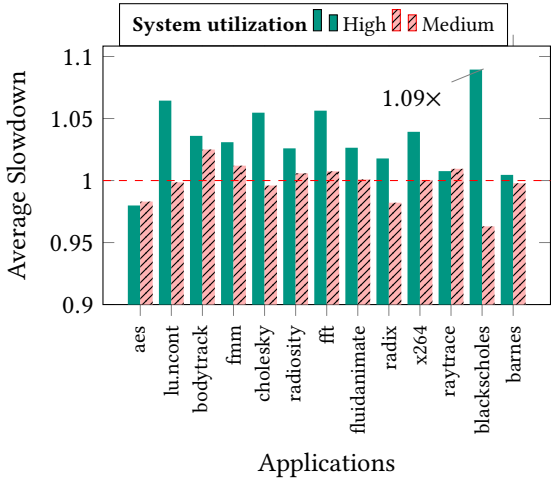


Figure 7.6: Average slowdown for benchmark applications under the secure migration heuristic for different utilization scenarios. Note that a slowdown ≥ 1 means a performance decrease.

blackscholes benchmark, which is the one with the highest slowdown, is also the one being migrated the most from all non-secure applications, which explains its behavior. It is worth noticing that the performance degradation is not exclusively linked to an application's own migration, as migrating other memory-intensive applications to the same cluster, will also contribute to negative effects on performance due to on-cluster memory contention.

Interestingly, some applications benefit from the migration policy in terms of performance, mostly for the medium utilization scenario. Since our migration policy focuses on migrating the secure application first, it gets migrated likely to an empty cluster. As new applications arrive, some of the threads might end-up executing in the same cluster as the secure application, which consecutively will trigger a migration on the secure application. As the migration keeps moving the secure application through the clusters, cores are being freed from these clusters. On lower utilization, these cores are likely not used anymore by other applications, relaxing the local cluster utilization which favors performance. On the full utilization scenario, it is more likely that the freed cores will be used again, since there are much more threads than the total number of cores in the system. This can explain

Table 7.2: Number of migrations and sleep time on full utilization scenario.

Applicaton	Nr. of Migrations	Sleep time (ms)
aes	4	0
splash2-lu.ncont	0	10
parsec-bodytrack	1	30
splash2-fmm	0	20
splash2-cholesky	1	10
splash2-radiosity	1	20
splash2-fft	0	20
parsec-fluidanimate	1	30
splash2-radix	0	10
parsec-x264	1	20
splash2-raytrace	1	10
parsec-blackscholes	3	10
splash2-barnes	2	10

why, on high utilization, the performance of all benchmark application is always decreased. As for the AES secure application, although being the one migrated the most, its performance is not negatively affected by the policy. On the contrary, once isolation has been ensured, the application continues the execution without any interference on its cluster, which explains its performance improvement.

7.7.3 Security and performance trade-off

As it was described in the security analysis, our migration heuristic is rather conservative in terms of co-residing applications. Since a migration will inevitably happen because of the shared time of the secure application and other co-residing apps, a potential attacker will be greatly mitigated, which is the goal of our solution. Although a slower attacker might be able to extract a small part of the sensitive information from the secure application before the migration occurs, this time is kept significantly small in our solution (t_{shared} in the order of 100 ms) and could be reduced even further. As mentioned before, a faster attacker, like the one shown in Fig. 7.2a, will cause a bigger degradation in the secure application's IPC, triggering the migration much faster than t_{shared}. Moreover, with our policy, the attacker will not be able to further execute in the same cluster as the secure application, making it harder

to complete the attack. The cost for these decisions is the overhead that our solution might cause in other applications, especially in a high-utilization system, where migration happens often for said applications. However, as our realistic scenario evaluation clearly shows, we keep the overhead very low (i.e., less than 10%) even for both medium and full utilization scenarios.

7.7.4 Comparison against state-of-the-art solutions

We compare our solution against three different state-of-the-art software-based countermeasures.The compiler-based solution from [39] produces an overhead ranging from 1.25× to 2.1× on the secure application, and up to almost 8× on a benchmark application. The cache flushing technique from [61] produces a maximum overhead of 15 % on the Apache benchmark. On the operating system level, the work from [108] produces a slowdown of 29 % on the same type of applications we use. Being on the operating system level and using the same type of applications make this solution a reference candidate to compare our results. As it can be seen, with a maximum overhead of 9 %, our solution outperforms the three different techniques from the state of the art, while mitigating the attacker on different scenarios.

7.7.5 System Run-time Overhead

Our solution introduces a periodic run-time overhead when executing the migration heuristic for every scheduling interval. This execution time depends on the system utilization, number of cores (n), and number of clusters (m). The worst case for the execution happens on full utilization, where all cores are occupied by threads. In that scenario, the selection of the new migration core for each thread (line 25 in Alg. 5) gets executed the most times (i.e., $n \times m$ times). This implies a time complexity of $O(n^2)$ if the number of clusters and cores is approximately the same. In a more realistic case, where $m \approx log_2(n)$, the corresponding complexity is $O(nlog(n))$.

To further evaluate this overhead, we run Algs. 5 and 6 as a single application that executes both sequentially one after the other. This represents a pessimistic scenario where the additional countermeasure is always executed. In this experiment, we keep the number of cores per cluster constant, but measure the overhead for a different number of cores in the system. The

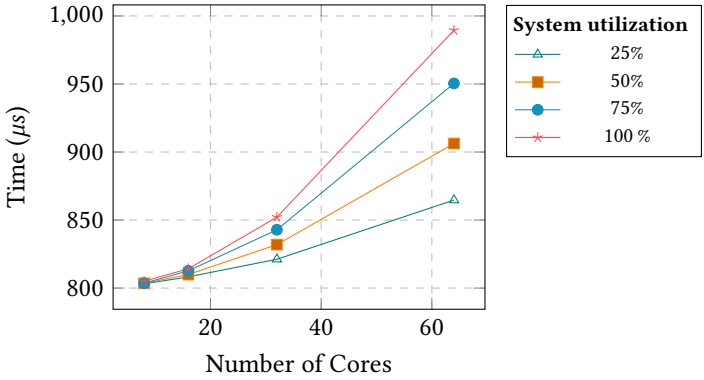


Figure 7.7: Migration heuristic run-time overhead vs. number of cores for different system utilization percentages.

reasoning behind this is that the outer loops in Alg. 5 mainly depend on the number of threads (secure and non-secure) which in turn depends on the total number of used cores, regardless to which cluster they belong.

The number of cores per cluster affects the *find_cluster* methods, since assuming a constant number of cores, more cores per cluster implies fewer total clusters, making these functions take less time to execute. On the other hand, fewer cores per cluster mean more clusters, which would affect these methods more. However, as a cluster-based architecture relies on clusters of cores, there will always be much more total cores than clusters, making the former number more impactful than the latter. Fig. 7.7 shows the results of this experiment. In the figure, the utilization represents the percentage of the cores used by the applications. Even for the worst scenario of 64 cores with 100 % utilization, the overhead time is kept as low as less than 1 ms, which represents a small fraction of the scheduling interval.

7.7.6 Power and Energy Overhead

As a final evaluation, we measure the effect our solution has on the power and energy of the system. To this end, we run the same full utilization scenario described in Section 7.7.2 and measure the total average core power and

execution time. We show the results from this experiment in Table III. As can be seen from the table, when executing our policy, we achieved a slightly reduced power, at the cost of a slightly higher execution time. Ultimately, this translates into an overall small increase in energy usage of 160 mJ (0.23%). This means that our solution does not produce a significant power or energy overhead in the system.

Table 7.3: Power and energy with and without our proposed policy

Configuration	Execution time (ms)	Power (W)	Energy (J)
Without policy	2608	26.96	70.31
With policy	2620	26.90	70.47

7.8 Summary

In this chapter, we introduced a novel cache-based side-channel attack mitigation strategy using dynamic task migration. Our policy prevents secure applications from sharing cache levels with potential slow or intermediate-strength attackers beyond t_{smax}, triggering migration when necessary. This ensures protection against such attacks. For faster attackers, early migration is initiated due to performance drops, preventing attacks. Our security-first approach avoids false negatives but allows for some false positives. Through analysis, we demonstrated secure execution for critical applications under our migration heuristic. Additionally, we proposed a resource management mechanism to prevent attacks when migration is not feasible due to high system utilization. The average performance overhead is minimal, with a worst-case 9 % slowdown during high utilization, outperforming current countermeasures. The migration heuristic incurs less than 1 ms execution time overhead in a fully utilized 64-core system.

8 System-informed Mitigation of Covert Channels

In order to mitigate the threat of TCCs, and power-based covert channels in general, several detection and countermeasure techniques have recently emerged [1, 77, 78, 158, 159], especially for general-computing devices. Even in such platforms, the challenge of such techniques resides in effectively detecting and mitigating the attack while reducing the performance impact on the system.

In the countermeasure domain for such channels, DVFS has been proposed [77, 78], as the default mechanism to tackle the attack. By dynamically switching between high and low frequencies for the processing device, the countermeasure technique affects the power consumption of the device and the system, and hence directly interferes with the covert communication. However, as it has been shown, reducing the frequency of the CPUs affects the performance of applications executing there. Current countermeasures do not consider the system information in the techniques, which can significantly affect its energy consumption and performance. For a many-core system, for example, when the attack is present at all times, the performance loss for a benchmark application set due to the countermeasure has been known to reach up to 25% [77], whereas the effect on the energy on the system due to the countermeasure has not properly been evaluated in the state of the art.

While energy efficiency might not be the most relevant metric for general-purpose computing platforms, for modern energy-constraint embedded systems it is a critical factor, hence in this chapter we target our work to such devices.

This chapter is mainly based on [7].

8.1 Motivational Example

Figure 8.1 shows different scenarios for an application set executed on an NVIDIA Jetson TX2 embedded board, detailed in Section 3.2.3.2. As shown, the device has two CPU clusters: an ARM CPU cluster with a Quad-Core ARM Cortex-A57, and a Dual-Core NVIDIA Denver 2 cluster. As initial state, five applications from the SPEC2006 [74] benchmark suite and a malicious thermal covert-channel transmitter (named "tcc") are executed on the system. Shortly after beginning the execution, the malicious application's core is detected, using a proven detection technique (e.g., [1, 159]). As a countermeasure, we first apply DVFS to the core where the malicious application executes, as it is the state-of-the-art countermeasure technique [77]. Because of the clustering, the DVFS technique is applied to the whole ARM cluster, producing high performance and energy penalties in the system since all other applications executing in the same cluster are affected.

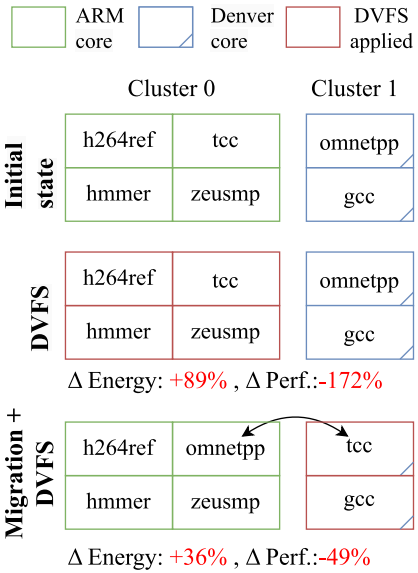


Figure 8.1: Effect of applying migration and DVFS on the energy and performance (makespan) of the system.

In order to show how this overhead can be reduced, we depict an alternative scenario, where we arbitrary migrate the malicious application from an ARM core to a Denver core, by dynamically exchanging the cores where *omnetpp* and *tcc* are executing, before applying DVFS. Because the offending application is now on the Denver cluster, we apply DVFS on that cluster, therefore affecting only one other application. By doing so, we are able to massively reduce the energy and performance overhead in the system.

As it is shown, while this arbitrary decision is able to reduce the overhead penalties in comparison to blindly applying DVFS, the performance and energy penalties are still significant.

In this chapter, we focus on the challenge of mitigating power-based attacks in an energy-efficient manner. We seek to highlight this problem, which has not been done properly for embedded devices, and address it by including information about the system as input to the countermeasure itself. We propose the use of system-informed techniques based on the combination of DVFS and dynamic task migration to mitigate power-based covert channels. By doing so, we are able to reduce the energy and performance penalties of the countermeasures on the real platform, while still mitigating the attack.

8.2 Problem Definition

As previously discussed in the motivational example in Fig. 8.1, blindly applying DVFS to mitigate power-based covert channels, as done in the state of the art, can lead to high performance and energy penalties in the rest of the system. In a real setup, the dynamic state of the system (i.e., type of applications, CPUs' frequencies, system load, etc.,) creates a complex environment where the ideal execution scenario for the current application set is not easy to predict. To tackle this problem, we intend to introduce system information into the covert-channel migration strategies by combining task migration with DVFS. Each mitigation strategy attempts to address the following challenge at runtime: *Once an attacker is detected, what is the best state the system should transition into (enforced by task migration) before applying DVFS, such that energy efficiency is maximized while the attack is mitigated?*

In the following sections, we show the design and implementation considerations for our proposed techniques, which seek to address exactly this challenge.

8.3 Novel Contributions

The contributions presented in this chapter are the following:

- We propose for the first time the use of system-informed techniques as countermeasures for power-based covert channels.
- We devise new countermeasures to power-based covert channels from the heuristics and ML domain that combine for the first time DVFS and dynamic task migration to tackle the attack.
- We deploy both our proposed countermeasures and the state-of-the-art DVFS approach on two real embedded platforms. Our extensive evaluation demonstrates how our techniques mitigate the attack while significantly reducing both energy and performance penalties.

