



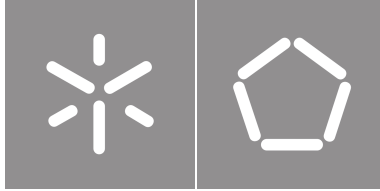
## Securing the Internet of Things through patchable hardware

Sérgio Augusto Gomes Pereira

**Universidade do Minho**  
Escola de Engenharia







**Universidade do Minho**

Escola de Engenharia

Sérgio Augusto Gomes Pereira

## **Securing the Internet of Things through patchable hardware**

Tese de Doutoramento

Programa Doutoral em Engenharia Electrónica e de  
Computadores

Trabalho efetuado sob a orientação de:

**Investigador Principal Doutor Sandro Emanuel  
Salgado Pinto**

**Professor Doutor Jorge Miguel Nunes dos  
Santos Cabral**

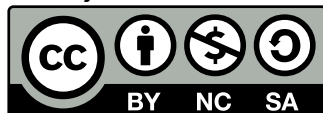
## **COPYRIGHT AND TERMS OF USE OF THIS WORK BY A THIRD PARTY**

This is academic work that can be used by third parties as long as internationally accepted rules and good practices regarding copyright and related rights are respected.

Accordingly, this work may be used under the license provided below.

If the user needs permission to make use of the work under conditions not provided for in the indicated licensing, they should contact the author through the RepositoriUM of Universidade do Minho.

### ***Licença concedida aos utilizadores deste trabalho***



**Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International**  
**CC BY-NC-SA 4.0**

<https://creativecommons.org/licenses/by-nc-sa/4.0/deed.en>

# Acknowledgements

I would like to begin by expressing my sincere gratitude to Professor Dr. Sandro Pinto. From my master's thesis onwards, you have been more than just my advisor; you have been a true mentor and a dear friend. Your guidance extended beyond academic matters, teaching me life lessons that I'll carry with me. Your endless patience, even when I doubted myself, have been invaluable. Through your example, I have learned how hard work, dedication, and genuine passion can lead to success.

To my incredible lab mates: Martins, Miguel, David, Daniel, and Roriz. Especially to Miguel Costa, who has been with me from the start of this journey and joined me in Thailand. Thank you all for making the days in the lab not just bearable, but truly enjoyable. I am also sincerely grateful to Professor Dr. José Cabral for believing in my potential from the start, and to my dear friend Professor Dr. Tiago Gomes for his constant encouragement and support.

To my dad and mom, thank you. Your unconditional love and support have always been present since childhood, nurturing my curiosity and love for electronics and engineering, which ultimately led to this very PhD dissertation. Along with my sisters, brothers-in-law, and my wonderful nephews, your collective love has been what keeps me grounded. Knowing I had your unwavering support made all the difference. And to my closest friends, thank you for being the source of joy and distraction I needed.

Finally, and certainly not least, to my love, Carolina. Thank you for your extraordinary patience, kindness, and unwavering belief in me. You've been by my side through my worst and best days. I truly couldn't have accomplished this journey without you. I'm eternally grateful for your love and sacrifice. And of course, a special nod to our loyal dog, Ollie, for all the happy distractions.

This thesis was supported by a PhD scholarship (grant SFRH/BD/145209/2019) from *Fundação para a Ciência e Tecnologia*.

### **STATEMENT OF INTEGRITY**

I hereby declare having conducted this academic work with integrity. I confirm that I have not used plagiarism or any form of undue use of information or falsification of results along the process leading to its elaboration.

I further declare that I have fully acknowledged the Code of Ethical Conduct of the Universidade do Minho.

\_\_\_\_\_, \_\_\_\_\_  
(Place) (Date)

\_\_\_\_\_  
(Sérgio Augusto Gomes Pereira)

” “We learn by expressing, not by thinking, knowledge  
doesn’t really exist until you can write it down.”

— **Evan Puschak**, from the book *Escape into Meaning*

# Resumo

## Potencialidades do Hardware Reconfigurável na Segurança de Sistemas IoT

A Internet das Coisas (IoT) está a transformar o nosso quotidiano ao interligar biliões de dispositivos em setores como o da saúde, o financeiro e o de transportes. A integração de processadores heterogêneos nestes sistemas (tais como GPUs, aceleradores de IA e FPGAs) origina arquiteturas complexas que ampliam a superfície de ataque. Estes desafios são ainda agravados pela limitada capacidade de atualização que caracterizam frequentemente os dispositivos IoT, deixando-os expostos a ameaças emergentes muito tempo após o seu lançamento. Consequentemente, os modelos de segurança convencionais, concebidos para sistemas estáticos e com bases de confiança (TCBs) fixas, têm dificuldade em acompanhar esta constante evolução.

Uma das soluções mais adotadas na indústria para proteger aplicações críticas são os *Trusted Execution Environments* (TEEs), que proporcionam isolamento e integridade através de extensões de hardware, reduzindo a TCB de software e limitando o impacto de sistemas operativos comprometidos. No entanto, a maioria dos TEEs mantém um foco em arquiteturas centradas no processador e em modelos de confiança fixos. Ao ignorar unidades de computação heterogênea fora do domínio da CPU, recorrem a firmware mais complexo, introduzindo novas vulnerabilidades de segurança.

Esta tese explora o conceito de *hardware atualizável* (*patchable hardware*) como um novo paradigma para proteger sistemas IoT. São apresentadas três contribuições principais. Em primeiro, um estudo sistemático que consolida a investigação sobre as implicações de segurança da computação reconfigurável, evidenciando como os FPGAs introduzem simultaneamente novas superfícies de ataque e oportunidades de defesa. Em segundo, o *TEEOD*, que evidencia a possibilidade de implementar ambientes de execução confiáveis no FPGA, superando as limitações dos TEEs convencionais e mantendo compatibilidade com a especificação *GlobalPlatform TEE*. Por fim, o *TREE*, que propõe uma framework unificada onde a reconfiguração parcial dinâmica é tratada como um serviço de TEE, permitindo criar, atualizar, ou remover enclaves de forma segura, sem necessidade de reiniciar o sistema nem de intervenção do software.

**Palavras-chave:** Internet das coisas (IoT), Hardware reconfigurável, FPGA, TEE, soft-MCU, Reconfiguração dinâmica parcial (DPR), Hardware-Software co-design

# Abstract

## Securing the Internet of Things through patchable hardware

The Internet of Things (IoT) is reshaping our digital ecosystem by connecting billions of devices across domains such as healthcare, finance, transportation, and beyond. This massive scale, combined with the ongoing trend of embedding heterogeneous processors (e.g., GPUs, AI engines, and FPGAs) beyond the CPU, gives rise to complex architectures that expand the attack surface. These security challenges are further increased by the long lifespans and limited updatability commonly observed in IoT devices, leaving them exposed to threats well after deployment. Conventional security models based on static trusted computing bases (TCBs) struggle to adapt to this dynamic landscape.

One well-established security in industry to protect the integrity and confidentiality of security-critical applications rely on Trusted Execution Environments (TEEs). TEEs provide hardware-assisted isolation and integrity protection by reducing the software TCB and limiting the impact of compromised operating systems. While widely adopted, most TEEs assume fixed, CPU-centric roots of trust that neglect the specialized units operating outside the CPU's domain, introducing gaps and firmware-induced vulnerabilities.

This thesis explores *patchable hardware* as a new paradigm for enabling trusted computing in modern IoT systems. Three main contributions are presented to support this hypothesis. First, a systematic survey consolidates research on the security implications of reconfigurable computing, highlighting how the FPGA fabric introduces both new attack surfaces and opportunities for security enforcement. Second, TEEOD demonstrates that isolated trusted execution can be implemented entirely within the reconfigurable fabric, overcoming limitations of mainstream TEEs, while preserving GlobalPlatform TEE specification compatibility. Third, TREE introduces a unified framework that turns dynamic partial reconfiguration (DPR) into a trusted execution service, allowing enclaves to be securely created, patched, or revoked at runtime.

**Keywords:** IoT, Patchable Hardware, FPGA, TEE, soft-MCU, Dynamic Partial Reconfiguration (DPR), Hardware-Software co-design

# Contents

|                                                                    |              |
|--------------------------------------------------------------------|--------------|
| <b>List of Figures</b>                                             | <b>xiii</b>  |
| <b>List of Tables</b>                                              | <b>xvi</b>   |
| <b>Acronyms</b>                                                    | <b>xviii</b> |
| <br>                                                               |              |
| <b>I Thesis Summary</b>                                            | <b>1</b>     |
| <br>                                                               |              |
| <b>1 Introduction</b>                                              | <b>2</b>     |
| 1.1 Motivation . . . . .                                           | 4            |
| 1.1.1 Security Challenges in Heterogeneous Architectures . . . . . | 4            |
| 1.1.2 The Limitations of Current IoT Security Approaches . . . . . | 5            |
| 1.2 Hypothesis . . . . .                                           | 6            |
| 1.3 Contributions . . . . .                                        | 7            |
| 1.3.1 Research Questions . . . . .                                 | 7            |
| 1.3.2 Main Contributions . . . . .                                 | 8            |
| 1.4 Thesis Outline . . . . .                                       | 10           |
| <br>                                                               |              |
| <b>2 Background</b>                                                | <b>11</b>    |
| 2.1 Field-Programmable Gate Arrays . . . . .                       | 11           |
| 2.1.1 General-purpose Computing on FPGA . . . . .                  | 12           |
| 2.1.2 Rise of SoC FPGA Architectures . . . . .                     | 13           |
| 2.1.3 Secure Reconfiguration Process . . . . .                     | 14           |
| 2.1.4 Partial Reconfiguration . . . . .                            | 15           |
| 2.2 Reconfigurable Computing Applications . . . . .                | 17           |
| 2.2.1 FPGA on the Cloud . . . . .                                  | 18           |
| 2.2.2 IoT Edge Applications . . . . .                              | 19           |

|          |                                                                                                                    |           |
|----------|--------------------------------------------------------------------------------------------------------------------|-----------|
| 2.3      | FPGA-specific Vulnerabilities and Attack Vectors . . . . .                                                         | 19        |
| 2.3.1    | Adversarial Access to the FPGA . . . . .                                                                           | 20        |
| 2.3.2    | Remote Physical Attacks . . . . .                                                                                  | 20        |
| 2.3.3    | Breaking Secure Boot . . . . .                                                                                     | 21        |
| 2.3.4    | DRAM Attacks . . . . .                                                                                             | 22        |
| 2.4      | Trusted Execution Environments . . . . .                                                                           | 22        |
| 2.4.1    | Arm TrustZone . . . . .                                                                                            | 23        |
| 2.4.2    | GlobalPlatform-compliant TEE Software Architecture . . . . .                                                       | 23        |
| 2.5      | Vulnerabilities in TrustZone-assisted TEE systems . . . . .                                                        | 25        |
| 2.5.1    | Architectural Issues . . . . .                                                                                     | 26        |
| 2.5.2    | Implementation Issues . . . . .                                                                                    | 26        |
| 2.5.3    | Hardware Issues . . . . .                                                                                          | 27        |
| <b>3</b> | <b>Related Work</b>                                                                                                | <b>29</b> |
| 3.1      | FPGA-based Countermeasures . . . . .                                                                               | 29        |
| 3.1.1    | Passive Countermeasures . . . . .                                                                                  | 30        |
| 3.1.2    | Runtime Monitoring and Enforcement . . . . .                                                                       | 30        |
| 3.1.3    | FPGA-based OS and Virtualization . . . . .                                                                         | 32        |
| 3.2      | TEE Systems . . . . .                                                                                              | 33        |
| 3.2.1    | TEE Systems based on CPU-based TEE extensions . . . . .                                                            | 34        |
| 3.2.2    | External Secure Processors . . . . .                                                                               | 34        |
| 3.3      | FPGA-based TEEs . . . . .                                                                                          | 35        |
| 3.3.1    | Extended CPU-centric TEE systems . . . . .                                                                         | 35        |
| 3.3.2    | Cloud FPGAs . . . . .                                                                                              | 37        |
| 3.3.3    | Trusted Execution on Soft-MCUs . . . . .                                                                           | 38        |
| 3.4      | Other Research . . . . .                                                                                           | 39        |
| 3.4.1    | New Hardware Architectures . . . . .                                                                               | 40        |
| 3.4.2    | GPU-based TEEs . . . . .                                                                                           | 40        |
| 3.4.3    | TEE for AI accelerators . . . . .                                                                                  | 41        |
| 3.5      | Gap Analysis . . . . .                                                                                             | 42        |
| 3.5.1    | Lack of a unified perspective on threats and under-explored security opportunities in FPGA-based systems . . . . . | 42        |
| 3.5.2    | Limitations of CPU-centric TEEs in heterogeneous systems . . . . .                                                 | 44        |
| 3.5.3    | Existing FPGA-based TEE solutions lack runtime flexibility and reconfigurable isolation . . . . .                  | 45        |
| <b>4</b> | <b>Concluding Remarks</b>                                                                                          | <b>47</b> |
| 4.1      | Conclusions . . . . .                                                                                              | 47        |

|     |                       |    |
|-----|-----------------------|----|
| 4.2 | Future Work . . . . . | 48 |
|-----|-----------------------|----|

## **II Publications 50**

### **5 Beneath the fabric: A survey on threats and opportunities in reconfigurable systems 52**

|       |                                                                    |    |
|-------|--------------------------------------------------------------------|----|
| 5.1   | Introduction . . . . .                                             | 53 |
| 5.2   | Background . . . . .                                               | 56 |
| 5.2.1 | FPGAs and SoC FPGAs architectures . . . . .                        | 56 |
| 5.2.2 | Built-in Reconfiguration Security . . . . .                        | 57 |
| 5.2.3 | TrustZone-assisted TEEs . . . . .                                  | 58 |
| 5.3   | Adversarial uses of reconfigurable hardware . . . . .              | 58 |
| 5.3.1 | Entry points to the reconfigurable fabric . . . . .                | 58 |
| 5.3.2 | FPGA-based physical attacks . . . . .                              | 59 |
| 5.3.3 | Breaking Secure Boot . . . . .                                     | 60 |
| 5.3.4 | DRAM attacks . . . . .                                             | 61 |
| 5.3.5 | TrustZone and AXI Interface Exploitation . . . . .                 | 61 |
| 5.4   | Defensive uses of reconfigurable hardware . . . . .                | 61 |
| 5.4.1 | Passive countermeasures . . . . .                                  | 62 |
| 5.4.2 | Runtime Monitoring and Enforcement . . . . .                       | 62 |
| 5.4.3 | FPGA-based OS and Virtualization . . . . .                         | 64 |
| 5.5   | Trusted execution in heterogeneous systems . . . . .               | 65 |
| 5.5.1 | Extensions of CPU-based TEEs . . . . .                             | 66 |
| 5.5.2 | Novel architectures . . . . .                                      | 67 |
| 5.5.3 | Cloud-Oriented FPGA Frameworks . . . . .                           | 67 |
| 5.5.4 | FPGA-based and reconfigurable TEEs . . . . .                       | 68 |
| 5.6   | Future Research Directions . . . . .                               | 69 |
| 5.6.1 | Threat models that reflect FPGA-specific vulnerabilities . . . . . | 69 |
| 5.6.2 | Beyond fixed hardware security extensions . . . . .                | 70 |
| 5.6.3 | Reconfiguration as an enabler of dynamic countermeasures . . . . . | 70 |
| 5.7   | Conclusion . . . . .                                               | 71 |

### **6 Rethinking Trusted Execution Environments in the Age of Reconfigurable Computing 72**

|       |                                          |    |
|-------|------------------------------------------|----|
| 6.1   | Introduction . . . . .                   | 73 |
| 6.2   | Background and Motivation . . . . .      | 75 |
| 6.2.1 | Trusted Execution Environments . . . . . | 75 |
| 6.2.2 | Field-Programmable Gate Arrays . . . . . | 76 |

|          |                                                                                     |            |
|----------|-------------------------------------------------------------------------------------|------------|
| 6.3      | Threat Model and Design Principles . . . . .                                        | 77         |
| 6.3.1    | Threat Model . . . . .                                                              | 77         |
| 6.3.2    | Requirements . . . . .                                                              | 78         |
| 6.3.3    | Design Principles . . . . .                                                         | 78         |
| 6.4      | TEEOD: Design . . . . .                                                             | 79         |
| 6.4.1    | TEEOD Elements . . . . .                                                            | 80         |
| 6.5      | Implementation . . . . .                                                            | 81         |
| 6.5.1    | Overview . . . . .                                                                  | 82         |
| 6.5.2    | CA-TA Communication . . . . .                                                       | 82         |
| 6.5.3    | TA Management . . . . .                                                             | 83         |
| 6.5.4    | Enclave Cleanup Process . . . . .                                                   | 83         |
| 6.5.5    | Secure Storage . . . . .                                                            | 83         |
| 6.5.6    | Enclave's Memory Access . . . . .                                                   | 84         |
| 6.6      | Evaluation . . . . .                                                                | 84         |
| 6.6.1    | FPGA Hardware Costs . . . . .                                                       | 85         |
| 6.6.2    | Benchmarks . . . . .                                                                | 87         |
| 6.6.3    | Real-world applications: Bitcoin Wallet . . . . .                                   | 89         |
| 6.7      | Security Analysis . . . . .                                                         | 90         |
| 6.7.1    | Real-World TEE Vulnerabilities . . . . .                                            | 90         |
| 6.7.2    | Threats Out-of-Scope . . . . .                                                      | 93         |
| 6.8      | Discussion . . . . .                                                                | 93         |
| 6.8.1    | TEEOD's Hardware Constraints . . . . .                                              | 93         |
| 6.8.2    | Usability and Practical Applicability . . . . .                                     | 94         |
| 6.8.3    | Integration with Existing TEEs . . . . .                                            | 94         |
| 6.8.4    | Dynamic Partial Reconfiguration . . . . .                                           | 95         |
| 6.8.5    | Physical Attacks Countermeasures . . . . .                                          | 95         |
| 6.9      | Related Work . . . . .                                                              | 96         |
| 6.9.1    | Hardware-based TEEs . . . . .                                                       | 97         |
| 6.9.2    | FPGA-based TEEs . . . . .                                                           | 97         |
| 6.10     | Conclusion . . . . .                                                                | 99         |
| <b>7</b> | <b>TREE: Bridging the gap between reconfigurable computing and secure execution</b> | <b>100</b> |
| 7.1      | Introduction . . . . .                                                              | 101        |
| 7.2      | Background and Overview . . . . .                                                   | 104        |
| 7.2.1    | SoC FPGA Partial Reconfiguration . . . . .                                          | 104        |
| 7.2.2    | Trusted execution environments . . . . .                                            | 105        |
| 7.3      | Threat Model & Design Goals . . . . .                                               | 106        |
| 7.3.1    | Threat Model . . . . .                                                              | 106        |

|       |                                                    |     |
|-------|----------------------------------------------------|-----|
| 7.3.2 | Design Requirements . . . . .                      | 106 |
| 7.4   | Overview & Architecture . . . . .                  | 107 |
| 7.4.1 | TrustShell . . . . .                               | 108 |
| 7.4.2 | Dynamic tiles . . . . .                            | 108 |
| 7.5   | Design & Implementation . . . . .                  | 110 |
| 7.5.1 | TA development process . . . . .                   | 110 |
| 7.5.2 | Boot-time process . . . . .                        | 111 |
| 7.5.3 | Runtime process . . . . .                          | 111 |
| 7.5.4 | Supporting Globalplatform specifications . . . . . | 111 |
| 7.5.5 | TA attestation mechanism . . . . .                 | 112 |
| 7.5.6 | Challenges of integrating DPR with TEE . . . . .   | 112 |
| 7.6   | Evaluation . . . . .                               | 113 |
| 7.6.1 | Reconfigurable hardware costs . . . . .            | 114 |
| 7.6.2 | hTAs support . . . . .                             | 115 |
| 7.6.3 | sTAs and GPTAs support . . . . .                   | 116 |
| 7.6.4 | hybridTAs support . . . . .                        | 117 |
| 7.7   | TREE: Use Cases . . . . .                          | 118 |
| 7.7.1 | Access control authenticator . . . . .             | 118 |
| 7.7.2 | Trust Bitcoin wallet . . . . .                     | 119 |
| 7.8   | Discussion . . . . .                               | 121 |
| 7.9   | Security Analysis . . . . .                        | 122 |
| 7.9.1 | Theoretical security analysis . . . . .            | 122 |
| 7.9.2 | CVE mitigation analysis . . . . .                  | 123 |
| 7.10  | Related Work . . . . .                             | 124 |
| 7.11  | Conclusion . . . . .                               | 125 |

|                     |            |
|---------------------|------------|
| <b>Bibliography</b> | <b>126</b> |
|---------------------|------------|

## List of Figures

|    |                                                                                                                                                                                                                                 |    |
|----|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----|
| 1  | Relationship between the main scientific contributions. . . . .                                                                                                                                                                 | 8  |
| 2  | High-level architecture of an SRAM-based FPGA. Bitstream is stored in non-volatile memory and loaded as a bitstream into the configuration memory through the configuration module. . . . .                                     | 11 |
| 3  | Generic example of a SoC FPGA secure boot process. . . . .                                                                                                                                                                      | 14 |
| 4  | Types of bitstream configurations supported in modern FPGAs: full, partial, and nested reconfiguration. . . . .                                                                                                                 | 16 |
| 5  | High-level view of a TEE within a system, including trusted and untrusted components and resources, highlighted in blue and grey, respectively . . . . .                                                                        | 22 |
| 6  | Typical software architecture of a TEE system compliant with the GlobalPlatform specification. . . . .                                                                                                                          | 24 |
| 7  | Taxonomy provided by Cerdeira et al. [30] of vulnerabilities in commercial TrustZone-assisted TEEs. . . . .                                                                                                                     | 25 |
| 8  | Comparison of conventional TEE approaches. . . . .                                                                                                                                                                              | 33 |
| 9  | Overview of representative TrustZone-assisted FPGA-based TEEs. . . . .                                                                                                                                                          | 36 |
| 10 | Overview of representative FPGA-based TEEs frameworks for cloud deployments. . . . .                                                                                                                                            | 37 |
| 11 | Overview of the system architecture proposed in BYOTee [186]. . . . .                                                                                                                                                           | 39 |
| 12 | Comparison between standalone SRAM-based FPGA architecture and a SoC FPGA architecture. . . . .                                                                                                                                 | 56 |
| 13 | Overview of representative FPGA-oriented TEE designs. . . . .                                                                                                                                                                   | 65 |
| 14 | Generic GP-compliant TEE and its major components. The trusted elements are highlighted in blue, whereas the non-trusted components are shown in gray. . . . .                                                                  | 76 |
| 15 | TEEOD core design elements. The TEEOD architecture includes enclaves and TEEOD agents in light green, secure storage in purple, and trusted software components in blue. Non-secure components are represented in grey. . . . . | 79 |

|    |                                                                                                                                                                                                                                                                                                                                                                                                                                         |     |
|----|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----|
| 16 | Example of TEEOD implementation in an SoC FPGA. The TEEOD secure elements are instantiated, and the TAs are executed in the PL, while the Rich OS, the CAs, and the TEE supplicant run in the PS. . . . .                                                                                                                                                                                                                               | 81  |
| 17 | The following are the hardware costs for our synthesized TEEOD design in four configurations: TEEOD-ENCLV1, TEEOD-ENCLV2, TEEOD-ENCLV3, and TEEOD-ENCLV4, with varying numbers of enclaves. Green cells indicate TEEOD agents' resources, blue cells show enclave resources, and yellow cells denote enclave private memory. All other hardware resources are represented in teal, and the REE runs on the grey-highlighted PS. . . . . | 86  |
| 18 | Performance speedup comparison between TEEOD and OP-TEE for each GP Client API operation. . . . .                                                                                                                                                                                                                                                                                                                                       | 88  |
| 19 | Normalized performance speedup of TEEOD and OP-TEE for AES and SHA <i>xtests</i> . . . .                                                                                                                                                                                                                                                                                                                                                | 89  |
| 20 | Normalized performance speedup of TEEOD and OP-TEE of each Bitcoin wallet operation. . . . .                                                                                                                                                                                                                                                                                                                                            | 89  |
| 21 | Representative CVE-driven privilege escalation path in conventional TrustZone-assisted TEEs, from REE malware to Trusted Applications and ultimately the Secure Monitor. This multi-layer attack surface, illustrated by CVEs such as CVE-2015-6639 (TA hijack), CVE-2016-2431 (Trusted OS buffer overflow), and CVE-2015-8999 (cross-TA interference), is eliminated in TEEOD by design. . . . .                                       | 91  |
| 22 | Generic TEE components complying with the GlobalPlatform specification. The REE, shown in brown, contains the Rich OS (e.g., Linux) providing non-trusted services, and CAs that interact with the TEE. The TEE, highlighted in green and blue, includes trusted components such as the TEE OS and TAs. The TEE Client API and TEE Internal Core API facilitate secure communication and provide TEE services, respectively. . . . .    | 105 |
| 23 | Design Elements of the Trusted Reconfigurable Environment (TREE) within an SoC-FPGA. . . . .                                                                                                                                                                                                                                                                                                                                            | 108 |
| 24 | Workflow of the TREE framework at development-time (a), boot-time (b), and runtime (c). During development, the TREE framework allows system designers to synthesize/compile custom hTAs, software-based TAs, and hybrid TAs. During boot, the TrustShell is loaded to the PL. During runtime, TrustShell facilitates IPC between REE-TAs, manages and attests TAs, and provides TEE services. . . . .                                  | 109 |
| 25 | Resource utilization across multiple tile configurations. The plot illustrates how the usage of various FPGA logic elements scales with the increase of tiles, from 1 tile to 4 tiles. . . .                                                                                                                                                                                                                                            | 114 |
| 26 | Performance speedup comparison between TREE and vanilla OP-TEE for GP Client API operations: open session, close session, and invoke command. . . . .                                                                                                                                                                                                                                                                                   | 117 |
| 27 | Performance comparison of TA loading models in hybrid TAs. The graph illustrates the relative execution time of the runtime loading model versus the development-time loading model. . . . .                                                                                                                                                                                                                                            | 118 |
| 28 | Performance speedup comparison between hardware, software and hybrid versions of the same TA against the same TA running in OP-TEE. . . . .                                                                                                                                                                                                                                                                                             | 119 |

|    |                                                                                                                           |     |
|----|---------------------------------------------------------------------------------------------------------------------------|-----|
| 29 | Normalized performance speedup of TREE (vanilla and hybrid versions) and OP-TEE of each Bitcoin wallet operation. . . . . | 120 |
|----|---------------------------------------------------------------------------------------------------------------------------|-----|

## List of Tables

|    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |     |
|----|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----|
| 2  | Comparison of soft-MCUs features. . . . .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                | 13  |
| 3  | Comprehensive comparison and gap analysis of prior work on FPGA security and TEE systems. Key: RoT – Root of Trust; TP – Target Platform; F/T Res. – Fault/Tamper Resilience; I/A Det. – Intrusion/Anomaly Detection; TA-Cfg – TA Configuration; TA-Conf – TA Confidentiality; S-MT – Spatial Multi-Tenancy; T-MT – Temporal Multi-Tenancy; HW – Hardware TAs; SW – Software TAs; Hybrid – Hybrid (HW/SW) TAs; SS – Secure Storage; Encrypt. – Cryptographic Services; Attest. – Attestation Service; GP - GlobalPlatform TEE compliant. ● – full support; ◐ – partial or limited support; ○ – no support; N/A – not applicable or not reported. . . . . | 43  |
| 4  | Overview of existing survey papers addressing FPGA and TEE security. . . . .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             | 54  |
| 5  | Comparison of soft-core MCUs features. . . . .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           | 77  |
| 6  | Synthesized results of each hardware element on the Ultra96-V2. . . . .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  | 85  |
| 7  | Synthesized results of each TEEOD configuration on the Ultra96-V2. . . . .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               | 87  |
| 8  | Power consumption of TEEOD configurations on Ultra96-V2, compared to the OP-TEE baseline. Measurements taken at 25°C, 250 LFM airflow, with Vivado default power optimization. . . . .                                                                                                                                                                                                                                                                                                                                                                                                                                                                   | 87  |
| 9  | Comparison of prior related TEEs. <b>Cat.:</b> design style; <b>HW Mods:</b> requires hardware modification or non-standard SoCs; <b>FW Indep.:</b> does not require firmware to manage enclaves; <b>GP API:</b> compatible with GlobalPlatform APIs; <b>Multi-TA:</b> supports concurrent isolated TAs; <b>TZ Indep.:</b> does not rely on TrustZone; <b>HLS Indep.:</b> does not require translation of TAs using HLS. . . . .                                                                                                                                                                                                                         | 96  |
| 10 | Architectural Features of Commercial FPGAs Supporting DPR technology. . . . .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            | 104 |
| 11 | Synthesized results of TrustShell module on the Ultra96-V2. . . . .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      | 114 |
| 12 | Microbenchmarks for hTA configuration, clean-up, and TREE-hTA round-trip communication on a tile covering an entire FPGA clock region (954 kiB). . . . .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 | 115 |

|    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |     |
|----|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----|
| 13 | The hardware storage operation reads 8 kiB of data in 1 kiB chunks. The scenarios are: vanilla hardware application configured in the static PL (Native App.) and hTA configuring within TREE. . . . .                                                                                                                                                                                                                                                                                           | 116 |
| 14 | Mitigation analysis of selected CVEs with critical CVSS scores: "✓- CVEs in scope; "— out of scope; "?- insufficient information. . . . .                                                                                                                                                                                                                                                                                                                                                        | 123 |
| 15 | FPGA-based TEEs state of the art comparison. Key: RoT - Root-of-trust Primitive; dE - Dedicated Execution Enviroment per TA; EC - Execution Enviroment clean-up; ER - Execution Enviroment reuse; TAC - TAs Concurrency; hTAs - Support for hardware TAs; sTAs - Support for software TAs; GPTAs - GP-compliant TAs; SS - Secure Storage; CRYPT - Encryption services; ATT - Verification/attestation services; ●- fares positively; ●- fares partially positively; ○- fares negatively. . . . . | 124 |

# Acronyms

|               |                                                     |
|---------------|-----------------------------------------------------|
| <b>AaaS</b>   | Acceleration-as-a-Service                           |
| <b>AES</b>    | Advanced Encryption Standard                        |
| <b>AI</b>     | Artificial Intelligence                             |
| <b>AP-TEE</b> | Application-Processor Trusted Execution Environment |
| <b>API</b>    | Application Programming Interface                   |
| <b>APU</b>    | Application Processing Unit                         |
| <b>ASIC</b>   | Application-Specific Integrated Circuit             |
| <b>ASLR</b>   | Address Space Layout Randomization                  |
| <b>AXI</b>    | Advanced eXtensible Interface                       |
|               |                                                     |
| <b>BBRAM</b>  | Battery-Backed RAM                                  |
| <b>BRAM</b>   | Block RAM                                           |
|               |                                                     |
| <b>CA</b>     | Client Application                                  |
| <b>CLB</b>    | Configurable Logic Block                            |
| <b>CPS</b>    | Cyber-Physical System                               |
| <b>CPU</b>    | Central Processing Unit                             |
| <b>CSP</b>    | Cloud Service Provider                              |
| <b>CSU</b>    | Configuration Security Unit                         |
|               |                                                     |
| <b>DDRAM</b>  | Double Data Rate Random Access Memory               |
| <b>DFX</b>    | Dynamic Function eXchange                           |
| <b>DMA</b>    | Direct Memory Access                                |
| <b>DPR</b>    | Dynamic Partial Reconfiguration                     |
| <b>DRAM</b>   | Dynamic Random-Access Memory                        |
| <b>DRM</b>    | Digital Rights Management                           |

|              |                                            |
|--------------|--------------------------------------------|
| <b>DSP</b>   | Digital Signal Processor                   |
| <b>ECDSA</b> | Elliptic Curve Digital Signature Algorithm |
| <b>FaaS</b>  | FPGA-as-a-Service                          |
| <b>FF</b>    | Flip-Flop                                  |
| <b>FPGA</b>  | Field-Programmable Gate Array              |
| <b>FSBL</b>  | First-Stage Bootloader                     |
| <b>GPU</b>   | Graphics Processing Unit                   |
| <b>HDL</b>   | Hardware Description Language              |
| <b>HLS</b>   | High-Level Synthesis                       |
| <b>HMAC</b>  | Hash-based Message Authentication Code     |
| <b>IC</b>    | Integrated Circuit                         |
| <b>ICAP</b>  | Internal Configuration Access Port         |
| <b>IO</b>    | Input/Output                               |
| <b>IoT</b>   | Internet of Things                         |
| <b>IP</b>    | Intellectual Property                      |
| <b>IPU</b>   | Intelligence Processing Unit               |
| <b>JTAG</b>  | Joint Test Action Group                    |
| <b>LUT</b>   | Look-Up Table                              |
| <b>MAC</b>   | Message Authentication Code                |
| <b>MCU</b>   | Microcontroller Unit                       |
| <b>ML</b>    | Machine Learning                           |
| <b>NoC</b>   | Network-on-Chip                            |
| <b>NPU</b>   | Neural Processing Unit                     |
| <b>NVM</b>   | Non-Volatile Memory                        |
| <b>OEM</b>   | Original Equipment Manufacturer            |
| <b>OS</b>    | Operating System                           |
| <b>OtA</b>   | Over-The-Air                               |

|                 |                                               |
|-----------------|-----------------------------------------------|
| <b>PCAP</b>     | Processor Configuration Access Port           |
| <b>PL</b>       | Programmable Logic                            |
| <b>PMU</b>      | Power Management Unit                         |
| <b>PR</b>       | Partial Reconfiguration                       |
| <b>PS</b>       | Processing System                             |
| <b>PUF</b>      | Physically Unclonable Function                |
| <b>REE</b>      | Rich Execution Environment                    |
| <b>RM</b>       | Reconfigurable Module                         |
| <b>RO</b>       | Ring Oscillator                               |
| <b>RPU</b>      | Real-time Processing Unit                     |
| <b>RR</b>       | Reconfigurable Region                         |
| <b>RSA</b>      | Rivest–Shamir–Adleman                         |
| <b>RTL</b>      | Register-Transfer Level                       |
| <b>SCR</b>      | Secure Configuration Register                 |
| <b>SHA</b>      | Secure Hash Algorithm                         |
| <b>SoC</b>      | System on Chip                                |
| <b>SoC FPGA</b> | System-on-Chip FPGA                           |
| <b>soft-MCU</b> | Soft Microcontroller Unit                     |
| <b>SRAM</b>     | Static Random-Access Memory                   |
| <b>SSBL</b>     | Second-Stage Bootloader                       |
| <b>TA</b>       | Trusted Application                           |
| <b>TCB</b>      | Trusted Computing Base                        |
| <b>TEE</b>      | Trusted Execution Environment                 |
| <b>TEEOD</b>    | Trusted Execution Environments On-Demand      |
| <b>TLS</b>      | Transport Layer Security                      |
| <b>TPM</b>      | Trusted Platform Module                       |
| <b>TPU</b>      | Tensor Processing Unit                        |
| <b>TREE</b>     | Trusted Reconfigurable Execution Environments |
| <b>VM</b>       | Virtual Machine                               |
| <b>xPU</b>      | Specialized Processing Unit                   |

# **Part I**

## **Thesis Summary**

# Introduction

Today, our world relies on a growing network of connected devices known as the Internet of Things (IoT). More than 27 billion such devices were already online in 2025, and forecasts expect the figure to exceed 40 billion by 2030 [1]. These devices underpin critical infrastructure in energy [2], healthcare [3], finance [4], and transportation [5]. At the same time, they are becoming essential in nearly every aspect of daily life, granting them access to our most personal data. For the most part, this rapid proliferation promises great excitement and opportunity for engineers and society at large. However, as we place more trust in these interconnected systems, the emerging threats are on a scale never experienced before [6].

**What makes IoT devices so vulnerable?** Four factors dominate. The first is their massive scale. This means that even a 0.01% defect rate translates into millions of exploitable devices [7]. Second is the intimate nature of the data they collect; the Vastaamo cyberattack in October 2020, where a single hacker leaked psychotherapy records for blackmail, starkly illustrates the disturbing stakes [8]. Third, their tight coupling with the physical world turns digital compromise into real-world harm; as Cyber-Physical Systems (CPSs), a hacked device can manipulate infusion pumps, disable automotive safety features, or halt industrial processes [9]. Finally, their lifespan often stretches far beyond that of consumer electronics. While people replace personal computers every three to four years and smartphones even more frequently, IoT devices (e.g., smart automotives, data center gear, industrial machinery, Internet-connected power meters, and smart traffic lights) are expected to operate for decades [10]. We cannot assume periodic replacement will purge latent design flaws, nor can manufacturers reliably predict the hardware resources their devices will need to repel the attacks they will face many years down the line.

**IoT growing complexity.** The inherent vulnerabilities of IoT devices are further intensified by the increasing demand for high-performance, low-latency, and energy-efficient computation. To meet these demands, modern systems integrate a wide range of heterogeneous compute units, commonly refereed as Specialized Processing Units (xPUs), on the same die [11]. Beyond general-purpose Central Processing Unit (CPU) cores, today's System on Chips (SoCs) often include Graphics Processing Units (GPUs), Real-time Processing Units (RPU), Artificial Intelligence (AI) accelerators, e.g., Tensor Processing Units

---

(TPUs), Neural Processing Units (NPU), vision processors, and Digital Signal Processors (DSPs), and Field-Programmable Gate Arrays (FPGAs), each designed to handle specific workloads efficiently [12]. While this architectural heterogeneity provides flexibility and performance, it also broadens the attack surface. A vulnerability in one computing element can undermine the security of the entire system, especially when these computing elements share memory hierarchies, interconnects, and other critical resources. As a result, ensuring robust isolation and protection across increasingly complex hardware platforms has become a central challenge in securing the IoT landscape [13].

**Built-in countermeasures and their limitations.** To thwart known threats, manufacturers increasingly rely on hardware-assisted security features. These include cryptographic engines, secure boot mechanisms, remote attestation protocols, and Trusted Execution Environments (TEEs). Each of these mechanisms is designed to shrink the attack surface by either enforcing strict hardware boundaries or reducing the size and complexity of the Trusted Computing Base (TCB), i.e., the minimal set of hardware and software components that must be correct for the system to enforce its security properties. A smaller TCB statistically reduces the likelihood of exploitable vulnerabilities [14]. Yet, these countermeasures are often conceived under outdated assumptions. TEE systems, for instance, tend to focus on isolating software running on a general-purpose CPU, assuming it is the dominant locus of execution. However, as modern systems increasingly offload workloads to a variety of xPUs, this assumption no longer holds. Security mechanisms tightly coupled to a specific processor architecture leave the rest of the system unprotected, and vulnerabilities in auxiliary compute engines or interconnects can bypass otherwise robust defenses [15]. For example, attacks on GPUs have shown how shader programs can read or manipulate CPU data [16], reconfigured FPGAs can compromise co-located CPU cores in SoCs [17], flaws in AI accelerators can corrupt machine-learning models [18], and on-chip interconnects can be exploited for covert channels [19]. Worse, many hardware security features are implemented as static logic, which cannot be modified after deployment. When a flaw is discovered post-silicon (as was the case with Spectre [20] and Meltdown [21]), it becomes difficult or impossible to revise without replacing the hardware entirely.

**Toward a more adaptive hardware security model.** The persistence of hardware-level zero-day vulnerabilities has made it clear that reactive software patching is no longer sufficient. These issues, often rooted in microarchitectural behaviors, cannot be fully addressed without fundamentally changing how we think about security in modern computing architectures. In some high-assurance domains, vendors have pursued formally verified designs or complete system redesigns, but such approaches are costly, inflexible, and impractical in IoT domains. A more practical solution lies in making the hardware itself more adaptable. Rather than viewing post-deployment immutability as a given, emerging research advocates for reconfigurable or *patchable* hardware security frameworks-capable of enforcing, and more importantly, being reprogrammable, ranging from lightweight cryptography and remote attestation to secure boot and TEEs. This idea was first articulated by Ray et al. [22], who proposed extending reconfigurability beyond logic functions to include security enforcement.

This shift toward *patchable* security aligns with broader trends in heterogeneous computing. As systems grow more complex, security must no longer be an afterthought, nor can it be confined to a fixed TCB. Instead, it must become a first-class, adaptive property of the device. In this thesis, we explore how the reconfigurable fabric can serve as a foundation for trusted execution, not merely by isolating workloads at runtime, but by enabling system designers to reconfigure, revoke, and restore trust dynamically.

## 1.1 Motivation

### 1.1.1 Security Challenges in Heterogeneous Architectures

In the context of the IoT, the growing reliance on heterogeneous computing architectures amplifies existing security concerns [7, 23, 24]. IoT workloads no longer depend solely on CPUs; instead, they increasingly offload computation to other xPUs within the SoC [24–26]. This architectural diversity delivers the performance and efficiency required for modern systems, but it also fragments the security landscape. Each computing unit introduces distinct interfaces, memory hierarchies, and execution models, meaning that a single weakness can cascade across the entire device and compromise its trustworthiness [27–30].

Among the different xPUs, FPGAs stand out because they consist of reconfigurable logic blocks and interconnects that can be programmed to implement custom hardware designs after manufacturing [31, 32]. This flexibility allows a single device to be repurposed across applications or adapted to evolving requirements, while still delivering performance and energy efficiency beyond what general-purpose CPUs can offer [33–35]. Early adoption of standalone FPGAs was driven by domains such as telecommunications [36–38] and data centers [39], where their inherent parallelism and low-latency execution enabled custom accelerators to achieve substantial gains over conventional processors. They also became indispensable in hardware prototyping [40], where their ability to instantiate application-specific designs reduced the cost and risk of Application-Specific Integrated Circuit (ASIC) production. Building on these advantages, vendors moved toward integrating reconfigurable computing with conventional processing cores, resulting in System-on-Chip FPGA (SoC FPGA) architectures [25, 41]. In these settings, FPGAs provide not only post-deployment adaptability but also tightly coupled interaction with system components, making them especially suited for diverse and security-sensitive workloads such as cryptography [42], secure communications [43, 44], and real-time applications [38].

However, the same properties that make FPGAs attractive for a wide range of applications also introduce their own security challenges. The reconfigurability that enables post-deployment adaptability can be exploited by adversaries to insert malicious logic or alter functionality at runtime. The parallel execution capabilities that accelerate cryptography or data analytics can equally enable high-bandwidth side-channel attacks, where many concurrent operations leak exploitable information about secret states [45–48]. Tight coupling with CPU cores and system interconnects further erodes isolation: malicious or faulty logic can snoop on shared buses, disrupt memory accesses, or degrade the performance of co-located workloads [17, 49]. Resource sharing between trusted and untrusted modules raises the risk of covert

channels and denial-of-service, especially when reconfigurable regions are multiplexed among multiple tenants or applications [48, 50–52]. In unattended IoT deployments, where physical access and environmental attacks are also plausible, maintaining the confidentiality, integrity, and availability of FPGA-based accelerators becomes a persistent challenge [23, 53].

### 1.1.2 The Limitations of Current IoT Security Approaches

Despite the growing adoption of heterogeneous computing, most existing IoT security mechanisms remain rooted in assumptions designed for homogeneous, CPU-centric architectures. As system complexity grows, so too does the size of the TCB, inherently introducing implementation flaws that adversaries can exploit as vulnerabilities. A widely adopted security strategy to counteract this growth has been to partition the system into trusted and untrusted environments. The TEE enforces this boundary by maintaining the integrity of a minimal TCB while isolating it from the untrusted environment, commonly referred to as the Rich Execution Environment (REE), which encompasses the full range of Operating System (OS) services and user applications. TEEs systems rely fundamentally on hardware support to enable this compartmentalization. In principle, the underlying assumptions are that hardware is less likely to have vulnerabilities and that by minimizing the complexity of security-critical software, the number of expected vulnerabilities is statistically reduced. In practice, however, this promise has been repeatedly undermined.

The protection offered by TEEs is only as strong as the correctness of the underlying hardware and its trusted software components. As past exploits have shown, implementation bugs, side channels, and microarchitectural flaws can all undermine the guarantees of isolation [30]. Examples of TEE-enabling technologies include Intel SGX [54], AMD SEV [55], ARM TrustZone [56], and RISC-V APTEE [57], which are widely used in industries ranging from cloud computing to mobile devices. Over the past decade, TEE systems based on these CPU-centric TEE-enabling technologies have been compromised by hundreds of practical attacks. Qualcomm’s TrustZone implementation, for example, was shown by Beniamini to contain vulnerabilities that allowed unprivileged applications to escalate privileges and compromise the secure kernel [58–60]. Intel SGX, despite its strong isolation model, has been repeatedly broken by side-channel and microarchitectural attacks, such as cache-based leakage, speculative execution exploits, and fine-grained stepping attacks [61–64]. These breaches expose the risks of overly complex TEE systems, inconsistent access control, and implicit trust in hardware components.

Moreover, attempts to extend TEE technology beyond isolated computation contexts have proven challenging, raising concerns about their overall security effectiveness [30]. For instance, Arm TrustZone has been extended for use in SoC FPGAs, incorporating a mechanism that allows all reconfigurable hardware to identify the CPU’s security state. However, this integration has also exposed vulnerabilities, as demonstrated by attacks that exploit malicious modifications to reconfigurable logic, undermining processes, i.e., secure boot or manipulating security state propagation [49].

Finally, current TEEs are often implemented as static logic in silicon, making them difficult or impossible to update after deployment. When post-silicon flaws are discovered, as with Spectre [20] and Melt-down [21], manufacturers are forced to rely on costly software mitigations that only partially restore trust, and IoT devices with decade-long lifespans cannot practically be recalled or replaced. These limitations highlight the urgent need to rethink TEEs for heterogeneous architectures: rather than static CPU-centric extensions, they must evolve into adaptive, system-wide frameworks capable of extending trust across diverse compute elements while remaining *patchable* long after deployment [22].

## 1.2 Hypothesis

**Patchable trust as a viable path for securing heterogeneous systems.** This thesis advances the hypothesis that reconfigurable computing can serve not only as a means of accelerating computation but also as an enabler of *patchable trust*, a paradigm in which the hardware TCB can be dynamically updated and verified after deployment. Such adaptability would allow systems to respond to newly discovered threats without requiring hardware replacement or compromising normal operation. This hypothesis is motivated by the structural limitations of conventional trusted computing architectures. Existing TEEs typically rely on fixed, CPU-centric hardware extensions, whose immutability constrains their capacity to accommodate evolving adversarial models. This rigidity is particularly problematic in heterogeneous IoT systems that often remain operational for extended lifespans and thus encounter emerging vulnerabilities long after deployment. In this context, reconfigurable logic could provide a promising foundation for developing adaptive security mechanisms. Through controlled reconfiguration, mechanisms such as isolation boundaries, attestation policies, and cryptographic accelerators could be selectively modified or replaced, thereby sustaining the integrity of the TCB and reducing the system’s long-term exposure to compromise.

However, achieving *patchable trust* brings new challenges that must be addressed. The dynamic nature of reconfiguration inherently expands the system’s attack surface, raising concerns related to reconfigurable Intellectual Property (IP) integrity, configuration-time vulnerabilities, and the secure management of runtime updates. Additional challenges include preserving isolation between trusted and untrusted domains, maintaining consistent attestation after reconfiguration, and preventing misuse of the reconfiguration process for unauthorized access. To address these challenges, this thesis investigates both the opportunities and risks of using reconfigurable logic as a root of trust. The central research question explored herein is whether *patchable hardware* can provide verifiable and enduring security guarantees in heterogeneous computing environments for IoT systems.

## 1.3 Contributions

### 1.3.1 Research Questions

In this PhD thesis, we address the security challenges that arise from the increasing integration of reconfigurable hardware into heterogeneous computing platforms. Motivated by the limitations of conventional trusted computing and inspired by the potential of patchable hardware mechanisms, we investigate whether FPGA can be leveraged to provide reconfigurable trust computing to sustain the dynamic and long-term deployment nature of modern IoT systems. To structure this research, we aim to answer the following pivotal research questions:

- RQ1.** *How does reconfigurable hardware reshape the security landscape of heterogeneous systems, in terms of the emerging threats and opportunities it introduces?*
- RQ2.** *How can trust be configured and maintained within the FPGAs fabric to address evolving security threats?*
- RQ3.** *What architectural challenges and design trade-offs arise when enabling reconfigurable TEEs within the FPGA fabric?*

To tackle **RQ1**, we conduct a systematic survey on the security implications of reconfigurable computing. We focus on existing research that investigates how reconfigurable logic can be leveraged both to realize and to mitigate security threats within heterogeneous systems. We organize the surveyed works on FPGA-based adversarial attack vectors according to their primary targets. Likewise, we classify FPGA-based protection mechanisms according to their underlying security goals. Additionally, we review TEEs frameworks that integrate reconfigurable hardware into the TCB, examining how they extend isolation guarantees and enable dynamic trust establishment. This systematization provides a unified perspective on how reconfigurable hardware simultaneously broadens the attack surface and creates new opportunities for adaptive, hardware-level defense. Finally, we outline future directions aimed at embracing reconfigurable computing not as a liability, but as a foundation for adaptable and verifiable security architectures.