8.4 Enabling System and Application Awareness

As previously introduced, in this chapter we propose system-informed techniques to mitigate power-based covert channels through the combination of DVFS and dynamic task migration. To support the techniques, we implemented a resource management orchestration application. This orchestration application, depicted as an overview in Fig. 8.2, is in charge of generating the experiment parameters (e.g., workloads and initial mapping configurations), launching the applications, periodically monitoring the system metrics, selecting the countermeasure policy, and finally enforcing the technique by migrating the applications and applying cluster-level DVFS where required by the countermeasure technique.

For the application set, we generate random workloads consisting of combinations of applications from the SPEC2006 benchmark and a functional power-based covert-channel transmitter in a one-application-per-core manner. Moreover, our monitoring tool periodically gathers execution metrics

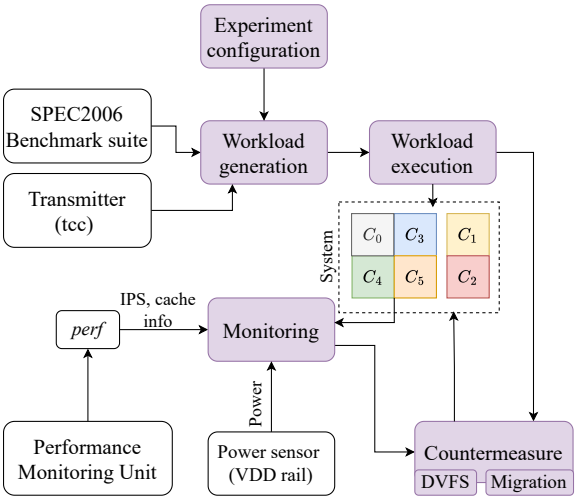


Figure 8.2: Overview of the orchestration resource management application.

from the system, CPUs, and cache, such as IPS, cache accesses, cache misses, and system power every 100 ms. We use *perf* [130] as the back-end mechanism to collect the performance counters' information (both CPU and cache). For the platform's power, we read the board's power sensor accessible through Linux the file system.

8.5 Heuristic-Based Mitigation

To reduce the overhead in the system due to an uninformed countermeasure, we propose a simple but effective technique that considers the performance of the cores to decide an efficient application mapping at runtime.

Our *Worst Performing Cluster - Best Performing Core* (WPCBPC) heuristic follows the principle of reducing the effect of performance penalty due to DVFS. It does so by moving the attacker application first to the cluster that has the worst performance. Then within that cluster, it selects the most performing core as the candidate for migration before enforcing the new application mapping dynamically in the next scheduling epoch. The full pseudo-algorithm for this technique is shown in Alg. 7. In the algorithm, we

Algorithm 7: Our WPCBPC Heuristic

```
1  Input: attack_core, curr_mapping
   Result: new_mapping: New application mapping configuration
2  cluster0_cores ← {0, 3, 4, 5};
3  cluster1_cores ← {1, 2};
4  IPS_cluster0 ← 0;
5  IPS_cluster1 ← 0;
6  all_IPS ← {};
7  for core in all_cores do
8      all_IPS.push(getIPS(core)) ;    /* Gets the performance for each
         core */
9  for core in cluster0_cores do
10     IPS_cluster0.push(all_IPS[core]) ; /* Performance for Cluster 0
         */
11 for core in cluster1_cores do
12     IPS_cluster1.push(all_IPS[core]); /* Performance for Cluster 1
         */
13 target_cluster ← cluster1_cores;
14 if average(IPS_cluster0) < average(IPS_cluster1) then
15     target_cluster ← cluster0_cores;
16 max ← 0;
17 cid ← −1;
18 for core in target_cluster do
19     if all_IPS[core] > max then
20         max ← all_IPS[core];
21         cid ← core ;          /* Finds the best performing core */
22 new_mapping ← curr_mapping;
23 moving_app ← curr_map[cid];
24 new_mapping[attack_core] = moving_app;
25 new_mapping[cid] = curr_mapping[attack_core];
26 return new_mapping;
```

first collect the performance information (IPS) from each core (lines 7 to 12). Then we identify the worst-performing cluster, by comparing the averages of the accumulated IPS (lines 13 to 15) over the last second of execution. The

reason for selecting the worst-performing cluster as the host to the potential attacker is that the application of the subsequent DVFS policy will affect the overall performance the least. To further enforce this, we then select the best-performing core from that cluster (i.e., the core with the highest IPS over the last second of execution) as the final candidate to which the malicious application should be migrated (lines 18 to 21). In this way, the best-performing application from the soon-to-be-affected cluster will not be affected by the performance penalty due to DVFS. Finally, we create the new mapping configuration in the system (lines 22 to 25) where the cores belonging to the attacker and the best performing application on the candidate cluster have been exchanged.

8.6 Machine Learning-Based Mitigation

To further explore other system-informed techniques, in this section, we introduce some machine learning-based countermeasures to power-based covert channels. Through different supervised ML algorithms, we seek for our models to learn the behavior of the system when mitigating the attack, and predict the best possible task migration scenarios at run-time. Instead of relying on heuristics, this approach attempts to quantify the impact of different mitigation strategies on the overall energy efficiency of the system.

Fig. 8.3 shows a high-level overview of our ML-based mitigation. Our approach follows a four-step process both for design and runtime. First, at design time, the process starts by generating a random workload from the SPEC2006 benchmark suite plus the malicious application, as an initial mapping (1). Then we start the execution of the workload and wait for a random delay (i.e., between 1 and 10 seconds) before starting the monitoring (2) to encounter different execution phases for the applications. After that, we collect the performance (IPS) for each core, cache misses and accesses, and system power information periodically every 100 ms for a window of 1 s. When the collecting period expires, we create a new random mapping for the current workload and then apply DVFS to the cluster to which the malicious application will be moved (3). Then we set the new core's affinity to each application, which enforces the dynamic task migration to the workload following the new mapping. Finally, we again collect the statistics for the

workload under the new mapping configuration (4). In addition to the mentioned metrics, we compute the energy efficiency (Instructions per Joule) obtained as a consequence of the migration and DVFS for the new mapping configuration. We repeated this process more than 5000 times, collecting over 180 individual data points per iteration. With the obtained metrics for all the iterations, we form a training dataset where each row contains a representation of the original mapping, its statistics, the representation of the new mapping, the new statistics, and the obtained energy efficiency. This dataset is then used to train the ML models we employ for the countermeasure technique.

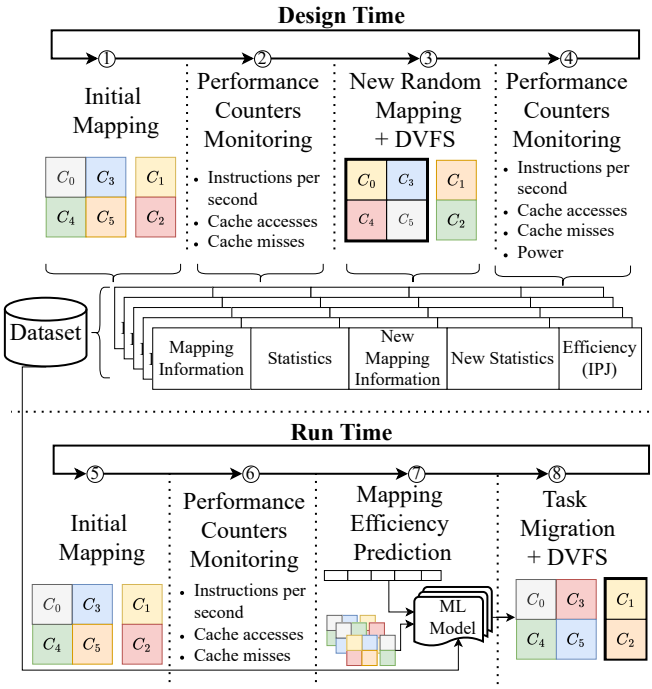


Figure 8.3: Overview of the ML-based countermeasure techniques.

At runtime, our techniques are applied in a continuous process which starts from an initial running mapping configuration (5). Then, we accumulate and collect the execution metrics over the most recent second of execution

Table 8.1: Prediction accuracy of different ML algorithms on the validation dataset

Regression Model	RMSE (10^9)	R^2 Score	MAE (10^9)
Linear	0.47	0.76	0.31
Ridge	0.49	0.73	0.32
Lasso	0.58	0.63	0.39
Elastic Net	0.53	0.69	0.34
K-Nearest Neighbors	0.49	0.73	0.30
Decision Trees	0.64	0.54	0.38
Random Forest	0.45	0.77	0.26
Neural Network	0.45	0.76	0.29
XGBoost	0.31	0.89	0.19

(6). After that, for each possible nonredundant mapping variation, we call the ML model to predict energy efficiency (7). It is important to note that in order to reduce the number of possible mapping predictions, we ignore mapping variants where all applications would reside in the same cluster but in different cores. Although technically different, these are *redundant mappings* in the sense that all applications are set to execute within the same DVFS domain, and the DVFS action would affect the same applications in the same manner. After the prediction of energy efficiency for all possible mappings is performed, we select the new mapping variant that produces the highest efficiency value as the new mapping configuration for the system. Finally, we enforce this new mapping configuration by applying task migration and then DVFS to the cluster where the malicious application is set to execute (8). This process is then repeated until the workload finishes the execution.

The following subsections describe in more detail each one of the steps involved in the design and implementation of the ML-based techniques.

8.6.1 Training Data Generation and Preprocessing

Following the steps depicted in the design-time phase of Fig. 8.3, we generate a dataset of ~1M data points. The dataset first undergoes standardization and scaling in order to adjust the distribution of each feature to have a mean of zero and a standard deviation of one, thereby enhancing the model's ability to converge during training. The scaling parameters are saved for usage at

runtime. Finally, we perform a random split of 80% / 20% training / testing of the dataset to prepare for the model training phase.

8.6.2 Feature Selection and Model Training

The problem at hand is a *regression* problem that can be solved with various machine learning algorithms, e.g., decision trees, random forests, neural networks, etc., where the label is set as the energy efficiency of the system *after* migration. Therefore, we first train different *regressors* from the *scikit-learn* Python library [129] with their default parameters using the raw dataset in order to identify the most promising algorithm for this specific problem. Table 8.1 shows the Root-mean-square Error (RMSE), R2 and Mean-absolute Error (MAE) scores achieved by the different models, with the eXtreme Gradient Boosting (XGBoost), *Random Forest* and NN models outperforming the other regressors. We then focus on training optimized models with each of the three selected algorithms as follows.

The XGBoost model is used for implicit feature selection during learning, identifying key features through its tree-building process and calculating a *gain* metric reflecting each feature's contribution to predictive accuracy. Higher values signify greater importance and correlation with the efficiency label. To refine feature selection, grid search hyperparameter tuning is performed, leveraging the *GridSearch* library from *scikit-learn*. Hyperparameters include estimators (up to 300), tree depth (up to 9), learning rate, subsample ratio, and column sample by tree. Exploration results in shortlisted features for each core: cache accesses/misses, retired instructions, encoded application IDs, and system energy efficiency *before* migration. The per-core feature grouping is crucial, as *it guides the model to learn the individual characteristics of the core as part of its cluster*. The final XGBoost model with 10 estimators and tree depth of 6 achieves high prediction accuracy with MAE and RMSE scores of barely 0.19×10^9 and 0.31×10^9.

Based on the feature importance insights obtained from training the XGBoost model, the same list of features is maintained for training the acNN model. The search for the model topology, including the depth and breadth of layers, is performed using the *Keras Tuner*. The non-linear ReLU activation function is incorporated in each hidden layer to introduce non-linearity and the *Adam* optimizer is used to effectively manage back-propagation and the learning rate during training. The final obtained NN model consists of 3 hidden layers

with 32, 32, and 16 neurons. Though slightly less accurate than the XGBoost model, the NN model also achieved a very high prediction accuracy of the energy efficiency label, with MAE and RMSE scores of 0.29×10^9 and 0.45×10^9, respectively.

Finally, with the same list of features as the two previous models, we train a Random Forest model, by using *GridSearch* to explore a search space of parameters, including the number of trees in the forest and the maximum depth of each tree. The final model, which uses 100 trees, achieves a slightly higher prediction accuracy compared to the NN model, with MAE and RMSE scores of 0.26×10^9 and 0.45×10^9, respectively.

8.7 Experimental Evaluation

8.7.1 Evaluation Platform

For our evaluation, we conducted experiments on two real-world commercial embedded boards: the NVIDIA Jetson TX2 and NVIDIA Jetson Orin Nano described in Section 3.2.3.2.

Both platforms run Ubuntu as the operating system (18.04.6 LTS on the Jetson TX2 and 20.04.6 LTS on Jetson Orin). Notably, while the different experiments are undergoing no other application is executing besides normal OS operation. Furthermore, we set the power management governor of the boards to “userspace", which avoids system-controlled changes in the CPUs' frequencies. Additionally, we restore the frequency level of the cores to the maximum value before executing each workload.

8.7.1.1 Benchmark Application set

As the application set for our experiments, we use two benchmark suites. First, for training the ML-based models and general evaluation purposes, we employed 18 applications from the SPEC2006 benchmark suite, all using the intermediate (i.e., the so-called *“train"*) input size from the suite. The set includes applications both from the integer and floating point benchmarks. The full list is the following: *gcc, milc, bzip2, sphinx3, astar, lbm, bwaves, mcf, zeusmp, namd, h264ref, gobmk, povray, gromacs, cactusADM, omnetpp,*

hmmer, and leslie3d. The remaining applications from the SPEC2006 suite were not used due to compilation or execution errors on the board. As a second application suite, we employ the full set (i.e., apps and kernels) from the PARSEC 2 benchmark [26], using the *simlarge* input size. These applications are exclusively used for evaluation purposes i,e., they not used for any training and hence are unseen to the techniques. In Section 8.7.5 we employ these applications to show how our proposed system-informed techniques can perform well independently of the application characteristics with which where they were trained.

8.7.1.2 Malicious application

The overview for both malicious transmitter and receiver applications is shown in Fig. 8.4. The malicious transmitter is a C++ functional covert-channel application that modulates the power of the system to transmit information. Similar to other power-based covert channels [46, 78, 103], we employ encoding and modulation mechanisms such as return-to-zero and on-off-keying on the transmission. When encoding a bit of 1, the application continuously performs a compute-intensive kernel that increases the power consumption of the system. It consists of floating-point operations (i.e., square-root) combined with a busy-waiting loop. For a bit of 0, the malicious transmitter sleeps to reduce the power consumption.

To evaluate the communication, we implement a simple off-line receiver which upon saving the power measurements from the sensors, filters them and then decodes and de-serializes the bits employing a threshold-based approach as done on other approaches (e.g., [78]). For the purposes of the evaluation, the channel frequency is set around 15 Hz. Due to modulation, the transmission speed of the channel is approximately 2.67 bits per second, which is in the normal range for power-based covert channels (e.g., thermal covert channels [2, 109]). As we show further in Section 8.7.3 when no countermeasure is present in the system, the channel can communicate information reliably with low error rates (i.e., less than 5%).

8.7.2 Baseline and Naive Policies

As a baseline for comparison with our proposed techniques, we implement the state-of-the-art DVFS technique from [77] (called simply "*DVFS*" in our

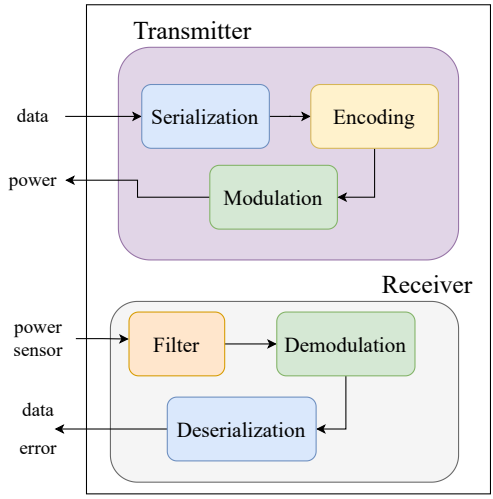


Figure 8.4: Overview of the transmitter and receiver malicious applications

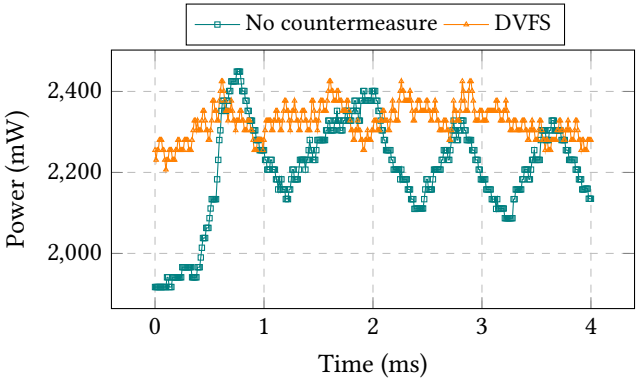


Figure 8.5: Power signal of the Jetson TX2 platform during the transmission of a packet of 0xb5 with no countermeasure and while applying DVFS to the attacking core

experiments). This technique periodically toggles the frequency level of the CPUs from the highest value to a random low value, and vice versa, to manipulate the power of the system and interfere with the attack. In our experiments, the high-value frequency is the maximum allowed frequency

for the boards (i.e., 2000 MHz for the TX2 and 1500 MHz for the Orin). As low frequencies, we employ the four lowest levels available in the boards (345 MHz, 500 MHz, 652 MHz, and 806 MHz for the TX2 and 115 MHz, 192 MHz, 268 MHz, and 345 MHz for the Orin). We employ a β value of 9, as used in [77]. In our setup, this means that while DVFS is applied, the affected cores execute at the high frequency for 0.25 ms and then at the lower frequency for 2.25 ms. To further visualize the effect of the DVFS on the malicious transmitter application, in Fig. 8.5 we show the power signal from the Jetson TX2 platform for the transmission of a packet of $0xb5$ without the countermeasure and while the DVFS countermeasure is active. As can be seen in the figure, the signal is significantly affected by changes in power. We properly evaluate the transmission error rates produced by the different countermeasure techniques in Section 8.7.3.

Furthermore, besides evaluating our system-information countermeasures, we develop two extra naive approaches and one semi-informed technique for comparison purposes. These approaches do not consider the system information explicitly but rather apply a fixed action.

The two naive techniques are **FC0** (*Fixed Core on Cluster 0*) and **FC1** (*Fixed Core on Cluster 1)*. In the FC0 approach, we always migrate the malicious application to the first core within the 4-processor cluster. In the FC1 technique, we move it to the first core within the 2-processor cluster. Additionally, in our evaluation we include an extra heuristic. This straightforward semi-informed heuristc, which we name *Worst-Performing Core* (**WPC**), finds the core with the lowest IPS value and assigns the attacking application to that core regardless of the cluster organization. For these three additional policies, when other applications are executing in the newly selected core for the malicious application, we exchange the applications' cores so that the policy is always followed in the same manner as our WPCBPC heuristic. After the migration happens, we apply DVFS to the affected cluster to mitigate the attack.

All the experiments that follow include the state-of-the-art DVFS approach [77], the naive techniques, and our system-informed approaches for comparison purposes.

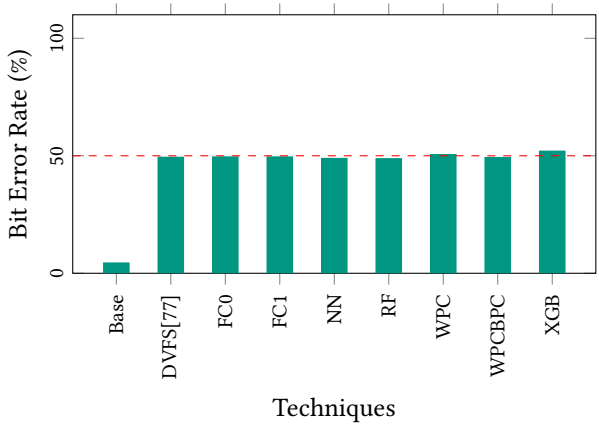


Figure 8.6: Bit error rate (BER) for the transmitter attack with no countermeasure applied (Base) and with each one of the evaluated countermeasure techniques on the Jetson TX2 platform.

8.7.3 Covert-channel Mitigation

In this first experiment, we evaluate the effectiveness of the different countermeasure techniques to mitigate the attack by affecting the transmission. To do so, we sent a total of 800 bits as 8-bit packets using the malicious transmitter. We implement a simple receiver that reads the power of the system and decodes the information being transmitted.

Figure 8.6 shows the results from this experiment. When no countermeasure is active (Base), the bit error rate (BER) from the transmission is very low (e.g., less than 5%). However, once the countermeasures are active, the BER increases drastically. Because of the transition to low frequencies in the DVFS, the power of the system tends to decrease, as seen in Fig. 8.5. This means that most of the bits of 1 would be interpreted as 0 while the bits of 0 are likely interpreted correctly. For a transmission with a balanced quantity of 1's and 0's, the expected error rate due to the countermeasure is then 50%. As seen in the figure, this is exactly the case for all of the techniques. Ultimately, this experiment shows that all the proposed countermeasures are effective for mitigating power-based covert channels.

8.7.4 Energy and Performance Penalty

In order to evaluate the energy and performance penalty of the different countermeasure techniques we devised an experiment where we generated 50 random workloads from the application set. After the workload generation, we simultaneously run the applications alongside the malicious transmitter application. The transmitter application executes at all times until the workload finishes. To replicate the behavior of a covert-channel detector, we wait for a period of 1 s after the workload is launched, before triggering the countermeasure technique. Then, we continue to apply the countermeasure until the full workload has finished the execution. This process is repeated for all the countermeasure techniques: the state-of-the-art DVFS approach (DVFS), Fixed Core on Cluster 0 (FC0), Fixed Core on Cluster 1 (FC1), Worst Performing Core (WPC), Worst Performing Cluster - Best Performing Core (WPCBPC), Neural Network (NN), Random Forest (RF), and XGBoost (XGB). Notably, to keep fairness, all techniques are evaluated with the same workload set. The orchestration of the experiment and corresponding monitoring is done by the resource management application, as described in Section 8.4. To reduce the effects of cached data in the experiments, we run all the workloads with one technique before moving to the next technique. The workloads are executed in the same order for all techniques. Additionally, we add delay of about 5 seconds between each workload to let the system return to a semi-idle state before a new execution.

Tables 8.2 and 8.3 show the averaged metrics obtained from the experiment for the baseline (no countermeasure applied) and all the different techniques on the two evaluation platforms. As a metric for performance, we report the average execution time of the whole workload from the moment we launch all the applications (done simultaneously) until the last application finishes its execution (i.e., makespan).

We measure the average power consumption of each run. Then, we compute the energy and Energy-delay Product (EDP), as a measurement of the efficiency of the system. Notably, since the resource management orchestration application executes in the system concurrently with the workload the overhead in the system due to techniques is already included as part of the obtained metrics.

From the tables, it is clear that all the countermeasures affect negatively the energy and performance of the system. It is important to notice that this

Table 8.2: Average results for the baseline and the different countermeasure approaches under 50 different workloads on the Jetson TX2 platform

Metric	Approach								
	Baseline	DVFS[77]	FC0	FC1	NN	RF	WPC	WPCBPC	XGB
Makespan (s)	210.12	358.63	407.7	284.81	280.47	301.26	282.51	269.38	268.03
Power (mW)	4999	4098	4009	4564	4609	4631	4489	4705	4759
Energy (J)	1050.48	1469.8	1634.64	1300.03	1292.63	1395.23	1268.32	1267.36	1275.63
EDP (Js)	220,727.79	527,116.38	666,442.94	370,260.59	362,544.76	420,326.52	358,313.13	341,403.66	341,909.21

effect is expected as it is the cost of mitigating the attack. Notably, the power in the system due to countermeasures is overall reduced, as it can also be seen in Fig. 8.7. From the normalized power, it would seem as if the state-of-the-art DVFS and FC0 are the best approaches. This again is a product of the frequency reduction due to the DVFS mechanism. However, as our further results show, from an energy and performance point of view the case is exactly the opposite. As our following results indicate, the power consumption of the system while the countermeasure is active is not an indication of the efficiency of the system, especially considering the performance penalty.

Table 8.3: Average results for the baseline and the different countermeasure approaches under 50 different workloads on the Jetson Orin Platform

Metric	Approach								
	Baseline	DVFS[77]	FC0	FC1	NN	RF	WPC	WPCBPC	XGB
Makespan (s)	183.36	296.83	418.89	234.44	239.8	191.26	291.76	184.92	213.22
Power (mW)	4960	4640	4368	4816	4918	5067	4723	5030	5080
Energy (J)	909.48	1377.2	1829.84	1129.08	1179.23	969.17	1377.87	930.22	1083.14
EDP (Js)	166,761.97	408,794.91	766,500.5	264,702.49	282,778.99	185,364.16	402,006.12	172,015.89	230,947.23

To better dissect and analyze the impact on the system's efficiency due to the countermeasures, we plot the performance and energy penalty for both platforms in Figs. 8.8 and 8.9. As can be seen, the state-of-the-art DVFS countermeasure has a high overhead of about 70% for the Jetson TX2 and about 62% for the Jetson Orin. This is a significant difference over the reported 25% for general purpose multi-core system [77]. This means that the performance penalty due to DVFS countermeasure is significantly higher on an embedded system. This is an interesting effect that has not been reported before this work.

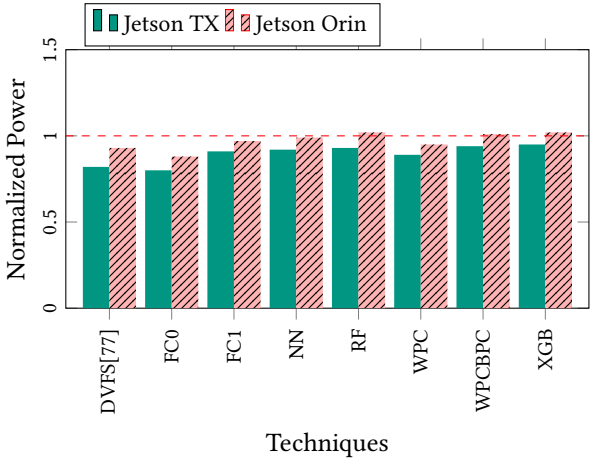


Figure 8.7: Normalized power in the system due to the different countermeasures on both evaluation platforms.

Moreover, the FC0 technique has the worst performance and energy penalty in both platforms. This outcome can be expected since this approach forces the DVFS to be applied to the bigger cluster, which consistently affects more cores (and applications) at all times when applying DVFS. On the other hand, the naive FC1 technique effectively reduces the performance and energy penalties when compared to the simple DVFS approach by affecting fewer cores.