To address **RQ2**, we first define a set of design principles derived from the vulnerabilities and limitations found in mainstream TEEs. Guided by these principles, we design and implement a prototype framework that leverages reconfigurable logic to instantiate the fundamental TEE primitives, including isolated execution environments, cryptographic services, secure storage, trusted time, and controlled communication channels with the untrusted domain. Finally, we evaluate the proposed framework by deploying unmodified software Trusted Applications (TAs) compliant with the GlobalPlatform TEE specifications, demonstrating that reconfigurable hardware can effectively support trusted execution while maintaining compatibility with established TEE standards.

To address **RQ3**, we investigate how Dynamic Partial Reconfiguration (DPR) can be elevated from a device management capability to a trusted execution service. We start by analyzing the assumptions underlying vendor-provided secure reconfiguration flows and identify their dependence on a monolithic trust model, where static and partial regions share the same privilege level. From this analysis, we establish the need for a fine-grained reconfiguration model that enforces controlled access to FPGA resources while preserving isolation and minimizing the TCB. We then design and implement a reconfigurable TEE framework that enables the dynamic deployment of hardware, software, and hybrid TAs. Finally, we evaluate the framework through theoretical security analysis and experimental validation on a commercial SoC FPGA, assessing its ability to mitigate practical threats and its feasibility for deployment in real-world scenarios.

### 1.3.2 Main Contributions

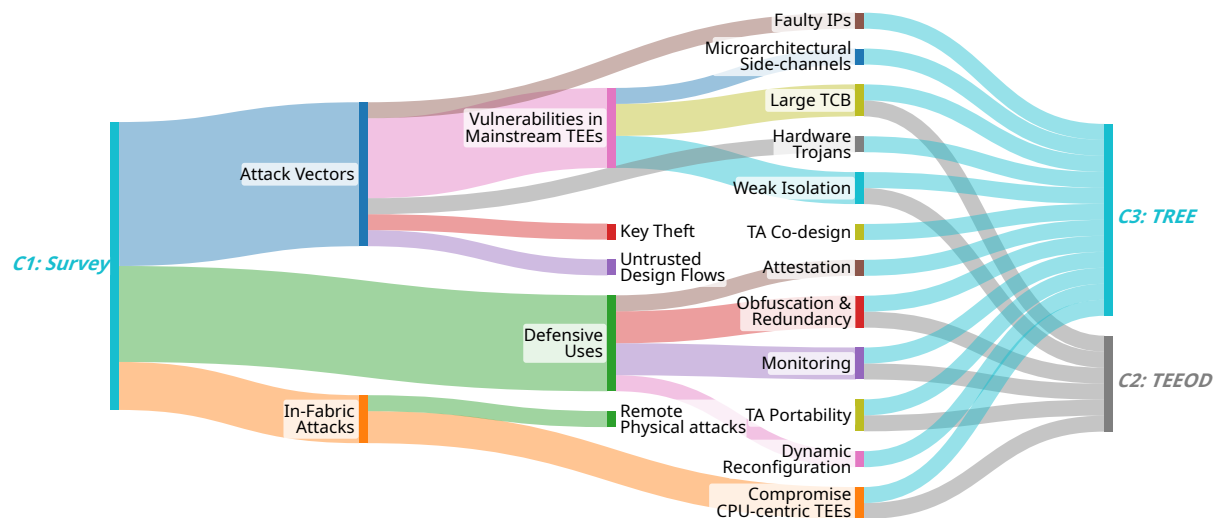


Figure 1: Relationship between the main scientific contributions.

Figure 1 summarizes the three main contributions of this PhD thesis [65–67]. Together, these contributions advance the vision of *patchable hardware*, demonstrating how reconfigurable computing potential in securing IoT systems. The first contribution (C1: *Survey*), presented in Chapter 5, provides a systematic study of the security implications of reconfigurable computing. It consolidates fragmented literature by examining both adversarial and defensive uses of the reconfigurable fabric. The study identifies the principal attack vectors through which adversaries can gain access to the reconfigurable logic, including those arising from vulnerabilities in mainstream TEEs. Once this access is achieved, new classes of in-fabric attacks can be mounted. The study also reviews countermeasures implemented directly in the fabric. The second contribution (C2: *TEEOD*), detailed in Chapter 6, introduces a novel TEE approach that enables dedicated and isolated execution environments within the reconfigurable fabric. Trusted Execution Environments On-Demand (TEEOD) addresses structural limitations of CPU-centric TEEs by relying exclusively

on reconfigurable logic to implement TEE services and manage TAs, while remaining fully compatible with existing GlobalPlatform TEE standards. The third contribution (C3: *TREE*), detailed in Chapter 7, presents Trusted Reconfigurable Execution Environments (TREE), a unified TEE framework that elevates DPR from a conventional device management feature to a trusted execution service. TREE mitigates in-fabric attacks by enabling secure and reconfigurable enclaves that can be instantiated, patched, or revoked entirely within the programmable logic at runtime, without requiring system reboot or software intervention. This contribution demonstrates how reconfiguration can be transformed from an operational mechanism into a security primitive, supporting *patchable* trust establishment and continuous hardware-level protection.

**Beneath the fabric: A survey on threats and opportunities in reconfigurable systems.** This survey presents a systematic analysis of the security implications and opportunities of reconfigurable computing. Rather than viewing FPGAs as peripheral accelerators, it focuses on works that exploit the reconfiguration logic itself, either as an attack surface or as a foundation for building security mechanisms. The study identifies how adversaries can inject or manipulate malicious circuits during configuration or runtime, leading to in-fabric attacks that compromise neighboring logic or trust boundaries while remaining within authenticated bitstreams. It further systematizes countermeasure mechanisms implemented directly in the reconfigurable fabric, including obfuscation, redundancy, runtime monitoring, attestation, and virtualization frameworks that enhance isolation and integrity. The survey also reviews emerging FPGA-based TEEs that integrate the reconfigurable fabric into the TCB, examining how they extend isolation guarantees and enable dynamic trust computing. Finally, it outlines future research directions toward reconfiguration-driven security architectures, positioning reconfigurable hardware not as a liability but as a foundation for adaptable and verifiable trust in modern computing platforms.

**Rethinking Trusted Execution Environments in the Age of Reconfigurable Computing.** This work introduces a novel TEE approach, TEEOD that proposes establishing TEEs directly within the FPGA fabric. Instead of relying on fixed hardware extensions or dedicated external secure elements, the proposed TEE framework relies on the FPGA fabric itself to provide dedicated enclaves with hardware-enforced isolation. Each enclave integrates: i) a dedicated Soft Microcontroller Unit (soft-MCU) executing a minimal TEE firmware alongside the software TA; ii) private on-chip memories; iii) an untrusted shared buffer to communicate with REE; and iv) optional hardware accelerators for cryptographic operations such as Advanced Encryption Standard (AES) and Secure Hash Algorithm (SHA). Sessions with the TAs are opened, invoked, and closed through dedicated reconfigurable logic that operates independently from the enclaves within the fabric. TEEOD demonstrates that relying solely on reconfigurable logic to provide isolated, dedicated, and standards-compliant TEEs is both feasible and practical. This was validated through the execution of an unmodified GlobalPlatform-compliant Bitcoin wallet TA. Furthermore, by operating entirely within the reconfigurable fabric, TEEOD avoids key limitations of mainstream TEEs, such as rigid trust boundaries, shared privilege models, and dependence on vulnerable processor extensions.

**TREE: Bridging the gap between reconfigurable computing and secure execution.** This work presents TREE, a reconfigurable TEE framework that integrates DPR as a TEE primitive. While TEEOD and other FPGA-based TEEs show that trusted environments can be fully implemented within the FPGA fabric, TREE focuses on bringing runtime flexibility to trusted execution. FPGAs are well known for enabling hardware-like acceleration, offering significant opportunities for performance-sensitive TAs, where critical routines can be offloaded to custom hardware modules or implemented as hybrid designs that balance flexibility and efficiency. However, enabling dynamic trust reconfiguration at runtime introduces several challenges: i) it shifts the trust model from static, pre-provisioned logic to runtime-managed reconfiguration; ii) it requires granting TAs controlled access to FPGA resources (e.g., DSPs, clocking and routing networks, or power-sensitive interconnects), without compromising isolation; and iii) demands secure handling of partial reconfigurations within a heterogeneous trust domain. TREE addresses these challenges by introducing a dual-layer architecture that separates static trusted management logic from reconfigurable enclaves, ensuring that dynamic reconfiguration can occur securely and independently across multiple trusted enclaves. This design enables fine-grained adaptability while preserving the integrity, isolation, and trust guarantees of conventional TEEs. This work also includes a security analysis and experimental evaluation, showing that TREE can mitigate practical vulnerabilities found in commercial TEEs, highlighting its potential for deployment in real-world heterogeneous systems.

## 1.4 Thesis Outline

This thesis is organized into two interconnected parts. Part ?? establishes the theoretical and contextual foundation of this work. Chapter 2 introduces core concepts essential to understanding FPGA-based systems and their security challenges, including an overview of FPGA architectures (Section 2.1) and TEEs (Section 2.4). Chapter 3 follows with a summary analysis of related work on security mechanisms proposed to counter FPGA-specific threats, trust architectures for modern heterogeneous systems, and existing trends in FPGA-based trust solutions. Chapter 4 concludes the foundational analysis by synthesizing key insights, identifying unresolved challenges in FPGAs security, and outlining promising directions for future research.

Part II presents the main contributions of this PhD, each corresponding to a publication that forms the core of this work. Chapter 5 provides a survey of the security implications of reconfigurable computing, consolidating fragmented literature by examining both adversarial and defensive uses of the reconfigurable fabric. Chapter 6 introduces TEEOD, a novel TEE framework that enables isolated execution environments entirely within the reconfigurable fabric, addressing structural limitations of CPU-centric TEEs while maintaining compatibility with GlobalPlatform standards. Finally, Chapter 7 presents TREE, a unified TEE framework that leverages DPR to elevate reconfiguration from a device management feature to a trusted execution service. By enabling fully adaptable and *patchable* trusted environments, TREE addresses the central hypothesis of this thesis.

## Background

This chapter provides a comprehensive overview of the concepts and technologies essential for understanding the work presented in this thesis. Section 2.1 introduces FPGA technology and presents a system-level view of a representative SoC FPGA architecture. Section 2.2 explores the growing adoption of FPGAs across heterogeneous computing environments, highlighting their role in emerging application domains such as edge computing and cloud infrastructure. Section 2.3 characterizes FPGA-specific vulnerabilities and attack vectors that go beyond secure reconfiguration. Section 2.4 introduces the concept of the TEE, with a focus on the Arm TrustZone architecture and the GlobalPlatform specifications. Finally, Section 2.5 discusses prominent vulnerabilities identified in existing TEE implementations.

### 2.1 Field-Programmable Gate Arrays

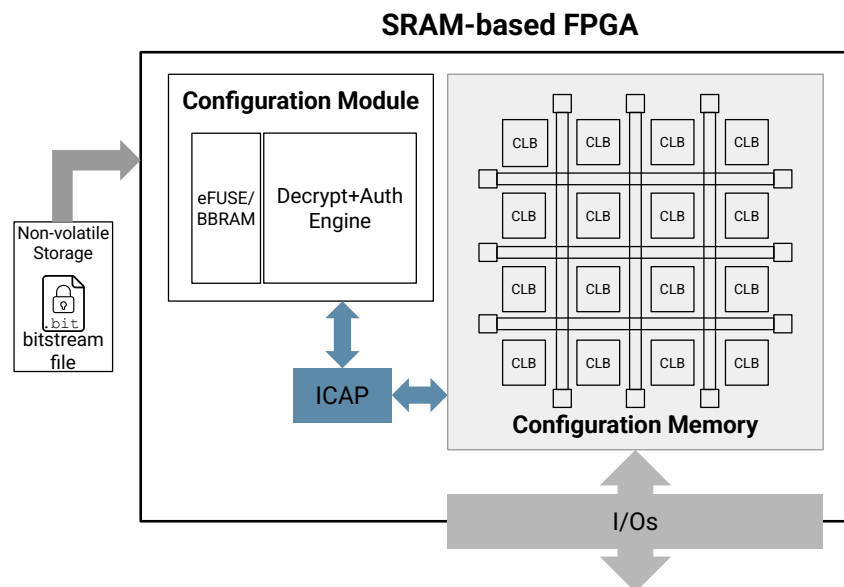


Figure 2: High-level architecture of an SRAM-based FPGA. Bitstream is stored in non-volatile memory and loaded as a bitstream into the configuration memory through the configuration module.

**What is an FPGA?** FPGA is an Integrated Circuit (IC) designed to be reconfigured by the user after manufacturing; hence, it is field programmable. At its core, an FPGA consists of an array of Configurable Logic Block (CLB) and a hierarchy of reconfigurable interconnects that allow these blocks to be wired together to implement various functions. These CLB consist of Look-Up Table (LUT) and Flip-Flop (FF) that, combined, create more complex logic functions (e.g., finite state machines or multi-bit arithmetic operations). Additionally, FPGA includes Block RAM (BRAM) for on-chip memory storage, DSP for high-performance arithmetic operations (e.g., multiply-accumulate for signal processing), and Input/Output (IO) blocks for interfacing with external devices. The behavior of the blocks is ultimately defined by a binary configuration file, known as the *bitstream*, which specifies how logic blocks and interconnects are programmed. Designers usually describe the intended hardware functionality in a Hardware Description Language (HDL), such as Verilog, SystemVerilog, or VHDL. Vendor toolchains (e.g., AMD Vivado, Intel Quartus) then synthesize, map, place, and route this HDL description to automatically generate the bitstream. There are two predominant FPGA technologies on the market: Static Random-Access Memory (SRAM)-based and flash-based. The leading vendors, AMD, Intel, Microsemi and Lattice, primarily offer SRAM-based devices. Figure 2 depicts the generic architecture of an SRAM-based device, showing the configuration path.

**The Popularity of SRAM-Based FPGA.** Because SRAM cells are volatile, SRAM-based FPGAs require an external Non-Volatile Memory (NVM) to store configuration data across power cycles, whereas flash-based devices can retain their configuration indefinitely. This volatility also means that the bitstream must be reloaded at every boot, exposing it to potential security risks. Despite this drawback, SRAM-based FPGAs remain the dominant choice, even in security-sensitive applications [68–70]. Their continued popularity is driven by several key advantages: i) manufacturing simplicity: their interconnect, configuration memory, and switches can be realized using only transistors and wires, making them easier to fabricate; ii) process node advantage: they typically adopt newer process nodes earlier than competing technologies, often staying two generations ahead; and iii) field update capability: they naturally support Over-The-Air (OtA) updates, much like software, which is highly attractive for long-lived and evolving systems.

### 2.1.1 General-purpose Computing on FPGA

The specific functionalities configured within the reprogrammable fabric are commonly implemented using reusable design modules known as IP cores. These IPs are not limited to custom application-specific designs; soon after the adoption of FPGA technology in broader computing domains, developers began implementing *soft* processors to handle general-purpose tasks [71]. These processors are referred to as *soft* because they are fully synthesized designs on the FPGA fabric. In contrast, *hard* ones refer to pre-designed, fixed-function ASIC. While *soft* processors are generally less efficient than *hard* cores in terms of performance and power consumption, they provide significant advantages in flexibility, rapid prototyping, and domain-specific adaptation, since they can be instantiated, modified and tested without the cost and

| Soft-MCU      | Architecture | Bit   | Pipeline | Area (LEs) | Open-source | TRM  |
|---------------|--------------|-------|----------|------------|-------------|------|
| Taiga         | RISC-V       | 32/64 | 5        | 5000       | •           | [72] |
| LEON3         | SPARC-v8     | 32/64 | 7        | 5000       | •           | [74] |
| OpenRISC 1200 | OpenRISC     | 32    | 5        | 6000       | •           | [75] |
| MicroBlaze    | MicroBlaze   | 32    | 3/5      | 1324       | ◦           | [73] |
| Nios II/f     | Nios II      | 32    | 6        | 1800       | ◦           | [76] |
| Cortex-M1     | Armv6        | 32    | 3        | 2600       | ◦           | [77] |
| PicoBlaze     | PicoBlaze    | 8     | no       | 192        | ◦           | [78] |
| LatticeMico8  | LatticeMico8 | 8     | no       | 200        | •           | [79] |

Table 2: Comparison of soft-MCUs features.

lead time of fabricating a dedicated ASIC. Moreover, these designs are often FPGA-agnostic, allowing deployment across different hardware platforms without major redesign effort. Both commercial vendors and open-source communities provide a wide range of implementations, from minimalistic processors with limited instruction sets to feature-rich cores with pipelines, caches, and multiprocessor support.

As FPGAs gained traction in embedded systems, soft processors evolved into soft-MCUs, integrating not only the processing core but also Microcontroller Unit (MCU) features such as timers, interrupt controllers, and peripheral interfaces. A notable case is RISC-V, a modern open-source and royalty-free instruction set architecture. Since commercial RISC-V chips were not yet available in its early years, FPGA soft-cores became the primary way to implement and experiment with the ISA, enabling researchers and developers to prototype designs, build toolchains, and evaluate extensions [72]. In parallel, FPGA vendors provided proprietary solutions tailored to their ecosystems. AMD's MicroBlaze became the most widely adopted, thanks to its seamless integration with design toolchains, flexible configuration options, and a mature ecosystem of IP blocks and drivers bundled with nearly every AMD development board [73]. Table 2 highlights representative soft-MCU designs and their availability as open-source implementations.

### 2.1.2 Rise of SoC FPGA Architectures

For many years, FPGAs have depended on either external off-chip Application Processing Units (APUs) or soft-MCUs instantiated in the fabric to handle general-purpose computation and system control. However, as pointed out, soft-MCUs introduce performance limitations because reprogrammable logic is inherently less efficient than a fixed ASIC implementation. Off-chip processors, on the other hand, provide higher performance but suffer from communication overhead and increased latency, since every interaction between the FPGA and the processor are external. As reconfigurable devices began to support increasingly complex applications, the need for integrated solutions that combine high-performance processing with tightly coupled reconfigurable logic became clear.

To address this demand, FPGA vendors explored integrating processors with FPGA fabric as early as the 2000s, but it only gained wide adoption in 2011 when Xilinx (now part of AMD) introduced the Zynq-7000 architecture [80]. The architecture combined a dual-core ARM Cortex-A9 processing system with reconfigurable logic, memory controllers, and common peripherals on a single die. Now, modern SoC

FPGA architectures have since evolved to incorporate even more heterogeneous components, including GPUs, DSPs, and custom hardware accelerators such as AI chips. As an example, AMD's new Versal AI Edge Gen 2 [12] integrates more RPU than general-purpose CPUs, complemented by AI Engines and DSP engines tailored for AI, vision, radar, and LiDAR processing.

Today, the terminology used to describe SoC FPGA architectures is largely shaped by AMD, given their dominant role in the commercial adoption of these platforms. According to AMD's current terminology, a generic SoC FPGA architecture is divided into two primary domains: the Processing System (PS) and the Programmable Logic (PL). The PS includes fixed-function components such as multicore CPU, xPUs, Configuration Security Unit (CSU), and various standard peripherals, while the PL provides the reconfigurable fabric. In parallel, other vendors have explored embedding smaller reconfigurable fabrics directly into ASIC or SoC designs as embedded FPGAs. These embedded FPGAs bring FPGA-like adaptability into otherwise fixed-function chips, enabling post-silicon updates or application-specific accelerators without requiring a full standalone SoC FPGA. Most commercial SoC FPGAs adopt Arm-based processor subsystems, typically with TrustZone extensions as part of their trusted execution foundation, discussed further in Section 2.4.1. To enable tight coupling between domains, the PS and PL are connected through high-performance interfaces, among which the Advanced eXtensible Interface (AXI) bus is the most widely used in AMD SoC FPGAs architectures.

### 2.1.3 Secure Reconfiguration Process

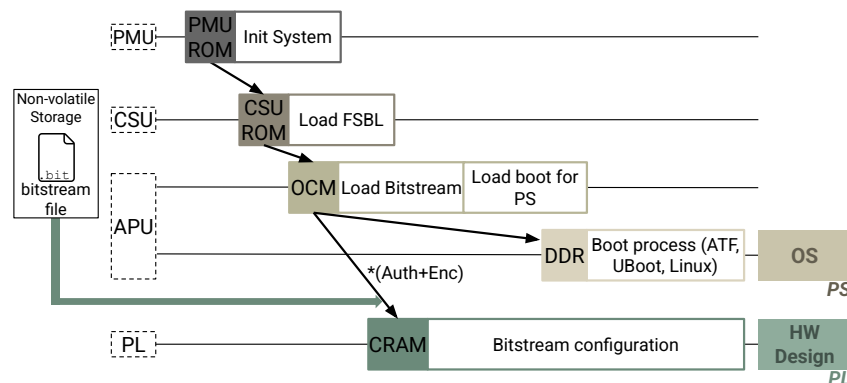


Figure 3: Generic example of a SoC FPGA secure boot process.

To ensure that the reconfigurable logic is treated as an integral system component subject to the same root-of-trust guarantees and not as a loosely attached accelerator, the reconfiguration process has been integrated into the secure boot routine of SoC FPGAs. In this setup, both the fixed PS and the PL are initialized under a common chain of trust. Similar to a standard secure boot, vendors implement a multi-stage procedure that combines immutable elements with user-defined programmable stages, so that configuration data, firmware, and bitstreams are all authenticated and decrypted before use. Immutable

elements include the platform management functions hardwired in the Power Management Unit (PMU), the boot code of the CSU. Together these elements manage early initialization and root key provisioning. The first programmable stage begins with the First-Stage Bootloader (FSBL) and extends to the initialization of the APU, peripheral subsystems, and the PL.

Figure 3 summarizes the secure boot process, highlighting how authentication and decryption steps are interleaved with processor and fabric initialization. At power-on, the PMU performs minimal initialization, i.e., power sequencing, reset management, and clock setup, before handing control to the CSU. The CSU then fetches the FSBL from external non-volatile storage (e.g., QSPI flash, SD card, or eMMC) into on-chip memory. In secure-boot mode, this process is protected through hardware-accelerated decryption and authentication, typically using AES-based encryption and optional Hash-based Message Authentication Code (HMAC) or signature verification with keys provisioned in eFUSE or Battery-Backed RAM (BBRAM). For example, AMD's Zynq UltraScale+ integrates AES-GCM decryption and Rivest–Shamir–Adleman (RSA)-4096 signature checking with SHA-3 hashing [81], while Intel's Stratix and Arria devices employ similar AES-GCM schemes with device-specific keys fused into the silicon [82]. This process ensures that only authenticated and confidential configurations are admitted into the system.

The FSBL is the earliest user-modifiable stage to execute on the main APU. It performs comprehensive system initialization, including enabling caches and memory controllers, configuring clocks and peripherals, and preparing higher-level boot stages such as the Second-Stage Bootloader (SSBL) or the OS. Crucially, the FSBL is also responsible for loading the bitstream for the PL. In secure-boot configurations, the bitstream is delivered together with its cryptographic hash; the CSU's authentication logic decrypts the bitstream, verifies its integrity, and only then releases it to configure the fabric. The exact cryptographic primitives used for this process vary across device generations. The Zynq-7000 family, for example, relies on AES-CBC for bitstream encryption and uses an HMAC for integrity verification [80]. By contrast, the more recent Zynq UltraScale+ devices strengthen these guarantees with authenticated encryption through AES-GCM [81]. Although these mechanisms provide a robust foundation for confidentiality and integrity, they follow a monolithic model in which a single device-specific key protects all configuration data. Consequently, every bitstream, whether partial or full, depends on the same root of trust.

#### 2.1.4 Partial Reconfiguration

Once an FPGA has been configured with an initial bitstream, its functionality is typically fixed until the next full reconfiguration. To overcome this limitation, modern FPGAs integrate DPR technology which allow Partial Reconfiguration (PR) of selected pre-defined Reconfigurable Regions (RRs) without disrupting the operation of the remaining logic. Both full and partial bitstreams are delivered through the same configuration interfaces, e.g., Joint Test Action Group (JTAG), SelectMAP, Internal Configuration Access Port (ICAP), or Processor Configuration Access Port (PCAP), in the case of SoC FPGAs. Figure 4a illustrates the configuration process for a full bitstream, which overwrites the entire contents of the FPGA configuration memory. Conversely, Figure 4b depicts the partial reconfiguration process, where only specific RR

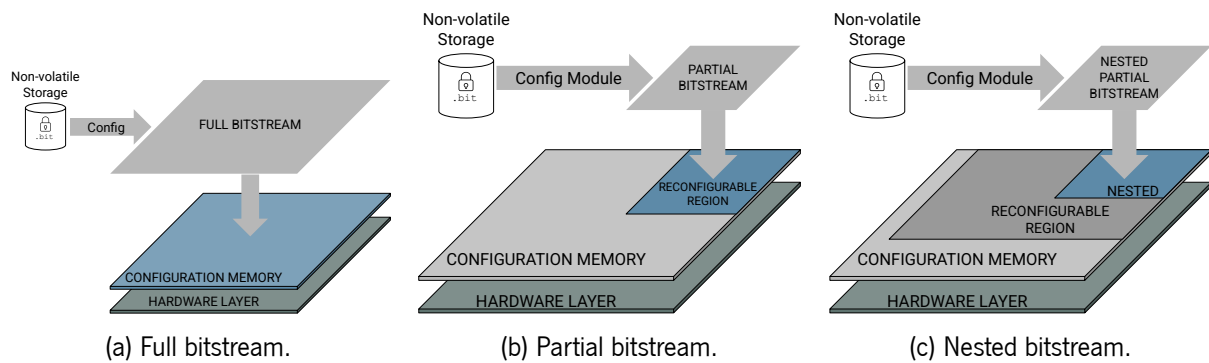


Figure 4: Types of bitstream configurations supported in modern FPGAs: full, partial, and nested reconfiguration.

updated, allowing new functionality to be introduced at runtime without interrupting the operation of the static design. More advanced reconfiguration schemes, such as hierarchical or *nested* partial reconfiguration, enable the subdivision of a single RR into smaller subregions that can be independently updated, as depicted in Figure 4c. Within a PR system, these reconfiguration mechanisms rely on different types of bitstreams, each serving a complementary purpose in managing and updating the device configuration:

- **Full bitstream:** Configures the entire FPGA fabric. It is typically employed during device initialization or following a system reset.
- **Partial bitstream:** Shares the same structural format as a full bitstream but applies exclusively to a defined subset of the device. Its size is proportional to the spatial extent of the targeted RR. Loading a partial bitstream replaces the current Reconfigurable Module (RM) with a new one while preserving the operation of the *static* region.
- **Blanking bitstream:** A specialized form of partial bitstream that removes the functionality of a previously instantiated module, effectively deactivating the corresponding region. The RR then behaves as a logical black box. In AMD documentation, this construct is referred to as a *greybox* module [83]. Blanking bitstreams are generated in the same way as standard partial bitstreams and are typically used to unload or disable a module prior to reconfiguration.
- **Clearing bitstream:** An auxiliary bitstream used in certain FPGAs (e.g., AMD UltraScale devices, though not UltraScale+) to prepare a RR for subsequent reconfiguration. It applies a global signal mask and disables clocks within the targeted region, placing it in an idle state. Although it does not modify the functional logic, it ensures a safe transition between modules. Clearing bitstreams are typically much smaller than standard partial bitstreams (often below 10% of their size) and are not considered full configurations, as their function is limited to disabling and isolating the region [83].

**PR Benefits.** PR significantly extends the flexibility of FPGAs by enabling logic updates during runtime with minimal system disruption. This capability is particularly advantageous in systems requiring

high availability or real-time responsiveness, as hardware modules can be dynamically replaced without resetting the device or halting its operation. Furthermore, the size of a partial bitstream scales proportionally with the area of the targeted RR, leading to shorter reconfiguration times and reduced configuration bandwidth. These characteristics facilitate efficient time-multiplexing of hardware resources and enable cost-effective implementation of complex designs on smaller devices [84]. Both major FPGA vendors have incorporated PR into their design toolchains: AMD provides the Dynamic Function eXchange (DFX) flow, while Intel offers the PR Design Flow [83, 85].

**Design Methodology.** Developing a DPR system requires deliberate partitioning of the design into *static* and reconfigurable regions, accompanied by explicit *floorplanning* to ensure proper isolation between them. Although modern synthesis tools automate much of the timing optimization [31], floorplanning remains essential to preserve functional independence and routing constraints: while signals from the static region may traverse a RR under controlled conditions, the reverse routing direction is typically restricted. Two design flows are commonly employed. The *top-down* approach synthesizes the entire system, including the static logic and all possible RMs, as a single entity, enabling global optimization at the cost of design complexity. In contrast, the *bottom-up* (out-of-context) approach treats the static region as a black box, allowing each RM to be developed, verified, and reused independently, which simplifies modular and scalable design.

**Nested Reconfiguration.** Recent advances in FPGA design have introduced nested partial reconfiguration, wherein a single RR can be hierarchically subdivided into smaller subregions that may be reconfigured independently [83]. This hierarchical capability enables finer-grained adaptability, allowing designers to modify only portions of a region while preserving the operation of the remaining logic. It further supports composable design strategies, in which RMs of different sizes and functionalities can coexist or be dynamically exchanged at runtime. As illustrated in Figure 4c, a reconfigurable region can be recursively partitioned to permit localized updates; however, each additional level of hierarchy increases the complexity of placement, routing, and bitstream management. Consequently, practical designs typically limit nesting to one or two levels, with three considered the feasible upper bound [84].

## 2.2 Reconfigurable Computing Applications

Over the last decades, the role of reconfigurable hardware has expanded significantly. In the early 2000s FPGAs were adopted primarily as prototyping platforms that allowed design teams to validate Register-Transfer Level (RTL) code before committing to an ASIC [40]. Today, this view is outdated. Instead of serving only as peripheral accelerators, reconfigurable computing is now entrusted with a growing share of computation that once relied exclusively on conventional processors. While they cannot provide the same degree of programmability as CPUs or GPUs, they achieve performance closer to ASICs without incurring the high cost and long development cycles of fixed-function hardware. As a result, FPGAs have become

a practical platform for diverse workloads, spanning from large-scale deployments in cloud datacenters to latency- and energy-sensitive tasks at the IoT edge. In the following, we discuss how reconfigurable hardware addresses the requirements of each domain.

### **2.2.1 FPGA on the Cloud**

In datacenter environments, the adoption of FPGAs is driven by the need to accelerate data-intensive workloads that exceed the efficiency of general-purpose CPUs and are not always well suited to GPUs. While GPUs remain predominant in massively parallel domains such as graphics rendering and large-scale model training, FPGAs offer fine-grained parallelism and near-I/O computational pipelines capable of achieving lower latency and higher energy efficiency for specific operators [39, 86]. Their deployment in domains such as network encryption (e.g., TLS/IPsec offload) [87–89], key-value stores [90, 91], and exploratory blockchain and cryptomining pipelines (e.g., SHA-256/PoW offload) [92, 93] illustrates ongoing debates regarding the boundaries of FPGA superiority relative to other accelerators.

Beyond raw computational acceleration, FPGAs in cloud infrastructures serve as flexible and shared hardware resources. Cloud Service Providers (CSPs) increasingly offer these devices either as managed acceleration services or as directly accessible virtualized instances, balancing accessibility with adaptability to evolving application domains. Two predominant service models have emerged: Acceleration-as-a-Service (AaaS), and FPGA-as-a-Service (FaaS). In AaaS, the provider preconfigures FPGAs for selected workloads and offers them transparently to clients, ensuring accessibility without requiring hardware expertise [39]. While this model simplifies adoption by hiding hardware complexity, it places full control in the provider’s hands, raising security implications: tenants must trust that the provider enforces bitstream integrity, prevents malicious updates, and protects proprietary accelerator designs. Conversely, in the FaaS model, tenants obtain direct programming access to their assigned devices, enabling domain-specific acceleration and rapid iteration as workloads evolve. Major providers such as Amazon, Huawei, and Alibaba currently support this model through dedicated FPGA instances.

However, the security implications are considerably more challenge in FaaS model. Since tenants supply their own bitstreams, providers must enforce strict validation and isolation mechanisms to prevent malicious or faulty configurations from compromising the device, accessing privileged resources, or degrading service quality. Typically, designs are submitted for provider-side verification prior to deployment, ensuring safety but requiring tenants to trust the provider with their proprietary IP. Maintaining robust isolation between user logic and the fixed system shell (e.g., PCIe, memory controllers, and network interfaces) remains essential, as faults or adversarial designs could propagate beyond a tenant’s partition. Beyond these commercial models, academic research has explored multi-tenant virtualization through PR, where multiple clients dynamically share a single physical device [50]. While this promises higher utilization, it also raises significant challenges for isolation and security.

### 2.2.2 IoT Edge Applications

Whereas cloud infrastructures centralize computational resources for large-scale analytics and interactive services where latency is tolerable, edge platforms must operate adjacent to sensors and devices, ensuring deterministic and real-time performance for applications such as voice assistants, biomedical wearables, and autonomous sensing nodes. General-purpose CPUs often fail to deliver this level of timing predictability, and while GPUs offer high throughput, they typically incur higher power consumption and exhibit variable tail latencies. FPGAs address this gap by enabling deeply pipelined, near-sensor processing that minimizes data movement and ensures consistent response times.

Beyond performance, adaptability is a defining requirement for edge devices. Conventional CPU and GPU architectures adapt only at the software level, often incurring performance overhead and energy inefficiency. In contrast, FPGAs support runtime adaptability through partial reconfiguration, enabling hardware accelerators to be dynamically exchanged according to workload demands. Frameworks such as mIoT [94] and ZyPR [95] demonstrate the feasibility of dynamic reconfiguration without rebooting, allowing long-lived edge systems to evolve with application requirements. Moreover, FPGAs frequently surpass GPUs in energy efficiency. For instance, an FPGA-based KV260 platform achieved comparable inference accuracy to an NVIDIA Jetson Nano while consuming approximately five times less energy per inference [26]. Surveys consistently identify FPGAs as an energy-optimized design point for IoT devices, offering a favorable trade-off between flexibility, throughput, and power consumption [96, 97].

Nevertheless, the security implications at the edge mirror, and in some cases intensify, those expected in the cloud. IoT devices frequently operate in untrusted environments, rendering configuration interfaces and in-fabric potential attack vectors. Malicious reconfiguration, remote physical attacks, or tampering with update flows can compromise system integrity and expose sensitive data. Although vendor mechanisms such as bitstream encryption and authentication mitigate these threats, their effectiveness ultimately depends on secure key management and protected update mechanisms. Hence, while FPGAs provide compelling advantages in latency, adaptability, and efficiency, their successful deployment at the edge requires a correspondingly robust security posture against both remote and physical adversaries.

## 2.3 FPGA-specific Vulnerabilities and Attack Vectors

Once an adversary gains access to the reconfigurable fabric, new types of threats can operate entirely inside the FPGA. FPGAs offer high-throughput parallel processing, fine-grained hardware customization, and the ability to be reconfigured at runtime. An adversary can therefore create logic that abuses shared on-chip resources, interferes with co-located modules, or subverts system trust anchors, while still appearing as a syntactically valid and cryptographically authenticated configuration. In this section, we first describe the primary vectors by which adversarial reconfigurable logic may be injected into the fabric, and then review representative attack classes that use these in-fabric capabilities to disrupt operation.

### 2.3.1 Adversarial Access to the FPGA

Built-in vendor protections, i.e., bitstream encryption, authentication, and secure boot chains, are primarily designed to preserve the bitstream confidentiality and integrity of the configuration process, ensuring that only authorized bitstreams are loaded onto the device. However, these mechanisms do not guarantee that the configured logic itself is trustworthy. A bitstream may be cryptographically valid yet still contain malicious or faulty components introduced during the design flow or through runtime reconfiguration [98].

**Untrusted bitstream development.** Zhang et al. [99] demonstrated that even trusted design tools can become vectors for Trojan insertion. A compromised CAD or synthesis toolchain can silently modify the netlist during high-level synthesis or optimization, embedding additional logic that remains unnoticed by the developer. Similarly, Ahmed et al. [98] showed that attackers can insert Trojans during placement and routing by directly manipulating the bitstream to include malicious reconfigurable regions that bypass standard design rule checks and post-layout validation. Because these modifications often occur after formal verification, the final bitstream remains syntactically valid, cryptographically authenticated, and therefore accepted by the device. The malicious payload may remain dormant until specific runtime conditions activate it, making detection highly challenging.

**Exposed reconfiguration interfaces.** Even when the design tools are trusted, other entry points may exist. Software vulnerabilities, such as buffer overflows and memory corruption bugs in privileged drivers or firmware, can allow arbitrary code execution on the host CPU, granting direct access to memory-mapped configuration registers and bitstream storage [100]. At the hardware level, Timmers and Spruyt [53] demonstrated that voltage fault injection on the Arm Cortex-A9 of a Zynq-7000 device can skip critical authentication instructions during secure boot, thereby bypassing bitstream verification and enabling unauthorized access to configuration control registers. Moreover, even when reconfiguration interfaces (e.g., ICAP or PCAP) are managed by a CPU-extension TEE, such as TrustZone-assisted implementations, the assumed isolation guarantees remain fragile, as discussed further in Section 2.5.

**Third-party IP dependency.** Modern reconfigurable systems frequently rely on third-party IP blocks that are opaque, insufficiently verified, or distributed as encrypted netlists. While these components accelerate development and reduce design effort, they also introduce trust dependencies that adversaries can exploit to embed reconfigurable hardware Trojans or other hidden functionality [101].

### 2.3.2 Remote Physical Attacks

Physical attacks exploit the electrical, electromagnetic, thermal, or timing behavior of hardware to compromise confidentiality, integrity, or availability. Traditionally, such attacks were associated with invasive laboratory techniques (e.g., microprobing or focused ion beam analysis). However, the term has expanded to include non-invasive fault injections, side-channel exploits, and covert channels that can be conducted

without direct physical access. Although these threats affect integrated circuits in general, FPGAs merit particular attention due to their reconfigurable nature. The intrinsic flexibility of FPGAs allows an adversary to instantiate malicious hardware primitives, such as oscillators, voltage sensors, delay lines, or power-hungry circuits. Such logic can generate or monitor physical effects that interfere with neighboring components, enabling remote fault injection, side-channel leakage, or denial-of-service conditions without external probing equipment.

**Fault injection attacks.** Gnad et al. [102] demonstrated that malicious circuits instantiated on an FPGA can induce faults in adjacent logic by manipulating the power distribution network. By deploying fast-switching Ring Oscillator (RO) to draw transient currents, they induced voltage fluctuations and timing instabilities that ranged from subtle computation errors to complete system resets. Building on this concept, Krautter et al. [103] introduced *FPGAhammer*, a fault attack inspired by the *Rowhammer* phenomenon. Their work showed that densely packed toggling logic on an SoC FPGA can generate localized power disturbances that fault cryptographic accelerators, allowing key recovery through Differential Fault Analysis (DFA) while occupying only a fraction of the logic fabric. Alam et al. [104] proposed a related mechanism based on dual-port block RAMs: by issuing conflicting writes to the same address, they induced short circuits and localized heating, leading to timing violations and configuration bit flips that could alter circuit behavior. More recently, Luo et al. [105] demonstrated that complex workloads, such as deep neural networks (DNNs), are also susceptible. By deploying oscillating circuits on an SoC FPGA, they injected timed power glitches into neighboring accelerators, significantly degrading inference accuracy.

**Side-channel attacks.** Several studies have shown that side-channel techniques can also be mounted internally within the reconfigurable fabric [45–47, 106]. These attacks exploit shared physical resources, such as power rails or electromagnetic coupling, to extract information about a neighboring circuit’s activity. Gravellier et al. [106] demonstrated cross-domain side-channel attacks on an SoC FPGA, where a malicious hardware module extracted AES keys from both a software implementation running on the PS and a hardware implementation in the PS. This shows that untrusted logic can monitor power or timing signatures of other processes on the same device. In multi-tenant cloud environments, such attacks are particularly concerning, as a malicious tenant could extract cryptographic secrets from another without requiring any physical access or hardware modification.

### 2.3.3 Breaking Secure Boot

Jacob et al. [107] demonstrated a notable in-fabric attack in which a FPGA-based hardware Trojan subverted the secure boot process. The Trojan was designed to monitor the boot sequence and modify specific memory contents once the bootloader (U-Boot) had been loaded into external Dynamic Random-Access Memory (DRAM). By manipulating these values, the adversary exploited a flaw in the bootloader validation flow, causing the processor to skip authentication checks and load an unsigned, attacker-controlled

kernel. This effectively bypassed the cryptographic integrity guarantees of the secure boot chain. Importantly, after the malicious bitstream was configured in the PL, the attack proceeded autonomously, without requiring further CPU involvement. This case illustrates how a seemingly legitimate FPGA configuration can compromise one of the system's most fundamental root of trust.

### 2.3.4 DRAM Attacks

The PL's ability to act as a high-performance bus master often gives it unrestricted access to shared Double Data Rate Random Access Memory (DDRAM). This opens the door to memory corruption attacks initiated from the FPGA side, often without the visibility or control mechanisms that apply to software running on the PS. Weissman et al. [108] JackHammer attack, exploited this access pattern to mount a *Rowhammer*-style fault injection against memory used by a CPU process. Their approach combined a CPU-driven hammering pattern with a custom logic circuit instantiated in the PL to generate high-frequency memory toggling and amplify electrical disturbance in adjacent DRAM rows. Deployed on an SoC FPGA-based acceleration platform, the coordinated attack successfully induced bit flips in DRAM used by a WolfSSL RSA application running on the host processor. These injected faults enabled the extraction of Transport Layer Security (TLS) session keys without triggering privilege violations or software-level alarms.

## 2.4 Trusted Execution Environments

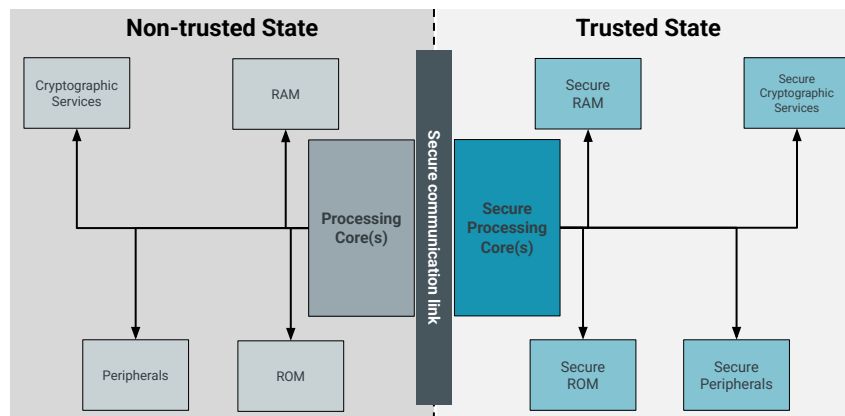


Figure 5: High-level view of a TEE within a system, including trusted and untrusted components and resources, highlighted in blue and grey, respectively

**What is a TEE?** A TEE is an isolated domain within a system. TEEs are designed to protect sensitive operations against unauthorized access or modification, and are used across a wide spectrum of devices from mobile to server platforms, providing security to applications demanding data and execution protection. Prominent examples include mobile banking [109], DRM [110], and key storage [111]. Figure 5 depicts a high-level view of a system featuring a TEE, including dedicated hardware resources. A TEE

operates in tandem with an REE. The REE often features a general-purpose OS that leverages most of a platform's hardware capabilities to implement rich functionalities.

Various TEE technologies have emerged, differing in their threat models and supported programming abstractions. Most create a trusted domain within the main processor, thereby sharing certain resources with the REE. For instance, ARM's TrustZone [56] splits the CPU into secure and non-secure worlds with minimal overhead, enabling fast context switching; Intel's SGX [54] creates protected enclaves within application space to ensure confidentiality even from privileged software; RISC-V's Application-Processor Trusted Execution Environment (AP-TEE) [57] adopt similar enclave-based approaches but offer open-source and modular implementations. Additionally, AMD's SEV [55] for virtualized environments represents further architectural explorations of TEE principles. Key characteristics of a TEE include [112]:

- **Isolation:** TEE are isolated from the main OS and other software/hardware running on the device. This isolation prevents unauthorized access and protects sensitive data and processes.
- **Verifiable Boot:** TEE often involves a secure boot process, ensuring that the environment starts in a trusted state. Secure boot thus protects the integrity of the TEE and the applications that run within it. Additionally, verifiable launch may also encompass remote attestation mechanisms that prove that the system instantiated an isolated environment correctly.
- **Trusted IO:** TEE may need to establish a secure path for accessing devices, ensuring confidentiality and integrity, and the protection of TEE data on the device through a trusted device support.
- **Secure Storage:** TEE applications frequently require maintaining persistent data across distinct invocations. This is often achieved through encryption in a process termed "sealing," while the subsequent decryption operation responsible for restoring the state is "unsealing."

### 2.4.1 Arm TrustZone

Arm TrustZone, introduced in 2004, is a hardware-based security extension for Arm APUs, specifically the Cortex-A series [113]. Its first implementation appeared in the Armv6-A architecture, and the overall design remained largely unchanged in Armv7-A [56]. As mentioned, TrustZone allows the system to be split into two isolated execution environments, the secure world and the normal world. The processor can only operate within one of these domains at a time, with its current state dictated by the non-secure bit located in the Secure Configuration Register (SCR). The context switch between these two worlds is handled by a privileged software mechanism, named secure monitor, which saves the state (registers, stack pointers, etc.) of the current world before switching to the other.

### 2.4.2 GlobalPlatform-compliant TEE Software Architecture

Before the introduction of the GlobalPlatform standard specification, TEE implementations were fragmented, with proprietary Application Programming Interface (API) and isolation mechanisms leading

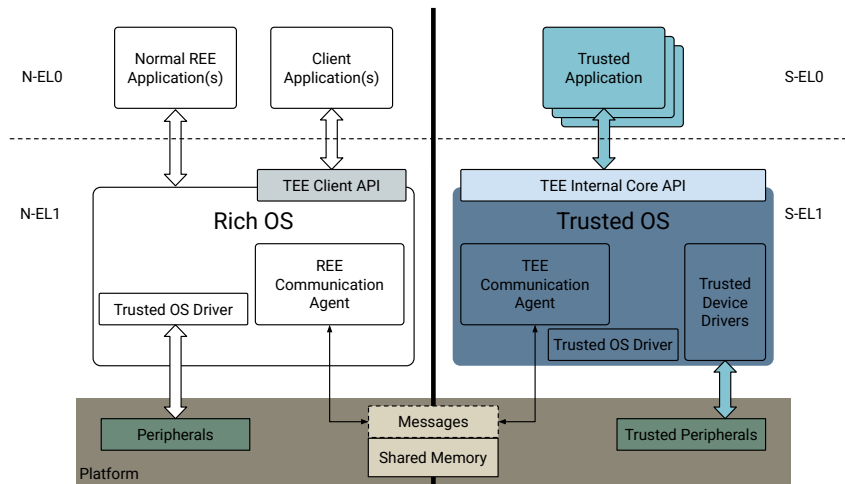


Figure 6: Typical software architecture of a TEE system compliant with the GlobalPlatform specification.

to compatibility challenges. The GlobalPlatform TEE specification addresses these issues by providing a standardized framework for implementing TEE systems. Figure 6 depicts the typical architecture of a TrustZone-assisted TEE system. Next, we provide a brief explanation of the main concepts that a GlobalPlatform-compliant TEE system considers.

**Trusted OS.** Operating in the secure state, the trusted OS provides runtime support for managing the lifecycle of TAs, ensuring controlled access to secure resources. It facilitates communication between secure and non-secure environments and implements key security functions such as cryptographic operations, secure input/output handling, and protected storage.

**Trusted Applications.** TAs leverage the security guarantees of the TEE to execute sensitive operations. They can be developed for various purposes, including system authentication methods (e.g., biometric verification), or third-party services (e.g., media decryption). The trusted OS manages their execution and enforces resource access policies.

**Client Applications.** Client Applications (CAs) are general-purpose applications that may interact with users and request secure services from the trusted OS and TAs. They operate in the non-secure state but rely on the TEE for security-critical functions.

**Rich OS.** The normal world hosts the conventional OS, often referred to as a rich OS, which is designed to handle non-critical functions. It provides a complete environment with multitasking, file systems, networking, and graphical interfaces, while managing system resources and exposing interfaces for CAs to interact with the trusted OS. When present, a hypervisor further introduces Virtual Machines (VMs), isolated environments running their own OS, similar to how an OS manages applications.