More importantly, our system-informed approaches reduce the performance penalty by up to 40% and up to 60% for the Jetson TX2 and Orin respectively, when compared to the state-of-the-art technique. From an energy perspective, our system-informed techniques reduce the penalty due to the DVFS state-of-the-art countermeasure by about 20% in the Jetson TX2 and up to 50% in the Jetson Orin. Moreover, when combining the effect of both energy and performance in EDP form, as can be seen in Fig. 8.10, it is clear that system-informed approaches are generally more energy-efficient than the reference and their naive counterparts. At their best, these techniques managed to reduce the EDP penalty by up to 84% and 142% for the TX2 and Orin boards respectively when compared to the simple state-of-the-art DVFS technique. Our other ML-based techniques slightly reduced EDP compared to XGB on

the TX2 platform, while RF outperformed XGB on the Orin board. On the TX2, NN reduced the penalty by about 70% and RF by 48% compared to the state-of-the-art DVFS technique. Although RF showed better prediction accuracy at design time (Table 8.1), it was less efficient due to its irregular memory access, increasing energy demand. Conversely, NN's structured memory access conserved energy. This shows how policy execution can alter expected behavior. Thus, RF's accuracy advantage was diminished, leading to longer makespan and higher energy consumption compared to NN. Despite this, our system-informed approach still outperforms the state-of-the-art approach.

Interestingly, on the Jetson Orin board, RF outperforms NN as expected by the training. In fact, as Fig. 8.9 shows, RF and WPCBPC manage to produce at most 4% performance and 7% energy penalties in the system, which is a major improvement when compared to the state-of-the-art technique. In fact, this penalty is close to insignificant on this board, when compared to the the case when no countermeasure is applied. We believe several factors contribute to this outcome. First, the Jetson Orin board features a unified 4MB L3 cache, which reduces the effect of the intense and irregular memory accesses the RF techniques had on the TX2 platform, which lacks an L3 cache. Moreover, the Orin board's homogeneous, modern, and more powerful CPUs further enhance performance, helping to achieve the expected results.

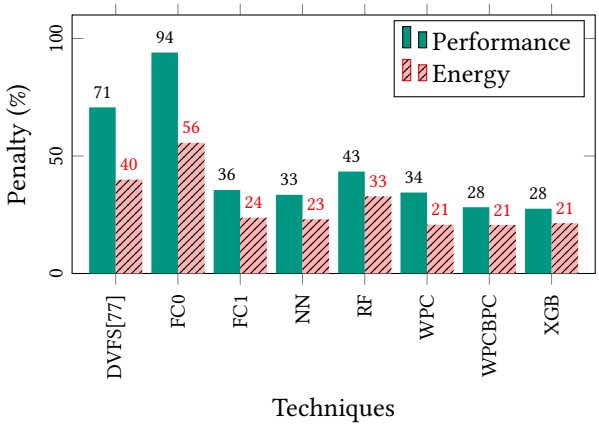


Figure 8.8: Performance and energy penalty over the baseline implementation in the system due to the different countermeasure techniques on the Jetson TX2 platform.

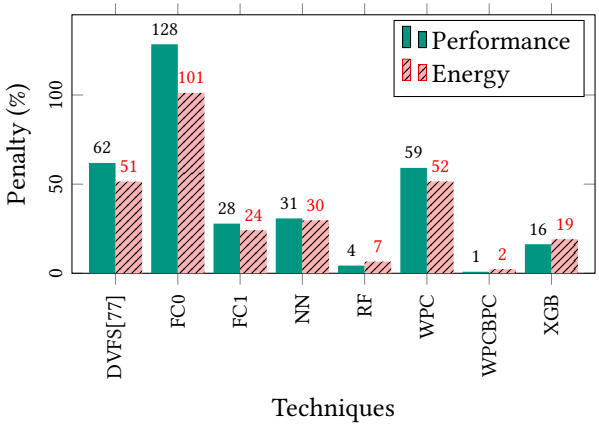


Figure 8.9: Performance and energy penalty over the baseline implementation in the system due to the different countermeasure techniques on the Jetson Orin platform.

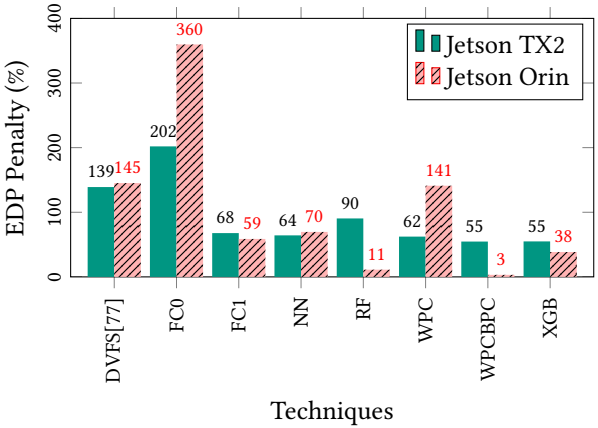


Figure 8.10: Energy-delay product (EDP) penalty in the system due to the different countermeasures on both evaluation platforms.

8.7.5 Generalization to unseen workloads

While our proposed system-informed heuristics are inherently application-agnostic (i.e., no application feature is considered in the migration logic), that

Table 8.4: Average results for the baseline, state-of-the-art, and system-informed countermeasures under 25 unseen workloads on the Jetson TX2 platform.

Metric	Approach						
	Baseline	**DVFS[77]**	**NN**	**RF**	**WPC**	**WPCBPC**	**XGB**
Makespan (s)	60.34	113.87	86.98	99.66	86.22	79.06	79.06
Power (mW)	4820	3707	4186	4091	4212	4250	4409
Energy (J)	290.83	422.08	364.09	407.75	363.13	336.02	348.61
EDP (Js)	17,548.89	48,062.73	31,668.36	40,636.06	31,309.16	26,566.06	27,560.82

might not necessarily be the case for the ML-based approaches. While we do not use features from the applications themselves as input to ML-models, since they are trained with execution traces from the SPEC2006 application set, it could be the case that the models are biased towards certain application behavior (e.g., memory or compute intensiveness).

In order to show the generality and the effectiveness of our ML-based techniques under a wider diversity of applications, we devised an experiment where we ran 25 completely new workloads, where each workload is fully comprised of apps never seen during training from the PARSEC 2 benchmark. In each one of these new workloads, all applications are selected randomly from the PARSEC 2 full application list. Additionally, we have ensured that each application from the set appears at least in one workload. The results from this test for both evaluation platforms can be seen in Tables 8.4 and 8.5.

For comparison purposes, we evaluate the baseline, the state-of-the-art approach and the system-informed countermeasure techniques. As shown for both platforms, even though new applications were unseen to the ML models during training, they still achieve a very good performance, which is consistent with our main experiments shown in Tables 8.2 and 8.3. Furthermore, as shown in Table 8.5, XGB has delivered the best performance out of the evaluated countermeasure techniques, surpassing even the WPCBPC heuristic in the Jetson Orin platform. This means that the ML-based techniques are not only able to successfully generalize to unseen applications, but can actually leverage the new workload characteristics to overperform the other approaches.

Table 8.5: Average results for the baseline, state-of-the-art, and system-informed countermeasures under 25 unseen workloads on the Jetson Orin platform.

Metric	Approach						
	Baseline	**DVFS[77]**	**NN**	**RF**	**WPC**	**WPCBPC**	**XGB**
Makespan (s)	69.64	125.48	119.2	108.98	117.55	99.7	97.15
Power (mW)	4550	4263	4323	4396	4354	4381	4447
Energy (J)	316.87	534.95	515.26	479.06	511.8	436.78	431.99
EDP (Js)	22,066.8	67,125.85	61,419.26	52,207.57	60,162.34	43,546.54	41,967.37

8.7.6 Runtime Overhead Analysis

As mentioned in Section 8.7.4, the overhead that each technique induces in the system from a performance and energy point of view is already included in the final result depicted in Table 8.2, as the resource management orchestration application runs in the system alongside the workload for all the experiments. Moreover, the actual cost of task migration in the applications themselves is also included already in the reported metrics.

Nonetheless, in this section, we provide a more detailed analysis on the part the overhead produced in the system by each of the system-informed techniques. We omit the overhead of the naive approaches (i.e., DVFS, FC0, and FC1) as no processing is needed in the selection of new mapping to be enforced by task migration. Table 8.6 shows the overhead of the system-informed techniques. As can be seen, the overhead due to the heuristics is significantly lower than the ML-based approaches since the computation needed to select the cluster and core needed for the migration is rather simple for both WPC and WPCBPC, and it only needs to be executed once at each acting epoch (of 1 second). The ML-based approaches, on the other hand, are called to predict the efficiency for each possible non-redundant mapping confirmation. For the configuration of our evaluation platforms, this represents a maximum of 15 non-redundant mapping configurations to be evaluated. The number reported in Table 8.6 is the accumulated overhead of the ML-base techniques for all calls. This means that in the worst case, the overhead of the techniques is rather small at about 128 ms. As a final remark, it should be noted that even though the heuristic approaches have much less overhead than the ML techniques, XGB is able to surpass the heuristics in terms of performance for the workloads as seen in Tables 8.2 and 8.5. In other words, the overhead difference between both approaches is balanced by the

improvement the XGB technique produces in the workload, which is in the end the relevant metric to compare both approaches on this platform.

Table 8.6: Overhead of the different system-informed techniques on the both evaluation platforms.

Platform	Overhead (ms)				
	NN	RF	WPC	WPCBPC	XGB
Jetson TX2	128.64	30.37	0.02	0.12	49.09
Jetson Orin	78.57	113.91	0.02	0.02	19.52

8.7.7 Machine learning vs. heuristics

As our experimental evaluation has shown, our system-informed approaches are effective at mitigating the attack while reducing the energy and performance penalty on the system.

While presenting techniques from both machine learning and heuristics domains, our intention in this chapter is not to indicate one *best* technique between the different approaches. On the contrary, as our results show, both approaches exhibit quite similar performance (the difference in EDP penalty in our main experiment between WPCBPC and XGB is less than 0.2%). We seek to show how both traditional and ML-based policies can effectively serve the purpose of an efficient countermeasure. Both approaches have advantages and disadvantages when used for this purpose. Both the WPC and our WPCBPC heuristics have low complexity and are very fast, as depicted in Table 8.6. These heuristics focus on optimizing performance, by reducing the negative effect of the DVFS mechanism. However, by only using IPS this approach does not consider the efficiency of the full system due to the current execution scenario. When dealing with diverse workloads, specially in a potentially more complex system (e.g., many-core), this information might not be sufficient to produce optimal results. The ML-based approaches, on the other hand, have a greater overhead when compared to the heuristics, but as just discussed in Section 8.7.6 they compensate for this overhead by producing efficient execution scenarios. Moreover, the ML-based approach utilizes execution features to learn the behavior of the system, even hidden or non-measurable parameters. This means that with enough training, the

approaches can be extended and adapted to perform well under diverse execution scenarios. Indeed, as we have demonstrated exactly this in Section 8.7.5, where the ML techniques were successfully able to generalize correctly to the new application set. Moreover, under this new execution scenario, the XGB model managed to outperform the best heuristic for the Jetson Orin board, showing the potential advantage of the ML approach vs the implemented heuristics.

By providing countermeasures from both heuristics and machine learning domains we presented two successful avenues to the problem of mitigating power-based covert channels in an efficient manner, Regardless of their domain, our system-informed techniques were able to defeat the state-of-the-art countermeasure, proving to be the better solution to the problem.

8.8 Summary

In this chapter, we have highlighted the performance and energy impact of traditional DVFS-based countermeasures to power-based covert channels on embedded systems. We have shown how the state-of-the-art DVFS method can produce up to 70% performance penalty on an embedded platform when the attack is present at all times, which differs greatly from the reported penalty for general-purpose multi-/many-core systems. Moreover, we have proposed different techniques from the heuristic and machine learning domain that, for the first time, combine dynamic task migration and DVFS to mitigate such attacks in an efficient and system-informed manner, significantly reducing both energy and performance penalties. From our experimentation on the commercial NVIDIA Jetson TX2 and Jetson Orin embedded platforms, we were able to successfully reduce the EDP penalty due to the state-of-the-art DVFS-only countermeasure by more than 84% and 142% respectively, proving that our system-informed techniques are a better approach to power-based covert-channel mitigation.

9 Conclusion

In this dissertation, a series of novel contributions aimed at disclosing and mitigating advanced threats in emerging computing systems were presented, with a particular focus on cross-layer security solutions against side- and covert-channel attacks. By exploring both heuristic and machine learning (ML) approaches, it was demonstrated how system-informed methodologies can effectively enhance security while maintaining performance and energy efficiency.

New covert-channel attack vectors were introduced in Chapter 4, where weaknesses in existing security models, particularly in embedded and heterogeneous computing environments, were identified. It was shown that these advanced attacks can compromise system integrity without being detected by traditional methods, highlighting the need for new defensive strategies.

In Chapters 5 and 6, ML-based detection mechanisms were introduced. In Chapter 5, a supervised learning framework was developed, utilizing CPU time-series performance data to predict covert-channel attacks. Chapter 6 explored an unsupervised learning approach within the context of Remote Attestation (RA) to detect anomalies in system execution. Both approaches were validated experimentally, showing significant improvements in threat detection accuracy while maintaining minimal system overhead.

The defensive mechanisms proposed in this dissertation (Chapters 7 and 8) revolved around two primary countermeasure strategies: task migration and DVFS. These techniques were evaluated on both simulation and real-world platforms, demonstrating their effectiveness. In Chapter 7, task migration techniques successfully mitigated cache side-channel attacks by preventing cluster co-residency between attacker and victim processes in simulated environments. Meanwhile, the combination of task migration and DVFS strategies in Chapter 8 effectively countered power-based covert channels by dynamically mapping applications to clusters and modulating the CPU frequency to disrupt covert communications on real hardware.

Overall, this research demonstrated two key findings. First, it highlighted how advanced threats can exploit computing resources in increasingly sophisticated ways, posing significant security risks. Second, it showed that cross-layer security solutions, when informed by system context and enhanced with machine learning techniques, are highly effective in detecting and mitigating these threats in modern computing systems.

9.1 Future Work

While this dissertation provides a strong foundation for addressing new threats in emerging computing systems from a cross-layer perspective, several avenues for future research can stem from the presented work.

Enhanced Detection Techniques and New Attack Vectors The detection methods described in Chapters 5 and 6, while robust, may need to be adapted to different computing environments and emerging threats targeting those environments. Specifically, integrating more advanced ML techniques such as reinforced and federated learning, where decentralized models may enable larger systems to detect emerging threats more rapidly and efficiently. These techniques would allow distributed systems to share knowledge of potential attacks without compromising the confidentiality of individual nodes.

Furthermore, new computational paradigms are likely to introduce additional attack vectors. Future research should focus on developing cross-layer security solutions tailored to these environments. For instance, attacks targeting the data privacy of federated learning in edge computing systems should be mitigated through decentralized, lightweight security measures.

Optimization of the Security-Performance Trade-off One of the key challenges highlighted through this dissertation is the balance between security and performance, especially with respect to countermeasure implementation. Further work is needed to refine this trade-off, particularly in systems with stringent energy and latency constraints. Novel optimization algorithms that dynamically adjust security mechanisms in real time could be explored to enhance both security and performance.

Similarly, while the experimental results in this dissertation were performed on different computing platforms, further research should aim to generalize these approaches to even a wider range of platforms, especially those with hard resource constraints. In such systems, restricted access to runtime system information (e.g., performance counters) might limit the capabilities of the defensive strategies.

Additionally, the application of these techniques in cloud and multi-tenant environments should be explored to further validate their scalability and effectiveness.

Cross-layer Integration with Emerging Technologies As highlighted in this dissertation, the intersection of security with emerging technologies such as AI accelerators, blockchain, and 5G/6G networks offers rich opportunities for future exploration. Research into how cross-layer security techniques can be integrated into these emerging domains will be critical for ensuring their resilience against advanced threats.

List of Abbreviations

CA Client Application

CFA Control-Flow Attestation

CFG Control-Flow Graph

DFT Discrete Fourier Transform

DVFS Dynamic Voltage and Frequency Scaling

ECC Error Correction Code

EDP Energy-delay Product

FPGA Field Programmable Gate Array

GPU Graphics Processing Unit

IFO Isolation Forest

IoT Internet of Things

IPC Instructions per Cycle

IPS Instructions per Second

LLC Last-level Cache

LOF Local Outlier Factor

MAE Mean-absolute Error

MI Mutual Information

ML Machine Learning

MPSoC Multi Processor System on Chip

NN Neural Network

NVM Non-Volatile Memory

OSVM One-class Support Vector Machine

PMU Performance Monitoring Unit

RA Remote Attestation

REE Rich Execution Environment

RMSE Root-mean-square Error

RoA Region of Attestation

SCA Side-channel Attack

SoC System on Chip

TA Trusted Application

TCC Thermal Covert Channel

TEE Trusted Execution Environment

V/f Voltage/frequency

VM Virtual Machine

XGBoost eXtreme Gradient Boosting

Bibliography

[1] Jeferson Gonzalez-Gomez, Mohammed Bakr Sikal, Heba Khdr, Lars Bauer, and Jörg Henkel. "Smart Detection of Obfuscated Thermal Covert Channel Attacks in Many-core Processors". In: *2023 60th ACM/IEEE Design Automation Conference (DAC)*. 2023, pp. 1–6. DOI: 10.1109/DAC56929.2023.10247844.

[2] Jeferson Gonzalez-Gomez, Kevin Cordero-Zuñiga, Lars Bauer, and Jörg Henkel. "The First Concept and Real-world Deployment of a GPU-based Thermal Covert Channel: Attack and Countermeasures". In: *2023 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. 2023, pp. 1–6. DOI: 10.23919/DATE56975.2023.10137090.

[3] Jeferson Gonzalez-Gomez, Hassan Nassar, Varun Manjunath, Lars Bauer, and Jörg Henkel. "Through Fabric: A Cross-world Thermal Covert Channel on TEE-enhanced FPGA-MPSoC Systems". In: *Asia and South Pacific Design Automation Conference (ASP-DAC)*. accepted to appear. 2025.

[4] Jeferson Gonzalez-Gomez, Jose Alejandro Ibarra-Campos, Jesus Yamir Sandoval-Morales, Lars Bauer, and Jörg Henkel. *MeMoir: A Software-Driven Covert Channel based on Memory Usage*. 2024. arXiv: 2409.13310 [cs.CR]. URL: https://arxiv.org/abs/2409.13310.

[5] Jeferson Gonzalez-Gomez, Hassan Nassar, Lars Bauer, and Jörg Henkel. "LightFAt: Mitigating Control-Flow Explosion via Lightweight PMU-Based Control-Flow Attestation". In: *2024 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*. 2024, pp. 222–226. DOI: 10.1109/HOST55342.2024.10545348.

[6] Jeferson Gonzalez-Gomez, Lars Bauer, and Jörg Henkel. "Cache-Based Side-Channel Attack Mitigation for Many-Core Distributed Systems via Dynamic Task Migration". In: *IEEE Transactions on Information Forensics and Security* 18 (2023), pp. 2440–2450. DOI: 10.1109/TIFS.2023.3266630.

[7] Jeferson González-Gómez, Mohammed Bakr Sikal, Heba Khdr, Lars Bauer, and Jörg Henkel. "Balancing Security and Efficiency: System-Informed Mitigation of Power-Based Covert Channels". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 43.11 (2024), pp. 3395–3406. DOI: 10.1109/TCAD.2024.3438999.

[8] Lars Bauer, Jörg Henkel, Timo Hönig, Wolfgang Schröder-Preikschat, Christian Eichler, Jeferson Gonzalez, Benedict Herzog, Tobias Langer, Sebastian Maier, Jonas Rabenstein, et al. "Invasive Run-Time Support System (iRTSS)". In: *Invasive Computing*. Ed. by Jürgen Teich, Jörg Henkel, and Andreas Herkersdorf. Cham: Springer International Publishing, 2022, pp. 285–305. ISBN: 978-3-96147-571-1. DOI: 10.25593/978-3-96147-571-1.

[9] Antonio González-Torres, Mónica Hernández, Jeferson González, Vetria L. Byrd, and Paul Parsons. "Information Visualization as a Method for Cybersecurity Education". In: *Innovations in Cybersecurity Education*. Ed. by Kevin Daimi and Guillermo Francia III. Cham: Springer International Publishing, 2020, pp. 55–70. ISBN: 978-3-030-50244-7. DOI: 10.1007/978-3-030-50244-7_4.

[10] Jeferson González-Gómez, Steven Ávila, Jonathan Rojas, Andres Stephen, Jorge Castro-Godínez, Carlos Salazar-García, Muhammad Shafique, and Jörg Henkel. "TailoredCore: Generating Application-Specific RISC-V-based Cores". In: *2021 IEEE 12th Latin America Symposium on Circuits and System (LASCAS)*. 2021, pp. 1–4. DOI: 10.1109/LASCAS51355.2021.9459152.

[11] Carlos Salazar-García, Jeferson González-Gómez, Kaleb Alfaro-Badilla, Ronny García-Ramírez, Renato Rímolo-Donadío, Christos Strydis, and Alfonso Chacón-Rodríguez. "PlasticNet: A low latency flexible network architecture for interconnected multi-FPGA systems". In: *2020 IEEE 3rd Conference on PhD Research in Microelectronics and Electronics in Latin America (PRIME-LA)*. 2020, pp. 1–4. DOI: 10.1109/PRIME-LA47693.2020.9062749.

[12] Carlos Salazar-García, Alfonso Chacón-Rodríguez, Renato Rímolo-Donadío, Ronny García-Ramírez, David Solórzano-Pacheco, Jeferson González-Gómez, and Christos Strydis. "A custom interconnection multi-FPGA framework for distributed processing applications". In: *2022 35th SBC/SBMicro/IEEE/ACM Symposium on Integrated Circuits*

and Systems Design (SBCCI). 2022, pp. 1–6. DOI: 10.1109/SBCCI55532.2022.9893238.

[13] Tigist Abera, N. Asokan, Lucas Davi, Jan-Erik Ekberg, Thomas Nyman, Andrew Paverd, Ahmad-Reza Sadeghi, and Gene Tsudik. "C-FLAT: Control-Flow Attestation for Embedded Systems Software". In: *Conf. on Computer and Communications Security*. 2016.

[14] Mohammad Abdullah Al Faruque, Thomas Ebi, and Jorg Henkel. "Configurable links for runtime adaptive on-chip communication". In: *2009 Design, Automation & Test in Europe Conference & Exhibition*. 2009, pp. 256–261. DOI: 10.1109/DATE.2009.5090667.

[15] Toqeer Ali, Roslan Ismail, Shahrulniza Musa, Mohammad Nauman, and Sohail Khan. "Design and implementation of an attestation protocol for measured dynamic behavior". In: *The Journal of Supercomputing* 74.11 (2018).

[16] *AMD "Zen 3" Core Architecture*. 2020. URL: https://www.amd.com/en/technologies/zen-core-3.

[17] Mahmoud Ammar and Bruno Crispo. "WISE: A Lightweight Intelligent Swarm Attestation Scheme for the Internet of Things". In: *ACM Trans. Internet Things* 1.3 (2020).

[18] ARM. *TrustZone for Cortex-A*. 04.09.23. 2023. URL: https://www.arm.com/technologies/trustzone-for-cortex-a.

[19] N. Asokan, Ferdinand Brasser, Ahmad Ibrahim, Ahmad Reza Sadeghi, Matthias Schunter, Gene Tsudik, and Christian Wachsmann. "SEDA: Scalable embedded device attestation". In: *Conf. on Computer and Communications Security*. 2015.

[20] Mohamed Azab and Mohamed Eltoweissy. "MIGRATE: Towards a Lightweight Moving-Target Defense Against Cloud Side-Channels". In: *2016 Security and Privacy Workshops (SPW)*. 2016, pp. 96–103. DOI: 10.1109/SPW.2016.28.

[21] Sahan Bandara and Michel A. Kinsy. "Adaptive Caches as a Defense Mechanism Against Cache Side-Channel Attacks". In: *Works. on Attacks and Solut. in Hardware Security Workshop*. 2019, pp. 55–64.

[22] Davide B. Bartolini, Philipp Miedl, and Lothar Thiele. "On the Capacity of Thermal Covert Channels in Multicores". In: *European Conference on Computer Systems (EuroSys)*. 2016. ISBN: 9781450342407.

[23] Lars Bauer, Muhammad Shafique, and Jörg Henkel. "Run-time instruction set selection in a transmutable embedded processor". In: *Proceedings of the 45th Annual Design Automation Conference*. DAC '08. Anaheim, California: Association for Computing Machinery, 2008, pp. 56–61. ISBN: 9781605581156. DOI: 10.1145/1391469.1391486. URL: https://doi.org/10.1145/1391469.1391486.

[24] Lars Bauer, Muhammad Shafique, and Jörg Henkel. "Run-time instruction set selection in a transmutable embedded processor". In: *Proceedings of the 45th Annual Design Automation Conference*. DAC '08. Anaheim, California: Association for Computing Machinery, 2008, pp. 56–61. ISBN: 9781605581156. DOI: 10.1145/1391469.1391486. URL: https://doi.org/10.1145/1391469.1391486.

[25] Omar Bawazeer, Tarek Helmy, and Suheer Al-hadhrami. "Malware Detection Using Machine Learning Algorithms Based on Hardware Performance Counters: Analysis and Simulation". In: *Journal of Physics: Conference Series* 1962.1 (July 2021), p. 012010. DOI: 10.1088/1742-6596/1962/1/012010.

[26] Christian Bienia and Kai Li. "Parsec 2.0: A new benchmark suite for chip-multiprocessors". In: *Proceedings of the 5th Annual Workshop on Modeling, Benchmarking and Simulation*. Vol. 2011. 2009, p. 37.

[27] Lilian Bossuet and Carlos Andres Lara-Nino. "Advanced Covert-Channels in Modern SoCs". In: *HOST*. 2023.

[28] Kenneth Brezinski and Ken Ferens. "Metamorphic Malware and Obfuscation: A Survey of Techniques, Variants, and Generation Kits". In: *Security and Communication Networks* 2023.1 (2023), p. 8227751. DOI: https://doi.org/10.1155/2023/8227751.

[29] Samira Briongos, Pedro Malagón, José L. Risco-Martín, and José M. Moya. "Modeling Side-Channel Cache Attacks on AES". In: *Summer Computer Simulation Conference*. 2016.