**Globalplatform TEE API specifications.** The key GlobalPlatform TEE APIs are: the TEE Internal API for TAs and the TEE Client API for communication between the REE and the TEE. The TEE Internal API provides core security functions including secure storage operations, cryptographic operations, and time management. The GlobalPlatform standard also defines the TEE Client API, which specifies the interface exposed to applications running in the REE (e.g., *TEEC\_InitializeContext*, *TEEC\_OpenSession*, *TEEC\_InvokeCommand*) and enables secure communication between CAs in the REE and TAs in the TEE. These standardized interfaces ensure interoperability and application portability while maintaining strong security guarantees across different TEE implementations.

## 2.5 Vulnerabilities in TrustZone-assisted TEE systems

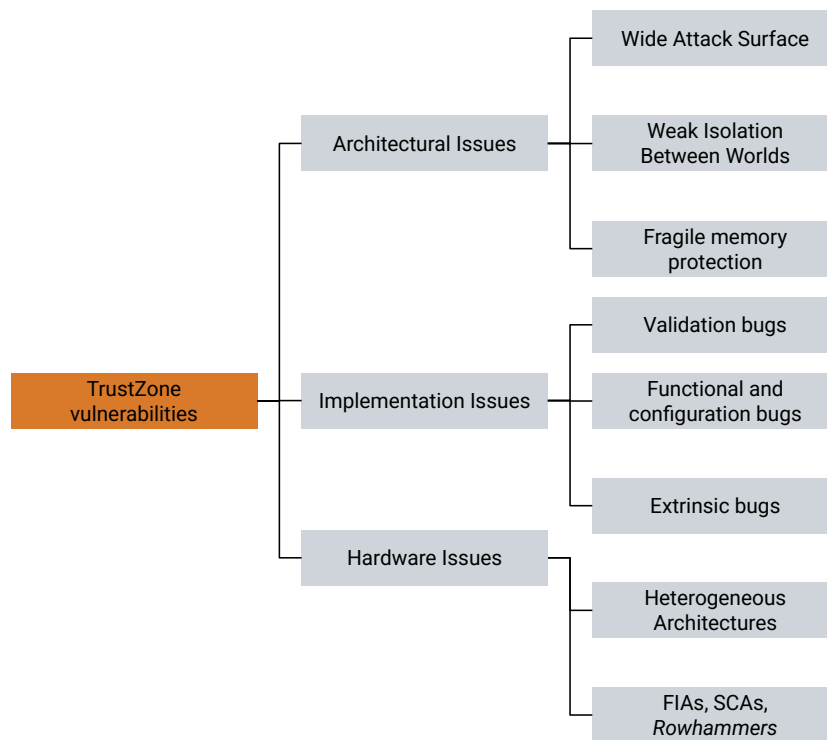


Figure 7: Taxonomy provided by Cerdeira et al. [30] of vulnerabilities in commercial TrustZone-assisted TEEs.

TrustZone-assisted TEEs rely on hardware-enforced isolation and a reduced TCB to achieve stronger security guarantees. However, real-world deployments have shown that these assumptions often do not hold. Cerdeira et al. [30] provide the most comprehensive study to date on the security of commercial TrustZone TEEs, focusing on systems implemented for Arm Cortex-A processors, the most prominent CPU architecture integrated on SoC FPGAs to date. Their analysis of 207 vulnerabilities across implementations from Qualcomm, Trustonic, Huawei, Nvidia, and Linaro reveals critical flaws at multiple levels. Figure 7

groups the prevailing problems into architectural, implementation, and hardware categories. In the following subsections, we summarize the main findings of Cerdeira et al.'s analysis, structured according to their taxonomy of architectural, implementation, and hardware issues.

### 2.5.1 Architectural Issues

**Excessive attack surface.** Existing commercial TEEs inherit long-standing OS design habits that repeatedly undermine their security. They adopt a monolithic architecture in which all privileged code runs in kernel space, expose an excessive number of system calls to TAs, and include TCBs that already span multiple megabytes, which weakens confidence in the correctness of these TAs. For instance, Qualcomm's TEE exposes 69 kernel syscalls to TAs [114]; Trustonic's TA controlling the fingerprint driver is reachable by virtually every TA; and Widevine alone implements 70 commands [115]. These broad interfaces have been instrumental for several real exploits, such as CVE-2015-6639 and CVE-2016-2431, where unchecked syscall parameters allowed normal-world attackers to overwrite secure kernel memory [110].

**Weak world isolation.** Some kernels allow any TA to map normal world physical memory and debug channels often leak stack traces or other sensitive data to the rich OS, giving attackers the information they need for privilege escalation. In Qualcomm's TEE, any TA can map normal world physical memory [114]. This includes the Linux kernel, turning a compromised TA into a trampoline to root Android. Trustonic restricts this operation to specific driver TAs, but it maintains a large whitelist of 34 privileged TAs that can still request normal world memory mappings [115]. In addition, Huawei exposes debug syscalls that dump stack traces into normal world memory, which were abused to mount reliable exploits [116].

**Fragile memory protection.** Memory protection mechanisms are inconsistently implemented across TrustZone TEEs. Address Space Layout Randomization (ASLR) is missing in some deployments or provides only minimal entropy (e.g., Qualcomm TA ASLR offers ~9 bits, while Trustonic loads all TAs at a fixed base address). Stack canaries and guard pages are also applied selectively: Trustonic lacks canaries altogether, Huawei lacks data execution prevention, and kernel-space ASLR is absent across all vendors studied. As a result, any vulnerability found in a TA can be exploited without the need for additional effort to determine code layout. Finally, trust bootstrapping remains fragile. TrustZone lacks a hardware root of trust for remote attestation, and most Original Equipment Manufacturers (OEMs) still permit outdated and vulnerable TAs to be reloaded, enabling trivial downgrade attacks [30].

### 2.5.2 Implementation Issues

**Validation bugs in TAs and kernels.** In addition to architectural weaknesses, many TEE vulnerabilities are caused by implementation bugs. Validation bugs appear when IOs are inadequately checked, leading to buffer or integer overflows, unchecked pointers, and other conditions that let normal world data corrupt trusted kernels or TAs. For instance, Qualcomm's trusted kernel failed to validate syscall

pointers (CVE-2016-2431), which enabled arbitrary secure-world writes from a compromised TA. Likewise, CVE-2016-2432 in Widevine exposed a TA command parsing flaw that led directly to code execution in the secure kernel. Such validation errors demonstrate how a single unchecked input can compromise the integrity of the entire TEE [30].

**Functional and configuration bugs.** Functional bugs arise when code deviates from its intended specification, often due to misconfigured memory protections, peripheral access, or cryptographic checks. OP-TEE, for example, was found to misconfigure page tables, leaving regions intended as read-only also executable, which undermined isolation guarantees [30]. NVIDIA's TEE suffered from CVE-2016-10423, where improper peripheral configuration allowed unintended TAs to access the SPI bus.

**Extrinsic bugs.** Defects also appear in the way TEEs handle external interactions, most notably through concurrency issues and software side-channels. OP-TEE contained a race condition in its secure storage subsystem, allowing a TA to delete a directory while another was creating it, leading to inconsistent state and potential privilege escalation. Samsung's TIMA framework reported similar races, and NVIDIA's DRM TA exhibited a TOCTOU bug (CVE-2017-6296) that enabled normal-world attackers to escalate privileges. Beyond races, side-channel vulnerabilities have also been observed. For example, OP-TEE's use of LibTomCrypt exposed a timing vulnerability (CVE-2017-1000413) in modular exponentiation, leaking RSA key bits until patched with constant-time operations.

### 2.5.3 Hardware Issues

**Heterogeneous architectures.** Since all hardware components are part of the TCB, a misconfigured or malicious peripheral can directly undermine TrustZone protections. Benhani et al. [15, 49] experimentally demonstrated that small reconfigurable hardware Trojans within the PL could: i) tamper with AXI protection bits (AWPROT/ARPROT) to mislabel non-secure transactions as secure; ii) manipulate AXI response signals (BRESP/RRESP) to falsely acknowledge illegal accesses; iii) alter the AXI crossbar logic to silently escalate privileges; iv) reconfigure the static security status of IPs; v) insert malicious FIFOs into the interconnect to eavesdrop on secure-world traffic; and vi) abuse the Accelerator Coherency Port (ACP) to gain direct access to external memory, including TrustZone configuration registers. These six proof-of-concept attacks required only minor RTL changes or malicious TCL scripts, yet were sufficient to bypass TrustZone's security model and compromise secure-world confidentiality and integrity. The results highlighted how reconfigurable logic, if not strictly controlled, can be used as a foothold to escalate privileges or tamper with trusted data.

**Fault and voltage attacks.** Hardware vulnerabilities also arise from the way processors manage energy and frequency scaling. The CLKSCREW attack exploited software control over dynamic voltage and frequency scaling on ARM Cortex-A processors to push them beyond safe operating parameters,

inducing timing faults in secure-world computations [117]. Using this method, attackers were able to extract AES keys and bypass authentication checks, proving that even privileged TrustZone code can be compromised without physical access when hardware management interfaces are exposed to software.

**Microarchitectural side-channels.** Shared microarchitectural resources further weaken world isolation. Cache-based attacks such as TruSpy [118] and ARMageddon [119] demonstrated that prime-and-probe techniques across the secure/non-secure boundary can recover AES keys or infer secure-world activity. Similarly, branch predictor attacks have been shown to extract 256-bit Elliptic Curve Digital Signature Algorithm (ECDSA) keys from Qualcomm’s secure keystore [120]. Finally, Rowhammer-style attacks have flipped bits across the world boundary, corrupting secure-world RSA private keys and enabling attackers to derive the secret from faulty signatures [121]. These results underscore that TrustZone’s hardware isolation is insufficient against microarchitectural side-channels.

## Related Work

In this chapter, we provide a comprehensive review of research relevant to the contributions of this thesis. The discussion is structured according to the main research directions that define security and trust in modern heterogeneous platforms, with particular emphasis on systems integrating reconfigurable computing. While not exhaustive, the selected works highlight key trends and directions in reconfigurable security research; a more detailed examination appears in the survey contribution (Chapter 5).

First, in Section 3.1, we examine academic proposals aimed at mitigating the prevailing threats to reconfigurable IPs, focusing on FPGA-based countermeasures that protect, isolate, and monitor sensitive IPs within the programmable fabric. We also review frameworks for secure third-party IP configuration, FPGA virtualization mechanisms, and FPGA-based OSs that manage access control and resource sharing across RRs. Second, in Section 3.2, we survey existing TEE systems, distinguishing between hardware-extension and external processor approaches, and covering both commercial and open-source implementations. Third, in Section 3.3, we review proposals for FPGA-based TEEs, which adapt these trusted execution concepts to the reconfigurable domain through TrustZone-assisted extensions, cloud-oriented frameworks, or fabric-level enclaves. Fourth, in Section 3.4, we extend the analysis to other accelerator domains, including GPUs, NPU, and novel design architectures, which pursue similar goals of enabling trusted execution and data protection outside the conventional CPU’s domain. Finally, Section 3.5 summarizes the key limitations identified across these research domains and outlines how this thesis aims to advance the state-of-the-art with the proposed TEEOD (Chapter 6) and TREE (Chapter 7) frameworks.

### 3.1 FPGA-based Countermeasures

As discussed in Section 2.3, built-in protections are largely limited to bitstream protection. These limitations have motivated extensive research into complementary countermeasures that extend security beyond the bitstream layer. Academic proposals in this domain can be broadly categorized into two major directions. The first focuses on passive protection countermeasures, which includes obfuscation and redundancy-based techniques that aim to conceal or harden logic against extraction, tampering, and fault

injection. The second centers on runtime monitoring and enforcement, encompassing attestation mechanisms, on-chip anomaly detection, frameworks for securing third-party IP integration, and FPGA-based virtualization schemes that enforce isolation between heterogeneous trust domains. These lines of research are particularly relevant to this thesis, as they suggest the feasibility of several security principles advanced in our work and, in some cases, may provide orthogonal mechanisms that can be integrated with the proposed TEEOD and TREE frameworks. Together, they offer valuable context and evidence for the broader applicability of reconfigurable and *patchable* enforcement mechanisms in FPGA-based systems.

### 3.1.1 Passive Countermeasures

The confidentiality and integrity of IP remain central concerns in reconfigurable computing, where third-party or proprietary modules are routinely deployed on shared hardware. Beyond vendor-provided encryption, numerous academic efforts have explored techniques to reduce information extraction, frustrate reverse engineering, and maintain correct operation even when exposed to tampering or fault injection.

**Obfuscation-based techniques.** When encryption is insufficient, obfuscation techniques conceal or disguise the underlying circuit structure to prevent unauthorized analysis or replication. Hoque et al. [122] propose leveraging unused LUTs to obfuscate sensitive regions of the bitstream by scrambling their configuration and introducing decoy paths to confuse tampering attempts. Other approaches equalize the timing of operations [123] or stabilize power consumption [124] to resist side-channel analysis. Dubey et al. [125] extend this to protect machine learning models. Moreover, Sasdrich et al. [126] propose a uniquely FPGA-based countermeasure to side-channels which involve using DPR to periodically relocate or shuffle sensitive logic, effectively randomizing physical placement and disrupting profiling.

**Redundancy-based techniques.** Redundancy-based defenses aim to preserve correctness in the presence of active perturbations, particularly fault injection attacks. Classical approaches rely on passive redundancy such as triple modular redundancy or algorithmic duplication, which ensure that transient or localized faults do not compromise the output of cryptographic units [42, 127]. Physical re-randomization of modules through partial reconfiguration helps prevent attackers from repeatedly targeting the same physical area [128]. In SoC FPGAs, software-level redundancy—such as repeated authentication checks—adds further protection against control flow tampering [53].

### 3.1.2 Runtime Monitoring and Enforcement

Whereas obfuscation and redundancy techniques aim to increase resilience by complicating reverse engineering or by tolerating injected faults, another line of research focuses on mechanisms that continuously verify the correctness of the system once reconfigurable logic is deployed. These works treat the FPGA as a dynamic environment in which modules may misbehave, either due to malicious intent or external perturbations, and introduce enforcement or verification steps that provide evidence of correct operation.

By shifting attention from static protections to runtime guarantees, they address adversarial actions that manifest after deployment and which encryption or diversity mechanisms alone cannot contain.

**Tampering and probing detection.** FPGA-based sensors can be used to detect physical disturbances caused by fault injection or probing. For example, He et al. proposed high-frequency RO watchdogs to detect changes in delay caused by laser pulses or voltage glitches [129–131]. These oscillators are sensitive to localized timing shifts and can trigger alarms or activate mitigation routines. Similarly, delay-sensing circuits can monitor path delays in real time and reduce clock frequency in response to anomalies, helping suppress transient faults [132]. Optical attacks can be detected using embedded Physically Unclonable Function (PUF)-based structures. Tajik et al. [133] demonstrate that RO-based PUFs, when placed in the FPGA fabric, show measurable disturbances under optical probing. By continuously sampling the output of these circuits, the system can detect and respond to invasive attacks that attempt to read internal signals or configuration bits through the silicon substrate.

**Attestation Mechanisms.** Drimer and Kuhn [134] introduced one of the first authenticated reconfiguration protocols for FPGAs, where bitstreams are digitally signed to prevent unauthorized configurations from being loaded remotely. Vliegen et al. [135] later proposed a secure remote reconfiguration scheme that maintains confidentiality and integrity during dynamic updates by combining bitstream encryption with mutual authentication. Kashyap and Chaves [136] extended these ideas by presenting a compact on-the-fly secure dynamic reconfiguration architecture for volatile FPGAs, ensuring only authorized bitstreams can occupy defined reconfigurable regions. Vliegen et al. [137] introduced SACHa, a self-attestation framework that allows FPGAs to autonomously verify their configuration integrity using lightweight cryptographic primitives, eliminating the need for continuous external supervision.

**On-chip monitors and anomaly detection.** Given that several runtime threats can bypass pre-deployment checks, embedding detection logic directly into the FPGA fabric has become a key defense strategy. Yao et al. [138] proposed a multipurpose ring-oscillator grid that continuously monitors the power distribution network for anomalies, enabling lightweight detection of side-channel attacks and fault-injection attempts. Alanwar et al. [139] developed a dynamic detection and protection framework against hardware Trojans in FPGA fabrics, combining runtime monitoring with logic-locking and configuration fingerprinting to identify malicious alterations after deployment. Ahmadi et al. [140] further advanced these ideas with FPGA-Patch, which dynamically mitigates remote side-channel attacks by applying reconfigurable countermeasures through PR. In shared FPGA environments, Krautter et al.'s FPGAhammer [103] demonstrates how aggressive ring-oscillator activation can induce voltage droops and timing faults across partitions; embedded sensors, such as timing or LUT-based voltage-drop detectors, can thus serve as early alarms against fault or power-injection attacks. By exploiting underutilized logic and routing resources, these monitoring schemes achieve low overhead while providing continuous vigilance against malicious or faulty behavior in reconfigurable systems.

**Frameworks for Third-party IP Security.** Restuccia et al. [141] introduced Bastion, a framework to secure the integration of third-party IP blocks in Network-on-Chip (NoC)-based SoCs by deploying distributed monitors that enforce traffic integrity, access control, and anomaly detection. This approach enables the isolation of compromised IPs without disrupting the platform, though it depends on accurate policy definition and incurs integration overhead. Tan et al. [142] proposed monitoring and mitigation patches, lightweight modules that attach to IP interfaces, observe I/O behavior, and enforce corrective actions at runtime. Unlike design-time verification, MoPs enable post-deployment patching of black-box IPs. Building on this, Dharavathu and Tan [143] explored using FPGA as reconfigurable monitors integrated at different system levels, such as tightly with IP blocks, coupled to Direct Memory Access (DMA) controllers, or as bus-compliant replacements. Their work demonstrates how FPGA logic can enforce corrective actions dynamically and provide a field-updatable enforcement layer against unforeseen vulnerabilities.

### 3.1.3 FPGA-based OS and Virtualization

The growing adoption of reconfigurable computing in shared infrastructures has motivated efforts to virtualize and manage FPGA resources with the same abstraction level as conventional processors. These efforts pursue higher utilization, programmability, and multi-tenancy, while raising new security concerns when several users or workloads share the same fabric. We focus on three main research directions: i) operating systems that expose reconfigurable hardware as managed computational resources; ii) virtualization frameworks that partition or multiplex FPGAs in cloud or cluster environments; and iii) embedded or IoT-oriented frameworks that adapt these concepts to constrained or long-lived edge platforms.

**OSs for reconfigurable devices.** Efforts to abstract reconfigurable hardware as managed computational resources have led to several OS frameworks that integrate hardware tasks into conventional software execution models. Early systems such as BORPH [144], FUSE [145], and ReconOS [146] extended Linux-like environments to treat FPGA modules as hardware threads or processes, enabling hybrid multiprocessing between CPUs and reconfigurable logic. Subsequent designs, including HybridOS [147], Feniks [148], LEAP [149], and AmorphOS [150], introduced modular abstractions and virtualization primitives to support multi-tenant management and resource sharing in cloud-scale deployments. More lightweight approaches such as FOS [151], ReproRun [152], and Arnold [153] adapt these principles to IoT and edge environments through reconfigurable runtime services and near-sensor hardware–software co-design. While these systems enhance programmability and flexibility, most lack strong isolation, authenticated context management, and secure module launch—leaving hardware-enforced security as an open research challenge.

**FPGA virtualization.** The increasing deployment of FPGAs in data centers has driven extensive research on virtualization frameworks that enable spatial and temporal sharing of the reconfigurable fabric across multiple tenants. Early efforts such as CoRAM [154] and OpenStack-based provisioning [155] explored memory and resource abstraction for accelerators through IOMMU-assisted DMA isolation and

cloud orchestration. Later works, including the FPGA hypervisors proposed by Fahmy et al. [156] and Knodel et al. [157], introduced virtual slot management to mediate configuration, I/O, and memory access. Asiatici et al. [158] extended these concepts to SoC-based platforms through runtime management of partial regions and DMA control, while Mandebi et al. [159] leveraged spatial partitioning and lightweight NoCs to create isolated virtual regions within shared devices. More recent systems such as Optimus [160] and the frameworks by Zha et al. [161, 162] enhance utilization and scalability by slicing FPGA memory pages and decomposing bitstreams for fine-grained placement across FPGA clusters.

## 3.2 TEE Systems

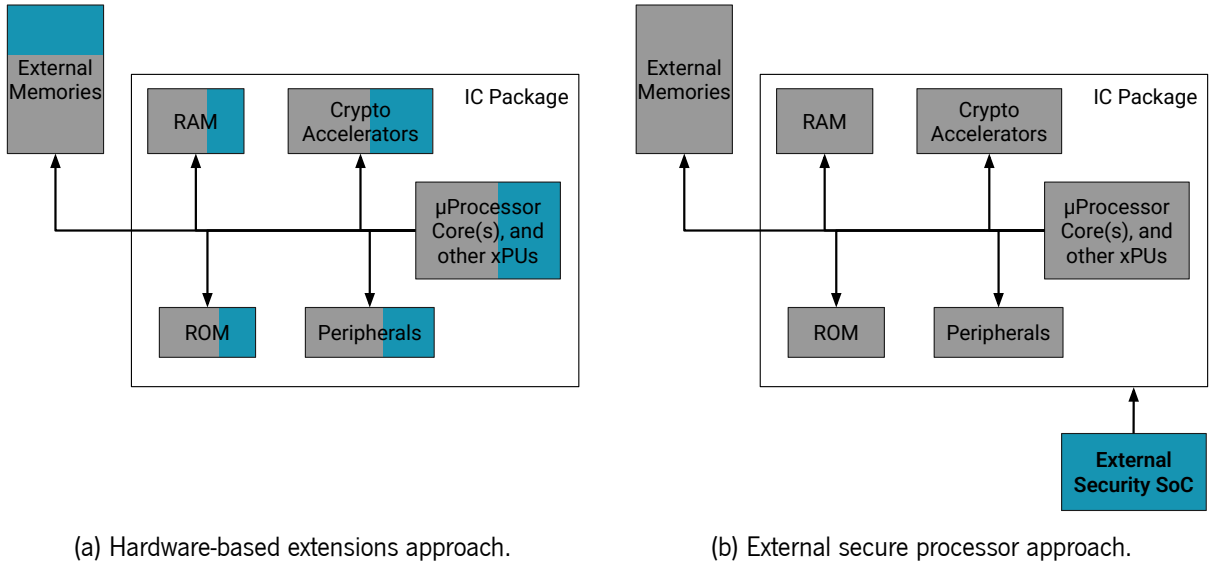


Figure 8: Comparison of conventional TEE approaches.

Existing TEE systems can be broadly categorized according to how they implement and anchor their root of trust. The most prominent designs rely on tightly coupled hardware extensions that embed security primitives directly into the CPU, such as Arm TrustZone [113], Intel SGX [54], AMD SEV, and Apple’s AP-TEE [57] (Figure 8a). An alternative approach relies on companion or external secure processors that operate independently from the main processing cores (Figure 8b). These architectures decouple the TEE from the host processor, delegating sensitive operations such as cryptographic processing, key management, and attestation to a physically or logically isolated subsystem. Examples include Apple’s Secure Enclave Processor (SEP), Google’s Titan [163], and Arm’s Platform Security Architecture (PSA) based frameworks [164]. By moving the TEE outside the main CPU, these designs strengthen resilience against host-level compromise, albeit at the cost of higher communication latency and performance overhead.

This section surveys the landscape of TEE systems, encompassing both commercial deployments and academic proposals that rely on either TEE-enabling approaches. The goal is to understand how these systems establish isolation, integrity, and attestation guarantees, and to identify architectural and scalability

limitations, particularly as computing platforms become increasingly heterogeneous. Recognizing these limitations is key to deriving the design principles behind the TEEOD and TREE frameworks proposed in this thesis, which extend trust beyond the general-purpose CPU toward reconfigurable logic fabrics.

### 3.2.1 TEE Systems based on CPU-based TEE extensions

Since most commercial SoC FPGAs adopt Arm-based processing subsystems, the TEE software and system designs discussed here are predominantly TrustZone-enabled. We group prior work according to where isolation is enforced (secure world vs. normal world) and the execution model adopted (monolithic trusted OS, virtualized partitions, or language/runtime sandboxes). This classification helps surface typical design patterns, TCB compositions, and trade-offs.

**Commercial TrustZone Implementation.** Mobile and embedded vendors typically ship proprietary trusted OSs and services running in the secure world. Examples include stacks from Qualcomm [114], Huawei [116], and Samsung [165], as well as third-party offerings such as SierraTEE [166] and Kinibi [115], which implement GlobalPlatform APIs and device services (e.g., secure storage, keymaster, or Digital Rights Management (DRM)) tightly integrated with boot firmware and SoC peripherals.

**Open-source and research TEE implementations.** Open-source stacks facilitate reproducibility and transparent evaluation of trusted execution environments, enabling both academic experimentation and industrial prototyping. Among the most widely adopted frameworks, OP-TEE [167] and Trusty [168] implement GlobalPlatform-compliant trusted OSs, while TF-A [169] provides a reference implementation for secure-world firmware initialization and exception-level mediation. OP-TEE and TF-A do not natively manage reconfigurable logic; however, they are frequently reused on SoC FPGA platforms to coordinate secure boot and PR, i.e., they offer partial integration with reconfigurable IPs. Trusty remains CPU-only with no such support. Several academic efforts extend these foundations by rethinking the TEE architecture itself. X. Yang et al. [170] propose Trust-E, an embedded TEE architecture on TrustZone. M. Paolino et al. [171] introduce T-KVM, a lightweight hypervisor extension that enables TEE functionality within isolated virtual machines, supporting concurrent enclave execution atop a secure monitor. B. McGillion et al. [172] develop Open-TEE, a hardware-independent GlobalPlatform-compliant emulator designed to facilitate rapid development and testing of TAs without physical hardware. N. Feske et al. [173] present the Genode framework with TrustZone backends, combining a microkernel-based architecture with fine-grained component isolation for trusted system composition.

### 3.2.2 External Secure Processors

**Commercial Secure processors.** Several vendors integrate dedicated secure processors alongside the main application core to handle trusted services. Apple's SEP [174] manages biometric data, key storage, and verified boot within a protected memory enclave. Qualcomm's SPU [175] and Google's

Titan [163] similarly offload cryptographic and attestation tasks to isolated hardware modules, either on-chip or as discrete secure elements. Intel's Management Engine (ME) [176] and AMD's Platform Security Processor (PSP) [177] embed MCUs within the chipset or CPU die to perform firmware validation, secure boot, and runtime management. While these systems differ in integration level and service scope, they all share a static trust model focused on predefined security tasks.

**Trusted Platform Modules.** Trusted Platform Modules (TPMs) provide standardized hardware roots of trust for key storage, platform measurement, and attestation, complementing secure boot and firmware validation flows. Traditionally deployed as discrete or firmware-integrated coprocessors bound to the CPU, they ensure integrity of static system components. Gross et al. [178] extend this model with a hybrid-TPM design for SoC FPGAs, combining a discrete TPM with reconfigurable logic to delegate selected security functions to the programmable fabric while preserving a hardware-protected root of trust. Beyond discrete implementations, virtualization frameworks such as vTPM [179] replicate TPM functionality in software to support isolated trust anchors across multiple virtual machines, enabling scalable attestation in cloud and multi-tenant environments.

### 3.3 FPGA-based TEES

The growing reliance on reconfigurable computing has motivated the adaptation of TEE concepts to FPGAs. Research efforts in this direction can be broadly categorized into three main trends: i) extensions of mainstream TEE technologies to include the reconfigurable fabric as part of the TCB; ii) cloud-oriented frameworks, which provide trusted execution capabilities for remote or multi-tenant FPGAs; and iii) alternative TEE systems, which aim to replace conventional CPU-centric hardware extensions with trust primitives implemented directly within the reconfigurable logic.

#### 3.3.1 Extended CPU-centric TEE systems

**SGX-FPGA.** Xia et al. [180] proposed SGX-FPGA, which extends Intel SGX enclaves to include FPGA accelerators. Figure 9a illustrates an overview of the proposed architecture. The system relies on PUF-based key derivation and inline encryption to secure data transfers between the CPU enclave and FPGA logic. This architecture is designed to ensure that sensitive workloads can offload computation to reconfigurable logic without exposing data outside the SGX enclave.

**HISA.** Ye et al. [41] introduced HISA, a TrustZone-assisted framework that extends hardware isolation to SoC FPGAs. Figure 9b illustrates an overview of the proposed architecture. It partitions the system into secure and normal worlds via AXI bus enforcement, isolating FPGA IPs from untrusted software. A secure agent mediates service requests and enforces access-control to block cross-domain attacks.

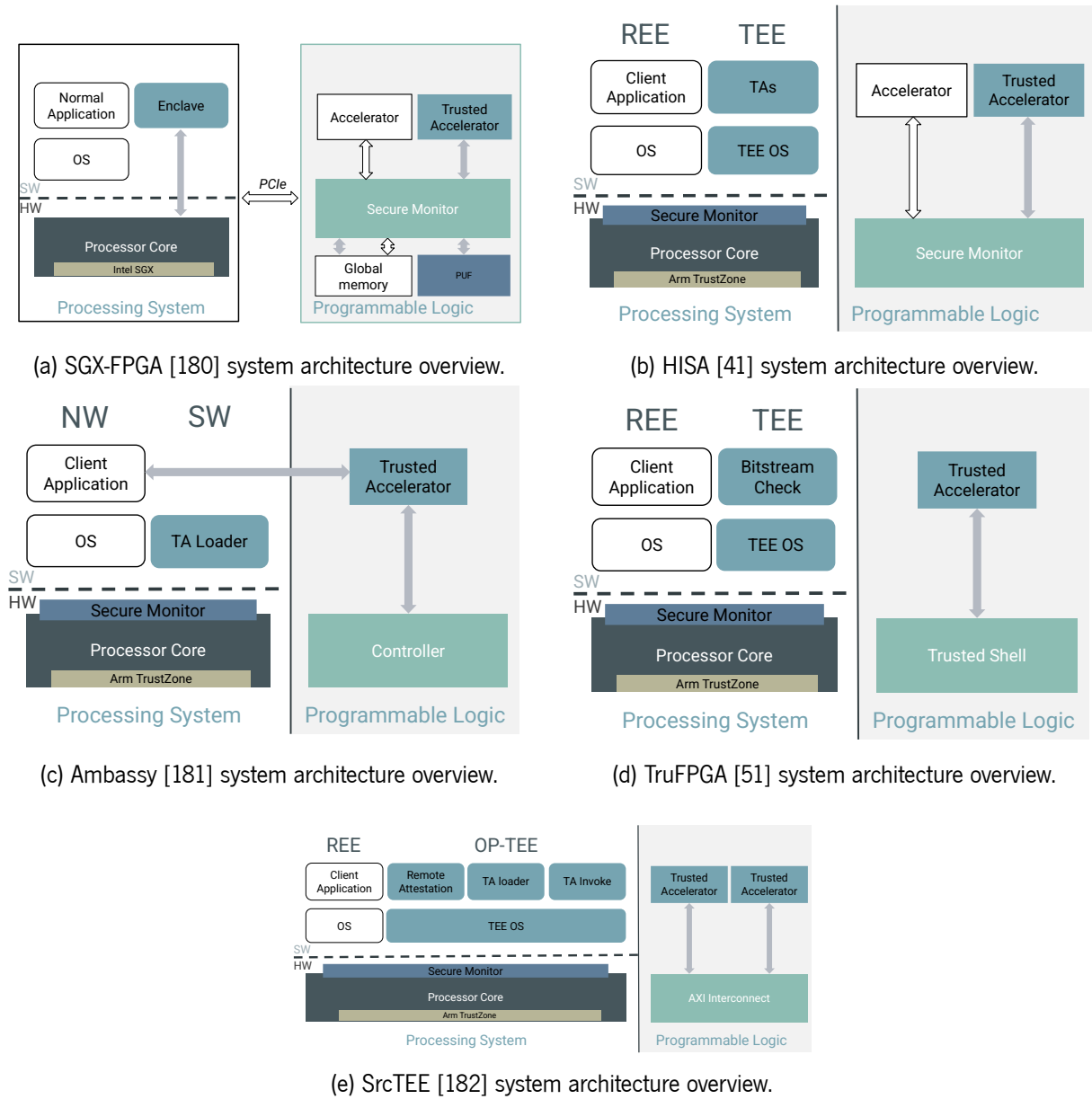


Figure 9: Overview of representative TrustZone-assisted FPGA-based TEEs.

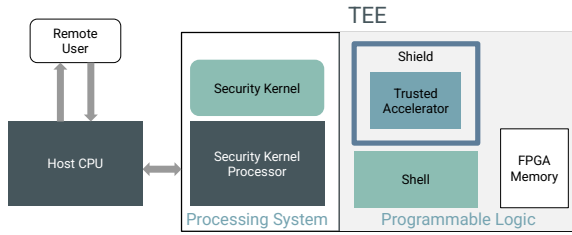
**Ambassy.** Hwang et al. [181] presented Embassy, a runtime framework for securely delegating TAs to PL. Figure 9c provides an overview of the proposed architecture. Built on Arm TrustZone, Embassy authenticates reconfigurable functions and manages their deployment dynamically. By combining secure world services with FPGA reconfiguration, Embassy enables trusted hardware functions to be instantiated on demand, while preserving the isolation guarantees of the TEE model.

**TruFPGA.** Giechaskiel et al. [51] proposed TruFPGA, a framework that establishes mutual trust between CPUs and FPGA logic. Figure 9d provides an overview of the proposed architecture. The system leverages Arm TrustZone to authenticate and authorize reconfiguration operations, ensuring that only verified logic

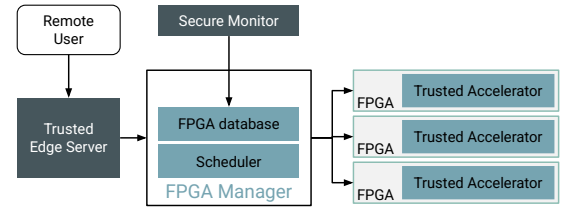
can be loaded into the FPGA. It integrates secure boot, cryptographic measurement of bitstreams, and configuration attestation to maintain the integrity of the PL throughout the deployment process. TruFPGA further supports secure DPR, enabling dynamic updates of reconfigurable regions while preserving system-wide integrity and providing built-in tamper detection.

**SrcTEE.** Ning et al. [182] present SrcTEE, a TrustZone-assisted framework that extends OP-TEE with hardware-aware runtime trusted software to support secure deployment, invocation, and attestation of reconfigurable IP blocks on SoC FPGAs. Figure 9e provides an overview of the proposed architecture. SrcTEE authenticates and decrypts bitstreams within the secure world.

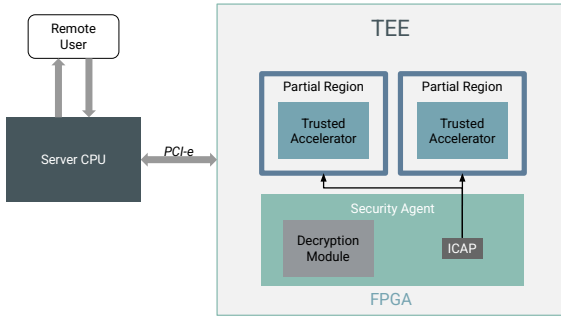
### 3.3.2 Cloud FPGAs



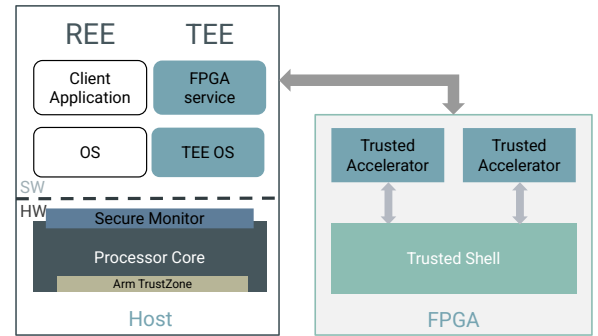
(a) Overview of the system architecture proposed in ShEF [52].



(b) Overview of the system architecture proposed in DF-TEE [183].



(c) Overview of the system architecture proposed in MeetGo [184].



(d) Overview of the system architecture proposed in AccGuard [185].

Figure 10: Overview of representative FPGA-based TEEs frameworks for cloud deployments.

**ShEF.** Zhao et al. [52] proposed ShEF, a TEE for cloud FPGAs that relies only on the FPGA for trust enforcement. Figure 10a shows an overview of the proposed architecture. ShEF enforces application-specific security policies, supports bitstream attestation, and leverages secure boot to ensure that only verified configurations are admitted.

**DF-TEE.** Xia and Wei [183] introduced DF-TEE, a framework for disaggregated multi-FPGA deployments in cloud environments. Figure 10b shows an overview of the proposed architecture. DF-TEE chains attestation across heterogeneous components, including CPUs and FPGAs, thereby extending the trust boundary beyond a single device. The architecture aims to enable tenants to verify the integrity of all interconnected accelerators, supporting a unified trust domain in scenarios where resources are pooled across multiple reconfigurable boards.

**MeetGo.** Zhao et al. [184] proposed MeetGo, an FPGA-based TEE that synthesizes enclave-style environments entirely within the reconfigurable fabric. Figure 10c illustrates the proposed architecture, which embeds a dedicated hardware security agent responsible for authenticating deployed modules, enforcing isolation policies, and performing runtime attestation. Remote tenants can independently verify the FPGA configuration without relying on the host system, thereby establishing stronger trust in remote FPGA workloads where the cloud provider cannot be fully assumed trusted.

**AccGuard.** Ren et al. [185] proposed AccGuard, a secure computation framework that extends trusted execution to remote FPGA accelerators in cloud environments. Figure 10d shows an overview of the proposed architecture. The architecture introduces a security manager within the FPGA fabric. This manager expects that the host platform already provides a TEE, such as Intel SGX or Arm TrustZone, to establish a secure channel with the FPGA and coordinate authenticated deployment and attestation of accelerators. A trusted authority audits and certifies the FPGA shell, provisions device-specific root keys, and validates attestation reports, enabling verifiable accelerator deployment without requiring pre-encrypted bitstreams.

### 3.3.3 Trusted Execution on Soft-MCUs

Enabling software TAs to execute directly within the reconfigurable fabric offers both security and practical advantages over conventional CPU-centric TEEs. From a security standpoint, this approach relocates the TCB entirely, or in large part, into the PL, thereby reducing, or in some cases eliminating, the reliance on fixed hardware TEE extensions such as those of TrustZone or SGX, and avoiding the cross-world vulnerabilities and software-layer attack surfaces discussed in Section 2.5. From a practical perspective, integrating general-purpose processing within the FPGA enables hybrid hardware/software co-designs that combine the flexibility of software with the performance and tight coupling of custom hardware accelerators.

Despite these advantages, the majority of FPGA-based TEEs proposals adopt a trust model centered on enabling secure execution for hardware accelerators, focusing primarily on the compartmentalization and attestation of reconfigurable IP blocks under the control of a trusted host processor. As a result, the programmable fabric remains a subordinate domain whose integrity ultimately depends on the main processor or management firmware. The only workaround proposed in these works has been the translation of software TAs into synthesizable hardware modules through High-Level Synthesis (HLS), as exemplified

by MeetGo [184], which achieves functional isolation but lacks runtime flexibility and dynamic reconfigurability. To overcome these constraints, a more recent research direction advocates the instantiation of both the enclave and its control logic entirely within the PL, fully decoupled from the host’s fixed TEE extensions. Notably, BYOTee [186], introduced in the same period as our proposed TEEOD framework, pursues a similar vision of FPGA-centric trusted execution, underscoring the growing recognition that reconfigurable logic itself can serve as a first-class trusted domain.

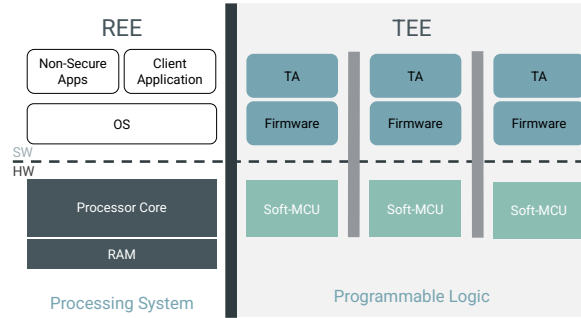


Figure 11: Overview of the system architecture proposed in BYOTee [186].

**BYOTee.** Armanuzzaman et al. [186] introduced BYOTee, a co-design framework that allows developers to construct application-specific enclaves on the FPGA. Figure 11 shows an overview of the proposed architecture. Rather than relying on a fixed hardware TCB, BYOTee lets users define the scope of the TCB by selecting which modules are included and attested. BYOTee architecture bears similarity to our proposed TEEOD framework in that both use soft-MCUs to execute TAs, decoupling from the PS dependency. However, BYOTEE relies extensively on firmware for TA management, including loading, scheduling, and cleanup, which must execute on each soft-MCU. In contrast, TEEOD implements these mechanisms as hardware-accelerated agents, eliminating dependence on enclave firmware and reducing the attack surface exposed to compromised firmware. This shift in the trust model substantially reduces firmware complexity and eliminates the need for advanced soft-MCUs to thwart privilege escalation attacks.

### 3.4 Other Research

Beyond FPGAs, a broader body of work explores trusted execution in other heterogeneous architectures. Although these systems target distinct xPUs, such as GPUs and NPUs, or propose new architectures, they share the same goals of extending enclave-level guarantees beyond the general-purpose CPU. We organize this discussion into three complementary threads: i) new hardware architectures; ii) GPU-based TEEs; and iii) NPU-based TEEs.

### 3.4.1 New Hardware Architectures

**HETEE.** Zhao et al. [28] proposed HETEE, an architecture that provides trust compartmentalization between CPU and FPGA domains. In HETEE, the FPGA is leveraged to isolate untrusted accelerators, preventing them from directly influencing secure system state. The CPU retains responsibility for attestation and management, delegating isolation TAs to the reconfigurable fabric. This division of labor reduces the software TCB while ensuring that heterogeneous accelerators are confined to well-defined domains.

**CURE.** Datta et al. [187] introduced CURE, a novel TEE architecture designed for heterogeneous systems. CURE introduces new hardware mechanisms that enable fine-grained resource isolation within the SoC, including exclusive assignment of CPU cores, caches, and peripherals to specific enclaves. These hardware extensions enforce multi-domain isolation and mitigate shared-resource side-channels while requiring only minimal modifications to the baseline RISC-V design. By combining flexible reconfiguration with hardware-assisted security primitives, CURE adapts to diverse application scenarios while maintaining verifiable and dynamically adjustable isolation guarantees.

### 3.4.2 GPU-based TEEs

The rise of general-purpose GPU computing has transformed GPUs from fixed-function graphics engines into programmable, high-throughput accelerators central to scientific, cloud, and AI workloads [188]. This shift has also exposed sensitive data and models to untrusted software stacks, motivating efforts to secure GPU execution environments. Recent works therefore pursue GPU-based TEEs that introduce enclave abstractions directly within GPU architectures.

**Graviton.** Schuett et al. [189] presented Graviton, a GPU-based TEE that introduces architectural extensions to support secure execution of GPU workloads. The system adds new instructions and access control mechanisms to GPUs, enabling them to isolate tasks at the hardware level. Graviton suggests that accelerators, traditionally treated as untrusted peripherals, can themselves provide enclave-style protection for high-performance parallel workloads.

**HIX.** Jang et al. [190] proposed HIX, which extends Intel SGX enclaves to cover GPU memory-mapped I/O regions. HIX enables fine-grained isolation of GPU memory, allowing enclave applications to securely offload tasks to GPUs without leaking sensitive data. By aligning GPU execution with SGX's protection model, HIX narrows the gap between CPU and accelerator trust domains in heterogeneous platforms.

**StrongBox.** Deng et al. [191] developed StrongBox, a TEE for Arm-based systems that enforces isolated GPU execution with a reduced TCB. StrongBox integrates secure drivers and hardware controls to

separate trusted GPU tasks from the untrusted operating system, limiting the attack surface while retaining performance benefits of GPU acceleration. Its lightweight design makes it suitable for mobile and embedded environments where resources are constrained.

**XpuTEE.** Fan et al. [192] recently introduced XpuTEE, a GPU-focused TEE that emphasizes efficiency in addition to isolation. XpuTEE aims to minimize memory copies and reduces encryption overheads when securing GPU tasks, addressing performance bottlenecks common in prior GPU-TEE designs. By optimizing both security and throughput, XpuTEE broadens the applicability of TEEs to data-intensive accelerator workloads.

**LITE.** Yudha et al. [193] introduced LITE, a low-cost and interoperable GPU-based TEE that achieves unified encryption domains between the CPU and GPU. Unlike prior hardware-only proposals, LITE relies on software-based memory encryption aided by minimal architectural support, allowing both CPU and GPU to share a common encryption key and avoid costly re-encryption during data transfers.

**GEVisor.** Wu et al. [194] proposed GEVisor, a formally verified framework that leverages Intel SGX to construct GPU TEEs without requiring hardware modifications. GEVisor introduces a minimal hypervisor that enforces trusted I/O paths between enclaves and GPUs, isolates device contexts, and monitors host software for unauthorized access.

**Telekine.** Hunt et al. [195] presented Telekine, a system enabling secure GPU computation in untrusted cloud environments. Building on the assumption of GPU-level TEEs, Telekine eliminates CPU-side side-channel attacks by transforming GPU API calls into fixed-rate, data-oblivious streams. This approach conceals timing and size-dependent information observable by privileged software or cloud administrators.

### 3.4.3 TEE for AI accelerators

**AccShield.** Ren et al. [196] propose AccShield, a TEE architecture targeting TPU-like accelerators that provides isolation and spatial multi-tenancy for Machine Learning (ML) workloads. The prototype leverages an FPGA-based implementation built upon the tinyTPU core and integrates the Coyote abstraction layer for virtual memory support alongside the AccGuard framework for secure task management. Together, these components enable concurrent protected execution of accelerator workloads while maintaining compatibility with existing hardware.

**Confidential AI Accelerator.** Vaswani et al. [197] extend Graphcore's GC200 Intelligence Processing Unit (IPU) with confidential computing support, establishing enclave-level confidentiality and integrity for offloaded ML tasks in cloud environments. Their design minimizes the trusted computing base by removing host CPU dependencies and introducing accelerator-local attestation and key management, thereby providing a self-contained trusted execution domain for distributed inference and training workloads.

**sNPU.** Feng et al. [198] present sNPU, an integrated NPU-centric TEE providing secure contexts, fine-grained memory protection, and attested firmware under a trusted scheduler. The system prevents model and activation data from traversing host-controlled paths, mitigating attacks from local or scratchpad memory [199], and achieves enclave-level confidentiality with low overhead.

**TNPU.** Lee et al. [200] propose TNPU, which replaces Merkle-tree integrity verification with a tree-less scheme optimized for tensor-oriented access patterns. This design maintains trusted execution while significantly reducing memory and bandwidth overheads.

**TensorTEE.** Han et al. [201] introduce TensorTEE, which unifies TEE granularity across heterogeneous CPU–NPU systems. It applies tensor-level protection and predictive Message Authentication Code (MAC) management, enabling enclave-to-enclave tensor transfers without re-encryption or host mediation, achieving high throughput with minimal training overhead.

## 3.5 Gap Analysis

The increasing adoption of heterogeneous computing architectures, particularly those integrating reconfigurable computing across embedded, edge, and cloud platforms, has driven extensive research into securing these systems and mitigating the expanding attack surface inherent to such systems. Table 3 consolidates all the related works discussed throughout this chapter, providing a comparative overview of their protection mechanisms, roots of trust, TA scope, and supported TEE services. Although these efforts collectively advance the state of reconfigurable system security, the literature remains fragmented across abstraction layers, deployment models, and trust assumptions. From this analysis, we identify three major gaps that hinder the realization of *unified*, *adaptive*, and *patchable* trust architectures for FPGA-based systems: i) the lack of a unified perspective on cross-domain threats and under-explored security opportunities within FPGAs; ii) the limitations of existing CPU-centric TEEs when extended to SoC FPGAs; and iii) the absence of runtime flexibility and reconfigurable isolation in current FPGA-based TEE frameworks.

Building on these observations, our comprehensive survey (Chapter 5) formalizes and contextualizes these gaps by unifying and further detailing the perspectives established through the works reviewed in this chapter. In the remainder of this section, we contextualize how our contributions, TEEOD (Chapter 6) and TREE (Chapter 7), directly address these identified gaps.