[30] Trevor E. Carlson, Wim Heirman, and Lieven Eeckhout. "Sniper: Exploring the Level of Abstraction for Scalable and Accurate Parallel Multi-Core Simulations". In: *International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*. Nov. 2011, 52:1–52:12.

[31] Xavier Carpent, Karim El Defrawy, Norrathep Rattanavipanon, and Gene Tsudik. "Lightweight swarm attestation: A tale of two LISA-s". In: *Asia Conference on Computer and Communications Security*. 2017.

[32] Shuai Che, Michael Boyer, Jiayuan Meng, David Tarjan, Jeremy W. Sheaffer, Sang-Ha Lee, and Kevin Skadron. "Rodinia: A benchmark suite for heterogeneous computing". In: *International Symp. on Workload Characterization (IISWC)*. 2009.

[33] Huili Chen, Cheng Fu, Bita Darvish Rouhani, Jishen Zhao, and Farinaz Koushanfar. "DeepAttest: An End-to-End Attestation Framework for Deep Neural Networks". In: *International Symposium on Computer Architecture (ISCA)*. IEEE, 2019.

[34] Ji Ming Chen, Shi Chen, Xiang Wang, Lin Lin, and Li Wang. "A Virtual Machine Migration Strategy Based on the Relevance of Services against Side-Channel Attacks". In: *Security and Communication Networks* 2021 (2021). ISSN: 19390122. DOI: 10.1155/2021/2729949.

[35] S. Chen, W. Xiong, Y. Xu, B. Li, and J. Szefer. "Thermal Covert Channels Leveraging Package-on-Package DRAM". In: *IEEE Int. Conf. on Trust, Security and Privacy In Computing and Com./IEEE Int. Conf. on Big Data Science and Eng. (TrustCom/BigDataSE)*. 2019, pp. 319–326.

[36] Shuai Chen, Wenjie Xiong, Yehan Xu, Bing Li, and Jakub Szefer. "Thermal Covert Channels Leveraging Package-on-Package DRAM". In: *2019 18th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/13th IEEE International Conference On Big Data Science And Engineering (TrustCom/BigDataSE)*. 2019, pp. 319–326. DOI: 10.1109/TrustCom/BigDataSE.2019.00050.

[37] Marco Chiappetta, Erkay Savas, and Cemal Yilmaz. "Real Time Detection of Cache-Based Side-Channel Attacks Using Hardware Performance Counters". In: *Appl. Soft Comput.* 49.C (Dec. 2016), pp. 1162–1174.

[38] Haehyun Cho, Penghui Zhang, Donguk Kim, Jinbum Park, Choong-Hoon Lee, Ziming Zhao, Adam Doupé, and Gail-Joon Ahn . "Prime + Count: Novel Cross-World Covert Channels on ARM TrustZone". In: *Annual Computer Security Applications Conference*. 2018.

[39] Stephen Crane, Andrei Homescu, Stefan Brunthaler, Per Larsen, and Michael Franz. "Thwarting cache side-channel attacks through dynamic software diversity." In: *NDSS*. 2015, pp. 8–11.

[40] Karim El Defrawy, Aurélien Francillon, Daniele Perito, and Gene Tsudik. "SMART: Secure and Minimal Architecture for (Establishing a Dynamic) Root of Trust". In: *Network & Distributed System Security Symposium*. 2012.

[41] Luca Demetrio, Scott E. Coull, Battista Biggio, Giovanni Lagorio, Alessandro Armando, and Fabio Roli. "Adversarial EXEmples: A Survey and Experimental Evaluation of Practical Attacks on Machine Learning for Windows Malware Detection". In: *ACM Trans. Priv. Secur.* 24.4 (Sept. 2021). ISSN: 2471-2566. DOI: 10.1145/3473039. URL: https://doi.org/10.1145/3473039.

[42] Ghada Dessouky, Tigist Abera, Ahmad Ibrahim, and Ahmad-Reza Sadeghi. "LiteHAX: Lightweight Hardware-Assisted Attestation of Program Execution". In: *Int. Conf. on Computer-Aided Design*. 2018.

[43] Ghada Dessouky, Shaza Zeitouni, Thomas Nyman, Andrew Paverd, Lucas Davi, Patrick Koeberl, N. Asokan, and Ahmad-Reza Sadeghi. "LO-FAT: Low-Overhead control Flow ATtestation in hardware". In: *Design Automation Conf.* 2017.

[44] *Developer Guide*. 6-9.3. Red Hat Enterprise Linux. 2017.

[45] Somdip Dey, Amit Kumar Singh, and Klaus McDonald-Maier. "ThermalAttackNet: Are CNNs Making It Easy to Perform Temperature Side-Channel Attack in Mobile Edge Devices?" In: *Future Internet* 13.6 (2021). ISSN: 1999-5903.

[46] Krithika Dhananjay, Vasilis F Pavlidis, Ayse K Coskun, and Emre Salman. "High bandwidth thermal covert channel in 3-d-integrated multicore processors". In: *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 30.11 (2022), pp. 1654–1667.

[47] Thomas Ebi, Mohammad Abdullah Al Faruque, and Jörg Henkel. "TAPE: thermal-aware agent-based power economy for multi/many-core architectures". In: *Proceedings of the 2009 International Conference on Computer-Aided Design*. ICCAD '09. San Jose, California: Association for Computing Machinery, 2009, pp. 302–309. ISBN: 9781605588001. DOI: 10.1145/1687399.1687457. URL: https://doi.org/10.1145/1687399.1687457.

[48] Thomas Ebi, David Kramer, Wolfgang Karl, and Jörg Henkel. "Economic learning for thermal-aware power budgeting in many-core architectures". In: *Proceedings of the Seventh IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis*. CODES+ISSS '11. Taipei, Taiwan: Association for Computing Machinery, 2011, pp. 189–196. ISBN: 9781450307154. DOI: 10.1145/2039370.2039401.

[49] Dmitry Efanov and Pavel Roschin. “The Port-in-Use Covert Channel Attack”. In: *Biologically Inspired Cognitive Architectures (BICA) for Young Scientists*. Ed. by Alexei V. Samsonovich and Valentin V. Klimov. Cham: Springer International Publishing, 2018, pp. 239–244. ISBN: 978-3-319-63940-6.

[50] R. Ernst, J. Henkel, and T. Benner. “Hardware-software cosynthesis for microcontrollers”. In: *IEEE Design & Test of Computers* 10.4 (1993), pp. 64–75. DOI: 10.1109/54.245964.

[51] Mark Evers, Leslie Barnes, and Mike Clark. “Next Generation “Zen 3” Core”. In: *2021 IEEE Hot Chips 33 Symposium (HCS)*. 2021, pp. 1–32. DOI: 10.1109/HCS52781.2021.9567108.

[52] Mohammad Abdullah Al Faruque, Thomas Ebi, and Jorg Henkel. “Run-time adaptive on-chip communication scheme”. In: *2007 IEEE/ACM International Conference on Computer-Aided Design*. 2007, pp. 26–31. DOI: 10.1109/ICCAD.2007.4397239.

[53] Anis Fellah-Touta, Lilian Bossuet, and Carlos Andres Lara-Nino. “Combined Internal Attacks on SoC-FPGAs: Breaking AES with Remote Power Analysis and Frequency-based Covert Channels”. In: *EuroS&PW*. 2023.

[54] Raspberry Pi Foundation. *Raspberry Pi 4 Model B Specifications*. Accessed: 2024-09-11. 2023. URL: https://www.raspberrypi.com/products/raspberry-pi-4-model-b/specifications/.

[55] Franklin, Dustin. *NVIDIA Jetson TX2 Delivers Twice the Intelligence to the Edge*. https://developer.nvidia.com/blog/jetson-tx2-delivers-twice-intelligence-edge/. Online; accessed 26 March 2024. 2017.

[56] Ilias Giechaskiel, Ken Eguro, and Kasper B. Rasmussen. “Leakier Wires: Exploiting FPGA Long Wires for Covert- and Side-Channel Attacks”. In: *ACM TRETS* (2019).

[57] Ilias Giechaskiel, Kasper Rasmussen, and Jakub Szefer. “Reading Between the Dies: Cross-SLR Covert Channels on Multi-Tenant Cloud FPGAs”. In: *ICCD*. 2019.

[58] Ilias Giechaskiel, Kasper Bonne Rasmussen, and Jakub Szefer . “C^3APSULe: Cross-FPGA covert-channel attacks through power supply unit leakage”. In: *2020 IEEE Symposium on Security and Privacy (SP)*. IEEE. 2020, pp. 1728–1741.

[59] Ilias Giechaskiel, Shanquan Tian, and Jakub Szefer. "Cross-VM Covert- and Side-Channel Attacks in Cloud FPGAs". In: *ACM TRETS* (2022).

[60] Dennis R. E. Gnad, Cong Dang Khoa Nguyen, Syed Hashim Gillani, and Mehdi B. Tahoori. "Voltage-Based Covert Channels Using FPGAs". In: *ACM TODAES* (2021).

[61] Michael Godfrey and Mohammad Zulkernine. "Preventing cache-based side-channel attacks in a cloud environment". In: *Transactions on Cloud Computing* 2.4 (2014), pp. 395–408.

[62] Mathieu Gross, Robert Kunzelmann, and Georg Sigl. *CPU to FPGA Power Covert Channel in FPGA-SoCs*. Cryptology ePrint Archive, Paper 2023/429. `https://eprint.iacr.org/2023/429`. 2023. URL: `https://eprint.iacr.org/2023/429`.

[63] Mathieu Gross, Robert Kunzelmann, and Georg Sigl. *CPU to FPGA Power Covert Channel in FPGA-SoCs*. Cryptology ePrint Archive, Paper 2023/429. 2023.

[64] Daniel Gruss, Erik Kraft, Trishita Tiwari, Michael Schwarz, Ari Trachtenberg, Jason Hennessey, Alex Ionescu, and Anders Fogh. "Page Cache Attacks". In: *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*. CCS '19. London, United Kingdom: Association for Computing Machinery, 2019, pp. 167–180. ISBN: 9781450367479. DOI: `10.1145/3319535.3339809`. URL: `https://doi.org/10.1145/3319535.3339809`.

[65] Jawad Haj-Yahya, Lois Orosa, Jeremie S. Kim, Juan Gómez Luna, A. Giray Yağlıkçı, Mohammed Alser, Ivan Puddu, and Onur Mutlu. "IChannels: Exploiting Current Management Mechanisms to Create Covert Channels in Modern Processors". In: *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*. 2021, pp. 985–998. DOI: `10.1109/ISCA52012.2021.00081`.

[66] R. W. Hamming. "Error detecting and error correcting codes". In: *The Bell System Technical Journal* 29.2 (1950), pp. 147–160. DOI: `10.1002/j.1538-7305.1950.tb00463.x`.

[67] Zecheng He and Ruby B. Lee. "How Secure is Your Cache against Side-Channel Attacks?" In: *Int. Symp. on Microarch.* 2017, pp. 341–353.

[68] J. Henkel. “A low power hardware/software partitioning approach for core-based embedded systems”. In: *Proceedings 1999 Design Automation Conference (Cat. No. 99CH36361)*. 1999, pp. 122–127. DOI: 10.1109/DAC.1999.781296.

[69] J. Henkel and Yanbing Li. “Energy-conscious HW/SW-partitioning of embedded systems: a case study on an MPEG-2 encoder”. In: *Proceedings of the Sixth International Workshop on Hardware/Software Codesign. (CODES/CASHE'98)*. 1998, pp. 23–27. DOI: 10.1109/HSC.1998.666233.

[70] J. Henkel, W. Wolf, and S. Chakradhar. “On-chip networks: a scalable, communication-centric embedded system design paradigm”. In: *17th International Conference on VLSI Design. Proceedings*. 2004, pp. 845–851. DOI: 10.1109/ICVD.2004.1261037.

[71] Jorg Henkel et al. “Special Session - Non-Volatile Memories: Challenges and Opportunities for Embedded System Architectures with Focus on Machine Learning Applications”. In: *Proceedings of the International Conference on Compilers, Architecture, and Synthesis for Embedded Systems*. CASES '23 Companion. Hamburg, Germany: Association for Computing Machinery, 2024, pp. 11–20. ISBN: 9798400702907. DOI: 10.1145/3607889.3609088.

[72] Jörg Henkel, Andreas Herkersdorf, Lars Bauer, Thomas Wild, Michael Hübner, Ravi Kumar Pujari, Artjom Grudnitsky, Jan Heisswolf, Aurang Zaib, Benjamin Vogel, Vahid Lari, and Sebastian Kobbe. “Invasive manycore architectures”. In: *17th Asia and South Pacific Design Automation Conference*. 2012, pp. 193–200. DOI: 10.1109/ASPDAC.2012.6164944.

[73] John L. Hennessy and David A. Patterson. “A New Golden Age for Computer Architecture”. In: *Commun. ACM* 62.2 (Jan. 2019), pp. 48–60.

[74] John L Henning. “SPEC CPU2006 benchmark descriptions”. In: *ACM SIGARCH Computer Architecture News* 34.4 (2006), pp. 1–17.

[75] D. Herrmann, J. Henkel, and R. Ernst. “An approach to the adaptation of estimated cost parameters in the COSYMA system”. In: *Third International Workshop on Hardware/Software Codesign*. 1994, pp. 100–107. DOI: 10.1109/HSC.1994.336718.

[76] Stefan Hristozov, Moritz Wettermann, and Manuel Huber. "A TOCTOU Attack on DICE Attestation". In: *Proceedings of the Twelfth ACM Conference on Data and Application Security and Privacy*. CODASPY '22. Baltimore, MD, USA: Association for Computing Machinery, 2022, pp. 226–235. ISBN: 9781450392204. DOI: 10.1145/3508398.3511507.

[77] Hengli Huang, Xiaohang Wang, Yingtao Jiang, Amit Kumar Singh, Mei Yang, and Letian Huang. "Detection of and Countermeasure Against Thermal Covert Channel in Many-Core Systems". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 41.2 (2022), pp. 252–265.

[78] Hengli Huang, Xiaohang Wang, Yingtao Jiang, Amit Kumar Singh, Mei Yang, and Letian Huang. "On Countermeasures against the Thermal Covert Channel Attacks Targeting Many-Core Systems". In: *Design Automation Conference (DAC)*. 2020. ISBN: 9781450367257.

[79] Wei Huang, Shougata Ghosh, Sivakumar Velusamy, Karthik Sankaranarayanan, Kevin Skadron, and Mircea R Stan. "HotSpot: A Compact Thermal Modeling Methodology for Early-Stage VLSI Design". In: *IEEE Trans. Very Large Scale Integration (VLSI) Systems* 14.5 (2006), pp. 501–513. DOI: 10.1109/TVLSI.2006.876103.

[80] Ralf Hund, Carsten Willems, and Thorsten Holz. "Practical Timing Side Channel Attacks against Kernel Space ASLR". In: *Symposium on Security and Privacy*. 2013, pp. 191–205.

[81] Tyler Hunt, Zhipeng Jia, Vance Miller, Ariel Szekely, Yige Hu, Christopher J Rossbach, and Emmett Witchel. "Telekine: Secure Computing with Cloud GPUs". In: *USENIX Symp. on Networked Systems Design and Implementation (NSDI)*. 2020.

[82] Dongdong Huo, Yu Wang, Chao Liu, Mingxuan Li, Yazhe Wang, and Zhen Xu. "LAPE: A Lightweight Attestation of Program Execution Scheme for Bare-Metal Systems". In: *(HPCC/SmartCity/DSS)*. 2020.

[83] Taras Iakymchuk, Maciej Nikodem, and Krzysztof Kepa. "Temperature-based covert channel in FPGA systems". In: *ReCoSoC*. 2011.

[84] Ahmad Ibrahim, Ahmad Reza Sadeghi, and Gene Tsudik. "HEALED: HEaling & Attestation for Low-End Embedded Devices". In: *Lecture Notes in Computer Science* (2019).

[85] Ahmad Ibrahim, Ahmad Reza Sadeghi, Shaza Zeitouni, and Gene Tsudik. "DARPA: Device attestation resilient to physical attacks". In: *Conf. on Security and Privacy in Wireless and Mobile Networks*. 2016.

[86] Gorka Irazoqui, Thomas Eisenbarth, and Berk Sunar. "S$A: A Shared Cache Attack That Works across Cores and Defies VM Sandboxing – and Its Application to AES". In: *Symposium on Security and Privacy*. 2015, pp. 591–604.

[87] Gorka Irazoqui, Mehmet Sinan Inci, Thomas Eisenbarth, and Berk Sunar. "Wait a Minute! A fast, Cross-VM Attack on AES". In: *Research in Attacks, Intrusions and Defenses*. 2014, pp. 299–319.

[88] Roslan Ismail, Toqeer Ali Syed, and Shahrulniza Musa. "Design and Implementation of an Efficient Framework for Behaviour Attestation Using N-Call Slides". In: *Int. Conf. on Ubiquitous Information Management and Comm.* 2014.

[89] Insu Jang, Adrian Tang, Taehoon Kim, Simha Sethumadhavan, and Jaehyuk Huh. "Heterogeneous Isolated Execution for Commodity GPUs". In: *International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*. 2019, pp. 455–468. ISBN: 9781450362405.

[90] Guy Jumarie. *Relative information*. Springer, 1990.

[91] Emilia Käsper and Peter Schwabe. "Faster and Timing-Attack Resistant AES-GCM". In: *Cryptographic Hardware and Embedded Systems*. Ed. by Christophe Clavier and Kris Gaj. 2009, pp. 1–17.

[92] Sebastian Kobbe, Lars Bauer, Daniel Lohmann, Wolfgang Schröder-Preikschat, and Jörg Henkel. "DistRM: distributed resource management for on-chip many-core systems". In: *Proceedings of the Seventh IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis*. CODES+ISSS '11. Taipei, Taiwan: Association for Computing Machinery, 2011, pp. 119–128. ISBN: 9781450307154. DOI: 10.1145/2039370.2039392.

[93] Florian Kohnhäuser, Niklas Büscher, and Stefan Katzenbeisser. "A Practical Attestation Protocol for Autonomous Embedded Systems". In: *European Symp. on Security and Privacy*. 2019.

[94] Jingfei Kong, Onur Aciicmez, Jean-Pierre Seifert, and Huiyang Zhou. "Hardware-software integrated approaches to defend against software cache-based side channel attacks". In: *Int. Symp. on High Perf. Computer Arch.* 2009, pp. 393–404.

[95] Joonho Kong, Farinaz Koushanfar, Praveen K. Pendyala, Ahmad-Reza Sadeghi, and Christian Wachsmann. "PUFatt: Embedded platform attestation based on novel processor-based PUFs". In: *Design Automation Conf.* 2014.

[96] Boyu Kuang, Anmin Fu, Willy Susilo, Shui Yu, and Yansong Gao. "A survey of remote attestation in Internet of Things: Attacks, countermeasures, and prospects". In: *Computers & Security* 112 (2022).

[97] Sheng Li, Jung Ho Ahn, Richard D Strong, Jay B Brockman, Dean M Tullsen, and Norman P Jouppi. "The McPAT Framework for Multicore and Manycore Architectures: Simultaneously Modeling Power, Area, and Timing". In: *ACM Trans. Arch. and Code Opt. (TACO)* (2013). DOI: 10.1145/2445572.2445577.

[98] Yanbing Li and J. Henkel. "A framework for estimating and minimizing energy dissipation of embedded HW/SW systems". In: *Proceedings 1998 Design and Automation Conference. 35th DAC. (Cat. No.98CH36175)*. 1998, pp. 188–193. DOI: 10.1109/DAC.1998.724464.

[99] Christian Lindenmeier, Jan Gruber, and Felix Freiling. "InvesTEE: A TEE-supported Framework for Lawful Remote Forensic Investigations". In: *Digital Threats* (July 2024). DOI: 10.1145/3680294. URL: https://doi.org/10.1145/3680294.

[100] Mario Lins, René Mayrhofer, Michael Roland, Daniel Hofer, and Martin Schwaighofer. *On the critical path to implant backdoors and the effectiveness of potential mitigation techniques: Early learnings from XZ*. 2024. arXiv: 2404.08987 [cs.CR].

[101] Fangfei Liu, Yuval Yarom, Qian Ge, Gernot Heiser, and Ruby B. Lee. "Last-Level Cache Side-Channel Attacks are Practical". In: *Symposium on Security and Privacy*. 2015, pp. 605–622.

[102] Renju Liu, Luis Garcia, Zaoxing Liu, Botong Ou, and Mani Srivastava. "SecDeep: Secure and Performant On-Device Deep Learning Inference Framework for Mobile and IoT Devices". In: *International Conference on Internet-of-Things Design and Implementation (IoTDI)*. 2021, pp. 67–79. ISBN: 9781450383547.

[103] Zijun Long, Xiaohang Wang, Yingtao Jiang, Guofeng Cui, Li Zhang, and Terrence Mak. "Improving the efficiency of thermal covert channels in multi-/many-core systems". In: *Design, Automation and Test in Europe Conference and Exhibition (DATE)*. Apr. 2018, pp. 1459–1464.

[104] Xiaoxuan Lou, Tianwei Zhang, Jun Jiang, and Yinqian Zhang. "A Survey of Microarchitectural Side-Channel Vulnerabilities, Attacks, and Defenses in Cryptography". In: *ACM Computing Surveys* (2021).

[105] Yangdi Lyu and Prabhat Mishra. "A Survey of Side-Channel Attacks on Caches and Countermeasures". In: *Journal of Hardware and Systems Security* 2.1 (Mar. 2018), pp. 33–50.

[106] Pieter Maene and Ingrid Verbauwhede. "Single-Cycle Implementations of Block Ciphers". In: *Lightweight Cryptography for Security and Privacy*. 2016.

[107] Theodoros Marinakis, Shivam Kundan, and Iraklis Anagnostopoulos. "Meeting Power Constraints While Mitigating Contention on Clustered Multiprocessor System". In: *IEEE Embedded Systems Letters* 12.3 (2020), pp. 99–102. DOI: 10.1109/LES.2019.2956990.

[108] Robert Martin, John Demme, and Simha Sethumadhavan. "TimeWarp: Rethinking timekeeping and performance monitoring mechanisms to mitigate side-channel attacks". In: *Int. Symp. on Computer Arch. (ISCA)*. 2012, pp. 118–129.

[109] Ramya Jayaram Masti, Devendra Rai, Aanjhan Ranganathan, Christian Müller, Lothar Thiele, and Srdjan Capkun. "Thermal Covert Channels on Multi-Core Platforms". In: *USENIX Conference on Security Symposium (SEC)*. 2015, pp. 865–880. ISBN: 9781931971232.

[110] Microsoft. *Hyper-V Overview*. https://learn.microsoft.com/en-us/windows-server/virtualization/hyper-v/hyper-v-overview. Accessed: 2024-09-11. 2023.

[111] Ivan Miketic, Krithika Dhananjay, and Emre Salman. "Covert Channel Communication as an Emerging Security Threat in 2.5D/3D Integrated Systems". In: *Sensors* (2023).

[112] J. Millen. "20 Years of Covert Channel Modeling and Analysis". In: *Symposium on Security and Privacy*. May 1999, pp. 113–114.

[113] Nimish Mishra, Anirban Chakraborty, Urbi Chatterjee, and Debdeep Mukhopadhyay. “Time’s a Thief of Memory”. In: *Smart Card Research and Advanced Applications*. Ed. by Ileana Buhan and Tobias Schneider. 2023.

[114] Amir Moradi. “Side-Channel Leakage through Static Power”. In: *Cryptographic Hardware and Embedded Systems*. Ed. by Lejla Batina and Matthew Robshaw. 2014, pp. 562–579.

[115] Hassan Nassar, Hanna AlZughbi, Dennis Gnad, Lars Bauer, Mehdi Tahoori, and Jörg Henkel. “LoopBreaker: Disabling Interconnects to Mitigate Voltage-Based Attacks in Multi-Tenant FPGAs”. In: *ICCAD*. 2021.

[116] Hassan Nassar, Lars Bauer, and Jörg Henkel. “CaPUF: Cascaded PUF Structure for Machine Learning Resiliency”. In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 41.11 (2022), pp. 4349–4360. DOI: 10.1109/TCAD.2022.3197539.

[117] Hassan Nassar, Simon Pankner, Lars Bauer, and Jörg Henkel. “Late Breaking Results: Configurable Ring Oscillators as a Side-Channel Countermeasure”. In: *2023 60th ACM/IEEE Design Automation Conference (DAC)*. 2023, pp. 1–2. DOI: 10.1109/DAC56929.2023.10247786.

[118] Nicholas Nethercote and Julian Seward. “Valgrind: A Framework for Heavyweight Dynamic Binary Instrumentation”. In: *SIGPLAN Not.* 42.6 (2007).

[119] Michael Neve and Jean-Pierre Seifert. “Advances on Access-Driven Cache Attacks on AES”. In: *Selected Areas in Cryptography*. Ed. by Eli Biham and Amr M. Youssef. 2007, pp. 147–162.

[120] NIST. *CVE-2022-23319 Detail*. July 2022. URL: https://nvd.nist.gov/vuln/detail/CVE-2022-23319.

[121] NIST. *CVE-2022-38529 Detail*. Sept. 2022. URL: https://nvd.nist.gov/vuln/detail/CVE-2022-38529.

[122] NIST. *CVE-2024-3094 Detail*. Mar. 2024. URL: https://nvd.nist.gov/vuln/detail/CVE-2024-3094.

[123] Ivan De Oliveira Nunes, Sashidhar Jakkamsetti, and Gene Tsudik. “Tiny-CFA: A minimalistic approach for control-flow attestation using verified proofs of execution”. In: *arXiv preprint arXiv:2011.07400* (2020).

[124] Dag Arne Osvik, Adi Shamir, and Eran Tromer. "Cache Attacks and Countermeasures: The Case of AES". In: *Topics in Cryptology*. 2006, pp. 1–20.

[125] Thales Bandiera Paiva, Javier Navaridas, and Routo Terada. "Robust covert channels based on DRAM power consumption". In: *Information Security: 22nd International Conference, ISC 2019, New York City, NY, USA, September 16–18, 2019, Proceedings 22*. Springer. 2019, pp. 319–338.

[126] Sining Pan and Kofi AA Makinwa. "A 0.25 mm 2-Resistor-Based Temperature Sensor With an Inaccuracy of 0.12° C (3σ) From- 55° C to 125° C". In: *IEEE Journal of Solid-State Circuits* 53.12 (2018), pp. 3347–3355.

[127] Dimitrios Papamartzivanos, Sofia Anna Menesidou, Panagiotis Gouvas, and Thanassis Giannetsos. "Towards Efficient Control-Flow Attestation with Software-Assisted Multi-level Execution Tracing". In: *Int. Mediterranean Conf. on Communications and Networking (MeditCom)*. 2021.