### 3.5.1 Lack of a unified perspective on threats and under-explored security opportunities in FPGA-based systems

Most research on FPGA-related security concentrates on two main directions. The first focuses on countermeasures against FPGA vulnerabilities, including bitstream protection, IP confidentiality, side-channel and fault resilience, and runtime monitoring. The second centers on FPGA-based TEEs, which mainly

| Reference                | Work                               | RoT                  | TP             | Passive F/T Res. | Protect I/A Det. | TA Management TA-Cfg | TA-Conf | S-MT | T-MT | TA scope HW | SW  | Hybrid | TEE services SS | Encrypt. | Attest. | GP  |
|--------------------------|------------------------------------|----------------------|----------------|------------------|------------------|----------------------|---------|------|------|-------------|-----|--------|-----------------|----------|---------|-----|
| [122–126]                | <b>IP Obfuscation</b>              | FPGA fabric          | FPGA, SoC FPGA | ○                | ○                | N/A                  | ●       | N/A  | N/A  | N/A         | N/A | N/A    | N/A             | N/A      | N/A     | N/A |
| [42, 53, 127, 128]       | <b>IP redundancy</b>               | FPGA fabric          | FPGA, SoC FPGA | ●                | ○                | N/A                  | N/A     | N/A  | N/A  | ●           | N/A | N/A    | N/A             | N/A      | N/A     | N/A |
| [129–131, 133]           | <b>Tamper/Prob. detect</b>         | FPGA fabric          | FPGA, SoC FPGA | ●                | ●                | N/A                  | N/A     | N/A  | N/A  | ●           | N/A | N/A    | N/A             | N/A      | N/A     | N/A |
| [134–137]                | <b>Attestation</b>                 | FPGA fabric          | FPGA, SoC FPGA | ○                | ●                | N/A                  | N/A     | N/A  | N/A  | ●           | N/A | N/A    | N/A             | N/A      | ●       | N/A |
| [138–140]                | <b>Monitors/Sensors</b>            | FPGA fabric          | FPGA, SoC FPGA | ●                | ●                | ○                    | ○       | N/A  | N/A  | ●           | N/A | N/A    | ○               | ○        | ○       | N/A |
| [141–143]                | <b>Third-party IP frameworks</b>   | FPGA fabric          | FPGA, SoC FPGA | ○                | ●                | ●                    | ●       | ●    | ●    | ●           | N/A | N/A    | ○               | ●        | ●       | N/A |
| [144–153]                | <b>FPGA-based OS</b>               | FPGA fabric          | SoC-FPGA       | ○                | ●                | ●                    | ○       | ●    | ●    | ●           | ○   | ○      | ○               | ○        | ●       | N/A |
| [154–162]                | <b>FPGA-based Virtualization</b>   | FPGA fabric          | Cloud-FPGA     | ○                | ●                | ●                    | ○       | ●    | ●    | ●           | ○   | ○      | ○               | ○        | ●       | N/A |
| [114–116, 165–173]       | <b>Extension-based TEE systems</b> | TrustZone, SGX       | CPU, SoC FPGA  | ○                | ●                | ●                    | ●       | ○    | ●    | ○           | ●   | ○      | ●               | ●        | ●       | ●   |
| [163, 174–179]           | <b>External Secure Processors</b>  | SEP, Titan, PSP, TPM | Mobile/Edge    | ○                | ●                | ○                    | ●       | ●    | ●    | ○           | ●   | ○      | ●               | ●        | ●       | ○   |
| <b>FPGA-based TEEs</b>   |                                    |                      |                |                  |                  |                      |         |      |      |             |     |        |                 |          |         |     |
| [180]                    | SGX-FPGA                           | SGX                  | FPGA           | ○                | ●                | ●                    | ○       | ○    | ○    | ●           | ●   | ●      | ●               | ○        | ○       | ○   |
| [41]                     | HISA                               | TrustZone            | SoC FPGA       | ○                | ○                | ○                    | ○       | ○    | ○    | ●           | ●   | ●      | ●               | ○        | ○       | ○   |
| [181]                    | Ambassy                            | TrustZone            | SoC FPGA       | ○                | ●                | ●                    | ●       | ○    | ○    | ●           | ●   | ○      | ●               | ○        | ●       | ○   |
| [51]                     | TruFPGA                            | TrustZone            | SoC FPGA       | ○                | ○                | ○                    | ○       | ○    | ○    | ○           | N/A | ○      | ○               | ○        | ○       | ○   |
| [182]                    | SrcTEE                             | TrustZone            | SoC FPGA       | ○                | ●                | ●                    | ●       | ○    | ○    | ●           | ○   | ○      | ○               | ○        | ○       | ○   |
| [52]                     | ShEF                               | FPGA fabric          | Cloud FPGAs    | ○                | ●                | ●                    | ●       | ○    | N/A  | ●           | ○   | ○      | ○               | ○        | ○       | ○   |
| [183]                    | DF-TEE                             | FPGA fabric          | Cloud FPGAs    | ○                | ●                | ●                    | ●       | ○    | N/A  | ○           | ○   | ○      | ○               | ○        | ○       | ○   |
| [184]                    | MeetGo                             | FPGA fabric          | Cloud FPGAs    | ○                | ●                | ●                    | ●       | ○    | N/A  | ●           | ○   | ○      | ○               | ○        | ○       | ○   |
| [185]                    | AccGuard                           | FPGA fabric          | Cloud FPGAs    | ○                | ●                | ●                    | ●       | ○    | N/A  | ○           | ○   | ○      | ○               | ○        | ○       | ○   |
| [186]                    | BYOTee                             | FPGA fabric          | SoC FPGA       | ○                | ●                | ●                    | ○       | ●    | ○    | ○           | ○   | ○      | ○               | ○        | ○       | ○   |
| <b>Other Research</b>    |                                    |                      |                |                  |                  |                      |         |      |      |             |     |        |                 |          |         |     |
| [28]                     | HETEE                              | Custom HW            | custom SoC     | N/A              | N/A              | ●                    | ●       | ○    | ●    | N/A         | ●   | N/A    | ●               | ●        | ●       | N/A |
| [187]                    | CURE                               | Custom HW            | custom SoC     | N/A              | N/A              | ●                    | ●       | ○    | ●    | N/A         | ●   | N/A    | ●               | ●        | ●       | N/A |
| [189]                    | Graviton                           | GPU                  | GPU            | N/A              | N/A              | ●                    | ●       | ○    | ●    | N/A         | ●   | N/A    | ○               | ○        | ○       | N/A |
| [190]                    | HIX                                | GPU                  | GPU            | N/A              | N/A              | ●                    | ●       | ○    | ●    | N/A         | ●   | N/A    | ○               | ○        | ○       | N/A |
| [191]                    | StrongBox                          | GPU                  | GPU            | N/A              | N/A              | ●                    | ●       | ○    | ●    | N/A         | ●   | N/A    | ○               | ○        | N/A     | N/A |
| [192]                    | XpuTEE                             | GPU                  | GPU            | N/A              | N/A              | ●                    | ●       | ○    | ●    | N/A         | ●   | N/A    | N/A             | ○        | ○       | N/A |
| [193]                    | LITE                               | GPU                  | GPU            | N/A              | N/A              | ○                    | ●       | ○    | ●    | N/A         | ●   | N/A    | ○               | ○        | ○       | N/A |
| [194]                    | GEVisor                            | GPU                  | GPU            | N/A              | N/A              | ●                    | ●       | ○    | ●    | N/A         | ●   | N/A    | ○               | ○        | ○       | N/A |
| [195]                    | Telekine                           | GPU                  | GPU            | N/A              | ○                | ●                    | ●       | ○    | ●    | N/A         | ●   | N/A    | ○               | ○        | N/A     | N/A |
| [196]                    | AccShield                          | FPGA fabric          | FPGA           | N/A              | N/A              | ●                    | ●       | ○    | ●    | ○           | ○   | N/A    | ○               | ○        | ○       | N/A |
| [197]                    | Confidential AI                    | Accel                | AI-Accel       | N/A              | N/A              | ●                    | ●       | ○    | ●    | N/A         | ●   | N/A    | ○               | ○        | ○       | N/A |
| [198]                    | sNPU                               | Accel                | AI-Accel       | N/A              | N/A              | ●                    | ●       | ○    | ●    | N/A         | ●   | N/A    | ○               | ○        | ○       | N/A |
| [200]                    | TNPU                               | Accel                | AI-Accel       | N/A              | N/A              | ●                    | ●       | ○    | ●    | N/A         | ●   | N/A    | ○               | ○        | N/A     | N/A |
| [201]                    | TensorTEE                          | Hybrid               | AI-Accel       | N/A              | N/A              | ●                    | ●       | ○    | ●    | N/A         | ●   | N/A    | ○               | ○        | N/A     | N/A |
| <b>Our Contributions</b> |                                    |                      |                |                  |                  |                      |         |      |      |             |     |        |                 |          |         |     |
|                          | <b>TEE0D</b>                       | FPGA fabric          | SoC FPGA       | ○                | ○                | ●                    | ○       | ●    | ○    | ○           | ○   | ○      | ●               | ●        | ○       | ●   |
|                          | <b>TREE</b>                        | FPGA fabric          | SoC FPGA       | ○                | ●                | ●                    | ●       | ●    | ○    | ●           | ●   | ●      | ●               | ●        | ●       | ●   |

Table 3: Comprehensive comparison and gap analysis of prior work on FPGA security and TEE systems. Key: RoT – Root of Trust; TP – Target Platform; F/T Res. – Fault/Tamper Resilience; I/A Det. – Intrusion/Anomaly Detection; TA-Cfg – TA Configuration; TA-Conf – TA Confidentiality; S-MT – Spatial Multi-Tenancy; T-MT – Temporal Multi-Tenancy; HW – Hardware TAs; SW – Software TAs; Hybrid – Hybrid (HW/SW) TAs; SS – Secure Storage; Encrypt. – Cryptographic Services; Attest. – Attestation Service; GP – GlobalPlatform TEE compliant. ● – full support; ○ – partial or limited support; ○ – no support; N/A – not applicable or not reported.

secure hardware accelerators through authenticated configuration, secure reconfiguration, and bitstream attestation. This focus is largely a result of how FPGAs have been traditionally integrated and utilized, as peripheral accelerators managed by a host processor, with limited visibility or control over the rest of the system. Consequently, most existing solutions target the same narrow threat surface within the programmable fabric or configuration flow.

Given the ongoing shift from CPU-centric designs toward heterogeneous computing architectures, this narrow perspective is no longer sufficient. Security challenges increasingly emerge from the interaction between xPUs. In such environments, untrusted software running on the CPU can target the PL, while malicious or compromised logic within the fabric can escalate privileges or interfere with trusted software, as discussed in Section 2.5. Additionally, most existing research overlooks the broader potential of FPGAs to contribute to system-wide trust. Only a few recent works have begun to recognize the active role of the PL in establishing trust, as discussed in Section 3.3.3.

**Our contribution.** We address this gap by demonstrating that FPGAs can evolve from passive accelerators into active roots of trust capable of hosting, enforcing, and maintaining trusted execution, while providing GlobalPlatform-compliant interfaces and standard TEE services. Both TEEOD and TREE aim to achieve this goal but address it from complementary perspectives. TEEOD aims to establish trusted execution within the fabric by instantiating pre-defined enclaves equipped with dedicated soft-MCUs, isolated memory, and optional hardware security services, thereby decoupling the trusted environment from CPU-centric assumptions. TREE, in turn, aims to extend this model by incorporating DPR as a trusted primitive to securely instantiate, update, and revoke hardware or hybrid TAs at runtime.

### 3.5.2 Limitations of CPU-centric TEEs in heterogeneous systems

Most existing TEEs technologies enforce isolation at the CPU level by partitioning execution contexts into secure and non-secure worlds. Despite their widespread adoption in mobile and embedded platforms, as well as in cloud and desktop environments, both have been compromised multiple times [30]. As discussed in Section 2.5, TrustZone-assisted TEEs exhibit architectural flaws (e.g., large TCBs and weak world isolation), implementation bugs (e.g., unchecked inputs and misconfigured access controls), and hardware-level vulnerabilities (e.g., microarchitectural side-channels, fault injection, and reconfigurable abuse). These weaknesses raise concerns about the reliability of any TEE system assisted by these TEEs technologies. Even considering FPGA-based TEEs that rely on TrustZone or SGX as their root of trust, such as SGX-FPGA [180], HISA [41], Embassy [181], TruFPGA [51], and SrcTEE [182], their security ultimately depends on the same vulnerable CPU-centric assumptions. Moreover, these systems further enlarge the TCB by integrating reconfiguration logic management, drivers, and interface layers into the secure world. Consequently, their security guarantees do not extend naturally to the reconfigurable domain, leaving the programmable fabric exposed once the processor-centric trust boundary is breached.

**Our contribution.** TEEOD operates entirely within the FPGA fabric and requires no trusted software on the PS, making it agnostic to conventional TEEs, i.e., Intel SGX and Arm TrustZone. Instead of extending their trust boundaries and thereby inheriting their associated vulnerabilities, TEEOD redefines them by establishing enclaves directly in the PL, each with its own minimal firmware, isolated memory, and optional hardware security services. This design removes dependency on vulnerable CPU-side privilege models and eliminates the need to enlarge the TCB with additional trusted drivers or configuration handlers. This design removes dependency on vulnerable TEEs hardware extensions and eliminates the need to enlarge the TCB with additional trusted drivers or configuration handlers. Moreover, by not relying on AXI-based tagging mechanisms, TEEOD inherently mitigates the hardware abuse scenarios demonstrated by Benhani et al. [15, 49] (Section 2.5.3), where malicious logic could manipulate bus protection signals to bypass TrustZone isolation. In TEEOD, each enclave operates under strict least-privilege access to shared interconnects, while access to secure storage and other services is exclusively mediated by loosely coupled, centralized FPGA-based agents, preventing privilege escalation or unauthorized data exposure within the fabric. As a result, trusted execution can be achieved within the reconfigurable fabric, independent of processor-centric assumptions, enabling fine-grained isolation and fabric-level attestation beyond what SGX- or TrustZone-assisted systems can provide.

### 3.5.3 Existing FPGA-based TEE solutions lack runtime flexibility and reconfigurable isolation

Most existing research on FPGA-based TEEs concentrates on extending conventional CPU-centric trust models, such as TrustZone and SGX, into the reconfigurable domain. These approaches primarily aim to protect hardware accelerators under the control of a trusted host processor by enabling authenticated configuration, secure bitstream loading, and attestation of deployed logic. As illustrated by frameworks such as SGX-FPGA [180], HISA [41], Embassy [181], TruFPGA [51], and SrcTEE [182], the programmable fabric is typically treated as a subordinate execution domain whose integrity ultimately depends on the host's secure world or management firmware. A second group of works targets cloud and multi-tenant environments, exemplified by ShEF [52], DF-TEE [183], and MeetGo [184], which focus on protecting reconfigurable accelerators in shared infrastructures through remote attestation, trusted provisioning, and inter-device trust chaining. Consequently, most FPGA-based TEEs are designed for edge or cloud scenarios, where powerful orchestration services and trusted runtime managers are assumed.

By contrast, only a few proposals explore trusted execution instantiated entirely within the reconfigurable fabric, independent of fixed CPU-based TEEs. Such FPGA-centric approaches, includes BY-OTee [186] and our contribution TEEOD, which relocate the TCB to the PL itself, enabling processor-independent enclaves with self-contained management, and runtime software programmability. Despite their advances, this line of reconfigurable security research remains largely underexplored. Existing solutions, while enabling TEE functionality within the fabric, do not support reconfigurable trust domains. In other words, they lack the capability to dynamically instantiate, update, or revoke enclaves at runtime

without interrupting execution or compromising isolation. The trust boundary in these systems remains static once the bitstream is provisioned, which prevents the deployment of adaptive or patchable trusted hardware, or hybrid modules that can respond to evolving application requirements.

**Our contribution.** The TREE framework aims to overcome these limitations by integrating DPR as a secure primitive for trusted execution. TREE partitions the programmable fabric into isolated and attestable RRs that can be securely reconfigured at runtime, allowing hardware, software, and hybrid TAs to coexist under a unified trusted framework. It enforces least-privilege access to shared TEE services and extends attestation to both static and dynamically reconfigured modules, ensuring that system integrity remains verifiable at all times. Through this design, TREE enables secure runtime flexibility and reconfigurable isolation, two capabilities that are absent from prior FPGA-based TEEs, and establishes a foundation for adaptive and patchable trusted systems.

## Concluding Remarks

In this chapter, we summarize the key contributions and findings of this thesis. Section 4.1 outlines the primary conclusions drawn from the work conducted for each publication. Section 4.2 discusses potential avenues for future research based on the insights gained from each work.

### 4.1 Conclusions

Modern SoCs integrate a wide range of xPUs, including SRAM-based FPGAs, to deliver higher performance and better energy efficiency. However, this heterogeneity also brings increased architectural complexity, and many of these platforms operate in untrusted environments where they face growing and unprecedented security threats. Ensuring that critical operations remain isolated and their integrity preserved has become significantly more difficult. A common belief is that hardwired security extensions are sufficient to protect sensitive assets since they are inherently immutable and help minimize the TCB. Although such mechanisms have proven effective against well-known threats, they remain poorly suited to evolving workloads and emerging attack vectors. This limitation has become evident in mainstream TEEs, where even extensive mitigation efforts have failed to fully address vulnerabilities stemming from fundamental design flaws in the underlying hardware. Furthermore, attempts to extend these fixed, CPU-centric protections across xPUs in modern systems have often fallen short or even introduced new vulnerabilities.

To address these challenges, we hypothesized that the reconfigurable nature of FPGAs could enable a new approach to trusted computing, one that combines the strong isolation guarantees of hardware with the ability to adapt defenses over runtime, a *patchable trust*. Accordingly, we focused our research on three central questions: i) what are the principal attack vectors in reconfigurable and heterogeneous systems, and how can FPGA capabilities be used to mitigate them; ii) how can trusted execution be instantiated and sustained directly within the reconfigurable fabric, independent of vulnerable CPU-centric TEEs; and iii) what are the architectural and performance trade-offs of dynamically deploying, updating, and enforcing isolation for security-critical modules within the fabric.

Through our systematic survey (Chapter 5), we identified fundamental gaps in existing reconfigurable security research. In particular, prior works lack a unified study on both in-fabric threats and potential

countermeasures, resulting in a fragmented understanding of how FPGA reconfigurability impacts the broader system security landscape. From this analysis, we introduced two complementary frameworks. The first, TEEOD, demonstrates how the reconfigurable fabric can be leveraged to instantiate isolated execution environments that extend conventional TEE concepts beyond fixed-function hardware. Second, we addressed the absence of runtime flexibility and reconfigurable isolation in existing FPGA-based TEEs by proposing TREE, a unified architecture that treats DPR as a secure primitive, enabling the creation, update, and revoke of enclaves at runtime without requiring system reboot or software intervention. Together, these contributions advance the notion of *patchable hardware* as an integral part of trusted computing. The resulting architectures, TEEOD and TREE, establish a new direction for reconfigurable-assisted security mechanisms that evolve alongside the systems they protect.

## 4.2 Future Work

While this thesis has demonstrated the potential of reconfigurable computing, particularly FPGA-related architectures, for realizing reconfigurable TEEs, several important research directions remain open for future exploration. These directions span both the technical deepening of mechanisms introduced here and the broader application of reconfigurable trust to heterogeneous computing platforms.

**Hierarchical and nested DPR for fine-grained reconfiguration.** The TREE framework demonstrates the viability of using DPR to support modular and adaptive TEEs. However, current implementations operate with relatively coarse-grained partitions due to the fixed tile sizes required by most FPGA design tools. Future work can explore nested DPR, in which reconfigurable modules themselves can contain sub-regions eligible for dynamic updates. This would enable faster switching between TAs and further decouple hardware accelerators from their managing soft-MCU instances. Such fine-grained reconfiguration could significantly reduce downtime and latency during context switches, and also allow for more compact and flexible co-designs of TAs and their required hardware functions.

**Extending FPGA-Based TEEs to other xPUs.** While this thesis primarily focused on establishing and enforcing trust within the FPGA fabric itself, and on enabling a broader range of TAs to execute securely within. A promising next step would be to explore how reconfigurable logic can serve as a system-wide TEE system for trust computation across the full SoC, such as emerging in-storage computing units. A particularly compelling direction arises from the work of Gaillardon et al. [202], who introduced the Programmable Logic-in-Memory (PLiM) concept. PLiM enables in-memory execution under the supervision of a lightweight controller that dynamically switches memory cells between storage and logic modes. Building upon this idea, we envision FPGAs acting as secure control and central trust management unit for such memory-centric architectures or other xPUs. In this role, the FPGA could enforce access control, attestation, and runtime policy management across distributed compute elements, effectively extending the principles of TEEOD to the system level. For example, in an smart storage computing context, the

FPGA could authorize which TAs may invoke logic operations on designated memory regions, isolate computation to trusted banks, and ensure continuous integrity verification of non-volatile states. This also opens avenues for secure non-volatile secure storage management.

**Integrating passive countermeasures against physical attacks.** While this thesis primarily addressed logical and architectural security challenges, physical attacks remain a critical concern, particularly in hostile or remote deployment environments. Passive protection techniques and runtime monitoring, discussed in Section 3.1, provide a foundation for detecting and reducing such threats, but the systematic integration of these techniques with FPGA-based TEE frameworks has not yet been fully explored. Although most of these countermeasures are orthogonal and do not necessarily affect or condition the security foundations of the TEEOD and TREE frameworks, their inclusion could complement them by hardening the programmable fabric against remote tampering, fault injection, and side-channel attacks. Future work should therefore explore how preventive solutions can be embedded within these frameworks to extend protection from logical isolation toward comprehensive physical-layer resilience.

# **Part II**

## **Publications**

# List of Publications

## Publications in this Thesis

- [65] **S. Pereira**, T. Gomes, J. Cabral, and S. Pinto. "Beneath the fabric: A survey on threats and opportunities in reconfigurable systems". [Submitted to Journal of Systems Architecture]
- [66] **S. Pereira**, D. Cerdeira, C. Rodrigues, and S. Pinto. "Rethinking Trusted Execution Environments in the Age of Reconfigurable Computing". [Submitted to IEEE Access]
- [67] **S. Pereira**, T. Gomes, J. Cabral, and S. Pinto. "TREE: Bridging the gap between reconfigurable computing and secure execution". IACR Transactions on CHES 25.

## Beneath the fabric: A survey on threats and opportunities in reconfigurable systems

***This chapter was submitted as:***

Sérgio Pereira, Tiago Gomes, Jorge Cabral, and Sandro Pinto. "Beneath the fabric: A survey on threats and opportunities in reconfigurable systems". [Submitted to Journal of Systems Architecture]

## Abstract

Modern heterogeneous systems increasingly rely on reconfigurable computing to achieve high performance, energy efficiency, and adaptability across diverse domains such as datacenters, AI accelerators, and Internet-of-Things (IoT) devices. However, the same reconfiguration mechanisms that provide flexibility also expand the attack surface, introducing security challenges that are beyond conventional threat models. This survey presents a comprehensive systematization of knowledge on FPGA security, emphasizing how reconfigurable logic can be either exploited to compromise systems or leverage to enhance the security of modern systems. We classify FPGA-based adversarial techniques across the FPGA design and runtime lifecycle and examine corresponding FPGA-based countermeasures, including obfuscation, redundancy, runtime monitoring, attestation, and secure reconfiguration. In addition, we review the emerging landscape of FPGA-based Trusted Execution Environments (TEEs) that extend or redefine trust boundaries within heterogeneous architectures. Finally, we identify future research directions toward FPGA-aware threat models and reconfiguration-driven security architectures that establish adaptability as a foundation for resilient and verifiable trust.

## 5.1 Introduction

Field-Programmable Gate Arrays (FPGAs) were originally introduced for rapid prototyping and hardware verification, allowing designers to test digital circuits directly in hardware without incurring the non-recurring engineering costs of ASIC development [203]. Over time, their role expanded beyond prototyping, and they became integral to systems that demand adaptability, fine-grained parallelism, and low-latency computation [204]. Today, reconfigurable computing spans a wide spectrum of computing domains: from high-performance computing (HPC) clusters and datacenters, where FPGAs accelerate computation-intensive workloads [39, 205], to deeply embedded and edge devices, where the same reconfigurable fabric provides energy-efficient, application-specific acceleration under the stringent constraints of Internet of things (IoT) systems [97, 206]. When properly configured, the reconfigurable fabric can behave as simple finite-state machines, up to fully featured processors or microcontrollers (MCUs) implementing advanced processing pipelines, memory hierarchies, and peripheral interfaces [207].

While FPGAs deliver near hardware-level performance and unmatched flexibility, these same characteristics make them a double-edged sword. Their programmable fabric, fine-grained parallelism, and precise timing control can be exploited to implement powerful attack vectors. Prior research have demonstrated how adversaries can leverage FPGA resources to mount side-channel attacks [45–47, 106], inject faults [102–105], or even compromise trusted computing infrastructures [49, 107, 108]. Consequently, the integration of reconfigurable computing within heterogeneous systems calls for a fundamental re-examination of established hardware trust assumptions.

At the same time, the intrinsic reconfigurability that exposes FPGA to adversarial exploitation also presents unique opportunities for security enhancement. Several works have explored reconfigurable

| Survey Paper                       | Year | Relevant Contributions                                                                                             |
|------------------------------------|------|--------------------------------------------------------------------------------------------------------------------|
| <b>FPGA-related</b>                |      |                                                                                                                    |
| A. Proulx et al. [209]             | 2022 | Surveys cybersecurity risks in SoC FPGAs, focusing on confidentiality, integrity, and validity.                    |
| S. Duan et al. [210]               | 2021 | Reviews side-channel, fault injection, and covert channel attacks on FPGAs and related defenses.                   |
| D. G. Mahmoud et al. [23]          | 2022 | Surveys electrical-level attacks across CPUs, FPGAs, and GPUs in heterogeneous systems.                            |
| M. K. Ahmed et al. [211]           | 2025 | Reviews security, trust, and privacy in multi-tenant cloud FPGAs; highlights isolation and cryptographic defenses. |
| P. D. Rosero-Montalvo et al. [212] | 2023 | Covers trusted computing on FPGAs; focuses on hardware security primitives.                                        |
| F. Turan et al. [69]               | 2021 | Surveys trust issues in FPGA cloud computing across the system lifecycle.                                          |
| G. Dessouky et al. [50]            | 2021 | Explores secure resource sharing and isolation in multi-tenant FPGA platforms; maps attack vectors and defenses.   |
| <b>TEE-related</b>                 |      |                                                                                                                    |
| A. Muñoz et al. [29]               | 2023 | Reviews TEE vulnerabilities across software, hardware, and side channels.                                          |
| D. Cerdeira et al. [30]            | 2020 | Examines TrustZone vulnerabilities, including threats via reconfigurable hardware in SoCs.                         |
| S. Pinto et al. [213]              | 2019 | Surveys ARM TrustZone: architecture, applications, and known attack vectors.                                       |

Table 4: Overview of existing survey papers addressing FPGA and TEE security.

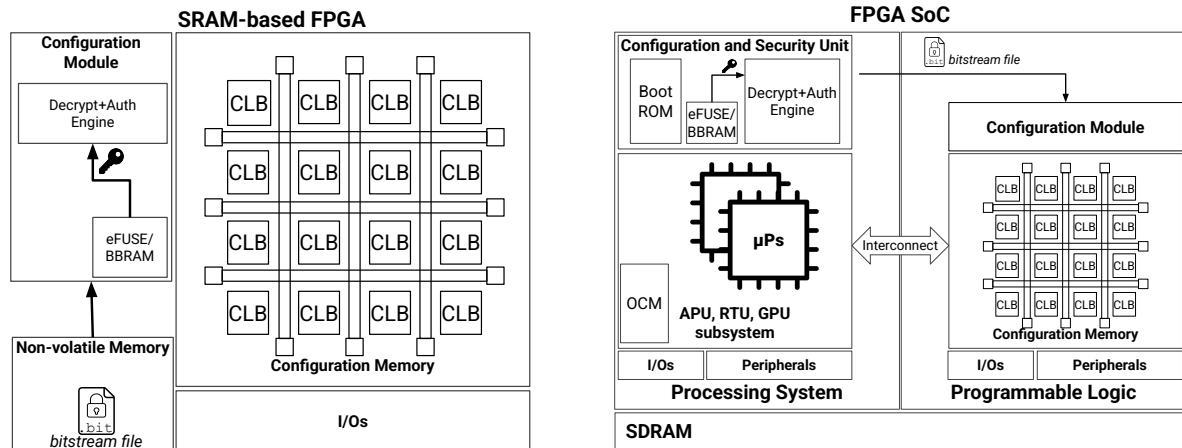
computing to mitigate specific threats. These include both passive techniques [122–126] and active monitoring [42, 53, 127–133, 138–140], as well as secure reconfiguration protocols that authenticate and attest bitstreams before they become operational [134–137]. Other works have further advanced system-level efforts that study the feasibility of using reconfigurable logic to host or co-manage operating system (OS) services [144–153] and hypervisors [154–162], abstracting hardware tasks as schedulable resources and virtualization within the fabric for multi-tenant use. More recently, research has leveraged FPGAs to construct Trusted Execution Environments (TEEs) that extend or replace traditional CPU-based models such as Arm TrustZone and Intel SGX [28, 41, 51, 52, 67, 180–187, 208].

This survey intend to provide a comprehensive overview of the current landscape of FPGA security, analyzing both the threats that emerge from reconfigurable architectures and the defensive mechanisms that exploit this same property. Despite the growing body of literature addressing FPGA security most existing surveys address FPGA security either from a device-centric or bitstream-centric viewpoint. Table 4 summarizes representative works and highlights their primary focus areas, revealing that prior efforts tend to cluster around three main dimensions. Some focus primarily on low-level attacks such as tampering, reverse engineering, or fault injection [23, 209, 210], while others emphasize cloud and virtualization aspects, where resource sharing introduces confidentiality and isolation challenges [50, 69, 211, 212]. Meanwhile, a third line of work investigates the vulnerabilities of CPU-based TEEs when extended to heterogeneous SoCs [29, 30, 213]. However, none of these efforts center their analysis on the reconfigurable logic itself as an active security agent or attack surface. They largely treat the FPGA as a peripheral or accelerator whose configuration requires protection.

Our goal is not merely to catalogue vulnerabilities of FPGA devices but to systematize how the reconfiguration capability itself can be abused to compromise systems or, conversely, be leveraged to implement adaptive, hardware-level security mechanisms. By reframing the analysis around the capabilities of reconfigurable logic, rather than around static architectural properties, we aim to bridge the gap between adversarial and protective uses of reconfigurable hardware. Accordingly, this work strives to provide a unified perspective that connects adversarial reconfiguration, in-fabric countermeasures, and trusted execution paradigms. We emphasize that reconfigurability is not inherently a vulnerability but rather a security-enabling property, capable of supporting adaptive isolation, runtime patching, and on-demand trust reestablishment in modern heterogeneous platforms.

To bring clarity to this landscape, our main **contributions** are as follows:

- **Systematization of reconfiguration attack vectors.** We focus our analysis on threats that exploit the reconfiguration logic of FPGA-based platforms. We first systematize the potential entry points through which adversaries may inject or modify malicious logic during the configuration or runtime phases. We then categorize and analyze the classes of in-fabric attacks that emerge once malicious logic is instantiated within the reconfigurable fabric.
- **Analysis of FPGA-based countermeasures.** We provide a comprehensive review of defensive mechanisms implemented directly in the reconfigurable fabric, such as obfuscation, redundancy, monitoring, and attestation frameworks, highlighting how the instantiation of custom hardware logic on FPGAs can be leveraged not only for functionality but also as a medium for IP protection, fault tolerance, and runtime enforcement.
- **Survey of FPGA-based Trusted Execution Environments.** We examine alternative TEE paradigms that extend beyond conventional CPU-centric designs to incorporate the reconfigurable fabric as part of the trusted domain. We categorize existing proposals into: i) extensions of CPU-based TEEs; ii) novel TEE architectures; iii) cloud-oriented frameworks; and iv) fully FPGA-based approach.
- **Perspective on future research directions.** Building on this analysis, we outline prospective research directions toward FPGA-aware threat models and reconfiguration-driven security architectures, where reconfigurability is elevated from a passive design property to an active security service leveraged for continuously reconfiguring, patching, and evolving hardware trust boundaries at runtime.



(a) Architecture of a typical SRAM-based FPGA, highlighting the configuration memory, configuration interfaces, and bitstream storage.

(b) Architecture of an SoC FPGA, highlighting configuration mechanisms and interfaces ports.

Figure 12: Comparison between standalone SRAM-based FPGA architecture and a SoC FPGA architecture.

## 5.2 Background

### 5.2.1 FPGAs and SoC FPGAs architectures

A FPGA device consists of arrays of configurable logic blocks (CLBs) interconnected through programmable routing. Each CLB contains basic logic elements, i.e., look-up tables (LUTs) and flip-flops (FFs), which, when combined, enable the implementation of complex logic functions (e.g., state machines or arithmetic operations). FPGAs also include block RAM (BRAM), digital signal processing units (DSPs), and input/output (IO) interfaces, which support memory storage, arithmetic processing, and external communication, respectively. Many designs also incorporate reusable intellectual property (IP) cores, that is, predefined logic blocks that encapsulate common functionalities such as communication protocols, cryptographic engines, or custom accelerators, greatly accelerating development and system integration. The FPGA's behavior is defined by a binary configuration file known as the bitstream. This file specifies how the CLBs and routing resources are arranged and connected to implement user defined logic circuits and is typically developed using hardware description languages (HDLs), e.g., Verilog, SystemVerilog, or VHDL [214].

**SRAM-based FPGAs.** Figure 12a illustrates a typical SRAM-based FPGA architecture. In such devices, the routing configuration is held in volatile SRAM cells, which are cleared upon power loss. To ensure configuration persistence, the bitstream is stored externally in non-volatile memory (NVM), from which it is reloaded each time the device powers up or is reconfigured. A dedicated configuration module is responsible for fetching the bitstream from the NVM, verifying its authenticity and integrity, and loading it into the configuration memory. Configuration is typically performed through dedicated interfaces such as JTAG, SelectMAP, or the internal configuration access port (ICAP), depending on the platform and system requirements.

**SoC FPGAs.** These architectures combine programmable logic (PL) with an integrated processing system (PS) on a single chip, as shown in Figure 12b. The PL retains the reconfigurable structure found in conventional FPGAs, while the PS typically consists of general purpose processor cores. Most architectures, such as those from AMD (formerly Xilinx) and Intel (formerly Altera), are based on Arm cores, although recent devices increasingly support RISC-V architectures [215]. Modern SoC FPGAs extend the PS with additional specialized units, including GPUs, RPU, DSPs, and AI engines [12]. Configuration of the PL is managed by a dedicated configuration and security unit (CSU). The CSU coordinates with immutable on-chip components such as the boot ROM and non-volatile key storage elements, i.e., eFUSEs or battery-backed RAM (BBRAM), to fetch, decrypt, and authenticate the bitstream before loading the configuration data into the programmable fabric. Bitstream configuration is achieved through conventional interfaces as well as a processor configuration access port (PCAP), which allows the PS to directly control the FPGA fabric. This port is often the default configuration interface, since configuring the PL is generally performed in tandem with the main system boot under the control of the CSU.

**Partial Reconfiguration (PR).** This mechanism enables the selective updating of specific regions of the FPGA fabric, designated as reconfigurable regions (RR), without disrupting the ongoing operation of other parts of the system. The FPGA is partitioned into static and dynamic regions. The static region continues normal operation, while the dynamic region can be updated independently, without requiring the static region to be reconfigured. When the reconfiguration process is performed at runtime, while the static region remains active, it is referred to as dynamic partial reconfiguration (DPR) [84]. This method significantly reduces reconfiguration time compared to full reconfiguration and is advantageous for applications where minimal downtime and adaptability are critical. PR is also useful in scenarios that require implementing large sets of logic functions, but not all at once, for example, in multi-tenant cloud environments [216].

### 5.2.2 Built-in Reconfiguration Security

The configuration process is a critical phase in FPGA operation, as it defines the logic and interconnects that determine the device's functional behavior. Consequently, protecting this process from tampering or unauthorized access is fundamental to ensuring system trust. Modern FPGA platforms therefore integrate a comprehensive set of built-in security mechanisms to protect the integrity and confidentiality of the configuration flow. For instance, AMD's Zynq UltraScale+ architectures [81] integrates AES-GCM decryption with integrity checking and supports RSA-4096 signature verification using SHA-3. Intel's Stratix 10 [82] implements similar protections using ECDSA signatures and SHA-based hashing. These cryptographic guarantees ensure that only authenticated bitstreams are loaded, mitigating the risk of tampering or IP theft. Additionally, mutual exclusion between PCAP and ICAP ports prevents concurrent configuration pathways, further reducing the attack surface.

### 5.2.3 TrustZone-assisted TEEs

TrustZone is a hardware security extension that partitions the system into two protection contexts: the secure world (SW) and the normal world (NW). These domains are enforced at both the processor and system levels, with transitions managed through secure monitor calls (SMCs). The active domain is tracked by the processor's non-secure (NS) bit, where a value of 0 indicates SW execution and a value of 1 indicates NW execution [56].

Since in most commercial SoC FPGAs the PS is Arm-based, they natively integrate TrustZone as a mechanism for isolating trusted and untrusted domains [81, 82]. In TrustZone-assisted SoC FPGAs, the guarantees provided by the TEE are extended beyond the PS to encompass the memory subsystem, peripheral interfaces, and the PL through the AMBA AXI interconnect. Each AXI transaction carries protection attributes (ARPROT and AWPROT) that encode the NS bit. Downstream components such as DRAM controllers, I/O peripherals, and custom logic instantiated in the PL interpret these attributes to enforce domain-specific access control. For instance, a reconfigurable IP is instantiated as a secure AXI slave within the PL accepts transactions exclusively from secure-world masters, thereby preventing non-secure entities from accessing its configuration or data interfaces. Conversely, non-secure logic such as user-space accelerators or DMA engines are restricted to the NW address space. If a mismatch occurs between the initiator's security attribute and the target's assigned domain, the AXI interconnect detects the violation and rejects the transaction [15, 49].

## 5.3 Adversarial uses of reconfigurable hardware

Once an adversary gains access to the fabric, new classes of threats emerge that operate entirely within the FPGA. Given that FPGAs provide high throughput parallel processing, fine-grained hardware customization, and runtime reconfigurability, adversaries can therefore craft logic that manipulates shared on-chip resources, interferes with co-located circuits, or compromise system trust domains, all while remaining within the syntactically valid and cryptographically authenticated configuration space. In this section, we first outline the primary entry points through which malicious logic can be introduced into the fabric, and then examine representative attack classes that exploit these capabilities to operate from within the FPGA itself.

### 5.3.1 Entry points to the reconfigurable fabric

Built-in vendor protections, i.e., bitstream encryption, authentication, and secure boot chains, primarily focus on preserving the integrity of the configuration flow, ensuring that only authorized bitstreams are loaded. However, these measures do not guarantee that the configured logic itself is trustworthy. Even a cryptographically valid bitstream may contain malicious or defective components introduced earlier in the design flow or during runtime reconfiguration [98].

**Untrusted bitstream development.** Zhang et al. [99] demonstrated that even trusted design tools can become a vector for Trojan insertion: a compromised CAD or synthesis suite can silently alter the netlist during high-level synthesis or optimization, embedding additional logic that remains undetected by the developer. Similarly, Ahmed et al. [98] showed that adversaries can inject Trojans during placement and routing, directly manipulating the bitstream to embed malicious reconfigurable regions that bypass standard design rule checks and post-layout validation. Because these manipulations occur often after formal verification, the resulting bitstream remains syntactically valid, cryptographically authentic, and thus trusted by the device. The embedded payload may remain dormant until specific runtime conditions trigger its activation, making detection extremely challenging.

**Exposed reconfiguration interfaces.** Even when the design tools are trusted, other entry points remain. Software exploits in the software, such as buffer overflows and memory corruption bugs in privileged drivers or firmware, have enabled arbitrary code execution on the CPU, allowing direct access to memory-mapped configuration registers and bitstream storage [100]. At the hardware level, Timmers and Spruyt [53] demonstrated that voltage fault injection on the ARM Cortex-A9 of a Zynq-7000 can skip critical authentication instructions during secure boot, thereby bypassing bitstream verification and unlocking the configuration control registers. Furthermore, even if the reconfiguration interfaces, i.e., ICAP or PCAP, are under the control of a CPU-extension TEE, such as TrustZone-assisted TEEs, the assumed isolation remain vulnerable. Notably, recent studies [30] have shown that commercial TEE systems contain numerous and severe vulnerabilities, many of which affect user-level TAs and the trusted OS itself. Such weaknesses can be chained to escalate privileges into the secure world, ultimately granting an adversary access to secure peripherals and configuration interfaces.

**Third-party IP dependency.** Modern reconfigurable designs increasingly rely on third-party IP blocks that are opaque, under-verified, or distributed as encrypted netlists. While this accelerates prototyping and reduces development costs, it also introduces unverifiable components that adversaries can exploit to embed reconfigurable hardware Trojans [101].

### 5.3.2 FPGA-based physical attacks

Physical attacks exploit the electrical, electromagnetic, thermal, and timing behavior of any hardware to compromise confidentiality, integrity, or availability. Traditionally associated with invasive laboratory techniques (e.g., microprobing, focused ion beam), the term has broadened to include non-invasive fault injections, side-channel exploits and covert channels that do not require physical access. Although these threats are common to integrated circuits in general, FPGAs deserve focused attention.

The intrinsic reconfigurability of FPGAs allows an adversary to instantiate malicious hardware primitives, e.g., such as oscillators, voltage sensors, delay lines, or power-hungry circuits, directly within the programmable fabric. These malicious logic can induce or monitor physical effects that interfere with

neighboring logic, enabling remote fault injection, side-channel attacks, or denial-of-service conditions without any external probing equipment.

**Fault injection attacks.** Gnad et al. [102] demonstrated that a malicious FPGA circuit can induce faults in co-located circuits by manipulating the power distribution network. On AMD Virtex-7, Kintex-7, and Zynq-7020 devices, they instantiated numerous fast-switching ring oscillators (ROs) to draw transient currents, generating voltage droop and timing instability. These fluctuations led to effects ranging from subtle computation errors to complete system resets. Expanding on this idea, Krautter et al. [103] introduced FPGAhammer, a fault attack modeled after Rowhammer. They densely packed toggling logic on an Intel Cyclone V SoC FPGA to brown-out an AES core, injecting faults that enabled recovery of 90% of the secret key using Differential Fault Analysis (DFA), all while occupying just 35–45% of the logic fabric. Alam et al. [104] proposed another fault mechanism using dual-port block RAMs: by issuing conflicting writes to the same address, they induced short circuits and localized heating, causing timing violations and bit flips in configuration memory. These faults could be targeted to alter circuit behavior. More recently, Luo et al. [105] demonstrated that complex workloads such as deep neural networks (DNNs) are also vulnerable. By deploying an oscillating circuit on a Zynq-7000 SoC, they injected timed power glitches into a neighboring DNN accelerator, degrading inference accuracy.

**Side-channel attacks.** Several studies have shown that side-channel techniques can be mounted internally within the FPGA fabric [45–47, 106]. These attacks exploit shared physical resources—such as power rails or electromagnetic coupling—to extract information about a neighboring circuit’s activity. Gravellier et al. [106] demonstrated cross-domain side-channel attacks on a Zynq-7000 SoC: a malicious hardware module successfully recovered AES keys from both a software implementation running on the ARM core and a hardware implementation in the PL. This shows that untrusted logic can monitor power or timing signatures of other processes on the same device. In multi-tenant cloud FPGAs, such attacks could be particularly dangerous, allowing one tenant to extract cryptographic secrets from another without any physical access or modification to the device.

### 5.3.3 Breaking Secure Boot

A notable FPGA-to-CPU attack was demonstrated by Jacob et al. [107], who embedded a hardware Trojan in the PL of an Zynq-7000 SoC to subvert the secure boot process. Masquerading as a benign cryptographic accelerator, the Trojan was programmed to monitor the boot sequence and modify specific memory contents once U-Boot was loaded into external DRAM. By altering these values, it exploited a flaw in the bootloader validation flow, causing the CPU to skip expected authentication checks and boot an unsigned, attacker-controlled kernel. This completely bypassed the secure boot’s cryptographic integrity guarantees. Critically, once the malicious bitstream was loaded into the PL, the attack proceeded without any further CPU interaction. This demonstrates how an apparently harmless FPGA configuration can

compromise one of the system's most foundational trust anchors.

### **5.3.4 DRAM attacks**

The PL's ability to act as a high-performance bus master gives it unrestricted access to shared DDR memory in many system designs. This opens the door to memory corruption attacks initiated from the FPGA side, often without the visibility or control mechanisms that apply to software running on the PS. In the JackHammer attack, Weissman et al. [108] exploited this access pattern to mount a Rowhammer-style fault injection against memory used by a CPU process. They combined a CPU-driven hammering pattern with a logic circuit instantiated in the PL to create high-frequency toggling and maximize disturbance in adjacent memory rows. Running on an Intel Arria 10 programmable acceleration card, the combined attack successfully induced bit flips in DRAM, targeting a WolfSSL RSA application, running on the host CPU. The injected faults allowed extraction of TLS session keys without triggering any privilege violations. This highlights how a malicious PL design can act as a stealthy adversary against software assets in the PS.

### **5.3.5 TrustZone and AXI Interface Exploitation**

Researchers have demonstrated that the PL can be used to bypass TrustZone protections by directly manipulating the AMBA AXI interconnect, which connects the PL and OS. Benhani et al. [15, 49] performed a series of six proof-of-concept attacks on an Zynq-7000 SoC, showing that a malicious or misconfigured FPGA bitstream could issue crafted transactions that accessed secure-world memory or impersonated trusted peripherals. These attacks required only minor alterations to the hardware design and exploited common misconfigurations in AXI access control logic.

## **5.4 Defensive uses of reconfigurable hardware**

Researchers have proposed solutions that use the reconfigurable logic not only to implement functionality, but also to monitor, isolate, and protect it. We categorize the academic proposals in this domain into two major directions. The first focuses on IP protection, which includes obfuscation and redundancy-based techniques that aim to conceal or harden logic against extraction, tampering, and fault injection. The second centers on runtime monitoring and enforcement, encompassing attestation mechanisms, on-chip anomaly detection, frameworks for securing third-party IP integration, and FPGA-based virtualization systems that enforce isolation between reconfiguration regions.

### 5.4.1 Passive countermeasures

The confidentiality and integrity of IP remain central concerns in reconfigurable computing, where third-party or proprietary modules are routinely deployed on shared hardware. Beyond vendor-provided encryption, numerous academic efforts have explored passive techniques to obscure design intent, prevent cloning, and ensure functional correctness under adversarial conditions. These works aim to reduce information extraction, frustrate reverse engineering, and maintain correct operation even when exposed to tampering or fault injection.

**Obfuscation-based techniques.** Obfuscation techniques make it harder for adversaries to reverse engineer or manipulate an FPGA design. Hoque et al. [122] use unused logic blocks to scramble sensitive areas of the bitstream and insert misleading paths to confuse tampering attempts. Other approaches equalize the timing of operations [123] or stabilize power consumption [124] to resist side-channel analysis. Dubey et al. [125] extend this to protect machine learning models. Moreover, Sasdrich et al. [126] propose a uniquely FPGA-based countermeasure to side-channels which involve using DPR to periodically relocate or shuffle sensitive logic, effectively randomizing physical placement and disrupting profiling.

**Redundancy-based techniques.** Redundancy techniques add replicated logic or computations to detect and tolerate faults. Triple Modular Redundancy (TMR) is often used to protect critical logic, including configuration control paths. In cryptographic cores, duplication or dual-rail encoding detects deviations due to fault injection [42, 127]. Physical re-randomization of modules through partial reconfiguration helps prevent adversaries from repeatedly targeting the same physical area [128]. In SoC FPGAs, software-level redundancy—such as repeated authentication checks—adds further protection against control flow tampering [53].

### 5.4.2 Runtime Monitoring and Enforcement

Unlike design-time protections, runtime monitoring mechanisms are implemented directly in the FPGA fabric and operate during execution. By leveraging reconfigurable logic resources, these techniques can be customized, updated, or relocated as needed, offering flexible defenses that are unique to FPGA platforms.

**Tampering and probing detection.** FPGA-based sensors can be embedded in the logic fabric to detect physical disturbances caused by fault injection or probing. For example, He et al. proposed high-frequency RO watchdogs to detect changes in delay caused by laser pulses or voltage glitches [129–131]. These oscillators are sensitive to localized timing shifts and can trigger alarms or activate mitigation routines. Similarly, delay-sensing circuits can monitor path delays in real time and reduce clock frequency in response to anomalies, helping suppress transient faults [132]. Optical attacks can be detected using embedded PUF-based structures. Tajik et al. [133] demonstrate that RO-based PUFs, when placed in the FPGA fabric, show measurable disturbances under optical probing. By continuously sampling the output

of these circuits, the system can detect and respond to invasive attacks that attempt to read internal signals or configuration bits through the silicon substrate.