[128] Anuj Pathania and Jörg Henkel. "HotSniper: Sniper-Based Toolchain for Many-Core Thermal Simulations in Open Systems". In: *IEEE Embedded Systems Letters* 11.2 (2019), pp. 54–57.

[129] F. Pedregosa et al. "Scikit-learn: Machine Learning in Python". In: *Journal of Machine Learning Research* 12 (2011).

[130] *perf: Linux profiling with performance counters*. June 2009. URL: https://perf.wiki.kernel.org/index.php/Main_Page.

[131] Behnaz Pourmohseni, Stefan Wildermann, Fedor Smirnov, Paul E. Meyer, and Jürgen Teich. "Task Migration Policy for Thermal-Aware Dynamic Performance Optimization in Many-Core Systems". In: *IEEE Access* 10 (2022), pp. 33787–33802. DOI: 10.1109/ACCESS.2022.3162617.

[132] Parisa Rahimi, Amit Kumar Singh, and Xiaohang Wang. "Selective Noise Based Power-Efficient and Effective Countermeasure against Thermal Covert Channel Attacks in Multi-Core Systems". In: *Journal of Low Power Electronics and Applications* 12.2 (2022). ISSN: 2079-9268. DOI: 10.3390/jlpea12020025.

[133] Martin Rapp, Hussam Amrouch, Yibo Lin, Bei Yu, David Z. Pan, Marilyn Wolf, and Jörg Henkel. "MLCAD: A Survey of Research in Machine Learning for CAD Keynote Paper". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 41.10 (2022), pp. 3162–3181. DOI: 10.1109/TCAD.2021.3124762.

[134] Jonas Röckl, Mykolai Protsenko, Monika Huber, Tilo Müller, and Felix C. Freiling. "Advanced System Resiliency Based on Virtualization Techniques for IoT Devices". In: *Proceedings of the 37th Annual Computer Security Applications Conference*. ACSAC '21. Virtual Event, USA: Association for Computing Machinery, 2021, pp. 455–467. ISBN: 9781450385794. DOI: 10.1145/3485832.3485836. URL: https://doi.org/10.1145/3485832.3485836.

[135] Chinmayee Rout, Srinivas Sethi, J Chandrakanta Badajena, and Ramesh Kumar Sahoo. "Secure Virtual Machine Allocation for Prevention of Side Channel Attacks in Cloud Computing". In: *2022 International Conference on Intelligent Controller and Computing for Smart Power (ICICCSP)*. 2022, pp. 1–6. DOI: 10.1109/ICICCSP53532.2022.9862404.

[136] Enrico Russo, Maurizio Palesi, Salvatore Monteleone, Davide Patti, Andrea Mineo, Giuseppe Ascia, and Vincenzo Catania. "DNN Model Compression for IoT Domain-Specific Hardware Accelerators". In: *IEEE Internet of Things Journal* 9.9 (2022), pp. 6650–6662. DOI: 10.1109/JIOT.2021.3111723.

[137] Mohamed Sabt, Mohammed Achemlal, and Abdelmadjid Bouabdallah. "Trusted Execution Environment: What It is, and What It is Not". In: *2015 IEEE Trustcom/BigDataSE/ISPA*. 2015.

[138] Aditya Kumar Sahu and Monalisa Sahu. "Digital image steganography and steganalysis: A journey of the past three decades". In: *Open Computer Science* 10 (Oct. 2020), pp. 1–47. DOI: 10.1515/comp-2020-0136.

[139] Gururaj Saileshwar, Christopher W. Fletcher, and Moinuddin K. Qureshi. "Streamline: a fast, flushless cache covert-channel attack by enabling asynchronous collusion". In: *ACM International Conference on Architectural Support for Programming Languages and Operating Systems* (2021).

[140] Christos Sakalis, Carl Leonardsson, Stefanos Kaxiras, and Alberto Ros. "Splash-3: A properly synchronized benchmark suite for contemporary research". In: *Int. Symp. on Performance Analysis of Systems and SW (ISPASS)*. 2016.

[141] Nitish Salwan, Sandeep Singh, Suket Arora, and Amarpreet Singh. "An Insight to Covert Channels". In: *CoRR* abs/1306.2252 (2013). arXiv: 1306.2252. URL: http://arxiv.org/abs/1306.2252.

[142] Steve Scargall. "Profiling and Performance". In: *Programming Persistent Memory: A Comprehensive Guide for Developers*. Berkeley, CA: Apress, 2020, pp. 295–312. ISBN: 978-1-4842-4932-1. DOI: 10.1007/978-1-4842-4932-1_15.

[143] Hannah M. Schlüter, Jeremy Tan, Benjamin Hou, and Bernhard Kainz. "Natural Synthetic Anomalies for Self-supervised Anomaly Detection and Localization". In: *Computer Vision – ECCV*. Ed. by Shai Avidan, Gabriel Brostow, Moustapha Cissé, Giovanni Maria Farinella, and Tal Hassner. Springer Nature Switzerland, 2022.

[144] Steffen Schulz, Ahmad-Reza Sadeghi, and Christian Wachsmann. "Short Paper: Lightweight Remote Attestation Using Physical Functions". In: *Conf. on Wireless Network Security*. 2011.

[145] Muhammad Shafique, Waqas Ahmad, Rehan Hafiz, and Jörg Henkel. "A low latency generic accuracy configurable adder". In: *Proceedings of the 52nd Annual Design Automation Conference*. DAC '15. San Francisco, California: Association for Computing Machinery, 2015. ISBN: 9781450335201. DOI: 10.1145/2744769.2744778.

[146] Chaoqun Shen, Jiliang Zhang, and Gang Qu. *MES-Attacks: Software-Controlled Covert Channels based on Mutual Exclusion and Synchronization*. 2022. arXiv: 2211.11855 [cs.AR].

[147] Lokesh Siddhu, Hassan Nassar, Lars Bauer, Christian Hakert, Nils Hölscher, Jian-Jia Chen, and Joerg Henkel. "Swift-CNN: Leveraging PCM Memory's Fast Write Mode to Accelerate CNNs". In: *IEEE Embedded Systems Letters* 15.4 (2023), pp. 234–237. DOI: 10.1109/LES.2023.3298742.

[148] Mohammed Bakr Sikal, Heba Khdr, Martin Rapp, and Jörg Henkel. "Machine Learning-based Thermally-Safe Cache Contention Mitigation in Clustered Manycores". In: *2023 60th ACM/IEEE Design Automation Conference (DAC)*. 2023, pp. 1–6. DOI: 10.1109/DAC56929.2023.10247708.

[149] Matt Spisak. “Hardware-Assisted Rootkits: Abusing Performance Counters on the ARM and x86 Architectures”. In: *Workshop on Offensive Technologies*. 2016.

[150] François-Xavier Standaert. “Introduction to Side-Channel Attacks”. In: *Secure Integrated Circuits and Systems*. Ed. by Ingrid M.R. Verbauwhede. Springer US, 2010, pp. 27–42.

[151] Toqeer Ali Syed, Roslan Ismail, Shahrulniza Musa, Mohammad Nauman, and Sohail Khan. “A Sense of Others: Behavioral Attestation of UNIX Processes on Remote Platforms”. In: *Int. Conf. on Ubiquitous Inform. Management and Comm.* 2012.

[152] The OpenSSL Project. “OpenSSL: The Open Source toolkit for SSL/TLS”. `www.openssl.org`.

[153] Theodoros Trochatos, Anthony Etim, and Jakub Szefer. “Security evaluation of thermal covert-channels on SmartSSDs”. In: *arXiv preprint arXiv:2305.09115* (2023).

[154] TrustedFirmware. *About OP-TEE*. 04.09.23. URL: `https://optee.readthedocs.io/en/latest/general/about.html`.

[155] Furkan Turan and Ingrid Verbauwhede. “Propagating Trusted Execution through Mutual Attestation”. In: *Workshop on System SW for Trusted Exec.* 2019.

[156] Stavros Volos, Kapil Vaswani, and Rodrigo Bruno. “Graviton: Trusted Execution Environments on GPUs”. In: *USENIX Symposium on Operating Systems Design and Implementation (OSDI)*. Oct. 2018, pp. 681–696. ISBN: 978-1-939133-08-3.

[157] Han Wang, Hossein Sayadi, Avesta Sasan, Setareh Rafatirad, Tinoosh Mohsenin, and Houman Homayoun. “Comprehensive Evaluation of Machine Learning Countermeasures for Detecting Microarchitectural Side-Channel Attacks”. In: *Proceedings of the 2020 on Great Lakes Symposium on VLSI*. ACM, 2020, pp. 181–186.

[158] Jiachen Wang, Xiaohang Wang, Yingtao Jiang, Amit Kumar Singh, Letian Huang, and Mei Yang. “Combating Enhanced Thermal Covert Channel in Multi-/Many-Core Systems With Channel-Aware Jamming”. In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 39.11 (2020), pp. 3276–3287.

[159] Xiaohang Wang, Hengli Huang, Ruolin Chen, Yingtao Jiang, Amit Kumar Singh, Mei Yang, and Letian Huang. "Detection of Thermal Covert Channel Attacks Based on Classification of Components of the Thermal Signal Features". In: *IEEE Transactions on Computers* (2022), pp. 1–14. DOI: 10.1109/TC.2022.3189578.

[160] Brett Weinger, Jinoh Kim, Alex Sim, Makiya Nakashima, Nour Moustafa, and K. John Wu. "Enhancing IoT anomaly detection performance for federated learning". In: *Digital Communications and Networks* 8.3 (2022).

[161] Steven Cameron Woo, Moriyoshi Ohara, Evan Torrie, Jaswinder Pal Singh, and Anoop Gupta. "The SPLASH-2 Programs: Characterization and Methodological Considerations". In: *Int. Symp. on Computer Arch.* 1995, pp. 24–36.

[162] Ran Wu, Xinmin Guo, Jian Du, and Junbao Li. "Accelerating Neural Network Inference on FPGA-Based Platforms—A Survey". In: *Electronics* 10.9 (2021). ISSN: 2079-9292. DOI: 10.3390/electronics10091025. URL: https://www.mdpi.com/2079-9292/10/9/1025.

[163] Kai Xiong, Supeng Leng, Chongwen Huang, Chau Yuen, and Yong Liang Guan. "Intelligent Task Offloading for Heterogeneous V2X Communications". In: *IEEE Transactions on Intelligent Transportation Systems* 22.4 (2021), pp. 2226–2238. DOI: 10.1109/TITS.2020.3015210.

[164] Zhixing Xu, Sayak Ray, Pramod Subramanyan, and Sharad Malik. "Malware detection using machine learning based analysis of virtual memory access patterns". In: *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2017*. 2017, pp. 169–174. DOI: 10.23919/DATE.2017.7926977.

[165] Chao Yang, Yunfei Guo, Hongchao Hu, Wenyan Liu, and Yawen Wang. "An effective and scalable VM migration strategy to mitigate cross-VM side-channel attacks in cloud". In: *China Communications* 16.4 (2019), pp. 151–171. DOI: 10.12676/j.cc.2019.04.012.

[166] Manzhi Yang and Qiaoyan Wen. "Detecting android malware by applying classification techniques on images patterns". In: *2017 IEEE 2nd International Conference on Cloud Computing and Big Data Analysis (ICCCBDA)*. 2017, pp. 344–347. DOI: 10.1109/ICCCBDA.2017.7951936.

[167] Ruikang Yang, Jianfeng Ma, Junying Zhang, Saru Kumari, Sachin Kumar, and Joel J. P. C. Rodrigues. "Practical Feature Inference Attack in Vertical Federated Learning During Prediction in Artificial Internet of Things". In: *IEEE Internet of Things Journal* 11.1 (2024), pp. 5–16.

[168] Yuval Yarom and Katrina Falkner. "FLUSH+RELOAD: A High Resolution, Low Noise, L3 Cache Side-Channel Attack". In: *USENIX Security Symposium*. Aug. 2014, pp. 719–732.

[169] Raz Ben Yehuda, Michael Kiperberg, and Nezer Jacob Zaidenberg. "Nanovised Control Flow Attestation". In: *Applied Sciences* 12.5 (2022).

[170] *ZCU102 Evaluation Board User Guide*. Xilinx. 2023.

[171] Shaza Zeitouni, Ghada Dessouky, Orlando Arias, Dean Sullivan, Ahmad Ibrahim, Yier Jin, and Ahmad Reza Sadeghi. "ATRIUM: Runtime attestation resilient under memory attacks". In: *Int. Conf. on Computer-Aided Design* (2017).

[172] Jiliang Zhang, Chaoqun Shen, and Gang Qu. "Mex+Sync: Software Covert Channels Exploiting Mutual Exclusion and Synchronization". In: *TCAD* (2023), pp. 1–1. DOI: 10.1109/TCAD.2023.3291669.

[173] Yumei Zhang, Xinzhi Liu, Cong Sun, Dongrui Zeng, Gang Tan, Xiao Kan, and Siqi Ma. "ReCFA: Resilient Control-Flow Attestation". In: *Computer Sec. Appl. Conf.* 2021.

[174] Jianping Zhu, Rui Hou, XiaoFeng Wang, Wenhao Wang, Jiangfeng Cao, Lutan Zhao, Fengkai Yuan, Peinan Li, Zhongpu Wang, Boyan Zhao, et al. "Enabling privacy-preserving, compute-and data-intensive computing using heterogeneous trusted execution environment". In: *arXiv preprint arXiv:1904.04782* (2019).