**Attestation Mechanisms.** Works in this line strengthen the reconfiguration process and provide cryptographic evidence of correct system state. Drimer and Kuhn [134] introduced one of the first authenticated reconfiguration protocols for FPGAs, where bitstreams are digitally signed to prevent unauthorized configurations from being loaded remotely. Vliegen et al. [135] later proposed a secure remote reconfiguration scheme that maintains confidentiality and integrity during dynamic updates by combining bitstream encryption with mutual authentication. Kashyap and Chaves [136] extended these ideas by presenting a compact on-the-fly secure dynamic reconfiguration architecture for volatile FPGAs, ensuring only authorized bitstreams can occupy defined reconfigurable regions. Building on these foundations, Vliegen et al. [137] introduced SACHa, a self-attestation framework that allows the fabric to autonomously verify their configuration integrity using lightweight cryptographic primitives, eliminating the need for continuous external supervision.

**Monitoring and anomaly detection.** Because many runtime threats can bypass pre-deployment checks, embedding detection logic directly into the FPGA fabric has become a key defense strategy. Yao et al. [138] proposed a multipurpose RO grid that continuously monitors the power distribution network for anomalies, enabling lightweight detection of side-channel attacks and fault-injection attempts. Alanwar et al. [139] developed a dynamic detection and protection framework against hardware Trojans in reconfigurable fabrics, combining runtime monitoring with logic-locking and configuration fingerprinting to identify malicious alterations after deployment. Ahmadi et al. [140] further advanced these ideas with FPGA-patch, which dynamically mitigates remote side-channel attacks by applying reconfigurable countermeasures through PR. In shared FPGA environments, Krautter et al.'s FPGAhammer demonstrates how aggressive RO activation can induce voltage droops and timing faults across partitions; embedded sensors, such as timing or LUT-based voltage-drop detectors, can thus serve as early alarms against fault or power-injection attacks [103]. By exploiting underutilized logic and routing resources, these monitoring schemes achieve low overhead while providing continuous vigilance against malicious or faulty behavior in reconfigurable systems.

**Frameworks for integration of third-party IPs.** Restuccia et al. [141] introduced Bastion, a framework to secure the integration of third-party IP blocks in network-on-chips (NoCs) by deploying distributed monitors that enforce traffic integrity, access control, and anomaly detection. This approach enables the isolation of compromised IP without disrupting the platform, though it depends on accurate policy definition and incurs integration overhead. Complementing this, Tan et al. [142] proposed monitoring and mitigation patches, lightweight modules that attach to IP interfaces, observe I/O behavior, and enforce corrective actions at runtime. Unlike design-time verification, modules-on-partition (MoPs) enable post-deployment patching of black-box IPs, extending survivability in long-lived devices such as IoT platforms.

Building on this, Dharavathu and Tan [143] explored using FPGA as reconfigurable monitors integrated at different system levels—tightly with IP blocks, coupled to DMA controllers, or as bus-compliant replacements. Their work demonstrates how FPGA logic can enforce corrective actions dynamically and provide a field-updatable enforcement layer against unforeseen vulnerabilities. Together, these frameworks highlight the need for runtime containment and adaptive monitoring to secure third-party IPs in heterogeneous SoCs.

### 5.4.3 FPGA-based OS and Virtualization

The growing adoption of reconfigurable computing in shared infrastructures has motivated efforts to virtualize and manage FPGA resources with the same abstraction level as conventional processors. This paradigm has led to the emergence of FPGA-based OSs and virtualization frameworks, which aim to provide process-like abstractions, standardized interfaces, and secure multiplexing of reconfigurable hardware resources.

**OSs for reconfigurable devices.** Efforts to abstract reconfigurable hardware as managed computational resources have led to several OS frameworks that integrate hardware tasks into conventional software execution models. Early systems such as BORPH [144], FUSE [145], and ReconOS [146] extended Linux-like environments to treat FPGA modules as hardware threads or processes, enabling hybrid multiprocessing between CPUs and reconfigurable logic. Subsequent designs, including HybridOS [147], Feniks [148], LEAP [149], and AmorphOS [150], introduced modular abstractions and virtualization primitives to support multi-tenant management and resource sharing in cloud-scale deployments. More lightweight approaches such as FOS [151], ReproRun [152], and Arnold [153] adapt these principles to IoT and edge environments through reconfigurable runtime services and near-sensor hardware–software co-design. While these systems enhance programmability and flexibility, most lack strong isolation, authenticated context management, and secure module launch—leaving hardware-enforced security as an open research challenge.

**FPGA virtualization.** The increasing deployment of FPGAs in data centers has driven extensive research on virtualization frameworks that enable spatial and temporal sharing of the reconfigurable fabric across multiple tenants. Early efforts such as CoRAM [154] and OpenStack-based provisioning [155] explored memory and resource abstraction for accelerators through IOMMU-assisted DMA isolation and cloud orchestration. Later works, including the FPGA hypervisors proposed by Fahmy et al. [156] and Knodel et al. [157], introduced virtual slot management to mediate configuration, I/O, and memory access. Asiatici et al. [158] extended these concepts to SoC-based platforms through runtime management of partial regions and DMA control, while Mandebi et al. [159] leveraged spatial partitioning and lightweight NoCs to create isolated virtual regions within shared devices. More recent systems such as Optimus [160] and the frameworks by Zha et al. [161, 162] enhance utilization and scalability by slicing FPGA memory pages and decomposing bitstreams for fine-grained placement across FPGA clusters.

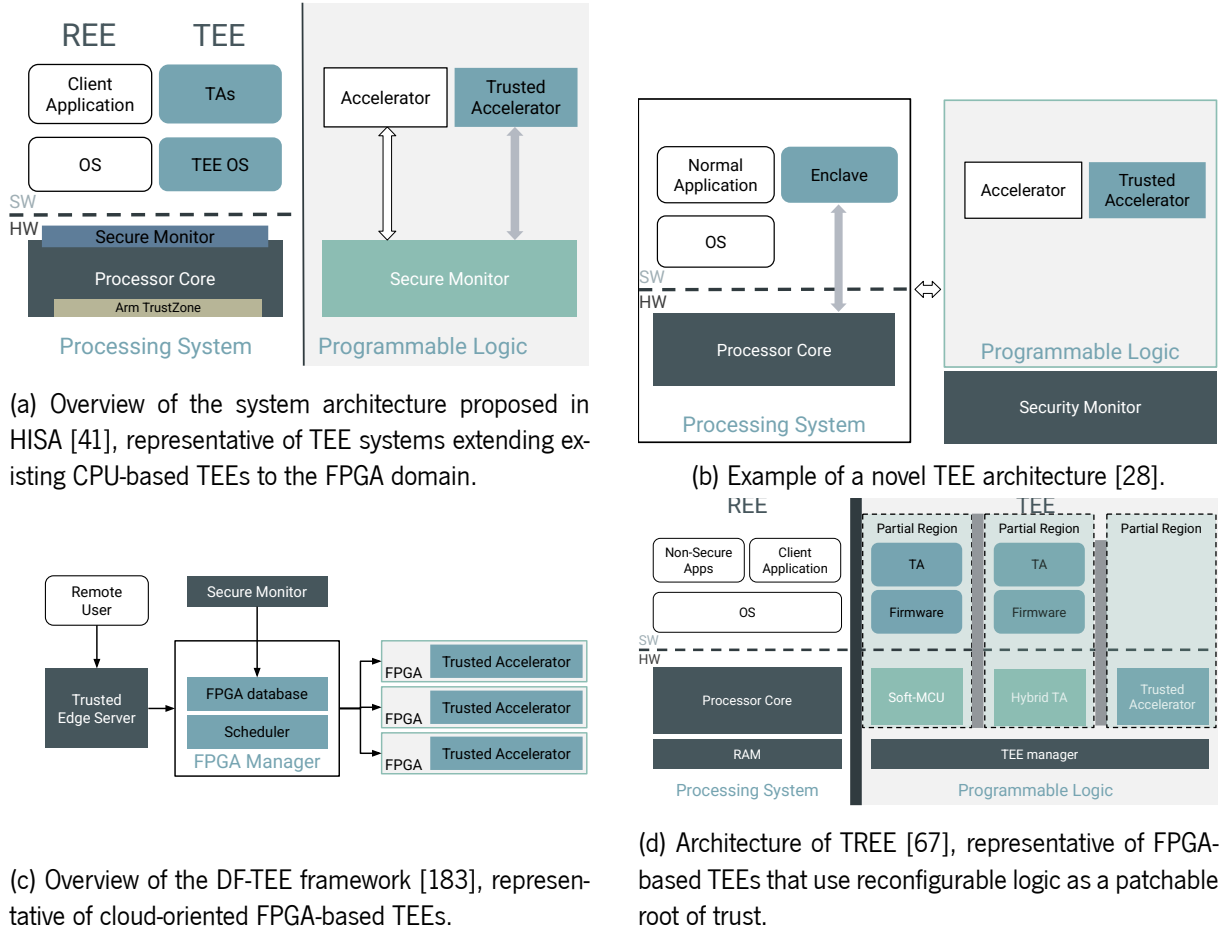


Figure 13: Overview of representative FPGA-oriented TEE designs.

## 5.5 Trusted execution in heterogeneous systems

TEEs have surged as a security paradigm to counteract the growing complexity and enlarged attack surface of modern computing platforms. By partitioning the system into trusted and untrusted domains, TEEs enforce hardware-assisted isolation to ensure the confidentiality and integrity of sensitive code and data from potentially compromised components, including OSs and user applications. At a minimum, a TEE provides: i) secure or measured boot, ensuring that only authenticated and integrity-verified firmware or bitstreams are executed; ii) isolated execution, establishing a protected environment that is inaccessible to untrusted software or hardware; and iii) trusted I/O, enabling secure communication with peripherals and other security-sensitive components. Additional primitives include remote attestation, monotonic counters, and sealed storage.

The growing reliance on reconfigurable computing has motivated the adaptation of TEE concepts to FPGAs, where hardware accelerators and dynamically reconfigurable logic may also require isolated execution. Research efforts in this direction can be broadly categorized into four main trends: i) extensions of existing CPU-based TEEs, which expand processor-centric trust models to include the reconfigurable

fabric as part of the trusted domain, as exemplified by HISA [41] (Figure 13a); ii) novel architectures, which propose new hardware isolation mechanisms for reconfigurable trust execution, as in HETEE [28] (Figure 13b); iii) cloud-oriented frameworks, which rely on the FPGA itself to provide trusted execution capabilities for remote or multi-tenant deployments, as in DF-TEE [183] (Figure 13c); and iv) alternative FPGA-based TEE systems, which aim to replace conventional CPU-centric hardware extensions by leveraging the reconfigurable hardware as the root-of-trust for flexible and patchable trust execution, as exemplified by TREE [67] (Figure 13d).

### 5.5.1 Extensions of CPU-based TEEs

**SGX-FPGA.** Xia et al. [180] proposed SGX-FPGA, which extends Intel SGX enclaves to include FPGA accelerators. The system relies on PUF-based key derivation and inline AES-GCM encryption to secure data transfers between the CPU enclave and FPGA logic. This architecture is designed to ensure that sensitive workloads can offload computation to reconfigurable logic without exposing data outside the SGX enclave, thereby bridging CPU-based trust with hardware acceleration.

**HISA.** Ye et al. [41] introduced HISA, a TrustZone-assisted framework that extends hardware isolation to SoC FPGAs. It partitions the system into secure and normal worlds via AXI bus enforcement, isolating FPGA IPs from untrusted software. A secure agent mediates service requests and enforces access-control and output-verification policies to block cross-domain attacks.

**Ambassy.** Hwang et al. [181] presented Embassy, a runtime framework for securely delegating trusted reconfigurable accelerators to the PL. Built on Arm TrustZone, Embassy authenticates reconfigurable functions and manages their deployment dynamically. By combining secure world services with FPGA reconfiguration, Embassy enables trusted hardware functions to be instantiated on demand, while preserving the isolation guarantees of the TEE model.

**TruFPGA.** Giechaskiel et al. [51] proposed TruFPGA, a framework that establishes mutual trust between CPUs and FPGA logic. The system leverages Arm TrustZone to authenticate and authorize reconfiguration operations, ensuring that only verified logic can be loaded into the FPGA. It integrates secure boot, cryptographic measurement of bitstreams, and configuration attestation to maintain the integrity of the PL throughout the deployment process. TruFPGA further supports secure DPR, enabling dynamic updates of reconfigurable regions while preserving system-wide integrity and providing built-in tamper detection.

**SrcTEE.** Ning et al. [182] present SrcTEE, a TrustZone-assisted framework that extends OP-TEE with hardware-aware runtime trusted software to support secure deployment, invocation, and attestation of reconfigurable IP blocks on SoC FPGAs. SrcTEE authenticates and decrypts bitstreams within the secure

world, enforces access control for hardware accelerators, and verifies the integrity of instantiated logic before it becomes operational.

### 5.5.2 Novel architectures

**HETEE.** Zhao et al. [28] proposed HETEE, an architecture that provides trust compartmentalization between CPU and FPGA domains. In HETEE, the FPGA is leveraged to isolate untrusted accelerators, preventing them from directly influencing secure system state. The CPU retains responsibility for attestation and management, delegating isolation TAs to the reconfigurable fabric. This division of labor reduces the software TCB while ensuring that heterogeneous accelerators are confined to well-defined domains.

**CURE.** Datta et al. [187] introduced CURE, a novel TEE architecture designed for heterogeneous systems. CURE introduces new hardware mechanisms that enable fine-grained resource isolation within the SoC, including exclusive assignment of CPU cores, caches, and peripherals to specific enclaves. These hardware extensions enforce multi-domain isolation and mitigate shared-resource side-channels while requiring only minimal modifications to the baseline RISC-V design. By combining flexible reconfiguration with hardware-assisted security primitives, CURE adapts to diverse application scenarios while maintaining verifiable and dynamically adjustable isolation guarantees.

### 5.5.3 Cloud-Oriented FPGA Frameworks

**ShEF.** Zhao et al. [52] proposed ShEF, an FPGA-resident TEE that introduces a reconfigurable “Shield” module integrated into the cloud FPGAs. The Shield enforces application-specific security policies, supports bitstream attestation, and leverages secure boot to ensure that only verified configurations are admitted. By relying only on the FPGA to trust enforcement, ShEF reduces reliance on the host processor and provides tenants with stronger assurance over deployed remote accelerators.

**DF-TEE.** Xia and Wei [183] introduced DF-TEE, a framework for disaggregated multi-FPGA deployments in cloud environments. DF-TEE chains attestation across heterogeneous components, including CPUs and FPGAs, thereby extending the trust boundary beyond a single device. The architecture aims to enable tenants to verify the integrity of all interconnected accelerators, supporting a unified trust domain in scenarios where resources are pooled across multiple reconfigurable boards.

**MeetGo.** Zhao et al. [184] proposed MeetGo, an FPGA-based TEE that synthesizes enclave-style environments entirely within the reconfigurable fabric. The architecture embeds a dedicated hardware security agent responsible for authenticating deployed modules, enforcing isolation policies, and performing runtime attestation. Remote tenants can independently verify the FPGA configuration without relying on the host system, thereby establishing stronger trust in remote FPGA workloads where the cloud provider cannot be fully assumed trusted.

**AccGuard.** Ren et al. [185] proposed AccGuard, a secure computation framework that extends trusted execution to remote FPGA accelerators in cloud environments. The architecture introduces a security manager within the FPGA fabric. The SM expects that the host platform already provides a TEE, such as Intel SGX or Arm TrustZone, to establish a secure channel with the FPGA and coordinate authenticated deployment and attestation of accelerators. A trusted authority audits and certifies the FPGA shell, provisions device-specific root keys, and validates attestation reports, enabling verifiable accelerator deployment without requiring pre-encrypted bitstreams.

#### 5.5.4 FPGA-based and reconfigurable TEEs

Enabling software TAs to execute directly within the reconfigurable fabric offers both security and practical advantages over conventional CPU-centric TEEs. From a security standpoint, this approach relocates the TCB entirely, or in large part, into the PL, thereby reducing, or in some cases eliminating, the reliance on fixed hardware TEE extensions such as those of TrustZone or SGX, and avoiding the intrinsic vulnerabilities and software-layer attack surfaces pointed out by Cerdeira et al. [30].

**BYOTee.** Armanuzzaman et al. [186] introduced BYOTee, a co-design framework that allows developers to construct application-specific enclaves on the FPGA. Rather than relying on a fixed hardware TCB, BYOTee let system designers define the scope of the TCB by selecting which modules are included and attested. Both the hardware configuration and the accompanying software stack are measured and verified, offering a customizable enclave model that adapts to diverse application requirements.

**TEEOD.** Pereira et al. [208] proposed TEEOD, a framework that redefines the trust boundary by treating the FPGA fabric itself as the primary trusted domain. In contrast to hybrid or CPU-dependent approaches, TEEOD migrates all TEE operations, including TA management, authentication, secure storage, and cryptographic services into the PL. This design offers an alternative to conventional TEEs that rely on fixed processor extensions such as TrustZone or SGX, while maintaining compatibility with the GlobalPlatform specifications that define the functional interfaces and life-cycle management of TAs. Therefore, TEEOD enables seamless portability of TAs to the FPGA domain without sacrificing interoperability or security semantics. Unlike BYOTee, which still depends on a firmware layer to authenticate and deploy TAs, TEEOD relies entirely on independent reconfigurable logic blocks for TA provisioning and control.

**TREE.** More recently, Pereira et al. [67] introduced TREE, a unified FPGA-based TEE framework that enables enclaves to be instantiated, patched, or revoked entirely within the FPGA through dynamic partial reconfiguration. Similar to other FPGA-based TEEs, TREE treats the FPGA fabric as the primary trust domain, solely rely on independent reconfigurable logic for enclave provisioning, attestation, and control. A distinctive feature of TREE is the integration of DPR as a TEE service, enabling TAs to be dynamically instantiated, updated, or revoked during operation while maintaining continuous isolation guarantees. In

doing so, TREE supports not only trusted accelerators within the reconfigurable trust domain but also general-purpose and hybrid TAs that seamlessly combine hardware and software execution.

## 5.6 Future Research Directions

As FPGAs continue to proliferate across datacenters, AI accelerators, and IoT devices, the security implications of the expanding attack surface introduced by reconfigurable hardware demand thorough investigation. While substantial progress has been achieved in identifying and mitigating FPGA-specific threats, a number of fundamental research challenges remain unresolved. Addressing these challenges is crucial to ensure that the growing integration of reconfigurable components does not compromise system trustworthiness. This section outlines emerging directions that can guide future research toward the design of reconfigurable security systems.

### 5.6.1 Threat models that reflect FPGA-specific vulnerabilities

Existing security models for heterogeneous computing systems often extend assumptions inherited from conventional CPU-centric architectures, where execution semantics are well-defined and isolation boundaries are enforced by fixed hardware privilege levels. However, FPGA-based systems introduce spatially and temporally dynamic hardware configurations, whose logic and interconnects can be reconfigured during runtime. This capability challenges traditional threat models, since system behavior and trust boundaries are no longer static but may change in response to reconfiguration events. In such environments, the attack surface is not fixed at boot time but expands and contracts dynamically as new modules are instantiated, relocated, or revoked within the FPGA fabric.

Consequently, the security guarantees of the system depend not only on the correctness of the initial configuration but also on the continuous integrity and isolation of reconfigurable regions during runtime. Adversaries have exploited this fluidity to introduce malicious logic, manipulate configuration flows, and establish fault injections and side-channels attacks between logic that were assumed isolated, as we discuss in section 5.3.

Future research should therefore aim to establish FPGA-aware threat models that accurately reflect these runtime dynamics. Such models must describe the temporal and spatial change of trust, including how the system transitions between secure and insecure states under partial reconfiguration. They should also consider the coexistence of multiple protection domains within the same fabric and the possibility that adversarial logic operates within cryptographically authenticated bitstreams. By capturing these aspects, formal FPGA-aware threat models would enable rigorous reasoning about containment, privilege separation, and fault propagation in reconfigurable systems.

### 5.6.2 Beyond fixed hardware security extensions

Hardware-based security primitives, such as Arm TrustZone or Intel SGX, have long been regarded as the fundamental root-of-trust for TEEs, built on the premise that isolation and integrity guarantees are most effectively maintained when enforced by immutable hardware rather than by software mechanisms. These mechanisms provide strong isolation guarantees, but their security logic, privilege hierarchies, and trust boundaries are fixed, leaving limited flexibility to accommodate evolving threats or reconfigurable system architectures. As highlighted by Cerdeira et al. [30], even mature TrustZone-assisted TEE systems exhibit a persistent class of vulnerabilities, many of which stem from the enlarged software TCB required to manage secure-world services, device drivers, and trusted applications. This expansion increases the attack surface and exposes critical components, such as the trusted OS and inter-world communication interfaces, to software-level exploits that can ultimately compromise hardware-enforced isolation.

With the paradigm shift toward increasingly heterogeneous and reconfigurable computing systems, future research must reconsider the role of the PL as a security enabler, rather than as a peripheral component under CPU control. The studies reviewed in Section 5.4 demonstrate how techniques such as logic obfuscation, redundancy, and DPR have been leveraged to protect IP, mitigate side-channel and fault attacks, and detect anomalous behavior at runtime. Similarly, the architectures discussed in Section 5.5 illustrate how FPGAs are progressively being integrated into trusted computing frameworks, either as extensions of CPU-centric TEEs or as standalone trusted domains capable of performing attestation, isolation, and secure reconfiguration.

These developments collectively point toward a broader research opportunity: to move beyond fixed hardware extensions and embrace reconfigurable hardware as an integral element of the security architecture. By enabling security mechanisms that can be instantiated, verified, and updated within the FPGA fabric itself, future systems may achieve an unprecedented level of adaptability, where trust boundaries evolve in tandem with workloads, and the security substrate itself becomes patchable, scalable, and context-aware.

### 5.6.3 Reconfiguration as an enabler of dynamic countermeasures

Reconfiguration has traditionally been perceived as a potential vulnerability, enabling adversaries to modify or inject malicious logic into the programmable fabric at runtime. However, the same mechanism can be repurposed to provide adaptive and fine-grained defensive capabilities. As pointed out by Sasdrich et al. [126], DPR can be exploited to achieve temporal diversity by periodically relocating or reshaping sensitive circuit regions within the fabric. This concept demonstrates how reconfigurability, when securely managed, can transform a traditional attack vector into a moving-target defense mechanism.

More recent work, such as TREE [67], extends this principle toward system-level trust management. TREE integrates DPR as a TEE service, allowing enclaves to be dynamically instantiated, updated, or revoked during operation without interrupting system execution. By providing reconfiguration control within the trusted domain, TREE enables hardware-level patching, secure updates, and context-aware reallocation

of trusted resources in response to changing operational or threat conditions. This dynamic adaptability illustrates a key opportunity for future research: to develop reconfiguration-aware security architectures that treat adaptability not as a side effect of FPGA design, but as an intentional and verifiable component of system resilience.

## **5.7 Conclusion**

Modern heterogeneous systems increasingly rely on FPGAs as reconfigurable accelerators that complement general-purpose processors, enabling high-performance and adaptable computing across domains from cloud to edge. This survey has shown, however, that such adaptability fundamentally reshapes the security landscape. The programmability that empowers FPGAs to serve diverse workloads also introduces a spectrum of vulnerabilities spanning design-time toolchains, runtime reconfiguration flows, and cross-domain communication channels. By systematizing these threats and consolidating existing countermeasures, this work highlights the dual role of reconfigurability as both a challenge and a security opportunity in modern heterogeneous platforms.

Ultimately, the convergence of reconfigurable logic and hardware-assisted TEEs underscores the need for a paradigm shift in secure system design. Future research should move beyond static, CPU-centric models and embrace FPGA-aware security architectures that integrate reconfiguration as an intrinsic defensive capability. Such architectures must formalize FPGA-specific threat models, enable dynamic and verifiable trust boundaries, and exploit runtime adaptability for proactive defense and secure patching. Achieving this vision would transform reconfigurable hardware from a perceived liability into a foundation for scalable, resilient, and context-aware trusted computing in next-generation heterogeneous systems.

# Rethinking Trusted Execution Environments in the Age of Reconfigurable Computing

***This chapter was submitted as:***

Sérgio Pereira, David Cerdeira, Cristiano Rodrigues, and Sandro Pinto. "Rethinking Trusted Execution Environments in the Age of Reconfigurable Computing"[Submitted to IEEE Access]

## Abstract

As systems become more interconnected and complex, relying solely on general-purpose operating systems (OS) to provide the necessary foundation for security-critical applications becomes challenging. Trusted execution environments (TEEs), i.e., Intel SGX and Arm TrustZone, have been widely adopted to minimize security vulnerabilities by reducing the attack surface. However, recent reported attacks on these well-known TEEs raise concerns about their fundamental security assurances. This work introduces the concept of Trusted Execution Environments On-Demand (TEEOD). This novel solution leverages Field-programmable gate array (FPGA) technology to provide customized and dedicated execution environments (enclaves) per application on a per-need basis. Each enclave includes only the necessary hardware blocks to ensure independent execution, i.e., a soft-core microcontroller unit (MCU), private memories, and access to TEE services and access to TEE services in compliance with the GlobalPlatform (GP) TEE specification. This industry-standard specification defines the architecture and requirements for TEEs, ensuring interoperability and security in trusted applications (TAs). Evaluation on an AMD Zynq UltraScale+ MPSoC (Ultra96-V2) with an entry-level FPGA demonstrates TEEOD's ability to synthesize up to four enclaves, surpassing the GlobalPlatform TEE specification's minimum requirement of two TAs. We also demonstrated TEEOD's practicality in real-world scenarios by successfully running a legacy open-source Bitcoin wallet. TEEOD achieves a 39.2x speedup over vanilla OP-TEE for straightforward operations. When dealing with more computationally intensive operations, a TEEOD configuration that leverages hardware-accelerated cryptographic services resulted in a 1.1x speedup compared to vanilla OP-TEE. In contrast, its software-based version showed a slowdown of 1.5x.

## 6.1 Introduction

Trusted execution environments (TEEs) have become widely adopted in academia and industry due to their ability to ensure confidential and anti-tamper execution of secure-sensitive processes and protect critical data within a broader system [217]. This security foundation differs from relying solely on the security of a full-blown operating system (OS), which increasingly has a large code base and extensive computing capabilities to manage the complex, open, and interconnected nature of modern systems [218–220].

The most common TEE design approach is to create a virtual secure processor in the main processor by leveraging specific security-oriented hardware extensions. Prominent examples of such technologies include Intel's Software Guard Extensions (SGX) [221] and Arm TrustZone [56], which are used in cloud and mobile applications. Despite their widespread adoption, both have been repeatedly compromised [30, 61, 119, 222–228]. Even with mitigation efforts, these TEEs remain vulnerable to attacks, e.g., Spectre and Meltdown, as these issues stem from fundamental flaws in the underlying hardware [20, 229].

Another TEE design approach relies on dedicated security elements, e.g., the Google Titan [163] and Qualcomm Secure Processing Unit (SPU) [175]. While offering strong isolation, this method comes with inherent drawbacks: i) upgrade and maintenance challenges hinder adaptation to evolving threats; ii)

compatibility issues make integration with existing systems difficult; and iii) limited customization reduces applicability, compromising effectiveness and usability [230]. Despite their differences, both TEE design approaches share common limitations, including a fixed hardware trusted computing base (TCB) and the potentially risky (and unnecessary) allocation of excessive privileges to specific protection modes. As a result, concerns arise regarding the practical effectiveness of security guarantees provided by current TEEs [30]. Addressing these shortcomings requires alternative TEE approaches that offer greater security and flexibility [49, 231].

Field-programmable gate array (FPGA) technology shows promise in overcoming the limitations of conventional TEE approaches. Due to their well-known advantages, such as reconfigurability and energy efficiency, FPGAs have already become a key technology for a wide range of applications, including security-sensitive ones [68–70]. Additionally, FPGA system-on-chips (SoCs), which combine the reconfigurable hardware of FPGAs with the software adaptability of microprocessors on the same die, have gained attention for their potential in secure applications. Recent studies have demonstrated the effective isolation of the FPGA fabric within these SoCs for secure storage applications [232].

Building on these observations, we introduce the concept of Trusted execution environments on-demand (TEEOD). This new TEE approach leverages the reconfigurable logic (a.k.a. FPGA fabric) of SoC FPGAs to provide additional security guarantees for security-critical applications. With proper control access mechanisms and security measures [52, 184, 212], FPGA fabric can act as an isolated and independent environment while the main processor executes the rich execution environment (REE). TEEOD instantiates customized and dedicated execution environments per trusted application (TA) on a per-need basis. These environments, named enclaves, encompass the hardware and software elements necessary to ensure the independent and isolated execution of TAs. Each enclave includes a dedicated soft-core microcontroller unit (MCU), private memory, security services, and a communication channel with the REE. TEEOD is compliant with the GlobalPlatform (GP) TEE specifications [233] to ensure interoperability with legacy TAs.

We extensively evaluated TEEOD using microbenchmarks and real-world applications, focusing on hardware costs and performance speedups compared to OP-TEE. We conducted these evaluations on an AMD Zynq UltraScale+ MPSoC development board (Ultra96-V2) with an entry-level FPGA. Regarding hardware costs, our evaluation showed TEEOD can support up to four enclaves running simultaneously, exceeding GP’s minimum requirement of two TAs<sup>1</sup>. TEEOD was evaluated under two configurations: software-based cryptographic services (TEEOD-sw) and hardware-accelerated cryptographic services (TEEOD-hw). TEEOD-sw exhibits normalized performance slowdowns (ranging from 1.1x to 2.3x) across all benchmarks. In contrast, TEEOD-hw outperforms vanilla OP-TEE, delivering speedups of 1.8x for advanced encryption standard in electronic codebook mode (AES-ECB), and 1.3x for SHA-1, highlighting the benefits of offloading some TEE services to FPGA fabric.

---

<sup>1</sup>According to the GP specification, a compliant TEE shall be able to host two TAs simultaneously.

We have deployed an open-source hardware Bitcoin wallet as our proof-of-concept in real-world scenarios. Our findings indicate that TEEOD outperforms OP-TEE by up to 39.2x when handling functions with simple operations, regardless of the TEEOD configuration. However, when it comes to computationally intensive operations that require cryptographic services, TEEOD-hw offers a speed-up improvement of 1.1x, whereas TEEOD-sw shows slowdowns of 1.5x. Additionally, TEEOD has proven to be a promising solution for security-critical applications that perform simple commands with frequent TA calls, while simultaneously freeing up the main processor for other REE processes.

In summary, this work offers the following **contributions**:

- We overview current TEE approaches, highlight their limitations, and emphasize the need for more flexible, customized security solutions (Section 6.2).
- We introduce TEEOD, a novel TEE approach that leverages reconfigurable logic within SoC FPGA to provide customized and dedicated execution environments per application on a per-need basis (Section 6.4).
- We discuss the implementation challenges of deploying TEEOD on an SoC FPGA while adhering to the GP TEE specifications (Section 6.5).
- We conducted a comprehensive evaluation of TEEOD, comparing hardware costs, microbenchmarks, and real-world scenarios with OP-TEE (Section 6.6).
- We perform a security analysis of TEEOD to assess its robustness against the defined threat model (section 6.7).

## 6.2 Background and Motivation

### 6.2.1 Trusted Execution Environments

TEEs provide a secure and isolated environment for processing and storing critical data within a broader system, offering a viable approach to minimizing the attack surface and reducing potential vulnerabilities [234]. The GlobalPlatform TEE Specification [233], widely adopted by the academia and industry and supported by numerous manufacturers and software developers, outlines the key components of a generic GP-compliant TEE (as shown in Figure 14). Key components outlined in the GlobalPlatform TEE Specification include a secure kernel, a trusted user interface, and secure storage, collectively represented as TEE OS. TAs are applications that execute secure-sensitive operations on top of the TEE OS and take leverage of the secure services provided by the GP TEE internal core API (TEE IC API). GP-compliant client applications (CAs) run outside the TEE and use the GP TEE Client API to communicate with TAs. Two primary types of TEE approaches are virtual secure processors and dedicated security elements, which have been implemented in both industry and academia over the years.

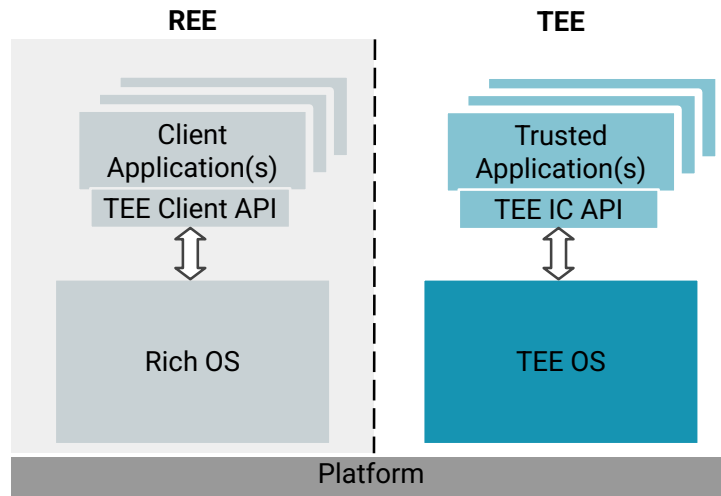


Figure 14: Generic GP-compliant TEE and its major components. The trusted elements are highlighted in blue, whereas the non-trusted components are shown in gray.

**Virtual Secure Processors.** This TEE approach integrates security-oriented hardware extensions, such as special instructions and access-control checks, directly into the main processor, offering a more cost-effective alternative to dedicated security elements. Prime examples include Intel’s SGX [54] and Arm’s TrustZone [56]. Arm TrustZone has become the de-facto hardware technology for implementing TEEs in mobile environments and is also used in industrial control systems [44], servers [235], and low-end devices [236]. The TrustZone in Armv7-A/Armv8-A provides two environments, the normal world and the secure world [56].

According to Cerdeira et al. [30], despite high deployment, TrustZone-assisted TEEs have several sources of vulnerabilities. For instance, the vulnerabilities demonstrated by Gal Beniamini [58–60], showed how an unprivileged user-level app can exploit excess privileges of the Qualcomm TEE’s core elements to hijack the TEE kernel. Additionally, cache-based attacks have successfully breached TrustZone-enabled TEEs by monitoring caches from the normal world [119].

**Dedicated security elements.** The industry has also adopted a different TEE approach by employing external coprocessors or integrating custom-designed on-SoC security chips, e.g., Google’s Titan [163] and Qualcomm’s SPU [175]. This approach is often used in cloud computing [230]. While it provides robust isolation between the REE and TEE, it is not entirely immune to attacks. The reported vulnerabilities in the secure element firmware enable attackers to perform a buffer overflow or exploit null-pointer dereferences [237, 238].

### 6.2.2 Field-Programmable Gate Arrays

SRAM-based FPGAs are highly versatile semiconductor devices that feature numerous logic gates, programmable routes, lookup tables (LUTs), input/output blocks, block-on-chip memory (BRAM), configurable

| Soft-core MCU | Architecture | Bit   | Pipeline | Area (LEs) | Open-source | TRM  |
|---------------|--------------|-------|----------|------------|-------------|------|
| Taiga         | RISC-V       | 32/64 | 5        | 5000       | ✓           | [72] |
| LEON3         | SPARC-v8     | 32/64 | 7        | 5000       | ✓           | [74] |
| OpenRISC 1200 | OpenRISC     | 32    | 5        | 6000       | ✓           | [75] |
| MicroBlaze    | MicroBlaze   | 32    | 3/5      | 1324       | ×           | [73] |
| Nios II/f     | Nios II      | 32    | 6        | 1800       | ×           | [76] |
| Cortex-M1     | Armv6        | 32    | 3        | 2600       | ×           | [77] |
| PicoBlaze     | PicoBlaze    | 8     | no       | 192        | ×           | [78] |
| LatticeMico8  | LatticeMico8 | 8     | no       | 200        | ✓           | [79] |

Table 5: Comparison of soft-core MCUs features.

logic blocks (CLB), and floating-point digital signal processing (DSP). These components allow FPGAs to efficiently implement logical, combinational, and/or sequential functions using hardware description languages (HDL), such as Verilog or VHDL [31].

SoC FPGAs, such as Zynq-7000 or Zynq Ultrascale+ families from AMD, integrate a programmable logic fabric (PL) with a processing system (PS) on the same die, combining FPGA reconfigurability with the power of traditional processors. PL and PS communicate through dedicated bus interfaces, i.e., Advanced eXtensible Interface (AXI). In some advanced SoC FPGA architectures, the PL can even undergo partial reconfiguration without requiring a system reset, allowing on-the-fly FPGA fabric changes without compromising data integrity or disrupting ongoing operations [239].

**General-Purpose computing on FPGAs.** FPGAs are not limited to application-specific designs and signal processing; they can also execute general-purpose tasks, i.e., software through soft-core MCUs. Unlike their hard-core counterparts, these computing units are entirely synthesized on the FPGA's fabric and can typically be tailored and optimized to meet specific application needs. Additionally, these soft-cores are often FPGA agnostic, allowing them to be deployed across various platforms. Both commercial vendors and open-source communities provide a wide range of soft-core MCUs, ranging from small to cost-effective options. Table 5 showcases some soft-core MCUs and highlights their essential attributes.

## 6.3 Threat Model and Design Principles

### 6.3.1 Threat Model

Our threat model adheres to the one commonly assumed for TEE architectures, e.g., as the threat model of Cerdeira et al. [30]. We consider a powerful adversary whose primary objective is to extract sensitive information from the TEE (kernel and secure monitor) or a compromised TA. Such an adversary can compromise all REE software components, including the OS, and has access to the platform's system interconnect and interfaces to the TEE.

Despite a powerful adversary potentially attempting to reconfigure the FPGA fabric, we also consider that necessary measures are in place to ensure the authenticity and integrity of the loaded bitstream. For instance, disabling the FPGA configuration ports during the secure boot process [52, 239]. We assume

that the SoC FPGA device and its FPGA fabric are secure and resistant to physical tampering. The FPGA isolation mechanisms are robust and effective in preventing unauthorized access. We also assume that the hardware and tools for programming the SoC FPGA are trustworthy. Despite the use of encryption and authentication measures for TAs, TEEOD does not inherently confer trust in them.

Physical attacks, such as fault injection, side-channel attacks, or connecting malicious peripherals, are not considered. Denial-of-service (DoS) attacks are also out-of-scope, as they typically aim to disrupt TEE availability and do not directly compromise the security of the TA's protected data or code. In summary, we expect a powerful adversary to have full control over the user applications, the OS, or the hypervisor executing on the REE, as well as the ability to develop malicious TAs or exploit faulty ones.

### 6.3.2 Requirements

A TEE must provide essential features encompassing the main building blocks to build secure execution environments for performing critical operations and data processing [234].

**R1: Isolate TEEOD from the REE.** TEEOD must provide isolation from the REE to prevent malicious, non-secure software or hardware components from accessing or tampering with the TEE and the TAs.

**R2: Isolate TAs from each other.** Each TA must have its unique physical address space, computing unit, system configuration registers, and essential peripherals to ensure an isolated and dedicated execution environment.

**R3: Protect sensitive data.** TEEOD must provide secure storage primitives to protect sensitive data from unauthorized access, tampering, or leakage. This includes isolation and access control mechanisms to ensure only authorized TAs can access the stored data.

**R4: Provide cryptographic primitives.** TEEOD must provide reliable cryptographic services, including encryption, decryption, digital signatures, and secure key management. These services should be easily accessible to TAs for efficient data protection and communication, and resilient against known cryptographic attacks.

**R5: GlobalPlatform Compliance.** TEEOD must be compliant with GP's API specifications [233] to provide a reliable framework for deploying and managing TAs on multiple TEEs, ensuring seamless functionality and interoperability with existing systems.

### 6.3.3 Design Principles

To avoid widely-known security pitfalls that have affected TEE systems in the past, i.e., TrustZone-assisted TEEs, we resorted to a set of principles to guide our design.

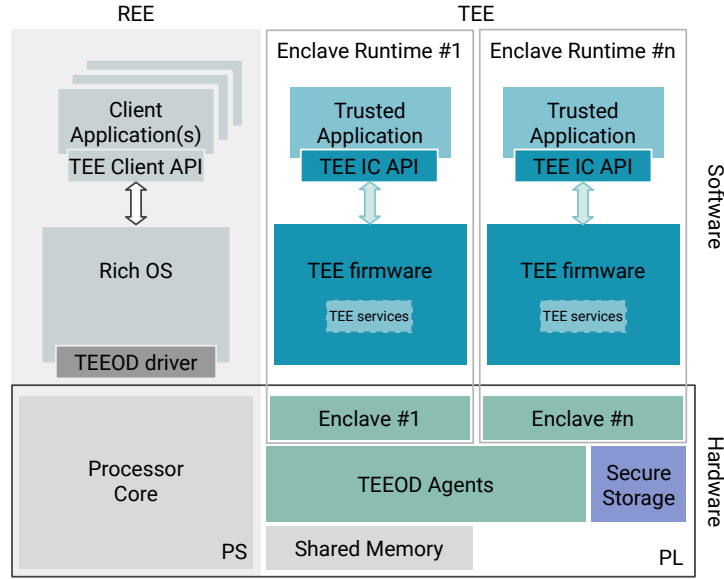


Figure 15: TEEOD core design elements. The TEEOD architecture includes enclaves and TEEOD agents in light green, secure storage in purple, and trusted software components in blue. Non-secure components are represented in grey.

**P1: Resource Isolation.** As found in the literature [117–121, 224, 227, 240, 241], shared resources have repeatedly been leveraged to extract secrets and manipulate core TA functionality. Sharing the software runtime has also compromised critical security functions [60, 242, 243]. In TEEOD, we strive to **avoid sharing hardware and software resources**.

**P2: Least Privilege.** Each TEEOD element should be granted the **minimum level of privilege required** for its functionality to prevent arbitrary code execution and privilege escalation [58–60]. This principle reduces the attack surface and limits the damage a compromised element can cause, e.g., TAs should not access main memory arbitrarily.

**P3: Modular Architecture.** A modular design strives for TEEOD to be composed of **decoupled modules**, each with its own interface and functionality. This principle allows TEEOD to be customized to meet the specific needs of each application and to mitigate potential vulnerabilities of interconnected complexity by selectively integrating only the necessary modules.

## 6.4 TEEOD: Design

Considering the aforementioned design principles, the TEEOD core design elements are shown in Figure 15. TEEOD provides TAs with an execution context, access to security mechanisms, and means to communicate with CAs. We designate the execution context in which TAs are loaded and executed as enclaves. Each enclave acts as a distinct execution context, running only one TA at a time. TAs can be

executed within the FPGA fabric using soft-core MCUs and/or custom accelerators. Our research focuses on the challenges of leveraging soft-core MCUs for TA execution since it simplifies porting legacy GP-compliant TAs to TEEOD. Using custom accelerators entails a new set of challenges that are out-of-scope in this work. We defer this exploration for future work.

### 6.4.1 TEEOD Elements

**Enclaves.** Enclaves ensure that TAs execute independently and are isolated from each other and from the REE. This is achieved through dedicated hardware blocks for each TA's execution. Each enclave provides a soft-core MCU for the TA's execution context and private memory to store code and data. It also provides hardware TEE services if required, a communication channel with REE clients, and interfaces with other TEEOD elements, i.e., shared memory, TEEOD Agents, and secure storage. No enclave element is shared among enclaves or with the REE. The enclave elements must be replicated to allow multiple TAs to run concurrently. This ensures that multiple TAs can run independently and simultaneously while maintaining isolation.

**Enclave Runtime.** In addition to executing a single TA at a time, the enclave's soft-core MCU also runs lightweight TEE firmware. This firmware provides essential kernel functionalities for booting, message parsing, and error handling, and acts as the backend for the TEE IC API, enabling TAs to interact with TEE services and hardware resources. Access to shared TEE secure services, i.e., secure storage, across enclaves is controlled and mediated by TEEOD agents. All TEE services are either connected exclusively to the enclave or mediated by TEEOD agents, meaning the TEE firmware does not require special access rights beyond those needed by the TAs. By eliminating privilege mode implementation, the soft-core and TEE firmware design is simplified. Since the soft-core memory is cleared when a TA finishes and the firmware contains no secrets, it is reloaded alongside the TAs.

**TEEOD Agents.** These dedicated hardware blocks are the backbone for managing enclaves, enabling secure communication, and ensuring the TEE's overall integrity and functionality. TEEOD agents are divided into three modules: i) the *TEE Communication Agent* (TCA), which facilitates communication between REE and TEE, parses received messages, and forwards them to the corresponding enclave, preventing direct access from the REE and enhancing access control; ii) the *TEE Manager Agent* (TMA), which manages enclave resources, tracks their availability, and assigns TAs dynamically, allowing the execution of multiple TAs and supporting scalability; iii) the *TA Loader Agent* (TLA), which loads TA binaries from non-secure memory to the enclave's private memory. Only the TLA can access this region; other agents, enclaves, and non-secure elements do not have access.

**TEE services.** A GP-compliant TEE offers the TAs a set of security-enabling services (i.e., secure storage, cryptographic operations, and time-related services). Most of these services do not require sharing or

interaction with other enclaves. Rather, they must be exclusively available for the TA during its execution. As a result, services such as cryptographic operations and time-related services are accessed within the enclave's boundaries and managed by TEE firmware, whether implemented as software TEE services or hardware blocks.

**Secure Storage.** Depending on the security policies, such as the need to retain a TA's data after its session ends, certain TEE services—like secure storage—must be able to operate independently of the TA's activity status. Since all enclave blocks are wiped out and reset when the TA is inactive, the secure storage is external to the enclaves to ensure the persistent storage of sensitive data. As a result, secure storage is accessible by all enclaves, allowing a TA to restart in a different enclave, and relies on TMA for TA identification.

**TEEOD driver.** The TEEOD driver is a kernel driver in Linux that functions as a bridge between the CAs and the TEE. The TEEOD driver works in conjunction with the TEE Client API, which specifies the communication protocol between CAs in the REE and the TAs in the TEE. The TEEOD driver forwards messages or requests from CAs to the TCA agent configured on the FPGA. This driver is classified as a non-secure element whose sole purpose is to facilitate communication between the REE and TEE. It is not considered trustworthy or secure.

## 6.5 Implementation

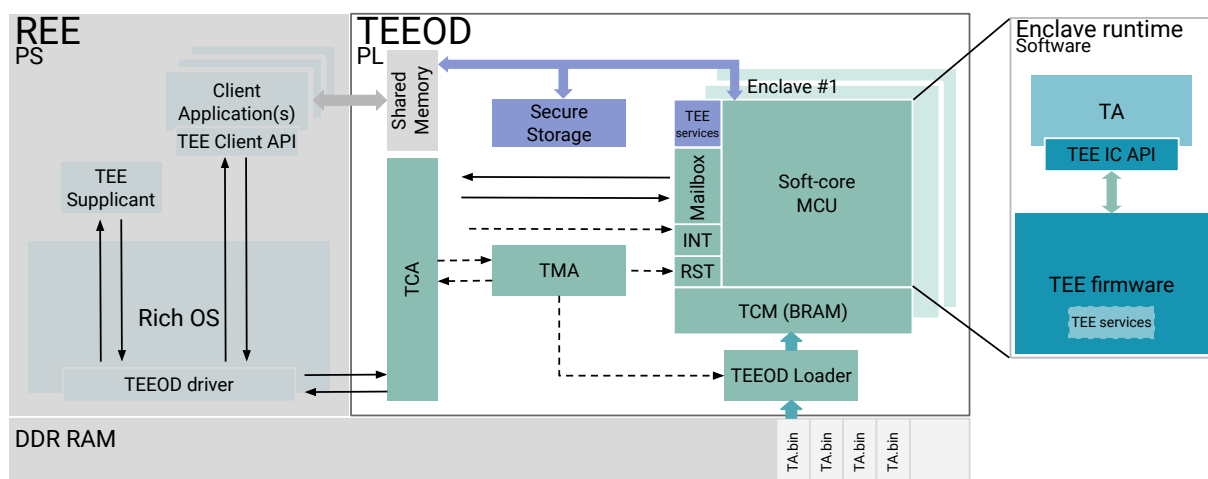


Figure 16: Example of TEEOD implementation in an SoC FPGA. The TEEOD secure elements are instantiated, and the TAs are executed in the PL, while the Rich OS, the CAs, and the TEE supplicant run in the PS.

### 6.5.1 Overview

Figure 16 illustrates an implementation of the TEEOD design in an SoC FPGA, showcasing the key components and their relationships. All secure TEEOD elements are synthesized as hardware blocks in the FPGA fabric, while the REE and the TEEOD driver run on the PS. DRAM and external memories are considered non-secure memories. REE can only access TEEOD through the TEEOD driver, which allows access to TCA and shared memory but no other TEEOD elements. To facilitate remote services required by the TEEOD, a TEE supplicant daemon operates in the REE as a Linux user space.

In the PL, each enclave is equipped with a single lightweight soft-core MCU (such as Arm Cortex-M1), private BRAM, a mailbox for transmitting information with REE, hardware TEE services, interfaces to secure storage and shared memory, as well as two interrupts: reset (RST signal) and TEE communication interrupt (INT signal). Each enclave's mailbox and INT signal are linked to the TCA, while the RST signal is connected to the TMA for control over the execution context of the enclaves. TLA is the only agent with access to the non-secure memory DRAM. It is also the only TEEOD element, besides the enclave itself, that has access to the enclave's private memory.

The interactions between the REE and TEEOD are similar to those found in other GP-compliant TEEs. However, TEEOD implementation faced unique challenges in meeting the requirements and adhering to the design guidelines described in section 6.3.1 and 6.3.3, respectively. These challenges encompass communication between CAs and TAs, managing TAs and sessions, cleaning up processes, secure storage, and TA access to non-secure memory. In the upcoming subsections, we examine these challenges in further detail.

### 6.5.2 CA-TA Communication

The communication between a CA and a TA follows a default RPC-model, where the CA sends a request and waits for a reply from the TA. TEEOD employs a shared mailbox mechanism to facilitate communication between CAs and TAs, with the TCA acting as a mediator. Enclaves have their own dedicated mailboxes that are also shared with the TCA, preventing unauthorized access between enclaves and upholding their isolation. The mailbox consists of several fields, including the intended TEE operation ID, the universally unique identifier (UUID) of the desired TA, parameters specific to the intended operation, and the result for the return value.

The TEEOD synchronization mechanism serves one request at a time, guaranteeing that each request is processed before the next one is taken up. Once the request is completed, TEEOD prepares to manage the next one. This is not a limitation of our approach, and in the future, we plan to add support for multiple concurrent requests. This is required because, unlike TrustZone, all requests are serviced asynchronously. In addition, we could also benefit from multiple channels between PS-PL so that the request can be processed in parallel.

### 6.5.3 TA Management

When a CA attempts to open a TA session, the TEEOD needs to create a new execution context for the TA (in case it did not exist before). This involves assigning an available enclave to the TA and loading the TA into the private memory of the target enclave. The number of available enclaves is defined at design time, while the number of TAs can change at runtime. Furthermore, each active TA context can have multiple sessions, which poses a management challenge for the hardware due to the variable number of TAs and sessions.

The solution was to delegate session management to the TEE firmware and TA context to the TEEOD agents. Since each enclave runs only one TA at a time and the total number of enclaves is known at design time, TA context management is trivial to implement in hardware. The TMA maintains a list of all enclaves available and, for each entry, includes a bit to indicate whether the enclave is running a TA and, if yes, also includes the TA's UUID. Regarding TA session management, even if the CA provides an incorrect session ID or enclave ID, the system's integrity and other TAs remain uncompromised.

When a CA requests an open session, the TMA checks all the enclaves with active bits to see if the received TA UUID matches the one in the list. If there is a match, it returns the enclave that is currently running the target TA. If there is no match, it signals the start of the TA loading process on a free enclave. Once the session is opened, the CA uses the enclave ID bound with the session ID created by the enclave to identify the desired TA and session each time.

### 6.5.4 Enclave Cleanup Process

TEEOD allocates clean-slate enclaves to each TA whenever possible. However, robust enclave cleanup and sanitization processes are required to prevent potential leakage or tampering between TAs if an enclave is reused.

The cleanup process implemented in TEEOD begins only after the last active session has closed and the return result has been sent to the REE. Once the former TA context is fully concluded, the TEEOD Agents start to overwrite the enclave's private memory with predefined patterns or zeroes, cleaning up the microarchitectural states of the soft-core MCU and resetting all the enclave's hardware. Once the cleanup process is complete, the TMA erases the TA context entry of the subject enclave, making it available. Only then does the TEEOD attribute the enclave to another TA, maintaining the isolation requirement throughout the transition.

### 6.5.5 Secure Storage

TEEOD provides a secure storage manager outside the enclave boundaries to ensure persistent storage even after the TA session ends and the enclave is reset. This additional hardware interfaces with all enclaves and not just a single enclave, thus guaranteeing that the sensitive data is accessible regardless of the enclave in which the TA is currently executing. Access to the secure memory is prevented as there is

no interface available for REE. However, ensuring no interface alone is not enough to guarantee persistent, secure storage.

The threat model, outlined in section 6.3.1, accounts for potential attacks by malicious TAs on other TAs. Therefore, the secure storage manager cannot rely on TAs to self-identify. TEEOD addresses this security challenge by relying on TMA to provide the UUID of the currently executing TA. This approach ensures that even if a malicious TA attempts to impersonate another TA, the secure storage manager can detect and prevent such unauthorized access.

### 6.5.6 Enclave's Memory Access

According to GP specifications, TEEOD requires access to REE's root file system to retrieve TA binaries and load them into the TEE. However, to provide such access to non-secure and secure memory without compromising the principle of least privilege (outlined in section 6.3.3) is crucial.

TEEOD uses non-secure and secure elements to address this challenge. Upon request from the TMA, the TEE Supplciant retrieves the required TA binary from the root file system and stores it temporarily in DRAM. The TLA then copies it from DRAM to the enclave's private memory. TLA has read-only access to DRAM and exclusive access to the enclave's memory, preventing overprivileged access. This simplifies TLA's tasks and minimizes data handling vulnerabilities.

Moreover, TAs require access to non-secure memory for data sharing with the REE. TEEOD opts to use the FPGA's internal memory, BRAMs, instead of granting TAs broader access to non-secure memory, i.e., DRAM. This ensures that shared memory requirements are met while preventing TAs from accessing non-secure resources excessively.

## 6.6 Evaluation

We evaluated TEEOD using the Ultra96-V2<sup>2</sup>, a development board based on Linaro 96Boards. For this evaluation, we use an Arm Cortex-M1 as the soft-core MCU used in each enclave. This 32-bit MCU is based on the Arm architecture v6-M, similar to the Cortex-M0, but highly optimized for FPGA implementation. Each Cortex-M1 IP was configured with the OS extensions option enabled to activate the SysTick timer while the debug extension option remained disabled. According to the GP specification, a TEE shall be able to execute TAs with a minimum binary TA code of 64 kiB [233]. To comply with this requirement, each enclave has 64 kiB of BRAM for code memory, 32 kiB of BRAM for data memory, and 8 kiB of BRAM for shared memory. For hardware TEE secure services, we synthesize hardware-based SHA-1 and SHA-256 hash functions and AES with support for both CBC and ECB modes. All FPGA modules were implemented using Verilog-HDL and configured to operate at 100 MHz using the AMD Vivado tool. The evaluation focused on:

---

<sup>2</sup>AMD Zynq UltraScale+ MPSoC ZU3EG, featuring an entry-level FPGA, a quad-core Arm Cortex-A53, and 2 GB of DDR4 SDRAM.

**FPGA Hardware Costs.** We synthesized TEEOD with a single enclave and the hardware TEE services and collected the required FPGA logic elements. To evaluate scalability, we expanded our synthesis to add multiple enclaves and collect the required number of FPGA logic elements for each corresponding TEEOD configuration. In addition, we evaluated power consumption under controlled conditions, capturing the incremental overhead introduced by scaling the number of enclaves.

**Benchmarks.** We compared TEEOD and vanilla OP-TEE performance speedup by measuring the execution time of GP API operations and calculating the speedup. We start by measuring the communication latency of open session, closed session, and invoke command operations. Then, we measured the execution time of the cryptographic services and compared the hardware and software implementations.

**Real-world applications.** We deployed an open-source Bitcoin wallet based on the Trezor wallet API [244] to demonstrate the practicality of TEEOD in real-world applications. Originally developed for OP-TEE, the wallet<sup>3</sup> offers several services, including: i) creating and deleting a master key, ii) accessing the mnemonic for a master key, iii) signing transactions, and iv) retrieving the wallet address. The TA leverages a subset of the TEE IC API, including random number generation for seed phrase creation, secure storage for protecting the master key, and cryptographic primitives for signing transactions and other cryptographic tasks.

| <b>Hardware Elements</b> |               | <b>LUT</b><br>(70560) | <b>LUTRAM</b><br>(28800) | <b>FF</b><br>(141120) | <b>BRAM</b><br>(216) | <b>DSP</b><br>(360) |
|--------------------------|---------------|-----------------------|--------------------------|-----------------------|----------------------|---------------------|
| TEEOD Agents             |               | 6919<br>(9.8%)        | 1055<br>(3.7%)           | 12821<br>(9.1%)       | 0                    | 0                   |
| Enclave                  | Soft-core MCU | 5382<br>(7.6%)        | 647<br>(2.2%)            | 5808<br>(4.1%)        | 9<br>(4.2%)          | 3<br>(0.8%)         |
|                          | Memory        | 318<br>(0.5%)         | 0                        | 333<br>(0.2%)         | 15<br>(6.9%)         | 0                   |
|                          | Shared Memory | 398<br>(0.6%)         | 0                        | 371<br>(0.3%)         | 2<br>(0.9%)          | 0                   |
| Secure Storage           |               | 406<br>(0.6%)         | 0                        | 383<br>(0.3%)         | 2<br>(0.9%)          | 0                   |
| <b>Total</b>             |               | 13423<br>(19.0%)      | 1702<br>(5.9%)           | 19716<br>(14.0%)      | 28<br>(12.9%)        | 3<br>(0.8%)         |

Table 6: Synthesized results of each hardware element on the Ultra96-V2.

### 6.6.1 FPGA Hardware Costs

At a minimum, the TEEOD hardware consists of the following elements: i) TEEOD agents, ii) one enclave with a soft-core MCU and private memory, iii) shared memory, and iv) secure storage. It is worth noting that the soft-core MCU can be customized as well as the memory allocations.

<sup>3</sup><https://github.com/Jamie-Cui/bitcoin-wallet>

**Logic Resource Breakdown.** Table 6 outlines the FPGA logic utilization based on the soft-core and memory allocations defined for this evaluation. TEEOD agents use 9.8% LUTs, 3.7% LUTRAM, and 0.9% FF. An enclave uses 8.1% LUT, 2.2% LUTRAM, 0.5% FF, 11.1% BRAM, and 0.8% DSP. Shared memory and secure storage have a small impact on FPGA logic elements, with only 0.6% LUT, 1.0% LUTRAM, and 0.4% FF. TEEOD agents prioritize combinational logic for enclave management and communication, eliminating the need for extensive memory or signal-processing operations, such as BRAM or DSP. In contrast, enclaves, shared memory, and secure storage consume most of the BRAM since they rely heavily on the memory size defined. Overall, the minimum resource utilization is marginal, considering the evaluation was performed on a small entry-level FPGA.

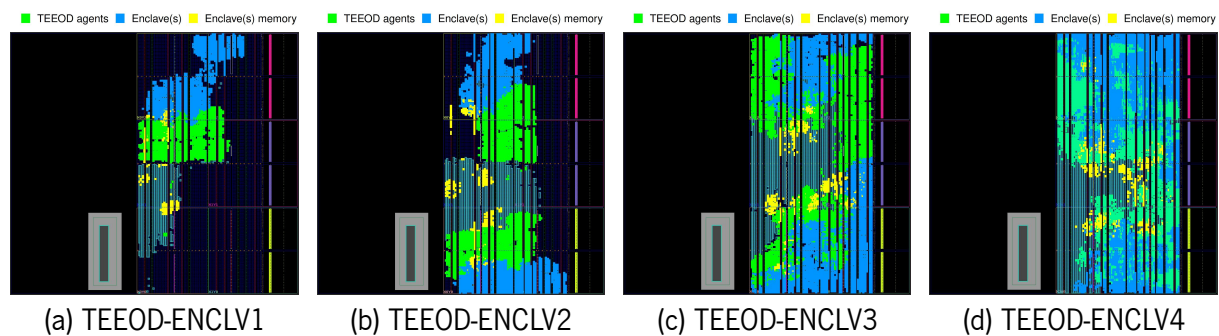


Figure 17: The following are the hardware costs for our synthesized TEEOD design in four configurations: TEEOD-ENCLV1, TEEOD-ENCLV2, TEEOD-ENCLV3, and TEEOD-ENCLV4, with varying numbers of enclaves. Green cells indicate TEEOD agents’ resources, blue cells show enclave resources, and yellow cells denote enclave private memory. All other hardware resources are represented in teal, and the REE runs on the grey-highlighted PS.

**Multiple Enclaves (Scalability).** We evaluated the impact of integrating multiple enclaves across four configurations: one enclave (TEEOD-ENCLV1), two enclaves (TEEOD-ENCLV2), three enclaves (TEEOD-ENCLV3), and four enclaves (TEEOD-ENCLV4). Figure 17 shows the hardware cost map for these configurations. Each map highlights the FPGA logic elements allocated to the TEEOD agents, enclaves, the private memory of each enclave, and all other elements used by peripherals and interconnects.

Table 7 presents the synthesis of total hardware costs for each configuration. On average, LUT usage increases by 10.9%, LUTRAM by 4.8%, FF by 8.1%, BRAM by 11.1%, and DSP by 0.83%. LUT and BRAM are the FPGA logic elements most impacted by the addition of new enclaves. This increase is due to the decision to use BRAM exclusively for memory, rather than relying on off-chip storage or DDRAM. We acknowledge this could become a bottleneck for TEEOD scalability, potentially limiting the number of active enclaves and TAs at any given time. However, we argue that the GP specification requires a compliant TEE to support two TAs simultaneously. In its current form, targeting the Ultra96-V2, TEEOD can host up to four enclaves without exhausting FPGA logic elements (noting that the target SoC’s FPGA is entry-level). This satisfies the GP specification. For future work, we will investigate how to allocate DRAM regions to enclaves securely and reduce BRAM resource consumption to enhance TEEOD scalability.

| <b>Configuration</b> | <b>LUT</b><br>(70560) | <b>LUTRAM</b><br>(28800) | <b>FF</b><br>(141120) | <b>BRAM</b><br>(216) | <b>DSP</b><br>(360) |
|----------------------|-----------------------|--------------------------|-----------------------|----------------------|---------------------|
| TEEOD-ENCLV1         | 13423<br>(19.0%)      | 1702<br>(5.9%)           | 19716<br>(14.0%)      | 28<br>(12.9%)        | 3<br>(0.8%)         |
| TEEOD-ENCLV2         | 21145<br>(30.0%)      | 3090<br>(10.7%)          | 31073<br>(22.0%)      | 52<br>(24.1%)        | 6<br>(1.6%)         |
| TEEOD-ENCLV3         | 29867<br>(42.3%)      | 4199<br>(14.6%)          | 42287<br>(29.9%)      | 76<br>(35.2%)        | 9<br>(2.5%)         |
| TEEOD-ENCLV4         | 36589<br>(51.9%)      | 5587<br>(19.4%)          | 53751<br>(38.1%)      | 100<br>(46.2%)       | 12<br>(3.3%)        |

Table 7: Synthesized results of each TEEOD configuration on the Ultra96-V2.

| <b>Configuration</b> | <b>CLKs</b> | <b>Signals</b> | <b>Logic</b> | <b>BRAM</b> | <b>PS</b> | <b>Total</b>  |
|----------------------|-------------|----------------|--------------|-------------|-----------|---------------|
| OP-TEE               | -           | -              | -            | -           | 1620 mW   | <b>1.62 W</b> |
| TEEOD-ENCLV1         | 45 mW       | 59 mW          | 64 mW        | 35 mW       | 1620 mW   | <b>1.83 W</b> |
| TEEOD-ENCLV2         | 78 mW       | 113 mW         | 121 mW       | 68 mW       | 1620 mW   | <b>2.00 W</b> |
| TEEOD-ENCLV3         | 105 mW      | 160 mW         | 175 mW       | 100 mW      | 1620 mW   | <b>2.18 W</b> |
| TEEOD-ENCLV4         | 120 mW      | 175 mW         | 190 mW       | 110 mW      | 1620 mW   | <b>2.35 W</b> |

Table 8: Power consumption of TEEOD configurations on Ultra96-V2, compared to the OP-TEE baseline. Measurements taken at 25°C, 250 LFM airflow, with Vivado default power optimization.

**Power consumption.** The power consumption evaluation was conducted under controlled conditions, with an ambient temperature of approximately 25°C and an airflow of 250 LFM, using Vivado’s default power optimization settings. The OP-TEE configuration, which runs entirely on the PS with the PL powered down, serves as the baseline, drawing 1.62 W. In contrast, Considering the same TEEOD configurations previously assessed in *Multiple Enclaves (Scalability)*, the total power usage increases with the number of instantiated enclaves: 1.83 W for one enclave (TEEOD-ENCLV1), 2.00 W for two (TEEOD-ENCLV2), 2.18 W for three (TEEOD-ENCLV3), and 2.35 W for four (TEEOD-ENCLV4), as detailed in Table 8. This gradual overhead stems from increased logic switching, signal activity, and BRAM usage in the FPGA fabric. Still, even at its peak, TEEOD’s consumption remains within 0.73 W of the OP-TEE baseline, demonstrating that scalable enclave deployment is achievable on entry-level FPGAs without excessive power demands.

## 6.6.2 Benchmarks

**GP API microbenchmarks.** To assess the communication latency between the CA-TA, we measured the round-trip execution time of four GP API operations: opening a session, opening a session after a TA is already loaded, closing a session, and invoking a command. The round-trip execution time encompasses the time for the CA to communicate with the GP Client API, the TA to acknowledge the request without performing any action, and the CA to receive acknowledgment. Figure 18 illustrates the performance speedup comparison between TEEOD and the vanilla OP-TEE.

TEEOD uses combinational logic to manage TAs, resulting in a significantly faster open session speed of 863.6  $\mu$ s compared to OP-TEE’s 72.1 ms. Once the TA is loaded, any subsequent session openings

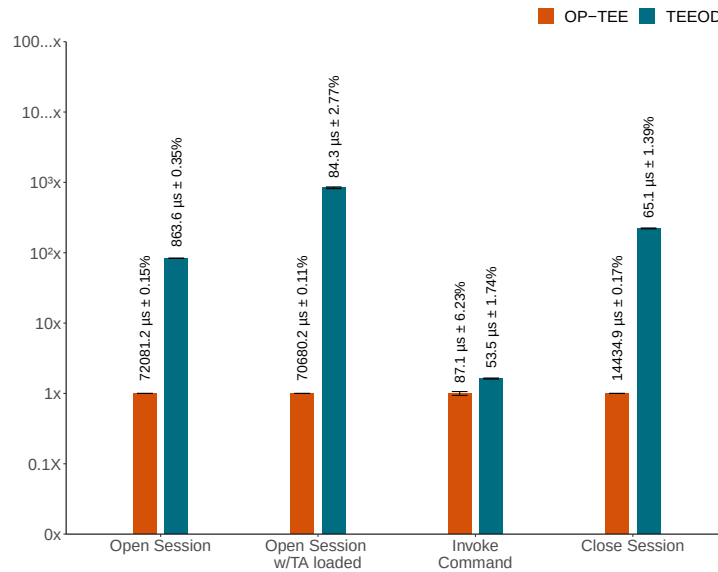


Figure 18: Performance speedup comparison between TEEOD and OP-TEE for each GP Client API operation.

are treated as invoked commands by TEEOD. Therefore, the time to execute the open session after a TA is loaded is similar to that of the invoked command operation, i.e., 84.3  $\mu$ s and 53.5  $\mu$ s, respectively. The only difference is that, in an open session, the TEE firmware needs to manage sessions before proceeding with the operation, while the invoke command does not require this step. Regarding close session, TEEOD has a significant speedup of 65.1  $\mu$ s compared to OP-TEE's 14.4 ms. This improvement is justified by TEEOD's efficient utilization of FPGA parallelism. While the TMA simultaneously performs enclave cleanup, the TCA independently acknowledges to the CA that the close session has been completed. Our evaluation revealed that TEEOD consistently outperformed OP-TEE in opening a session, opening a session after a TA is already loaded, closing a session, and invoking a command. Regarding the coefficient of variation, TEEOD shows reduced variances in all operations, showcasing its determinism.

**TEE secure services benchmarks.** OP-TEE provides a suite of evaluation tests, called *xtests*, which perform TEE sanity checks to assess integrity and functionality. These checks benchmark cryptographic algorithms, including AES in ECB mode, AES in CBC mode, SHA-1, and SHA-256. The benchmarks were conducted under two distinct TEEOD configurations: one with cryptographic services implemented within the TEE firmware (TEEOD-sw) and another with cryptographic services accelerated by hardware (TEEOD-hw). To compare performance between TEEOD's TAs, running on a soft-core Cortex-M1 at 100MHz, and OP-TEE's TAs, running on a 64-bit Cortex-A53 at 1 GHz, clock frequency normalization was applied. The results of vanilla OP-TEE were multiplied by the ratio of its frequency (1 GHz) to the reference frequency (100 MHz) set to match TEEOD's FPGA elements.

Figure 19 illustrates the normalized performance speedups of the two TEEOD configurations compared to vanilla OP-TEE, with clock frequency normalized. TEEOD-sw shows slowdowns ranging from 1.1x to 2.3x

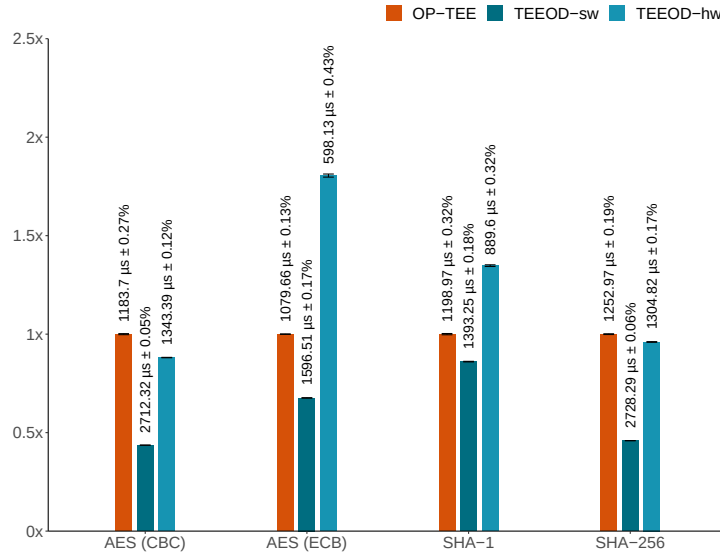


Figure 19: Normalized performance speedup of TEEOD and OP-TEE for AES and SHA xtests.

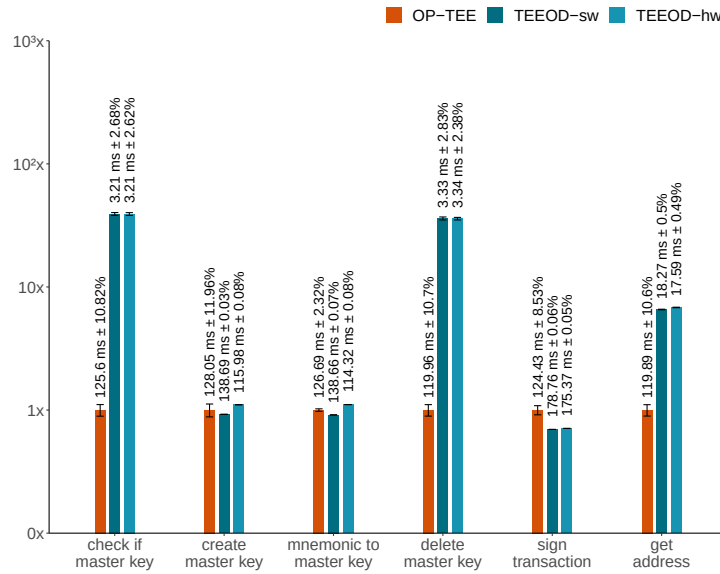


Figure 20: Normalized performance speedup of TEEOD and OP-TEE of each Bitcoin wallet operation.

across all benchmarks. In contrast, TEEOD-hw shows a speedup of 1.8x for AES in ECB mode and 1.3x for SHA-1. For AES in CBC mode and SHA-256, TEEOD-hw experiences a modest 1.1x slowdown.

### 6.6.3 Real-world applications: Bitcoin Wallet

The Bitcoin wallet is an ideal example for assessing TEEOD, as it involves security-sensitive operations such as managing keys and signing transactions that demand confidentiality and integrity. Since the Bitcoin wallet was originally developed for OP-TEE and follows the GP specification, porting to TEEOD was trivial. Both the CA and the TA were able to execute on TEEOD without any modifications.

Users engage with the CA by providing a command ID, access pin, and optional arguments for specific operations. Then, CA uses the TEE Client API to perform security-sensitive operations via the TA. Regardless of the user command, CA communicates with the TA by opening a session, invoking a command, and closing the session upon completion.

Similar to the TEE secure services benchmarks, the results presented in Figure 20 show the normalized performance speedups of TEEOD under two configurations (TEEOD-sw and TEEOD-hw) compared to vanilla OP-TEE. Regarding operations that do not require extensive mathematical computation, such as checking for an existing master key, erasing a master key, and obtaining the Bitcoin public address, both TEEOD configurations outperform OP-TEE. Since these operations only involve simple secure storage verification, clearing, and address retrieval operations and do not benefit from the offload of some cryptographic services to the hardware provided by TEEOD-hw, both configurations have similar speedups compared to OP-TEE, with up to 39.2x speedup.

For operations that demand more cryptographic services, such as creating a master key or converting a mnemonic to a master key, TEEOD-hw demonstrates a 1.1x speedup, while TEEOD-sw shows a 1.5x slowdown. This disparity stems from the hardware-accelerated cryptographic services employed by TEEOD-hw. The sign transaction operation shows a 1.1x slowdown in both configurations due to the fact that this operation requires cryptographic services that are not available as hardware accelerators in TEEOD-hw.

Our findings have shown that TEEOD-hw and TEEOD-sw have similar normalized performance improvements in handling operations related to secure storage and basic functions. However, when it comes to computationally intensive operations that can benefit from cryptographic hardware accelerations, the difference between TEEOD-hw and TEEOD-sw becomes more noticeable. This highlights the advantages of utilizing FPGA fabric for TEE services instead of relying on slower software-based implementations.

## 6.7 Security Analysis

### 6.7.1 Real-World TEE Vulnerabilities

Publicly disclosed Common Vulnerabilities and Exposures (CVEs) provide empirically validated examples of how commercially available TrustZone-assisted TEEs have been compromised. We revisit a selection of relevant CVEs compiled by Cerdeira et al. [30] cross multiple platforms to illustrate recurring attack patterns targeting different layers of the TEE stack, including user-level TAs, the Trusted OS, and cross-TA interactions. In many cases, these vulnerabilities can be chained to escalate privileges progressively, from a malicious REE application, to hijacking a TA, compromising the Trusted OS, and ultimately gaining control of the secure monitor. Such escalation attacks are enabled by shared execution stacks, excessive privilege allocation to trusted components, and weak isolation boundaries within conventional TrustZone-assisted TEEs. Figure 21 provides a high-level overview of this multi-layered attack progression, which is further detailed below. TEEOD breaks this chain at every level by enforcing the security requirements

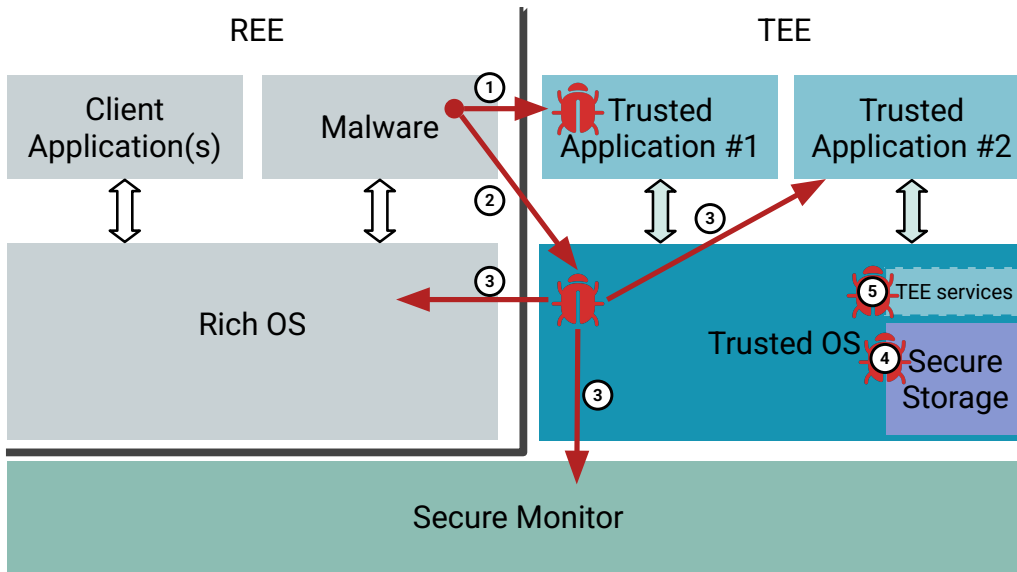


Figure 21: Representative CVE-driven privilege escalation path in conventional TrustZone-assisted TEEs, from REE malware to Trusted Applications and ultimately the Secure Monitor. This multi-layer attack surface, illustrated by CVEs such as CVE-2015-6639 (TA hijack), CVE-2016-2431 (Trusted OS buffer overflow), and CVE-2015-8999 (cross-TA interference), is eliminated in TEEOD by design.

defined in section 6.3.2. For each class of vulnerability, we show how TEEOD mitigates the threat either by eliminating vulnerable interfaces or by enforcing architectural separation that renders the exploit path ineffective.

**User-level TA threats.** Several vulnerabilities have shown that user-level TAs in TrustZone-assisted TEEs can be hijacked directly from the REE due to insufficient input validation and weak isolation. For example, CVE-2015-6639, CVE-2015-9000, CVE-2015-9002, and CVE-2015-6647 exploit memory corruption and unsafe interface handling to redirect TA control flow, while CVE-2015-4422 and CVE-2015-9183 demonstrate how unvalidated parameters in system calls enable arbitrary code execution or unauthorized access to cryptographic functions. These attacks rely on the ability of REE applications to interact directly with TAs via shared memory and standardized interfaces. In contrast, TEEOD enclaves execute each TA on a dedicated soft-core MCU within the PL, with private memory and no shared execution context or memory accessible to the REE. All communication is strictly mediated by the TCA, which parses REE messages and forwards them in a controlled format using an untrusted BRAM channel. The only shared medium is a BRAM region explicitly marked as untrusted and mapped with a known fixed layout for controlled message-passing. Additionally, enclaves have no access to DDR and are fully isolated from the PS address space. This design removes the REE-to-TA attack surface and enforces the *R1: Isolate TEEOD from the REE* security requirement.

**Trusted OS compromise.** Other CVEs have demonstrated flaws in the Trusted OS of TrustZone-assisted TEEs that can be exploited directly from the REE, enabling full compromise of the secure world.

For example, CVE-2016-2431, CVE-2017-6292, and CVE-2018-5870 describe unchecked memory operations and missing bounds checks that may allow unprivileged REE applications to overflow buffers, corrupt kernel objects, or perform out-of-bounds writes within the Trusted OS. Once compromised, attackers may bypass TA isolation, invoke secure services arbitrarily, or tamper with global secure state. These vulnerabilities stem from the monolithic architecture of conventional TEEs, where a privileged Trusted OS centrally manages all TAs and shares memory space with their logic and services. TEEOD eliminates this attack surface by removing the need for a centralized software Trusted OS: each TA runs on a dedicated soft-core MCU within a hardware enclave, alongside minimal firmware that executes in the same privilege as the TA and exposes only a limited interface, tightly scoped to a single enclave. Shared services such as TA loading and secure storage are managed by dedicated, functionally isolated hardware agents. This decentralization of control and absence of a privileged software kernel in the TEE eliminates the trusted OS attack surface exploited in the aforementioned CVEs.

**TA-to-TA attacks.** Certain CVEs expose how insufficient intra-TEE isolation in TrustZone-assisted designs allows compromised TAs to interfere with or escalate privileges through other co-resident TAs. For instance, CVE-2015-8999 and CVE-2015-9108 describe scenarios where a malicious or hijacked TA leverages shared services or memory regions to influence the behavior of the Trusted OS and, indirectly, affect the execution of other TAs. These attacks rely on the fact that conventional TEEs often permit multiple TAs to run concurrently within the same secure memory space, with access to shared system resources and indirect communication paths via global service interfaces. TEEOD eliminates this attack surface by ensuring that each TA executes in a fully isolated enclave, implemented as a hardware block with a dedicated soft-core MCU and private memory. No enclave shares state, logic, or service interfaces with others. Shared services, such as secure storage, are accessed only through stateless TEEOD agents, which enforce strict access control and reveal no enclave-specific data. Furthermore, enclaves are reset and cleared upon each TA termination, preventing residual state persistence. This architectural isolation enforces the *R2: Isolate TAs from each other* security requirement and ensures that a compromised TA cannot interfere with or gain visibility into the execution of others.

**Sensitive data exposure.** CVE-2016-2432 reports that secure-world memory was not properly cleared after use, allowing a normal-world application to retrieve residual sensitive data—such as cryptographic keys or previously processed content—during subsequent interactions. This type of vulnerability arises from shared memory usage, persistent TA state, or improper teardown procedures, all common in conventional TEEs. TEEOD prevents such exposures by instantiating each TA in a dedicated enclave with private memory that is fully wiped upon session termination. Since enclaves are rebooted for every execution and the TEE firmware retains no secrets between sessions, no residual data persists beyond the enclave's lifetime, except when explicitly stored through secure storage. This design eliminates unintended data leakage to shared memory spaces and enforces the *R3: Protect sensitive data* requirement.

**Cryptographic misuse.** Vulnerabilities such as CVE-2015-9000 and CVE-2015-9002 highlight how insecure or inconsistent use of cryptographic primitives (such as IV reuse or misconfigured cipher modes) can result in severe confidentiality breaches when performed in generic software environments. TEEOD addresses these risks by encapsulating cryptographic operations within enclaves and exposing only a minimal, well-defined API that enforces correct usage. By preventing shared access with the REE to cryptographic services and enabling hardened FPGA-based execution paths, TEEOD reduces the attack surface and mitigates risks such as CWE-1240 (use of weak or risky cryptographic implementations), thereby fulfilling the *R4: Secure cryptographic primitives* requirement (see section 6.3.2).

### 6.7.2 Threats Out-of-Scope

While TEEOD addresses a wide range of TEE-related threats, certain classes of vulnerabilities remain beyond its scope due to architectural boundaries and threat model assumptions. For instance, CVE-2017-18310 originates from insecure TA design, where a TA unintentionally exposes internal services to the REE, allowing unauthorized access or misuse by malicious CAs. Although TEEOD enforces strong isolation between enclaves respecting the security requirement *R2: Isolate TAs from each other*, it does not control how individual TAs define their service interfaces or implement internal logic. These types of design flaws must be addressed during TA development. Likewise, hardware misconfiguration issues such as CVE-2017-18128, which result from insecure platform initialization (for example, enabling readback of sensitive configuration regions), fall outside the scope of TEEOD's protection model. Such threats originate prior to the TEE's runtime operation and must be mitigated through platform-level controls, particularly during secure boot and system provisioning. Although TEEOD may reduce the potential impact of these issues, their prevention is ultimately the responsibility of system designers and TA developers.

## 6.8 Discussion

### 6.8.1 TEEOD's Hardware Constraints

The current implementation of TEEOD relies exclusively on FPGA logic resources, specifically BRAM, to provide isolated memory for each enclave, thereby establishing the hardware-based TCB. BRAM offers low-latency access and strong physical separation, which are essential properties for enforcing strict spatial isolation among enclaves. However, its limited capacity poses scalability challenges. Even in high-end devices such as the Xilinx Virtex-7 series, BRAM availability is constrained to approximately 68 MiB [245], which inherently limits both the number of concurrently supported enclaves and the memory footprint of individual TAs.

Despite these limitations, our experimental evaluation (Section 6.6.1) demonstrates that TEEOD supports up to four concurrent enclaves, each allocated 64 KiB of private BRAM, on a resource-constrained SoC FPGA. To accommodate more demanding or memory-intensive workloads, we envision extending the

enclave memory space using off-chip DRAM, while maintaining confidentiality and integrity via a Memory Encryption Engine (MEE) or isolating access through a Memory Protection Unit (MPU). In this hybrid model, BRAM could function as a secure cache, with the bulk of enclave data stored in encrypted or integrity-protected DRAM or flash memory.

However, shifting from the BRAM-only constraint introduces new security challenges. Offloading enclave state to shared DRAM potentially weakens Design Principle *P1: Resource Isolation* and exposes the system to additional attack vectors. For example, Krautter et al. [103] demonstrated that malicious memory access patterns can induce remote voltage faults, an attack known as *FPGAhammer*, which compromises data integrity across logic regions. Such fault injection techniques may be adapted to interfere with DRAM-based enclaves. Moreover, using BRAM as a cache layer introduces susceptibility to cache-based side channels that could be exploited to extract sensitive information from neighboring enclaves. These considerations highlight the need for further investigation into scalable memory management mechanisms that preserve strong isolation guarantees.

### 6.8.2 Usability and Practical Applicability

TEEOD is designed to operate under realistic constraints typical of embedded and edge computing environments. It requires no external secure elements, trusted hypervisors, or new design architectural platforms. Instead, it establishes a self-contained hardware root of trust entirely within the FPGA fabric, making it deployable even on commercial available resource-constrained SoC FPGA platforms.

In practice, TEEOD supports a diverse range of TAs, including lightweight cryptographic primitives, secure sensor data processing, and hardware-assisted attestation. Section 6.6.3 demonstrates its practical viability through successful deployment on a low-cost SoC FPGA. From a usability standpoint, TEEOD adopts a modular hardware/software architecture that abstracts away many of the low-level hardware details traditionally associated with secure system development. This allows developers to concentrate on the TAs functionality, without requiring deep expertise in secure hardware design.

To facilitate portability and adoption, we provide firmware implementations that expose GP-compliant TEE interfaces for enclaves based on Cortex-M0 MCUs. These implementations are highly configurable and adaptable to other MCU architectures, as the interface between the TEE runtime and the enclaves remains lightweight and standards-compliant. This design simplifies integration, accelerates development, and reduces the likelihood of misconfiguration or security-critical implementation issues that commonly affect TrustZone-based TEE deployments [30].

### 6.8.3 Integration with Existing TEEs

TEEOD operates entirely within the FPGA fabric and requires no trusted software on the PS, making it agnostic to conventional TEEs, i.e., Intel SGX and Arm TrustZone. On Arm-based SoC FPGAs, TrustZone extends beyond the PS by propagating AXI security tags into the programmable logic (PL). These tags

enforce access control at the interconnect level, preventing non-secure masters from accessing secure-world peripherals. By connecting TEEOD agents to secure-world AXI interfaces, our design can leverage this mechanism to isolate PL-based enclaves from untrusted software in the REE.

TrustZone support enables the partitioning of the PL itself into secure and non-secure regions. This makes it possible to allow portions of the FPGA fabric to operate in the REE while confining TEEOD enclaves to secure regions—broadening deployment options in mixed-trust environments. Moreover, when deployed within the secure world, TEEOD further strengthens isolation by providing a secondary layer of protection. It enables finer-grained, hardware-enforced separation of sensitive logic and data beyond the coarse-grained boundaries natively supported by TrustZone.

However, prior work has demonstrated that adversarial reconfigurable logic can bypass TrustZone protections. For example, Jacob et al.[107] showed how malicious hardware can compromise the secure boot process on SoC FPGAs. Similarly, Benhani et al.[15] experimentally evaluated TrustZone on a Zynq-7000 SoC, revealing six proof-of-concept attacks that exploited minor modifications in PL logic to access secure-world memory or impersonate trusted peripherals. These findings underscore that further investigation is needed to define, implement, and validate a robust and secure integration model between TrustZone and TEEOD.

#### **6.8.4 Dynamic Partial Reconfiguration**

Dynamic partial reconfiguration (DPR) enables runtime updates of specific FPGA regions while leaving the rest of the configured logic and PS operational. In TEEOD, this could enable dynamic deployment or revocation of enclaves, tailored to system demand or changing trust relationships. Such flexibility could optimize enclave resource usage, reduce downtime, and enable reactive security postures. However, DPR introduces non-trivial security challenges. Improper validation of new bitstreams or reconfiguration errors could undermine isolation guarantees. While we acknowledge its potential, a secure DPR workflow is outside the current scope and will be explored in future work. Key directions include secure bitstream delivery, atomic enclave reconfiguration, and rollback protection.

#### **6.8.5 Physical Attacks Countermeasures**

TEEOD already integrates hardware-based protections to resist physical tampering, notably by relying on an authenticated and encrypted initial bitstream that contains all TEE agents, enclave partitions, and system services. This ensures that only trusted logic is configured during secure boot, with bitstream decryption and authentication enforced via the FPGA's AES-GCM engine and BBRAM-stored keys [246]. Reconfiguration interfaces such as JTAG and ICAP are disabled post-boot to prevent external access to configuration memory [239]. Once instantiated, enclaves remain spatially isolated within the FPGA fabric, while their execution contexts are dynamically defined through the loading of TA binaries—further narrowing the attack surface available to untrusted logic.

| System          | Cat.           | HW Mods | FW Indep. | GP API | Multi-TA | TZ Indep. | HLS Indep. |
|-----------------|----------------|---------|-----------|--------|----------|-----------|------------|
| HECTOR-V [254]  | New design     | ×       | ×         | ×      | ×        | ✓         | ✓          |
| Sanctum [255]   | New design     | ×       | ×         | ×      | ≈        | ✓         | ✓          |
| Sanctuary [256] | Resource reuse | ✓       | ≈         | ×      | ×        | ×         | ✓          |
| Graviton [189]  | Resource reuse | ✓       | ≈         | ×      | ×        | ✓         | ✓          |
| StrongBox [191] | Resource reuse | ✓       | ≈         | ×      | ×        | ✓         | ✓          |
| Ambassy [181]   | FPGA-based TEE | ✓       | ≈         | ×      | ≈        | ×         | ✓          |
| SrcTEE [182]    | FPGA-based TEE | ✓       | ≈         | ×      | ≈        | ×         | ✓          |
| ShEF [52]       | FPGA-based TEE | ✓       | ✓         | ×      | ×        | ✓         | ✓          |
| MeetGo [184]    | FPGA-based TEE | ✓       | ✓         | ×      | ×        | ✓         | ×          |
| BYOTee [186]    | FPGA-based TEE | ✓       | ×         | ×      | ✓        | ✓         | ✓          |
| TruFPGA [51]    | FPGA-based TEE | ✓       | ✓         | ×      | ×        | ✓         | ✓          |
| DF-TEE [183]    | Multi-FPGA TEE | ×       | ≈         | ×      | ✓        | ✓         | ✓          |
| <b>TEEOD</b>    | FPGA-based TEE | ✓       | ✓         | ✓      | ✓        | ✓         | ✓          |

Table 9: Comparison of prior related TEEs. **Cat.**: design style; **HW Mods**: requires hardware modification or non-standard SoCs; **FW Indep.**: does not require firmware to manage enclaves; **GP API**: compatible with GlobalPlatform APIs; **Multi-TA**: supports concurrent isolated TAs; **TZ Indep.**: does not rely on TrustZone; **HLS Indep.**: does not require translation of TAs using HLS.

To reinforce protection against fault injection, TEEOD could adopt redundancy-based strategies explored in recent FPGA security literature. For example, deploying multiple enclaves executing identical TAs and validating consistency among their results, as in Triple Modular Redundancy (TMR), can expose transient faults or active perturbations [247]. Likewise, future TEEOD designs could leverage partial reconfiguration to randomize enclave placement across boots, disrupting spatial targeting by attackers [248]. Embedding fault sensors—such as ring oscillators [249] or delay monitors [250]—alongside enclaves would also allow for real-time detection of anomalous behavior caused by laser or voltage-based glitching.

While invasive probing is limited by packaging and access controls, optical key extraction and remote physical attacks have been demonstrated on commercial FPGAs [251, 252]. In particular, remote fault injection via power transients (e.g., *FPGAhammer* [103]) and side-channel attacks via co-tenant logic [106] present credible threats in multi-tenant scenarios. TEEOD could be extended with lightweight passive countermeasures such as constant-latency execution, power masking [253], or placement obfuscation [122]. Combined with active monitoring of power and thermal variations, these mitigations would enhance enclave integrity and confidentiality even under remote or non-invasive attack vectors.

## 6.9 Related Work

Table 9 presents a comparative overview of recent hardware-based and FPGA-based TEE systems across key architectural and functional dimensions, including hardware modification requirements, firmware dependence, TrustZone reliance, High-Level Synthesis (HLS) translation, multi-TA support, and GlobalPlatform API compatibility. Rather than surveying commercial TEEs, we focus on research-driven proposals that explore novel TEE designs or repurpose SoC FPGA resources for trusted execution.

### 6.9.1 Hardware-based TEEs

Modern computing systems increasingly rely on hardware-based TEEs. While commercial solutions such as Intel SGX [54], Arm TrustZone [56], and RISC-V Keystone [257] or dedicated coprocessors like Google Titan [163] and Apple SEP [174] are widely deployed, their security has been repeatedly undermined by critical vulnerabilities. As extensively analyzed by Cerdeira et al. [30], such vulnerabilities stem from architectural flaws (e.g., excessive attack surfaces, coarse-grained isolation, and absent memory protections), implementation bugs (e.g., input validation errors and concurrency issues), and hardware-level weaknesses (e.g., side-channels attacks and insecure boot flows). In response, research has explored both alternative TEE designs and reuses of existing SoC resources to build TEEs without relying on proprietary CPU extensions. TEEOD contrasts with these solutions by providing isolation purely within FPGA logic, avoiding shared CPU microarchitectures and enabling FPGA-based trusted execution contexts.

**New hardware designs.** HECTOR-V [254] integrates a secure enclave into RISC-V cores, achieving tight coupling between secure and normal execution environments. However, its reliance on a custom ASIC limits deployability. In contrast, TEEOD builds on commercially available SoC FPGAs without requiring new silicon. Similarly, Sanctum [255] enforces strong isolation via modified caches and memory management, but its security guarantees rely on invasive CPU modifications. TEEOD avoids this by decoupling secure execution from the main processor entirely, isolating TAs in PL with minimal software dependencies.

**Non-CPU TEEs.** Sanctuary [256] partitions TrustZone-enabled cores to dedicate entire CPUs to TAs, achieving isolation at coarse granularity. Graviton [189] and StrongBox [191] use GPUs for isolation but are constrained by hardware availability and do not support concurrent multi-TA execution. TEEOD avoids such resource constraints by allocating enclaves within the PL, supporting up to four concurrent TAs even on low-cost SoC FPGAs. Unlike accelerator-bound designs, TEEOD’s resource scalability stems from the FPGA fabric, enabling more flexible and portable instantiations.

### 6.9.2 FPGA-based TEEs

FPGAs are increasingly used to support custom TEEs that implement hardware-enforced isolation [52, 181, 182, 184, 186]. These designs often prioritize hardware TA security but lack compatibility with standard TEE software stacks. TEEOD bridges this gap by supporting GP-compliant software TAs in FPGA logic.

**Secure reconfiguration and attestation.** TruFPGA [51] secures the FPGA bitstream via attestation and authenticated configuration but does not manage TAs at runtime or support legacy GP-based interfaces. TEEOD complements secure reconfiguration with runtime enclave management and standardized APIs, enabling software-level compatibility without sacrificing reconfigurability.

**Multi-FPGA isolation.** DF-TEE [183] extends trust across multiple FPGAs through partial reconfiguration and CPU attestation. Although effective for cloud-scale deployments, it lacks GP API compatibility and targets disaggregated architectures. TEEOD, by contrast, focuses on tightly coupled SoC FPGAs and supports legacy TAs through standardized interfaces, making it suitable for embedded deployments with standalone trust roots.

**TrustZone-enabled FPGA TEEs.** Embassy [181] and SrcTEE [182] both depend on TrustZone to manage and secure FPGA-based TAs. However, TrustZone’s known vulnerabilities [15, 107] limit their trust model. TEEOD eliminates TrustZone dependencies by enforcing all isolation and TA lifecycle management entirely in hardware, thereby reducing the trusted computing base and avoiding firmware-based vulnerabilities.

**Hardware-based TAs.** ShEF [52] creates secure IP modules for cloud FPGAs, but requires full hardware synthesis of application logic. MeetGo [184] shares architectural similarities with TEEOD, offloading TAs to the FPGA independently of the host. However, MeetGo mandates HLS translation of software TAs into HDL, requiring significant code adaptation and hampering GP compatibility. Additionally, it relies on asymmetric encryption (ECIES) for bitstream protection by default, which imposes considerable performance overhead. TEEOD supports unmodified TAs through a lightweight soft-core MCU and hardware backend, allowing seamless execution of legacy applications.

**Software-based TAs.** BYOTee [186], which Armanuzzaman et al. independently developed around the same time, is most related to TEEOD, as both instantiate isolated soft-core MCUs to run software TAs within the FPGA fabric. However, BYOTEE relies extensively on firmware for TA lifecycle management, including loading, scheduling, and cleanup, which must execute on each soft-MCU. In contrast, TEEOD implements these mechanisms as hardware-accelerated agents, eliminating dependence on enclave firmware and reducing the attack surface exposed to compromised TAs. This shift in the trust model substantially reduces firmware complexity and eliminates the need for security-enhanced soft-MCUs to thwart privilege-escalation attacks by a compromised TA. TEEOD also enforces a hardware-level reset and zeroization of enclave memory after each TA session, ensuring that no residual data, malicious code, or modified control flow persists across executions. This guarantees strong temporal isolation and limits the long-term impact of a compromised TA. Furthermore, TEEOD is, to our knowledge, the first FPGA-based TEE to support the GlobalPlatform API directly, enabling the execution of standard TA binaries without requiring any modifications. This provides improved portability and usability compared to previous proposals.

## 6.10 Conclusion

We introduced the concept of TEEOD, a new TEE approach that provides enhanced security guarantees by leveraging FPGA's reconfigurability to provide custom and dedicated execution environments per application on a per-need basis. TEEOD's evaluations have shown that it has significant potential for security-critical applications that require frequent TA calls, while at the same time freeing up the REE for other processes. Real-world application evaluations with a Bitcoin wallet demonstrate the practicality of TEEOD in ensuring confidentiality and integrity during operations. Future research intends to explore challenges related to user-defined custom hardware-accelerated TAs and explore enhanced isolation methods (i.e., dynamic function exchange technology), striving to fully unleash FPGA technology's potential for trusted computing to adapt to ever-evolving threats.

## TREE: Bridging the gap between reconfigurable computing and secure execution

***This chapter was published as:***

Sérgio Pereira, Tiago Gomes, Jorge Cabral, and Sandro Pinto. (2025). TREE: Bridging the gap between reconfigurable computing and secure execution. IACR Transactions on Cryptographic Hardware and Embedded Systems, 2025(3), 115-138. <https://doi.org/10.46586/tches.v2025.i3.115-138>

## Abstract

Trusted Execution Environments (TEEs) have become a pivotal technology for securing a wide spectrum of security-sensitive applications. With modern computing systems shifting to heterogeneous architectures, integrating TEE support into these systems is paramount. One promising line of research has proposed leveraging FPGA technology to provide promising TEE solutions. Despite their potential, current implementations of FPGA-based TEEs have a set of drawbacks. Some solutions (i.e., MeetGo and ShEF) prioritize the secure loading of reconfigurable modules but lack compatibility with established legacy TEE specifications and services. On the other hand, those that aim to establish legacy compatibility (i.e., TEEOD and BYOTee) fail to fully utilize the dynamic reconfigurability and parallel processing capabilities inherent in FPGAs. In this context, we introduce Trusted Reconfigurable Execution Environments (TREE), a novel framework that fulfills the gaps existing in current FPGA-based TEE approaches. TREE enables system designers to fully leverage the reconfigurability capabilities of FPGAs without compromising compatibility with existing TEE specifications. Our reference TREE implementation ensures secure execution of user-customized hardware, legacy software trusted applications (TAs), and TAs that combine both custom hardware and software components, by fully exploiting the FPGA's dynamic partial reconfiguration capabilities. TREE's root of trust relies on conventional SoC-FPGA mechanisms including secure initial reconfiguration and memory protection, to ensure the initial bitstream integrity is kept after loaded and that reconfiguration access is restricted to the FPGA fabric after boot. Additionally, TREE provides essential TEE services within the FPGA fabric, including secure storage and cryptographic functions, enabling TAs to securely store sensitive data and perform critical operations in an isolated environment. Our evaluation on an entry-level FPGA, involved assessing TREE using microbenchmarks and real-world applications to compare its hardware costs and performance speedups against OP-TEE. The results showed that TREE's hardware costs are minimal, while it achieves significant performance speedups, especially when compared to hardware TAs. For empirical demonstrations, we assess two real-world TA examples on TREE: an access control authenticator and a Bitcoin wallet.

## 7.1 Introduction

As modern computing systems transition from CPU-centric designs to heterogeneous computing, the security implications associated with these new architectures have not yet been met with adequate solutions [25, 26, 33]. Heterogeneous architectures (also known as xPU) combine different processing units, such as CPUs, graphics processing units (GPUs), Field Programmable Logic Arrays (FPGAs), and domain-specific accelerators (DSAs), to meet the growing computational needs and energy efficiency. Despite the performance benefits and the security measures in place for each unit individually, xPU architectures face unattended challenges in security [28, 41]. For example, adversaries can exploit one unit's specific vulnerabilities to compromise others and potentially compromise data confidentiality and integrity across the system [180].

Trusted Execution Environments (TEEs) have become a pivotal and continuously evolving technology for protecting security-sensitive applications [30]. Available on most commercial off-the-shelf (COTS) platforms, TEEs isolate the execution of trusted applications (TAs) from the untrusted main operating system, or Rich Execution Environment (REE) [258]. Moreover, to standardize and enhance TA development, the GlobalPlatform [233] TEE specifications are widely adopted. Intel SGX [54], Arm TrustZone [56], and AMD SEV [55] are well-established TEE technologies designed primarily for CPU-centric architectures.

Unfortunately, extending TEE support to xPU architectures poses several challenges. High-throughput xPUs (e.g., GPUs and FPGAs) achieve high performance by integrating multiple cores and using high-bandwidth memory. Adding features like memory confidentiality and integrity via encryption engines or Merkle Trees can reduce available memory capacity and bandwidth, critical for these xPUs [11]. Similarly, enforcing memory isolation through SGX-like checks during address translation would severely under-utilize accelerators due to their sensitivity to address translation latency [13].

With the rise of general-purpose GPUs, several works have successfully implemented TEEs on GPUs, with minimal or no hardware modifications required [189–191, 259]. These initiatives have paved the way for enhanced security guarantees on these systems. However, the security landscape differs significantly for other high-throughput xPUs, such as FPGAs. Unlike GPUs, which are tightly controlled by CPUs, FPGAs operate independently as hardware accelerators [31]. This introduces potential risks as CPU vulnerabilities could compromise the integrity of the reconfigurable logic in FPGA systems-on-chips (SoC-FPGA). Adversaries exploiting a compromised CPU could manipulate FPGA operations or access sensitive data [15, 107].

Researchers have explored the challenges of implementing isolated execution environments on SoC-FPGAs to protect sensitive operations from unauthorized access and tampering. ShEF [52] and MeetGo [184] have taken initial steps towards FPGA-based TEEs by relying on conventional FPGA mechanisms to secure loading custom hardware designs onto Cloud’s FPGAs. Other works (i.e., TEEOD [208] and BYOTee [186]) focus on isolating sensitive applications from malicious software, or hardware logic. This second approach enables TEEs by synthesizing soft microcontroller units (MCUs) at start-up, providing dedicated execution environments for software TAs.

Despite the progress made by these FPGA-based TEE solutions, none fully support unified TA management from development through runtime. MeetGo and ShEF are tailored for hardware-focused applications and lack support for software and GlobalPlatform-compliant TAs. While TEEOD and BYOTee address this gap, they fall short on leveraging FPGA’s partial reconfigurability. They rely on full system reconfiguration for new TA hardware requirements, rather than supporting independent isolation, updating, or optimization of TAs through partial reconfigurability. As a result, these post-boot fixed environments may struggle to accommodate unexpected runtime workloads, such as increased data processing (e.g., machine learning inference or real-time analytics) or adapting to evolving cryptographic algorithms (e.g., transitioning to new digital signature algorithms).

This paper builds on existing FPGA-based TEE approaches, advocating for the adoption of Trusted Reconfigurable Execution Environments (TREE). The TREE framework allows system designers to synthesize

(or compile) and execute customized TAs—whether hardware-based, software-based, or hybrid—isolated from the REE. Similar to other FPGA-based TEEs, TREE’s secure system resides in the FPGA fabric, while the remaining SoC units belong to the REE. TREE stands out by leveraging dynamic partial reconfiguration (DPR) technology, prevalent in most SoC-FPGAs, to facilitate the loading of TAs’ hardware, while ensuring their integrity. Additionally, TREE’s root of trust relies on standard SoC-FPGA mechanisms, such as secure reconfiguration and memory protection, removing the need for specialized hardware or proprietary extensions. TREE is designed to integrate seamlessly into existing development workflows, providing tools and processes that simplify TA development and deployment. It also offers TEE services like secure storage and cryptography, compliant with GlobalPlatform specifications.

Our evaluation of TREE was performed on an AMD Zynq UltraScale+ MPSoC development board (Ultra96-V2), featuring an entry-level SoC-FPGA. Our analysis of the reconfigurable hardware costs showed that TREE requires minimal FPGA logic elements, even on a low-resource SoC-FPGA, leaving ample capacity for TA hardware. To assess different TA deployment models, we compared TREE with the vanilla OP-TEE implementation, a widely used GlobalPlatform-compliant TEE on Arm TrustZone. Results show that TREE achieves faster round-trip times in GlobalPlatform-compliant CA operations (e.g., session opening, closing, and command invocation). TREE’s TAs outperform OP-TEE’s in basic computational tasks—such as checks and secure storage accesses. For more complex tasks, including cryptographic operations, TREE’s TAs show notable speedups over OP-TEE’s standard TAs. However, software-based TAs experience a slight slowdown, justified by the use of a small soft MCU. We demonstrate TREE’s real-world applicability with two proof-of-concept applications: an access control authenticator and an open-source, hardware-based Bitcoin wallet.

In summary, with TREE, we make the following **contributions**:

1. **Conceptualization of TREE:** Introduce the concept of TREE to address the limitations of current FPGA-based TEEs, focusing on dynamic reconfigurability.
2. **TREE Framework Design:** Develop a comprehensive security infrastructure within the TREE framework to ensure integrity and confidentiality while providing access to essential TEE features, such as secure storage and cryptographic engines, from the boot process onwards.
3. **Open-Source Reference Implementation:** Provide an open-source reference implementation of the TREE framework, facilitating adoption and further development by the research community and industry practitioners.
4. **Comprehensive Evaluation and Benchmarking:** Conduct thorough evaluation and benchmarking of the TREE framework to demonstrate its security, performance, and flexibility compared to existing solutions.
5. **Empirical Evaluation of TREE in Practical Applications.** To demonstrate TREE’s practical applicability, we evaluate its performance in real-world scenarios through two case studies: an

| Architecture family       | PR Granularity        | PR Primitive |
|---------------------------|-----------------------|--------------|
| AMD Virtex-5/6/7          | One clock region high | ICAP         |
| AMD Zynq                  | One clock region high | ICAP/PCAP    |
| AMD Ultrascale            | One CLB               | ICAP/MCAP    |
| Altera Stratix-V/Arria-10 | One ALM               | PR-IP        |
| Altera Stratix-10         | One sector            | SDM          |

Table 10: Architectural Features of Commercial FPGAs Supporting DPR technology.

access control authenticator and a Bitcoin wallet.

## 7.2 Background and Overview

### 7.2.1 SoC FPGA Partial Reconfiguration

SoC-FPGAs, e.g., AMD Zynq Ultrascale+ [81], integrate a programmable logic (PL) and a processing system (PS) within the same die. The PS encompasses non-reconfigurable units, i.e., application processor units (APUs), platform management units (PMUs), and configuration security units (CSUs), which handle processing tasks, system management, and security operations, respectively [32]. The PL's functionality is defined by bitstreams generated from hardware designs synthesized using hardware description languages (HDL), e.g., Verilog or VHDL [214]. FPGAs support various applications, including flexible software execution via soft microcontrollers (soft MCUs) within their fabric [31, 73].

**Dynamic partial reconfiguration.** SoC-FPGAs, e.g., AMD Zynq, enable partial reconfiguration of the PL during system initialization through various ports: i) the internal configuration access port (ICAP) for self-reconfiguration, ii) the processor configuration access port (PCAP) from the PS, and iii) the joint test action group (JTAG) interface for external configuration and debugging [239]. In applications requiring new functionality or feature updates, some FPGAs provide DPR technology, i.e., the ability to reprogram specific regions of the fabric, known as partial reconfigurable regions (PRRs), on the fly without disrupting the rest of the device [84, 216]. Table 10 compares the architectural features of commercially available FPGAs supporting DPR.

**Reconfiguration access control.** Under normal conditions, the PS has full control over the PL, allowing the PS to reconfigure, read data, clear, or power down either the entire PL or the PRRs. However, this control can be restricted to prevent unauthorized access. Manufacturers have implemented security measures to establish a loose coupling between the PL and the PS. These include having PCAP and ICAP interfaces mutually exclusive by design, disabling reset and readback ports, enabling encryption-only reconfigurations, implementing memory protection units (MPUs), enabling secure boot processes, and using physically unclonable functions (PUFs). These standard security features may serve as a strong

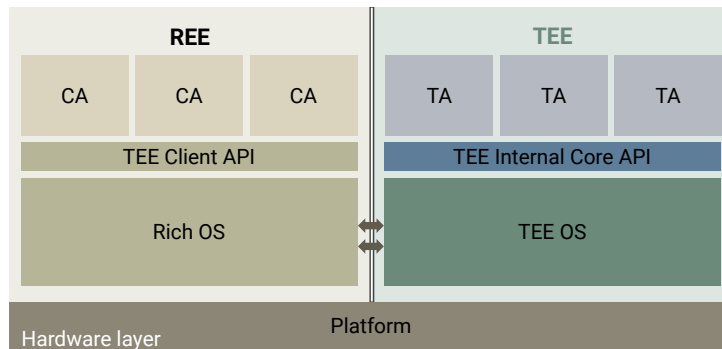


Figure 22: Generic TEE components complying with the GlobalPlatform specification. The REE, shown in brown, contains the Rich OS (e.g., Linux) providing non-trusted services, and CAs that interact with the TEE. The TEE, highlighted in green and blue, includes trusted components such as the TEE OS and TAs. The TEE Client API and TEE Internal Core API facilitate secure communication and provide TEE services, respectively.

foundation but could fall short in addressing scenarios where different security domains must co-exist within the PL. In such cases, additional security techniques (e.g., fine-grained isolation using TEEs or hardware-based memory partitioning) must be employed to establish isolated and independent regions within the PL [212].

## 7.2.2 Trusted execution environments

TEEs protect the processing of sensitive data, such as biometric data or cryptographic keys, in an untrusted environment. These secure environments comprise hardware and software features and the trusted computing base (TCB) [258]. The GlobalPlatform TEE Specification [234], widely adopted by the academia and industry and supported by numerous manufacturers and software developers, outlines the key components of a generic GP-compliant TEE (as shown in Figure 22). The Regular Execution Environment (REE), which runs outside the TEE and contains non-trusted applications and services, interacts with the TEE through the GP TEE Client API. Key trusted components include the TEE OS, which consists of a secure kernel, a trusted user interface, and secure services; TAs, which are the applications that execute secure-sensitive operations on top of the TEE OS and take leverage of the secure services provided by the GP TEE internal core API; and GP-compliant client applications (CAs), applications that run outside the TEE and use the GP TEE Client API to communicate with TAs [233].

**TEEs in xPUs.** With the shift from CPU-centric designs to heterogeneous computing architectures, TEEs now face the challenge of isolating trusted applications from malicious operating systems and malicious hardware and software across various processing units (xPUs), such as GPUs and TPUs [28, 29, 180]. This shift brings additional complexity to the design and implementation of TEEs. Additionally, TAs increasingly leverage these heterogeneous architectures to improve performance, utilizing specialized processors to accelerate tasks like cryptographic operations [260].

## 7.3 Threat Model & Design Goals

### 7.3.1 Threat Model

We consider a powerful adversary whose primary goal is to extract sensitive information from the TEE or a compromised TA, either locally or remotely, without resorting to intrusive methods. Such an adversary is capable of controlling and compromising all REE software and hardware components, such as the OS, memory, storage, and the platform's system interconnect, including the interfaces connecting to the TEE hosted on the FPGA fabric. The adversary might attempt to load, read back, or tamper with the hardware designs loaded on the FPGA fabric. However, it is assumed that access to the FPGA is attempted through standard operations (i.e., loading, reading back, or modifying via conventional channels) rather than through intrusive physical tampering.

Additionally, we assume that the development framework, compilers, and synthesis tools are trustworthy. We assume the SoC has not been subjected to tampering, thereby preserving the integrity of its hardware resources and standard SoC FPGA functionalities (e.g., secure boot and reconfiguration access control), as guaranteed by the manufacturer.

**SoC FPGA specific threats.** Our threat model adheres to the one commonly assumed for TEE architectures, e.g., as outlined by Cerdeira et al. [30], while accounting for the unique characteristics and known vulnerabilities of SoC FPGA platforms. Specifically, it considers potential attacks on the dynamic partial reconfiguration process, where an adversary might attempt to inject malicious bitstreams and software applications. We also account for the risk of a TA being compromised, and an adversary potentially escalating access and privileges to gain unauthorized control over other TAs or TEE resources. Intrusive physical attacks (e.g., micro-probing or fault injection) and side-channel attacks (e.g., power analysis or electromagnetic leakage) are out of scope, as their mitigations are orthogonal and can be integrated independently (see section 7.8 for further details).

### 7.3.2 Design Requirements

Considering the aforementioned threat model, we resorted to a set of principles to guide our design. We propose introducing the concept of reconfigurable TEEs. Reconfigurable TEEs address the shortcomings observed in existing FPGA-based TEEs by focusing on two key aspects: dynamic reconfigurability and comprehensive security features. The goal is to create a flexible framework for SoC-FPGAs, ensuring the integrity and confidentiality of various security-sensitive applications within the PL. To effectively achieve this goal, we propose the following design requirements:

**R1: Isolate FPGA fabric from the REE.** Create an isolated environment within the FPGA fabric to prevent unauthorized access and interference from the rest of the SoC-FPGA and to prevent malicious, non-secure software or hardware components from accessing or tampering with the FPGA fabric logic

and the TAs. This principle aligns with resource isolation guidelines in secure system design [261], which advocate avoiding shared hardware and software resources to prevent compromise and manipulation. As found in the literature [117, 120, 121, 241], shared resources have repeatedly been leveraged to extract the secrets and manipulate TA functionality.

**R2: Isolate each TA execution environment.** Establish dedicated and distinct secure environments for each TA, ensuring minimal sharing of FPGA logic elements to prevent potential interference and maintain integrity and confidentiality. Hardware TAs require dedicated FPGA resources such as logic elements and memory spaces, while software TAs require dedicated physical address spaces, computing units, and peripherals. Implement an efficient clean-up process to facilitate the reuse of secure environments for hosting different TAs during runtime. This approach supports the least privilege principle by limiting each TA to only the necessary for its operation, thus reducing the attack surface [60].

**R3: Ensure TA sensitive data protection.** TREE must provide secure storage primitives to protect sensitive data from unauthorized access, tampering, or leakage. This includes access control to guarantee only authorized TAs can access the stored data. This requirement aligns with guidelines to protect sensitive information through strict access controls and encryption [233, 234, 262].

**R4: Enable versatile TA synthesis and compilation.** Enable TA developers to design and compile custom hardware TAs, software-based TAs (including legacy TAs), and hybrid TAs (software TAs running on specialized hardware). This functionality should include both development-time tools and runtime processes, allowing developers to create and deploy their TA configurations within the FPGA fabric—whether hardware, software-based, or hybrid. This flexibility facilitates the integration of various TA configurations while managing complexity and mitigating potential vulnerabilities [187].

## 7.4 Overview & Architecture

Guided by the threat model and design goals in sections 7.3.1 and 7.3.2, the TREE design assumes all software running on the PS and associated hardware components as insecure, belonging to the REE, while the FPGA fabric is considered trusted. TREE divides the FPGA fabric into two modules: the **TrustShell** and **Dynamic Tiles**. The TrustShell incorporates several key elements that ensure secure operation and communication within the FPGA. Dynamic Tiles are the different dedicated execution environments for TAs and their exclusive interfaces to the TrustShell. Figure 23 illustrates TREE’s core design elements, including the REE, TrustShell, and Dynamic tiles.

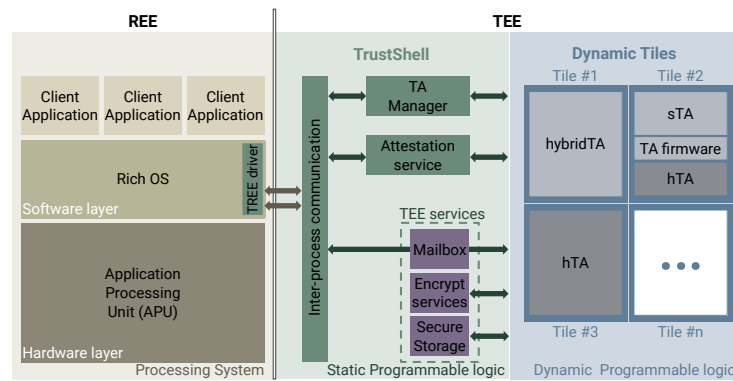


Figure 23: Design Elements of the Trusted Reconfigurable Environment (TREE) within an SoC-FPGA.

### 7.4.1 TrustShell

Configured securely into the FPGA during boot, the TrustShell corresponds to the TREE's static portion and is responsible for: i) enabling interprocess communication (IPC) between the CAs and TAs; ii) managing, loading, and attesting TAs; and iii) providing TEE services to TAs. The main components of TrustShell are:

**Tile manager engine.** This engine handles loading and managing available Tiles. It verifies the integrity and authenticity of each TA before loading it into a dynamic tile. The engine manages the full lifecycle of TAs, including loading, execution, and cleanup. Additionally, it provides to the TEE services information about the current TAs in execution, removing the need to rely on the TAs themselves for identification.

**TrustShell IPC.** This mediator manages the communication between the REE and TAs within dynamic tiles. It parses incoming REE messages, updates the Tile Manager Engine, and forwards them to the respective TA upon authorization.

**Attestation service.** This service verifies the integrity of the TrustShell configuration and active TAs, ensuring that the static configuration and TAs remain genuine and untampered.

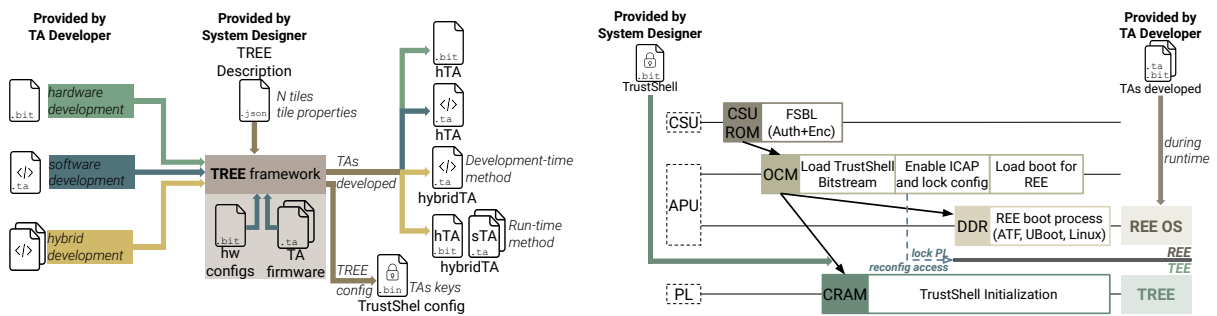
**TEE services and secure storage.** TrustShell provides TAs with a range of TEE services, including secure storage, cryptographic operations, and time services.

### 7.4.2 Dynamic tiles

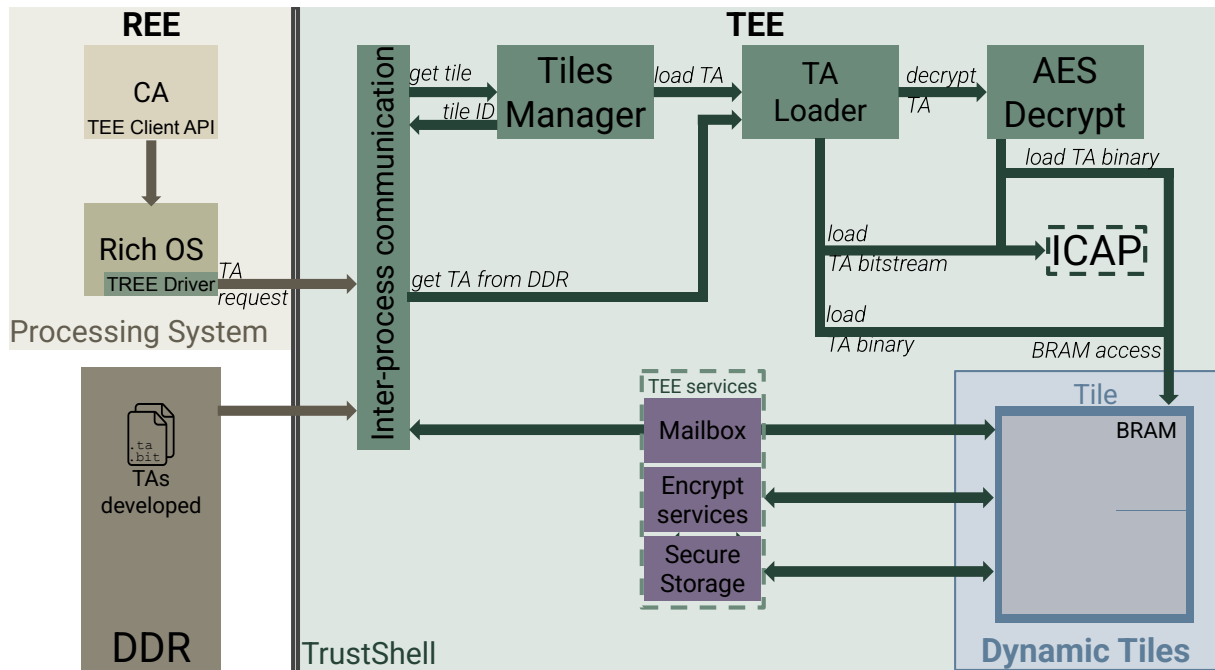
Each tile executes a single TA at a time and has exclusive interfaces to the TrustShell, shared memory, and TEE services. To support the execution of multiple TAs concurrently, additional dynamic tiles must be configured, with corresponding interfaces replicated for each TA. This architecture ensures independent and simultaneous TA execution. The main components of dynamic tiles include:

**Partial reconfigurable regions.** Each dynamic tile contains a PRR, a region within the FPGA fabric where a TA is loaded and executed. These regions can be dynamically reconfigured to load different TAs, enabling the execution of various hardware designs.

**TA firmware.** To support software TAs, compatible soft computing hardware must be configured within the PRR before loading the TA. The firmware, loaded with the TA, provides the runtime environment for execution, including essential kernel functions for booting, message parsing, error handling, and communication with TEE services.



(a) hTAs, sTAs and hybridTAs development differences (b) Boot-time sequence to configure the TrustShell and restrict the REE from accessing the PL.



(c) TREE runtime modules to manage, load, and monitor TAs and provide TEE services to them.

Figure 24: Workflow of the TREE framework at development-time (a), boot-time (b), and runtime (c). During development, the TREE framework allows system designers to synthesize/compile custom hTAs, software-based TAs, and hybrid TAs. During boot, the TrustShell is loaded to the PL. During runtime, TrustShell facilitates IPC between REE-TAs, manages and attests TAs, and provides TEE services.

## 7.5 Design & Implementation

TREE is highly modular and can be configured to provide single or multiple dynamic tiles with varying FPGA fabric allocations of PRRs for each tile. All TrustShell components are synthesized as hardware blocks in the PL. Figure 24 outlines the key differences in the TREE framework across the three processes: i) **development process**, which focuses on the TA development stage, designed to comply with GP specifications and offers similar TA development and TEE-REE interactions as other GP-compliant TEEs; ii) **boot process**, which involves the boot sequence and TREE configuration in the PL, where the secure environment is initialized to ensure proper operation; and iii) **runtime process**, which centers on running and managing TAs within a reconfigurable, secure environment, ensuring that the TREE framework meets its unique requirements.

TREE strives to align with GP specifications but faces unique challenges. These include handling encrypted TAs, proving GP support, and reusing and cleaning up tiles. These challenges are discussed in detail in Section 7.5.6.

### 7.5.1 TA development process

TREE allows system designers to synthesize/compile: i) custom hardware TAs (hTAs); ii) software-based TAs (sTAs); iii) GP-compliant TAs (GPTAs); and iv) hybrid TAs. Its modular design supports multiple tiles, each with its own PRR. System designers require a TREE configuration file that specifies the sizes and locations of these PRRs.

**hTAs development.** Developing hTAs is straightforward, as tiles are primarily composed of PRRs. System Designers only need to ensure that their designs adhere to timing constraints and fit within the dynamic regions defined by the TREE configuration.

**sTA development.** sTAs require a compatible hardware execution environment. TREE provides pre-designed trusted hardware configurations, abstracting hardware design complexities. Additionally, TREE offers compatible, GP-compliant firmware that supports kernel functionalities like booting, message parsing, and error handling. This compliance ensures adherence to widely accepted security standards, facilitating TAs' integration into diverse environments without compromising compatibility or security. Alternatively, system designers can create custom hardware, as detailed in the hybrid TA development.

**hybridTA development.** TAs that integrate both hardware and software, provide system designers flexibility in leveraging FPGA hardware acceleration alongside software functionalities. Depending on the use case, system designers may need either tightly coupled integration for optimal performance or loose coupling for flexibility in updates and third-party involvement. TREE supports two development models: i) *development-time model*: The software and hardware are integrated during development. In this case, TREE treats the hybrid TA as an hTA during runtime. This approach is ideal for tightly coupled systems that

do not require post-deployment updates or external contributions; and ii) *runtime model*: The hardware and software are compiled and synthesized separately. During runtime, the hardware is loaded first, followed by the verification and loading of the software. While introducing some performance overhead, this model offers flexibility, allowing the hardware to support various software versions and third-party applications. TREE ensures compatibility by attesting the hardware and verifying software on-the-fly to prevent unauthorized execution.

### 7.5.2 Boot-time process

TREE relies on the secure boot to guarantee: i) the integrity and secure loading of TrustShell into the PL; ii) exclusive TrustShell access to the FPGA configuration port (i.e., ICAP); and iii) restricted access to the PL's reconfiguration port (i.e., PCAP and JTAG), including reconfiguration, readback, clearing, and power management. These measures are necessary to prevent PS and external interactions with the reconfigurable logic, thus ensuring a loosely coupled relationship between the PS and PL.

### 7.5.3 Runtime process

After TrustShell is configured and booted, it can load TAs on demand. TREE supports one PRR per tile and one TA per tile at a time, adhering to design requirement *R2* (see Section 7.3.2). For hTAs, or the hardware components of sTAs and hybrid TAs, TrustShell verifies the hardware's integrity and decrypts it if required before partially reconfiguring the selected TA on an available tile. Additionally, for sTAs and hybrid TAs using the *runtime model*, TrustShell's verification process is repeated to now ensure the integrity of the TA binaries before loading them into the tile's memory. Once the TA is loaded, the REE and TEE communication follows a standard RPC model, facilitated by a shared mailbox infrastructure with IPC mediation. TrustShell handles one request at a time per dynamic tile. During the TA's lifecycle, TrustShell: i) provides attestation services to verify the TA's integrity; ii) maintains tile isolation to ensure security; and iii) delivers core TEE services, including secure storage and cryptographic engines.

### 7.5.4 Supporting Globalplatform specifications

**CA-TA communication.** The communication between a CA and a TA follows a default RPC model, where the CA sends a request and waits for a reply from the TA. Each tile has a mailbox shared with the REE, facilitating this communication. CAs determine which mailbox to use via a TA identifier register. This register, managed by the Tile manager and read-only accessible by the REE, contains the TA identification running in that tile. The mailbox is inactive if the register is zero, indicating no TA is running in that tile. The mailboxes also include fields for operation-specific parameters and return values, ensuring accurate request processing and correct routing of responses to the requesting CA.

**TEE services.** A TEE usually offers the TAs a set of security-enabling services. Most of them, such as cryptographic operations and time-related services, are only necessary during TA execution and do not require sharing or interaction with other TAs, except for secure storage. As a result, these kinds of services are integrated into the hTA design or as software services embedded in the TA firmware. These services are optional and do not require special access rights beyond what TAs need, removing the need for privilege mode implementation, as it's up to the TA developer to add them.

**Secure Storage.** Cryptographic operations and time-related services are necessary only during TA execution, while secure storage operates independently of the TA's activity status. Tiles are wiped and reset when the TA is inactive. Secure storage, therefore, is a TEE service external to the tiles, ensuring the persistent storage of sensitive data. This service requires the universally unique identifier (UUID) of TAs attempting to access their data. However, secure storage cannot rely on TAs for self-identification, as the threat model in section 7.3.1 considers attacks by malicious TAs on others. TREE addresses this by using the Tiles Manager to provide correct TA identification, ensuring that malicious TAs cannot impersonate others and access data.

### 7.5.5 TA attestation mechanism

TREE allows CAs to verify the authenticity and integrity of TAs that are executed within the tiles at any time. The TrustShell carries out proof whenever a request for remote attestation exists. For this, the SA employs a static measurement scheme. Whenever the TrustShell loads an application into a tile, it measures (computes the hash of) the bitstream and stores it in a table along with the TA's UUID. Such static measurement is trustworthy because, in the TREE design, TrustShell is the only entity authorized to access the dynamic area where applications are installed.

### 7.5.6 Challenges of integrating DPR with TEE

TREE heavily relies on DPR technology to create a flexible FPGA-based TEE. DPR controller and decoupler features enable TREE to instantiate TAs as partial bitstreams at runtime without disrupting the operation of other tiles, the TrustShell, or the REE. However, the DPR capabilities alone are insufficient and limited to provide a TEE. Key challenges include supporting encrypted TAs to ensure confidentiality, efficiently managing the clearing and reuse of tiles to maintain both security and resource availability, and configuring TAs onto any available tile without any interference or resource contention. These challenges require additional mechanisms and strategies to enhance the basic DPR technology to meet the stringent requirements of a secure and efficient TEE environment.

**Encrypted TAs.** Most SoC-FPGAs only support reconfiguring unencrypted partial bitstreams or encrypted partial bitstreams with the same key as the full bitstream instantiated. This presents a challenge in offering system designers confidential loading capabilities without requiring them to expose their TAs

to the REE, or revealing sensitive keys within TrustShell belonging to other system designers. TREE addresses this challenge by synthesizing an AES symmetric decryption IP before accessing the ICAP. This benign man-in-the-middle decrypts and directly forwards the unencrypted partial bitstream to the ICAP. Our implemented mechanism does not require having all the plain partial bitstreams at once in a trusted storage. Instead, each decrypted frame is sent directly to the ICAP, avoiding the plaintext temporary storage management.

**GPTAs support.** Regarding sTAs that are Globalplatform-compliant, compatible hardware must be configured to execute them. System designers can use TREE's trusted hardware configurations that support GPTAs to abstract the complexities of hardware design and maintain their legacy. TA runtime environment support, i.e., TA firmware, is also provided; this firmware provides essential trusted kernel functionalities for booting, message parsing, and error handling. It also serves as the TEE IC API's backend, allowing TAs to interact with the TEE services and hardware resources seamlessly.

**Tiles reconfiguration and cleanup.** Regarding the cleanup process, TREE keeps blanking bitstreams for each tile. Once the former TA context fully concludes, TREE starts overwriting the PRR with the blanking bitstream. Once the partial reconfiguration is complete, the tiles manager erases the TA entry and resets the mailbox. Only then does the TREE attribute the tile to another TA, maintaining the isolation requirement.

**Reconfiguring TAs onto any available tile.** The size and location of PRRs must be fixed during development; they cannot be modified without fully reconfiguring the PL. Also, partial bitstreams are tied to specific PRRs, preventing them from being reassigned to different PRRs. TREE tackles this challenge by ensuring all tiles have similar sizes and the hTAs are synthesized for all tiles available. At runtime, TREE selects the suitable PR associated with the available tile, preventing resource allocation issues where multiple TAs demand the same PRR concurrently while others remain unused.

## 7.6 Evaluation

We evaluated TREE using the Ultra96-V2<sup>1</sup> development board. TREE heavily relies on DPR technology. The Ultra96-V2 supports dynamic function exchange (DFX), a DPR technology for AMD FPGAs, allowing up to 32 PRRs. We decided only to consider one PRR per tile configured. All FPGA modules were synthesized using the AMD Vivado tool, configured to operate at 100 MHz.

<sup>1</sup>AMD Zynq UltraScale+ MPSoC ZU3EG, featuring an entry-level FPGA, a quad-core Arm Cortex-A53, and 2GB of DDR4 SDRAM.

| TrustShell module |                     | LUT<br>(70336) | LUTRAM<br>(28720) | FF<br>(141120) | BRAM<br>(216) | DSP<br>(360) |
|-------------------|---------------------|----------------|-------------------|----------------|---------------|--------------|
| Tile Manager      | DFX Controller      | 969<br>(1.4%)  | 169<br>(0.6%)     | 1149<br>(0.8%) | 0             | 0            |
|                   | AES decryption IP   | 2365<br>(3.3%) | 172<br>(0.6%)     | 2970<br>(2.1%) | 0             | 0            |
|                   | Attestation Tester  | 90<br>(0.1%)   | 33<br>(0.1%)      | 318<br>(0.2%)  | 0             | 0            |
| TEE services      | Shared Memory       | 398<br>(0.5%)  | 0                 | 371<br>(0.3%)  | 2<br>(0.9%)   | 0            |
|                   | Encryption Services | 2435<br>(3.5%) | 174<br>(0.6%)     | 3260<br>(2.3%) | 0             | 0            |
|                   | Secure Storage      | 406<br>(0.5%)  | 0                 | 383<br>(0.3%)  | 2<br>(0.9%)   | 0            |
| Total             |                     | 4228<br>(6.0%) | 374<br>(1.3%)     | 5191<br>(3.7%) | 4<br>(1.8%)   | 0            |

Table 11: Synthesized results of TrustShell module on the Ultra96-V2.

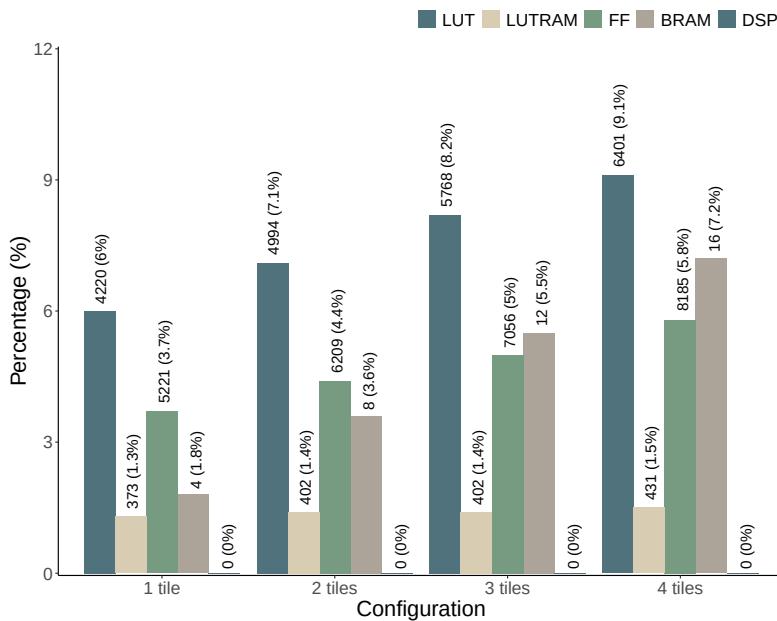


Figure 25: Resource utilization across multiple tile configurations. The plot illustrates how the usage of various FPGA logic elements scales with the increase of tiles, from 1 tile to 4 tiles.

### 7.6.1 Reconfigurable hardware costs

We evaluate TREE’s FPGA logic resource allocation in single-tile and multi-tile configurations. This serves two goals: i) to assess the minimal overhead by measuring the FPGA logic allocation efficiency of the minimal configuration; and ii) to evaluate scalability by analyzing how resource usage increases with additional tiles.

**Methodology.** For each TREE configuration evaluated, we measured the FPGA logic elements required, including look-up tables (LUTs), LUTRAMs, flip-flops (FFs), Digital Signal Processors (DSPs), and Block RAMs (BRAMs). We conducted resource utilization tests to quantify the logic elements used by the static

portion of TREE (TrustShell), which includes the Tile Manager, TA loader mechanism, TA decryption engine, and TEE services which include mailboxes for communication between CA-TAs, cryptographic engines, and secure storage. Each configuration considers 8 kiB of secure storage per tile of BRAM. Tile granularity is not relevant to this evaluation, as it does not belong to TrustShell.

**Single-Tile configuration.** Table 11 details the FPGA logic elements usage of each TrustShell component for the single-tile configuration. The Tile Manager, composed mainly of combinational logic, consumes 4.8% of LUTs, 1.3% of LUTRAM, and 3.1% of FFs, while the shared memory and secure storage, which are dependent on the defined memory size, use 1% of LUTs, 1.8% of LUTRAM, and 0.6% of FFs. TrustShell, despite being central to TREE, only requires a small portion of an entry-level FPGA, according to the results. Consequently, most FPGA logic elements remain available for the TAs loaded on the tiles.

|               | Config (ms) | Clean-up (ms) | Communication ( $\mu$ s) |
|---------------|-------------|---------------|--------------------------|
| Raw hTA       | 2.387       | 2.425         | 35.5                     |
| Encrypted hTA | 57.4        | 2.425         | 35.5                     |

Table 12: Microbenchmarks for hTA configuration, clean-up, and TREE-hTA round-trip communication on a tile covering an entire FPGA clock region (954 kiB).

**Multi-Tile configuration.** As shown in Figure 25, the resource usage for memory logic elements scales predictably with the number of tiles. Each additional tile increases the consumption of logic elements—LUTs, FFs, and BRAMs—by 1.1%, 0.7%, and 1.85%, respectively. LUTRAM usage, however, grows only slightly (<0.1%) since most of the added resources go to shared memory and secure storage, which do not rely on LUTRAM. The tile manager’s resource usage remains constant, confirming TREE’s scalable design.

## 7.6.2 hTAs support

The support for hTAs in TREE was evaluated by measuring the execution times of key operations across various tile configurations. Additionally, we assessed the overhead introduced by TREE by comparing the execution time of a hardware storage application with that of a similar native application running independently, with no trusted computing.

**Methodology.** To assess how TREE handles the lifecycle of hTAs, we measured the execution times for hTA configuration, clean-up, and round-trip communication between the CA and TA. Execution time was defined as the duration from the CA’s request to the successful return of a status response. The evaluation was conducted using tiles configured to an entire clock region (954 kiB) of the SoC-FPGA.

|             | <b>Overhead (%)</b> | <b>Average (ms)</b> | <b>Coeff. of Var. (%)</b> |
|-------------|---------------------|---------------------|---------------------------|
| Native app. | -                   | 21.4 ms             | 0.05                      |
| hTA         | 0.9                 | 21.6 ms             | 0.06                      |

Table 13: The hardware storage operation reads 8 kiB of data in 1 kiB chunks. The scenarios are: vanilla hardware application configured in the static PL (Native App.) and hTA configuring within TREE.

**Micro-operations.** Table 12 summarizes the microbenchmark results for the lifecycle of hTA operations on a single tile. The extended configuration time observed for encrypted hTA instances stems from the block-by-block AES decryption process, which could benefit from parallelization optimizations. Once configured, TREE-hTA operations exhibit minimal communication latency, remaining largely unaffected by tile dimensions.

**Hardware Storage Application.** The results for the hTA overhead are summarized in Table 13. Our evaluation indicates that the performance of hTAs remains comparable to native applications, with an average overhead of less than 1%. This minimal overhead can be attributed to the efficiency of the CA-hTA communication. The results indicate that TREE do not compromise operational efficiency.

### 7.6.3 sTAs and GPTAs support

We measure the round-trip communication time for GP-compliant CA operations: opening and closing sessions, and invoking commands. We compared TREE with the vanilla OP-TEE [167], a software-based GP-compliant TEE. The goal is to measure the TREE’s IPC communication latency while ensuring GP compliance.

**Methodology.** To enable the execution of software TAs, including GPTAs, TREE must first configure a tile with reference hardware. The reference hardware uses an Arm Cortex-M1 as the soft-core MCU with 64 kiB of BRAM for code and data memory. For the cryptographic operations and Bitcoin TA, we used Linux time API. Due to potential deviations between the performance overhead on an FPGA-based soft MCU at 100 MHz and a hard processor in silicon at 1GHz, we have performed clock frequency normalization. The results of vanilla OP-TEE were normalized by multiplying them by the ratio of its frequency (1 GHz) to the reference of 100 MHz.

**Benchmarks.** Figure 26 illustrates the execution time comparison between TREE and the vanilla OP-TEE. Opening a session while the TA is already loaded with TREE has a significant speedup compared to OP-TEE, i.e., 83.8x. This improvement is due to TREE’s design that treats any subsequent open session as invoked commands, once the TA is loaded. Another significant speedup is regarding the closing session by 22.1x. This improvement is justified by TrustShell’s efficient utilization of FPGA parallelism.

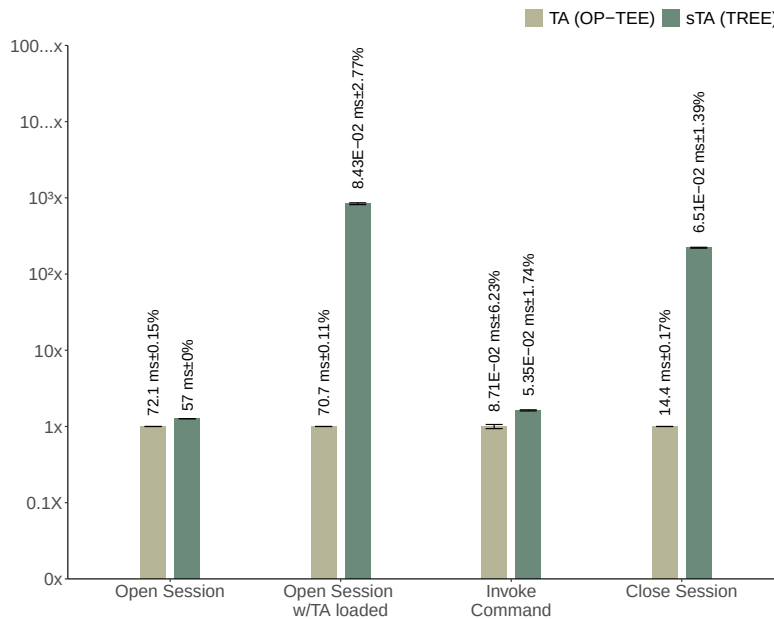


Figure 26: Performance speedup comparison between TREE and vanilla OP-TEE for GP Client API operations: open session, close session, and invoke command.

### 7.6.4 hybridTAs support

TREE provides two models for supporting hybrid TAs: i) *development-time model*; and ii) *runtime model*. Since the primary difference between the models is the software loading mechanism, this evaluation focuses on the performance impact of the TA loading process.

**Methodology.** We measure the execution time of TA loading operation for both models on the same hybrid application under different conditions. Specifically, we fixed the tile size to one clock region (954 kiB) and varied the software binary size (from 2 kiB, 4 kiB, 8 kiB, up to 16 kiB) to identify the performance trade-offs between the two approaches.

**Impact of TA loading.** As shown in figure 27, the relative performance of the *runtime model* compared to the *development-time model* decreases as memory size increases. For example, at 16 kiB, the *runtime model* achieves 98.4% of the development-time performance, but this drops to 48.9% at 1 MiB. This is due to the additional time required to load the software after hardware reconfiguration in the *runtime model*, which is more noticeable for larger binaries. For instance, the 1 MiB binary requires an additional 60.1 ms. In contrast, the *development-time model*'s performance is unaffected by binary size, as the software is directly loaded into the BRAM, which serves as on-chip memory for the soft MCU, during partial reconfiguration without requiring additional setup.

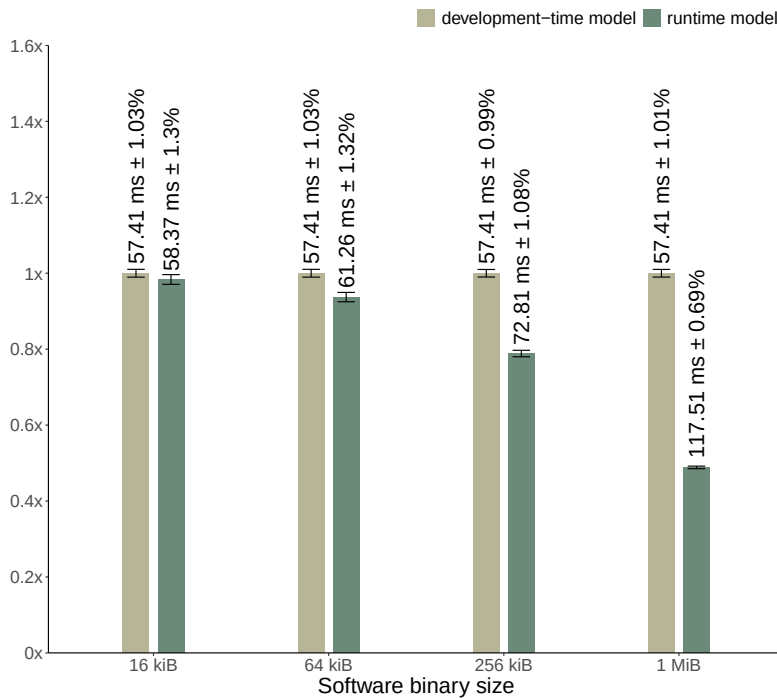


Figure 27: Performance comparison of TA loading models in hybrid TAs. The graph illustrates the relative execution time of the runtime loading model versus the development-time loading model.

## 7.7 TREE: Use Cases

### 7.7.1 Access control authenticator

The access control authenticator operation is a requirement in a wide range sectors- such as finance, healthcare, and data protection. Its primary goal is to verify user credentials and gate access to protected assets. Given the heterogeneity of workloads, compliance regulations, and security requirements in these industries, developers benefit from customizable solutions. To evaluate these requirements, we developed a minimal access control authenticator in three configurations: i) hardware-only version (hTA); ii) GP-compliant version (sTA); and iii) hybrid version combining both approaches (hybridTA). Each version performs the same core task—digesting new user credentials and comparing them sequentially against a predefined list of authorized credentials. Upon verification, the system returns an authentication result. We also run the same software TA on OP-TEE (TA) as a baseline for comparison.

**Methodology:** We focused on two key operations: the authentication process (Authentication) and the sequential checking of credentials against the authorized list containing 1,024 entries (Check loop). For each version, we measured the round-trip execution time of the invoke command operation calling the respective operation.

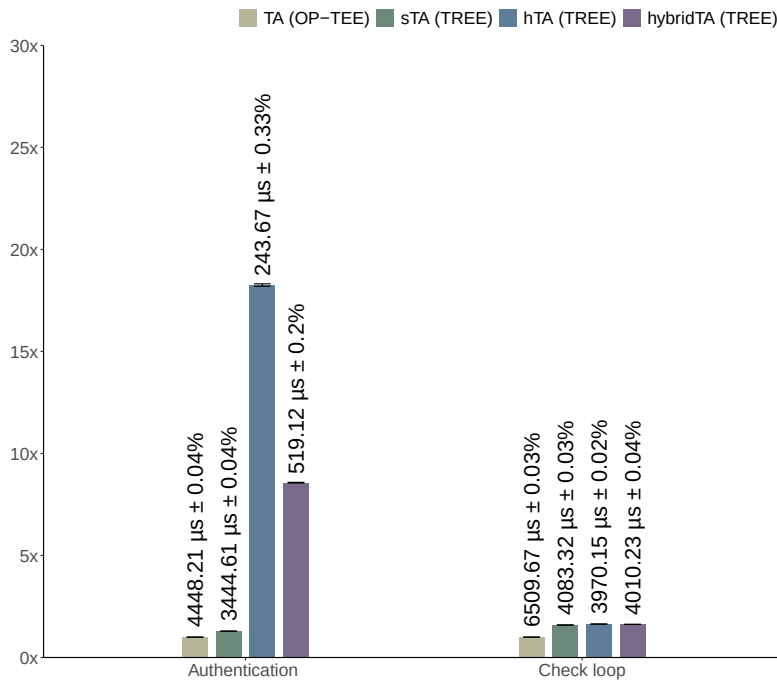


Figure 28: Performance speedup comparison between hardware, software and hybrid versions of the same TA against the same TA running in OP-TEE.

**Evaluation:** Figure 28 presents the performance speedup across the hardware, software, and hybrid implementations compared to the TA running in OP-TEE. The hTA achieved up to 18x speedup for the authentication service, leveraging hardware-accelerated logic. The hybrid version also showed notable speedups but was slower than the hardware-only solution due to its mix of hardware acceleration and software fallback. However, it outperformed the software TA, which achieved a 1.1x speedup. In contrast, for the check loop process, which primarily involves reading and verifying credentials in secure storage, there were no significant performance gains when using the hardware-only or hybrid versions. This is because the check loop is largely bound by read operations rather than intensive calculations, limiting the impact of hardware acceleration.

### 7.7.2 Trust Bitcoin wallet

The Bitcoin wallet is a well-fitted real-world example for evaluating TREE, as it handles security-sensitive tasks like key management and transaction signing, which would benefit from the reconfigurability provided by TREE. Since the wallet<sup>2</sup> was initially built for OP-TEE and follows the GP specification, porting it to TREE was straightforward. Both the CA and sTA ran on TREE without any changes. Additionally, we evaluated a modified version of the wallet, in which we migrated the wallet-specific cryptographic algorithms to hardware to assess the performance benefits of TREE’s reconfigurable computing.

<sup>2</sup><https://github.com/Jamie-Cui/bitcoin-wallet>

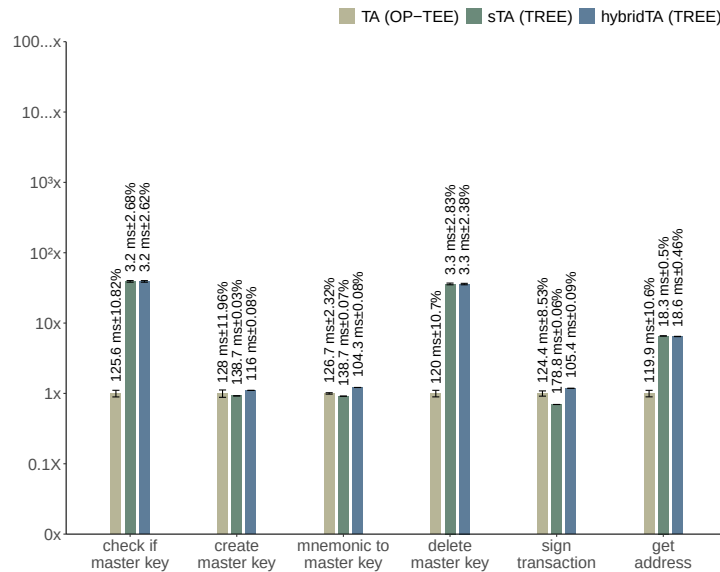


Figure 29: Normalized performance speedup of TREE (vanilla and hybrid versions) and OP-TEE of each Bitcoin wallet operation.

**Methodology:** To enable the execution of software TAs, including GPTAs, TREE must first configure a tile with a reference hTA. We use an Arm Cortex-M1 soft-core MCU with 64 kiB of BRAM for code and data. Due to potential deviations between the performance overhead on an FPGA-based soft MCU at 100MHz and a hard processor in silicon at 1 GHz, we have performed clock frequency normalization. We measure the execution time of six operations: i) check if a master key exists; ii) create a new master key; iii) create a new master key from a mnemonic; iv) delete key; v) sign transaction; vi) get public address. Regardless of the operation, CA communicates with the TA by opening a session, invoking a command, and closing the session upon completion.

**Evaluation:** Figure 29 presents the results for the Bitcoin wallet TA. Our findings show that the vanilla (sTA) and hybrid (hybridTA) versions running on TREE achieve comparable normalized performance improvements for operations involving simple secure storage verification, such as checking or deleting a master key. This similarity arises because these operations rely only on basic functionalities (e.g., straightforward secure storage access), which do not benefit from hardware acceleration. However, both versions significantly outperform the vanilla version running on OP-TEE, which is justified by the closer access to secure storage and the reduced overhead of GlobalPlatform operation calls. Regarding more intensive cryptographic services (e.g., creating a master key or signing a transaction), the performance gap between the two versions becomes evident. The vanilla version experiences a 1.5x slowdown, while the hybrid version delivers a 1.1x speedup. Additionally, the comparable performance of hybrid TAs and OP-TEE can be explained by the hardware-accelerated cryptographic enhancements, which offset the slower execution of software on a small soft MCU, resulting in overall performance similar to OP-TEE.

## 7.8 Discussion

**Encrypted TA key Provisioning.** TREE securely loads TAs using an AES symmetric decryption IP to decrypt partial bitstreams before reaching the ICAP, ensuring isolation from the REE and other TAs. While TA key provisioning is not currently supported, our core mechanisms lay the groundwork for its future implementation. Some key exchange protocols necessitate a secure channel for exchange, whereas others like Elliptic Curve Diffie-Hellman (ECDH) do not require a pre-shared secret but should be paired with authentication services. Future iterations will analyze and assess the suitability of these methods for the TREE concept.

**SoC-FPGA's main memory management.** TREE's threat model assumes DRAM and other SoC-FPGA memories are untrusted. Sharing DDRs among multiple entities (trusted and untrusted) raises significant security concerns particularly side-channel. Adversaries might exploit shared memory access to infer information via timing, power consumption, or electromagnetic emissions, compromising confidentiality and integrity. Future research will analyze the security impacts of allocating some DRAM regions to Tiles and TrustShell services, sharing the DRAM with REE and TEE. This strategic approach aims to enhance scalability and efficiency across the framework.

**REE performance impact.** Since TREE leverages FPGA solely to provide the execution environment necessary for TAs, it presents a potential to benefit the REE performance as the TEE within the FPGA operates in parallel with the REE. Further research will quantify these performance benefits by analyzing the nature of TAs that would provide the most overall system efficiency. Additionally, this research will determine the suitable use cases for reconfigurable FPGA-based TEEs, identifying scenarios where the unique capabilities of FPGAs can be maximally leveraged.

**Tile temporal and spatial multi-Tenancy.** TREE treats each tile as a single execution environment, supporting one TA at a time with fixed FPGA space. However, we believe that this could be further enhanced. Some SoC-FPGAs, e.g., AMD's Zynq, support nested DPR, allowing dynamic subdivision of regions. This could overcome the tile size configuration limitation, offering more flexibility in adjusting tile sizes for different hTAs. Additionally, more advanced soft-MCUs with TEE features could be synthesized on TREE, enabling concurrent execution of multiple software TAs and adding an extra security layer. While technically feasible, these multi-tenancy models introduce complexity that may outweigh the benefits for the lightweight TEEs in this research. Exploring these ideas may benefit future systems, but is beyond TREE's current scope.

**Physical and side-Channel attacks.** FPGA-based attacks, e.g., power analysis, electromagnetic leakage, and fault injection, could potentially compromise the integrity of both the TEE and TAs. TREE

does not directly address these specific, indeed intrusive attack scenarios, but many mitigation techniques can be applied. For instance, fault injection resistance can be achieved through redundancy techniques, where a dynamic tile is replicated and the outputs are compared to detect inconsistencies [263]. Side-channel attack defenses e.g., noise generation, dynamic voltage, and frequency scaling, and power analysis countermeasures can be implemented to mask signals from being exploited in power or electromagnetic analysis [48]. Physical probing can be mitigated through techniques, e.g., physical unclonable functions (PUFs) [264]. Similar techniques can be considered for future enhancements to TREE, but their mitigation will not impact the current focus on reconfiguration and software isolation security.

## 7.9 Security Analysis

### 7.9.1 Theoretical security analysis

We analyze TREE's security by theoretically assessing its implementation of security properties. This includes addressing the security requirements and threat vectors detailed in sections 7.3.2 and 7.3.1, and emphasizing TREE's protective mechanisms.

**Isolate FPGA fabric from the REE (R1).** During boot time, access to reconfiguration ports is restricted to maintain a secure boundary between the FPGA fabric and the rest of the SoC, including the APU and peripherals. This restriction is enforced by disabling the external reconfiguration ports (PCAP and JTAG) and allowing only the ICAP to remain active. This ensures that unauthorized software, including potentially compromised applications on the REE OS, cannot reconfigure, read back, clear, or power down the entire or partial FPGA fabric. Consequently, they ensure that TREE modules and loaded TAs remain protected against tampering and malicious interference.

**Isolate each TA execution environment (R2).** Each TA has its own dedicated execution environment, i.e., a tile, which includes the necessary resources, dedicated access to TEE services (e.g., encryption and secure storage), and essential peripherals for independent execution. Regarding sTAs, this also includes a unique physical address space, CPU, system configuration registers, and TA firmware. To prevent any potential interference or tampering, TAs do not share resources. However, for services like secure storage that cannot be uniquely dedicated to a single TA, TREE does not trust TAs to self-identify. Instead, it uses a backend process that handles the identification of the TA attempting to access it. This approach protects against threats, e.g., resource contention, unauthorized data access, and tampering between TAs. It also protects against attacks where a compromised TA could misuse shared services or alter its own tile's hardware logic to gain unauthorized access or interfere with other TAs.

**Ensure TA sensitive data protection (R3).** TAs, when idle, remain in the untrusted memory of the REE. Some TAs contain sensitive data, e.g., cryptographic keys, or user credentials, which must be

| CVE        | C    | CVE        | C    | CVE        | C       | CVE        | C       |
|------------|------|------------|------|------------|---------|------------|---------|
| 2015-4422  | TA ✓ | 2015-9108  | TO ✓ | 2017-18133 | TO ✓    | 2015-8997  | TO/TA ✓ |
| 2015-6639  | TA ✓ | 2015-9112  | TO ✓ | 2017-18311 | TO ✓    | 2015-8998  | TO/TA ✓ |
| 2015-6647  | TA ✓ | 2015-9113  | TO ✓ | 2017-18314 | TO ?    | 2015-9005  | TO/TA ✓ |
| 2015-9000  | TA ✓ | 2015-9198  | TO ✓ | 2017-18315 | TO ✓    | 2015-9007  | TO/TA ✓ |
| 2015-9002  | TA ✓ | 2015-9199  | TO ✓ | 2017-6290  | TO ✓    | 2016-10297 | TO/TA ✓ |
| 2015-9162  | TA ✓ | 2015-9200  | TO ✓ | 2017-6292  | TO ✓    | 2016-2432  | TO/TA ✓ |
| 2015-9174  | TA ✓ | 2016-10238 | TO ✓ | 2017-6294  | TO ✓    | 2017-14913 | TO/TA ✓ |
| 2015-9183  | TA ✓ | 2016-10239 | TO ✓ | 2017-8274  | TO ✓    | 2017-18293 | TO/TA ✓ |
| 2017-18310 | TA — | 2016-10432 | TO ✓ | 2018-11950 | TO ✓    | 2017-18296 | TO/TA ? |
| 2017-18312 | TA ? | 2016-2431  | TO ✓ | 2018-3588  | TO ✓    | 2017-18297 | TO/TA ✓ |
| 2017-18317 | TA ? | 2017-11010 | TO ✓ | 2018-5870  | TO ✓    | 2017-18298 | TO/TA ✓ |
| 2017-6293  | TA ✓ | 2017-11011 | TO ✓ | 2014-9932  | TO/TA ✓ | 2017-6289  | TO/TA ✓ |
| 2018-5210  | TA ✓ | 2017-14912 | TO ✓ | 2014-9935  | TO/TA ✓ | 2018-5866  | TO/TA ✓ |
| 2018-5885  | TA ✓ | 2017-14916 | TO ✓ | 2014-9936  | TO/TA ✓ | 2017-18282 | HW ✓    |
| 2014-9979  | TO ✓ | 2017-14917 | TO ✓ | 2014-9937  | TO/TA ✓ | 2015-9003  | CI —    |
| 2015-8999  | TO ✓ | 2017-17176 | TO ✓ | 2014-9945  | TO/TA ✓ | 2016-10398 | CI —    |
| 2015-9070  | TO ✓ | 2017-18071 | TO ✓ | 2014-9948  | TO/TA ✓ | 2017-14907 | CI —    |
| 2015-9071  | TO ✓ | 2017-18128 | TO — | 2014-9949  | TO/TA ✓ | 2017-18146 | CI —    |
| 2015-9072  | TO ✓ | 2017-18129 | TO ✓ | 2015-8995  | TO/TA ✓ | 2016-10458 | BL ✓    |
| 2015-9073  | TO ✓ | 2017-18132 | TO ✓ | 2015-8996  | TO/TA ✓ | 2017-14911 | BL —    |

Table 14: Mitigation analysis of selected CVEs with critical CVSS scores: "✓- CVEs in scope; "— out of scope; "?- insufficient information.

kept confidential even at rest. The TREE framework enables system designers to encrypt their TAs using symmetric cryptography (AES). During secure loading, the encrypted TA is decrypted block by block, with each block sent directly to the ICAP for configuration, avoiding temporary storage of plaintext TAs and reducing exposure risk. Cryptographic keys required for encryption and decryption are securely managed by a hardware security module (HSM), which provides a trusted environment for key generation, storage, and usage. This mechanism protects against threats, e.g., unauthorized memory access, plaintext data exposure, and key extraction attempts.

## 7.9.2 CVE mitigation analysis

We assess the potential of deploying TREE in real-world systems by analyzing its practical effectiveness in mitigating threats associated with TEE-related vulnerabilities. Our analysis relies on the CVE list compiled by Cerdeira et al. [265], which provides an extensive collection of TEE-related CVEs. The goal is to understand how TREE's security features, e.g., isolation mechanisms and data protection strategies, directly address these threats. The analysis, summarized in Table 14, focuses on 80 critical CVEs and categorizes the CVEs by the affected components: Trusted OS (TO), Trusted Application (TA), Cryptographic Implementation (CI), Hardware (HW), and Bootloader (BL).

**Mitigation of CVEs within scope.** TREE addresses 69 of the 80 CVEs analyzed. Most of these vulnerabilities target the trusted OS or TAs, with the potential for privilege escalation attacks. Since TREE meets the security requirement *R2*—isolate each TA execution environment—by ensuring that each TA executes within its own dedicated tile, which includes a unique physical address space, TAs do not share resources with one another nor with the TrustShell, significantly reducing the potential risk of privilege escalation attacks. Additionally, TREE addresses vulnerabilities, e.g., CVE-2016-10458, which exploits improper memory handling between the REE and TEE. By meeting the security requirement *R1*—isolating

|                     |               |           | TEE management |    |    |       | TA compatibility |      |       | TEE Services |       |     |
|---------------------|---------------|-----------|----------------|----|----|-------|------------------|------|-------|--------------|-------|-----|
| RoT                 |               |           | dENCLV         | EC | ER | TAcrr | hTAs             | sTAs | GPTAs | SS           | CRYPT | ATT |
| CPU-driven TEEs     | HISA [41]     | TrustZone | ○              | ○  | ○  | ○     | ●                | ●    | ○     | ●            | ○     | ○   |
|                     | Ambassy [181] | TrustZone | ●              | ○  | ○  | ●     | ●                | ○    | ○     | ●            | ●     | ●   |
| FPGA TEEs           | MeetGo [184]  | FPGA      | ●              | ○  | ○  | ○     | ●                | ●    | ○     | ○            | ○     | ●   |
|                     | ShEF [52]     | FPGA      | ●              | ○  | ○  | ○     | ●                | ○    | ○     | ●            | ●     | ●   |
| Soft MCU TEEs       | BYOTee [186]  | SoC-FPGA  | ●              | ●  | ●  | ●     | ○                | ●    | ○     | ○            | ○     | ●   |
|                     | TEEOD [208]   | SoC-FPGA  | ●              | ●  | ●  | ●     | ○                | ●    | ●     | ●            | ●     | ○   |
| Reconfigurable TEEs | TREE          | SoC-FPGA  | ●              | ●  | ●  | ●     | ●                | ●    | ●     | ●            | ●     | ●   |

Table 15: FPGA-based TEEs state of the art comparison. Key: RoT - Root-of-trust Primitive; dE - Dedicated Execution Environment per TA; EC - Execution Environment clean-up; ER - Execution Environment reuse; TAC - TAs Concurrency; hTAs - Support for hardware TAs; sTAs - Support for software TAs; GPTAs - GP-compliant TAs; SS - Secure Storage; CRYPT - Encryption services; ATT - Verification/attestation services; ● - fares positively; ● - fares partially positively; ○ - fares negatively.

the FPGA fabric from the REE—TREE ensures a loosely coupled interaction between the REE and TEE, avoiding the need for trusted memory allocation in the same address space as the REE OS.

**CVEs out of scope.** Out of the 80 CVEs analyzed, 7 CVEs fall outside TREE’s scope. For instance, CVE-2017-18310 involves a TA exposing unnecessary services to the REE, potentially allowing malicious actions by CAs. While TREE does not control which services are exposed, it enforces strict isolation between TAs, as required by security requirement R2, minimizing the impact of such exposures. TREE also does not address hardware misconfiguration, e.g., CVE-2017-18128, that could result in sensitive data leakage. Since TREE relies on the boot configuration to protect against FPGA readback access to the REE, this vulnerability is also excluded from its scope. Bootloader authentication issues are also unaddressed by TREE, e.g., CVE-2017-14911, as TREE relies on the platform’s secure boot process for correct system initialization, it does not address this type of vulnerability either. Cryptographic vulnerabilities, e.g., CVE-2017-18146, are not considered, as the framework assumes the proper implementation of cryptographic algorithms and primitives. However, TREE’s flexible architecture, with its FPGA-based TEE, could potentially address these vulnerabilities by enabling over-the-air updates for cryptographic modules, allowing prompt remediation of any weaknesses in cryptographic protocols.

## 7.10 Related Work

We compare TREE with other FPGA-based TEE solutions. Table 15 summarizes our findings, focusing on the critical features of TEE design, including the underlying TEE technology, management capabilities, TA compatibility, and TEE services. We also classify these works into three FPGA-based TEE categories: i) extended CPU-driven (CPU-driven TEEs), ii) secure reconfiguration TEEs (FPGA TEEs), and iii) soft MCU-enabled TEEs (Soft MCU TEEs).

**Extended CPU-driven TEEs.** Well-established processor-built-in TEEs, such as Arm TrustZone [56], have been extended to support reconfigurable computing. Solutions such as HISA [41] and Embassy [181] use TrustZone capabilities to ensure secure enclave operations. They establish an additional trusted environment on FPGA by leveraging TrustZone’s secure world to oversee the secure loading of TAs in bitstream form onto the FPGA. However, this approach inherits inherent microarchitectural vulnerabilities in the underlying hardware’s TEE architecture [20, 30, 229, 265].

**Secure reconfiguration TEEs.** MeetGo [184] and ShEF [52] propose mechanisms to load custom hardware designs onto a remote FPGA securely and independently of the host system architecture. However, both present limited TA support and lack TEE features such as secure storage. In MeetGo, TA sources are converted to HDL code through high-level synthesis (HLS) tools and mapped onto the FPGA as a partial bitstream, requiring users to translate software applications into hardware logic. In contrast, TREE’s secure hardware TA development mechanism eliminates the need for software-to-hardware translation, enabling a more seamless integration for existing applications without compromising interoperability or legacy support for existing TAs.

**Soft MCU-enabled TEEs.** TEEOD [208] and BYOTee [186] synthesize soft-core MCUs on an SoC-FPGA to provide secure environments for software TAs to run isolated. TEEOD even complies with the GP TEE specifications. The difference is in the TA cleanup, loading, and management method; BYOTee relies on the firmware running on the soft-MCU and so depends on the enclave’s state, while TEEOD offloads these services to dedicated hardware accelerators that are independent of the enclave’s state. However, the hardware for executing TAs is allocated at development-time, with no support for dynamic hardware swapping. Any hardware change on the TAs requires rebooting the entire system. As a result, the framework does not support hardware-based TAs, limiting its ability to fully leverage the potential of reconfigurable computing. In contrast, TREE stands out by leveraging DPR technology, commonly found in SoC-FPGAs, to enable loading of TAs’ hardware without requiring system reboots, while ensuring their integrity.

## 7.11 Conclusion

In this work, we introduce TREE, a novel FPGA-based TEE framework that leverages conventional FPGA mechanisms to enhance current FPGA-based TEEs. Our evaluation reveals minimal overhead, less than 1%, compared to native hardware applications. Additionally, TREE demonstrates significant speedups for TAs optimized for FPGA fabric, while also maintaining compatibility with legacy software TAs. We also demonstrate TREE’s ability to handle real-world scenarios, such as access control authentication and Bitcoin wallets without notable performance degradation and, in some cases, speedups.

## Bibliography

- [1] S. Al-Sarawi et al. "Internet of Things Market Analysis Forecasts, 2020–2030". In: *WorldS4*. 2020. doi: 10.1109/WorldS450073.2020.9210375 (cit. on p. 2).
- [2] F. Al-Turjman and M. Abujubbeh. "IoT-enabled smart grid via SM: An overview". In: *Future generation computer systems* (2019) (cit. on p. 2).
- [3] M. H. Kashani et al. "A systematic review of IoT in healthcare: Applications, techniques, and trends". In: *Journal of Network and Computer Applications* (2021) (cit. on p. 2).
- [4] M. Fahlevi, A. Purnomo, et al. "The integration of internet of things (IoT) and cloud computing in finance and accounting: systematic literature review". In: *ICBIR*. IEEE. 2023 (cit. on p. 2).
- [5] H. Khaleel et al. "Heterogeneous applications, tools, and methodologies in the car manufacturing industry through an IoT approach". In: *Systems journal* (2015) (cit. on p. 2).
- [6] S. Pinto et al. "IIoTEED: An enhanced, trusted execution environment for industrial IoT edge devices". In: *IEEE Internet Computing* (2017) (cit. on p. 2).
- [7] P. Schaumont. "Security in the Internet of Things: A challenge of scale". In: *DATE*. IEEE. 2017 (cit. on pp. 2, 4).
- [8] H. Ghanbari and K. Koskinen. "When data breach hits a psychotherapy clinic: the Vastaamo case". In: *Journal of Information Technology Teaching Cases* (2024) (cit. on p. 2).
- [9] D. Oliveira et al. "Investigating and mitigating contention on low-end multi-core microcontrollers". In: *Proceedings of Cyber-Physical Systems and Internet of Things Week*. ACM, 2023 (cit. on p. 2).
- [10] G. Shirvani and S. Ghasemshirazi. "Towards Sustainable IoT: Challenges, Solutions, and Future Directions for Device Longevity". In: *arXiv preprint arXiv:2405.16421* (2024) (cit. on p. 2).
- [11] G. Suh et al. "Efficient memory integrity verification and encryption for secure processors". In: *Proc. in IEEE/ACM MICRO*. 2003. doi: <https://doi.org/10.1109/MICRO.2003.1253207> (cit. on pp. 2, 102).

- 
- [12] AMD. *AMD Versal AI Edge Series Adaptive SoCs*. <https://www.amd.com/en/products/adaptive-socs-and-fpgas/versal/ai-edge-series.html>. Accessed: 2025-06-27. 2023 (cit. on pp. 3, 14, 57).
  - [13] J. Park, Y. Park, and S. Mahlke. "Chimera: Collaborative Preemption for Multitasking on a Shared GPU". In: *Proc. in ASPLOS* (2015). url: <https://api.semanticscholar.org/CorpusID:7937269> (cit. on pp. 3, 102).
  - [14] A. Al-Omary et al. "Survey of hardware-based security support for IoT/CPS systems". In: *KnE Engineering* (2018) (cit. on p. 3).
  - [15] E. M. Benhani et al. "On the security evaluation of the ARM TrustZone extension in a heterogeneous SoC". In: *2017 30th IEEE International System-on-Chip Conference (SOCC)*. 2017, pp. 108–113. doi: 10.1109/SOCC.2017.8226018 (cit. on pp. 3, 27, 45, 58, 61, 95, 98, 102).
  - [16] H. Naghibijouybari et al. "Rendered Insecure: GPU Side Channel Attacks are Practical". In: *Proceedings of the 2018 ACM SIGSAC. CCS '18*. ACM, 2018. doi: 10.1145/3243734.3243831 (cit. on p. 3).
  - [17] M. Gross et al. "Breaking TrustZone Memory Isolation through Malicious Hardware on a Modern FPGA-SoC". In: *Proceedings of the 3rd ACM Workshop on Attacks and Solutions in Hardware Security Workshop. ASHES'19*. ACM, 2019. url: <https://doi.org/10.1145/3338508.3359568> (cit. on pp. 3, 4).
  - [18] Y. Xiang et al. "Open DNN Box by Power Side-Channel Attack". In: *IEEE Transactions on Circuits and Systems II: Express Briefs* 67.11 (2020), pp. 2717–2721. doi: 10.1109/TCSII.2020.2973007 (cit. on p. 3).
  - [19] H. Ragab et al. "CrossTalk: Speculative Data Leaks Across Cores Are Real". In: *2021 IEEE Symposium on Security and Privacy (SP)*. 2021, pp. 1852–1867. doi: 10.1109/SP40001.2021.00020 (cit. on p. 3).
  - [20] P. Kocher et al. "Spectre Attacks: Exploiting Speculative Execution". In: *2019 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2019, pp. 1–19. doi: 10.1109/SP.2019.000002 (cit. on pp. 3, 6, 73, 125).
  - [21] M. Lipp et al. "Meltdown: Reading Kernel Memory from User Space". In: *27th USENIX Security Symposium (USENIX Security 18)*. USENIX Association, 2018, pp. 973–990. url: <https://www.usenix.org/conference/usenixsecurity18/presentation/lipp> (cit. on pp. 3, 6).
  - [22] S. Bhunia, S. Ray, and A. Basak. *To Secure the Internet of Things, We Must Build It Out of "Patchable" Hardware*. Accessed: 2025-06-27. 2017. url: <https://spectrum.ieee.org/to-secure-the-internet-of-things-we-must-build-it-out-of-patchable-hardware> (cit. on pp. 3, 6).

- [23] D. G. Mahmoud, V. Lenders, and M. Stojilović. “Electrical-Level Attacks on CPUs, FPGAs, and GPUs: Survey and Implications in the Heterogeneous Era”. In: *ACM Comput. Surv.* 55.3 (2022). issn: 0360-0300. doi: 10.1145/3498337. url: <https://doi.org/10.1145/3498337> (cit. on pp. 4, 5, 54).
- [24] Y. Gao and P. Zhang. “A survey of homogeneous and heterogeneous system architectures in high performance computing”. In: *SmartCloud*. IEEE. 2016 (cit. on p. 4).
- [25] Y. Choi et al. “A quantitative analysis on microarchitectures of modern CPU-FPGA platforms”. In: *ACM/ EDAC/ IEEE DAC*. 2016. url: <https://doi.org/10.1145/2897937.2897972> (cit. on pp. 4, 101).
- [26] H. Ning et al. “Heterogeneous edge computing open platforms and tools for internet of things”. In: *Future Generation Computer Systems* (2019). url: <https://doi.org/10.1016/j.future.2019.12.036> (cit. on pp. 4, 19, 101).
- [27] A. Koshiba et al. “Trusted Heterogeneous Disaggregated Architectures”. In: *ACM APSys*. 2023. url: <https://doi.org/10.1145/3609510.3609812> (cit. on p. 4).
- [28] J. Zhu et al. “Enabling Rack-scale Confidential Computing using Heterogeneous Trusted Execution Environment”. In: *Proc. in IEEE S&P*. 2020. doi: 10.1109/SP40000.2020.00054 (cit. on pp. 4, 40, 43, 54, 65–67, 101, 105).
- [29] A. Muñoz et al. “A survey on the (in)security of trusted execution environments”. In: *Computers & Security* (2023). doi: <https://doi.org/10.1016/j.cose.2023.103180> (cit. on pp. 4, 54, 105).
- [30] D. Cerdeira et al. “SoK: Understanding the Prevailing Security Vulnerabilities in TrustZone-assisted TEE Systems”. In: *Proc. in IEEE S&P*. 2020. url: <https://doi.org/10.1109/SP40000.2020.00061> (cit. on pp. 4, 5, 25–27, 44, 54, 59, 68, 70, 73, 74, 76, 77, 90, 94, 97, 102, 106, 125).
- [31] S. Trimberger and J. Moore. “FPGA Security: Motivations, Features, and Applications”. In: *IEEE Access* (2014) (cit. on pp. 4, 17, 77, 102, 104).
- [32] Taylor, B. “State-of-the-Art Programmable Logic”. In: *Designing with Xilinx® FPGAs: Using Vivado*. Springer International Publishing, 2017, 1–15. url: [https://doi.org/10.1007/978-3-319-42438-5\\_1](https://doi.org/10.1007/978-3-319-42438-5_1) (cit. on pp. 4, 104).
- [33] T. El-Ghazawi et al. “The Promise of High-Performance Reconfigurable Computing”. In: *Computer* (2008). url: <https://doi.org/10.1109/MC.2008.65> (cit. on pp. 4, 101).
- [34] S. Biokaghazadeh, M. Zhao, and F. Ren. “Are {FPGAs} suitable for edge computing?” In: *USENIX workshop on hot topics in edge computing (HotEdge 18)*. 2018 (cit. on p. 4).
- [35] H. Sharma et al. “From high-level deep neural models to FPGAs”. In: *Proc. in IEEE/ACM MICRO*. 2016 (cit. on p. 4).

- 
- [36] E. Monmasson et al. "FPGAs in Industrial Control Applications". In: *Proc. in IEEE Trans. Ind. Electron.* 2011 (cit. on p. 4).
  - [37] E. Monmasson and M. Cirstea. "FPGA Design Methodology for Industrial Control Systems—A Review". In: *Proc. in IEEE Trans. Ind. Electron.* 2007 (cit. on p. 4).
  - [38] E. Hartley et al. "Predictive Control Using an FPGA With Application to Aircraft Control". In: *IEEE Trans. on Control Systems Technology* (2014) (cit. on p. 4).
  - [39] C. Bobda et al. "The future of FPGA acceleration in datacenters and the cloud". In: *ACM Transactions on Reconfigurable Technology and Systems (TRETs)* 15.3 (2022), pp. 1–42 (cit. on pp. 4, 18, 53).
  - [40] R. Cofer and B. F. Harding. *Rapid System Prototyping with FPGAs: Accelerating the Design Process*. Elsevier, 2006 (cit. on pp. 4, 17).
  - [41] M. Ye, X. Feng, and S. Wei. "HISA: Hardware Isolation-based Secure Architecture for CPU-FPGA Embedded Systems". In: *IEEE/ACM ICCAD*. 2018, pp. 1–8. doi: 10.1145/3240765.3240814 (cit. on pp. 4, 35, 36, 43–45, 54, 65, 66, 101, 124, 125).
  - [42] M. Doulcier-Verdier et al. "A Side-Channel and Fault-Attack Resistant AES Circuit Working on Duplicated Complemented Values". In: *Proceedings of the 2011 IEEE International Solid-State Circuits Conference (ISSCC)*. IEEE, 2011, pp. 274–276. doi: 10.1109/ISSCC.2011.5746316 (cit. on pp. 4, 30, 43, 54, 62).
  - [43] E. Owusu et al. "OASIS: On Achieving a Sanctuary for Integrity and Secrecy on Untrusted Platforms". In: *Proc. in ACM CCS*. 2013 (cit. on p. 4).
  - [44] C. Lesjak, D. Hein, and J. Winter. "Hardware-security technologies for industrial IoT: TrustZone and security controller". In: *Proc. in IEE IECON*. 2015 (cit. on pp. 4, 76).
  - [45] M. Zhao and G. E. Suh. "FPGA-based remote power side-channel attacks". In: *2018 IEEE symposium on security and privacy (SP)*. IEEE. 2018, pp. 229–244 (cit. on pp. 4, 21, 53, 60).
  - [46] F. Schellenberg et al. "An inside job: Remote power analysis attacks on FPGAs". In: *IEEE Design & Test* 38.3 (2021), pp. 58–66 (cit. on pp. 4, 21, 53, 60).
  - [47] C. Ramesh et al. "FPGA side channel attacks without physical access". In: *2018 IEEE 26th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*. IEEE. 2018, pp. 45–52 (cit. on pp. 4, 21, 53, 60).
  - [48] S. Mirzargar and M. Stojilović. "Physical Side-Channel Attacks and Covert Communication on FPGAs: A Survey". In: *FPL*. 2019. doi: 10.1109/FPL.2019.00039 (cit. on pp. 4, 5, 122).
  - [49] E. Benhani, L. Bossuet, and A. Aubert. "The Security of ARM TrustZone in a FPGA-Based SoC". In: *IEEE Transactions on Computers* (2019) (cit. on pp. 4, 5, 27, 45, 53, 58, 61, 74).

- [50] G. Dessouky, A.-R. Sadeghi, and S. Zeitouni. “SoK: Secure FPGA Multi-Tenancy in the Cloud: Challenges and Opportunities”. In: *EuroS&P*. IEEE, 2021. url: <https://ieeexplore.ieee.org/document/9581180/> (cit. on pp. 5, 18, 54).
- [51] S. Zeitouni et al. “Trusted Configuration in Cloud FPGAs”. In: *IEEE FCCM*. 2021. doi: 10.1109/FCCM51124.2021.00036 (cit. on pp. 5, 36, 43–45, 54, 66, 96, 97).
- [52] M. Zhao, M. Gao, and C. Kozyrakis. “ShEF: shielded enclaves for cloud FPGAs”. In: *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*. ASPLOS '22. New York, NY, USA: Association for Computing Machinery, 2022, pp. 1070–1085. isbn: 9781450392051. doi: 10.1145/3503222.3507733. url: <https://doi.org/10.1145/3503222.3507733> (cit. on pp. 5, 37, 43, 45, 54, 67, 74, 77, 96–98, 102, 124, 125).
- [53] N. Timmers and A. Spruyt. “Bypassing secure boot using fault injection”. In: *Black Hat Europe 2016* (2016) (cit. on pp. 5, 20, 30, 43, 54, 59, 62).
- [54] S. Johnson et al. “Intel software guard extensions: EPID provisioning and attestation services”. In: *White Paper 1.1-10* (2016), p. 119 (cit. on pp. 5, 23, 33, 76, 97, 102).
- [55] AMD. *AMD Secure Encrypted Virtualization*. 2019. url: <https://www.amd.com/en/developer/sev.html> (cit. on pp. 5, 23, 102).
- [56] Arm. *ARM Security Technology: Building a Secure System using TrustZone Technology*. White Paper. Arm, 2009 (cit. on pp. 5, 23, 58, 73, 76, 97, 102, 125).
- [57] R. Sahita et al. “CoVE: Towards confidential computing on RISC-V platforms”. In: *Proceedings of the 20th ACM International Conference on Computing Frontiers*. 2023, pp. 315–321 (cit. on pp. 5, 23, 33).
- [58] G. Beniamini. *TrustZone Kernel Privilege Escalation (CVE2016-2431)*. <http://bits-please.blogspot.com/2016/06/trustzone-kernel-privilege-escalation.html>. 2016 (cit. on pp. 5, 76, 79).
- [59] G. Beniamini. *QSEE privilege escalation vulnerability and exploit*. <http://bits-please.blogspot.com/2016/05/qsee-privilege-escalation-vulnerability.html>. 2016 (cit. on pp. 5, 76, 79).
- [60] G. Beniamini. *Extracting Qualcomm's KeyMaster Keys Breaking Android Full Disk Encryption*. <https://bits-please.blogspot.com/2016/06/extracting-qualcomms-keymaster-keys.html>. 2016 (cit. on pp. 5, 76, 79, 107).
- [61] F. Brasser et al. “Software Grand Exposure: SGX Cache Attacks Are Practical”. In: *Proc. in USENIX WOOT*. 2017 (cit. on pp. 5, 73).

- [62] A. Moghimi, G. Irazoqui, and T. Eisenbarth. “CacheZoom: How SGX Amplifies the Power of Cache Attacks”. In: *International Conference on Cryptographic Hardware and Embedded Systems (CHES)*. 2017, pp. 69–90 (cit. on p. 5).
- [63] D. Moghimi et al. “CopyCat: Controlled Instruction-Level Attacks on Enclaves”. In: *29th USENIX Security Symposium*. 2020, pp. 469–486 (cit. on p. 5).
- [64] J. V. Bulck, F. Piessens, and R. Strackx. “SGX-Step: A practical attack framework for precise enclave execution control”. In: *Proceedings of the 2nd Workshop on System Software for Trusted Execution*. 2017, pp. 1–6 (cit. on p. 5).
- [65] S. Pereira et al. “Beneath the fabric: A survey on threats and opportunities in reconfigurable systems”. In: *Submitted to Journal of Systems Architecture* (2025) (cit. on pp. 8, 51).
- [66] S. Pereira et al. “Rethinking Trusted Execution Environments in the Age of Reconfigurable Computing”. In: *Submitted to IEEE Access* (2025) (cit. on pp. 8, 51).
- [67] S. Pereira et al. “TREE: Bridging the gap between reconfigurable computing and secure execution”. In: *IACR Transactions on Cryptographic Hardware and Embedded Systems* 2025.3 (2025), pp. 115–138 (cit. on pp. 8, 51, 54, 65, 66, 68, 70).
- [68] V. Jyothi et al. “FPGA Trust Zone: Incorporating trust and reliability into FPGA designs”. In: *Proc. in IEEE ICCD*. 2016 (cit. on pp. 12, 74).
- [69] F. Turan and I. Verbauwhede. “Trust in FPGA-accelerated Cloud Computing”. In: *ACM Comput. Surv.* (2021). url: <https://dl.acm.org/doi/10.1145/3419100> (cit. on pp. 12, 54, 74).
- [70] M. Will and R. Ko. “Secure FPGA as a Service — Towards Secure Data Processing by Physicalizing the Cloud”. In: *Proc. in IEEE Trustcom/BigDataSE/ICCESS*. 2017 (cit. on pp. 12, 74).
- [71] P. Yiannacouras, J. Rose, and J. G. Steffan. “The microarchitecture of FPGA-based soft processors”. In: *Proceedings of the 2005 International Conference on Compilers, Architectures and Synthesis for Embedded Systems. CASES '05*. New York, NY, USA: Association for Computing Machinery, 2005, pp. 202–212. isbn: 159593149X. doi: 10.1145/1086297.1086325. url: <https://doi.org/10.1145/1086297.1086325> (cit. on p. 12).
- [72] E. Matthews and L. Shannon. “TAIGA: A new RISC-V soft-processor framework enabling high performance CPU architectural features”. In: *Proc. in FPL*. 2017 (cit. on pp. 13, 77).
- [73] A. Xilinx. *MicroBlaze Soft Processor Core*. <https://www.xilinx.com/products/design-tools/microblaze.html> (cit. on pp. 13, 77, 104).
- [74] Gaisler. *LEON3 Processor*. <https://www.gaisler.com/index.php/products/processors/leon3> (cit. on pp. 13, 77).
- [75] OpenRISC. *OPENRISC Project Overview*. <https://openrisc.io/> (cit. on pp. 13, 77).

- [76] Intel. *Nios II Classic Processor Reference Guide*. <https://www.intel.com/content/www/us/en/docs/programmable/683620/current/nios-ii-processor-system-basics.html> (cit. on pp. 13, 77).
- [77] Arm. *Cortex-M1 Technical Reference Manual*. <https://documentation-service.arm.com/static/5e8e18c2fd977155116a3d48> (cit. on pp. 13, 77).
- [78] AMD. *PicoBlaze 8-bit Microcontroller*. <https://www.xilinx.com/products/intellectual-property/picoblaze.html> (cit. on pp. 13, 77).
- [79] Lattice. *LatticeMico8 Processor Reference Manual*. <https://www.latticesemi.com/media/LatticeSemi/Documents/UserManuals/JL/LatticeMico8ProcessorReferenceManual35.pdf> (cit. on pp. 13, 77).
- [80] AMD-Xilinx. *UG585 Zynq-7000 SoC Technical Reference Manual v1.12.2*. Retrieved January 30, 2022 from [https://www.xilinx.com/support/documentation/user\\_guides/ug585-Zynq-7000-TRM.pdf](https://www.xilinx.com/support/documentation/user_guides/ug585-Zynq-7000-TRM.pdf). AMD-Xilinx. 2019 (cit. on pp. 13, 15).
- [81] AMD. *AMD Zynq UltraScale+ MPSoCs Family Reference Guide*. <https://www.amd.com/en/products/adaptive-socs-and-fpgas/soc/zynq-ultrascale-plus-mpsoc.html>. 2024 (cit. on pp. 15, 57, 58, 104).
- [82] T. Lu, R. Kenny, and S. Atsatt. *Secure Device Manager for Intel® Stratix®10 Devices Provides FPGA and SoC Security*. <https://cdrdv2-public.intel.com/650483/wp-01252-secure-device-manager-for-fpga-soc-security.pdf>. White paper. Accessed June 2025. 2025 (cit. on pp. 15, 57, 58).
- [83] AMD. *Vivado Design Suite User Guide: Partial Reconfiguration (UG909)*. <https://docs.amd.com/r/en-US/ug909-vivado-partial-reconfiguration>. Accessed June 2025. 2023 (cit. on pp. 16, 17).
- [84] K. Vipin and S. A. Fahmy. "FPGA Dynamic and Partial Reconfiguration: A Survey of Architectures, Methods, and Applications". In: *ACM Comput. Surv.* (2019), pp. 1–39. doi: 10.1145/3193827. url: <https://dl.acm.org/doi/10.1145/3193827> (visited on 2025-07-08) (cit. on pp. 17, 57, 104).
- [85] Intel Corporation. *Quartus® Prime Pro Edition User Guide: Partial Reconfiguration*. <https://www.intel.com/content/www/us/en/docs/programmable/683090>. Accessed June 2025. 2024 (cit. on p. 17).
- [86] A. M. Caulfield et al. "A Cloud-Scale Acceleration Architecture". In: *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 2016. url: <https://www.microsoft.com/en-us/research/wp-content/uploads/2016/10/Cloud-Scale-Acceleration-Architecture.pdf> (cit. on p. 18).

- 
- [87] M. Hamilton and et al. "Implementation of a Secure TLS Coprocessor on an FPGA". In: *Computer Communications* (2016). doi: 10.1016/j.comcom.2015.12.016. url: <https://www.sciencedirect.com/science/article/abs/pii/S0141933115001647> (cit. on p. 18).
  - [88] M. Vajaranta and et al. "Feasibility of FPGA-Accelerated IPsec on Cloud". In: *Computer Communications* (2019). doi: 10.1016/j.comcom.2018.12.019. url: <https://www.sciencedirect.com/science/article/abs/pii/S014193311830512X> (cit. on p. 18).
  - [89] J. Lu and J. W. Lockwood. "IPSec Implementation on Xilinx Virtex-II Pro FPGA and Its Performance Analysis". In: *RAW 2005*. 2005. url: <https://www.arl.wustl.edu/Publications/2005-09/raw05.pdf> (cit. on p. 18).
  - [90] M. Blott et al. "Achieving 10Gbps Line-Rate Key-Value Stores with FPGAs". In: *USENIX HotCloud*. 2013. url: <https://www.usenix.org/system/files/conference/hotcloud13/hotcloud13-blott.pdf> (cit. on p. 18).
  - [91] Z. István et al. "A Flexible Hash Table Design for 10Gbps Key-Value Stores on FPGAs". In: *Proceedings of the ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA)*. 2013. url: [https://zistvan.github.io/doc/paperM3C\\_3.pdf](https://zistvan.github.io/doc/paperM3C_3.pdf) (cit. on p. 18).
  - [92] N. T. T. Thuy and et al. "A Fast Approach for Bitcoin Blockchain Cryptocurrency Based on a Two-Level Pipeline Hardware Architecture for SHA-256". In: *Journal of Systems Architecture* (2020). doi: 10.1016/j.sysarc.2020.101804. url: <https://www.sciencedirect.com/science/article/abs/pii/S0167926020300146> (cit. on p. 18).
  - [93] Y. Sakakibara et al. "Accelerating Blockchain Transfer System Using FPGA-Based NIC". In: *IEEE International Symposium on Parallel and Distributed Processing with Applications (ISPA)*. 2018. url: [https://www.arc.ics.keio.ac.jp/~matutani/papers/sakakibara\\_ispa2018.pdf](https://www.arc.ics.keio.ac.jp/~matutani/papers/sakakibara_ispa2018.pdf) (cit. on p. 18).
  - [94] D. Lee et al. "mIoT: Metamorphic IoT Platform for On-Demand Hardware Replacement in Large-Scaled IoT Applications". In: *Sensors* 20.12 (2020), p. 3337. doi: 10.3390/s20123337 (cit. on p. 19).
  - [95] A. R. Bucknall and S. A. Fahmy. "ZyPR: End-to-End Build Tool and Runtime Manager for Partial Reconfiguration of FPGA SoCs at the Edge". In: *ACM Transactions on Reconfigurable Technology and Systems* (2023). doi: 10.1145/3585521 (cit. on p. 19).
  - [96] M. Qasaimeh et al. "Comparing Energy Efficiency of CPU, GPU and FPGA Implementations for Vision Kernels". In: *arXiv preprint arXiv:1906.11879* (2019) (cit. on p. 19).
  - [97] A. Magyari and Y. Chen. "Review of state-of-the-art FPGA applications in IoT Networks". In: *Sensors* 22.19 (2022), p. 7496 (cit. on pp. 19, 53).

- [98] Q. A. Ahmed, T. Wiersema, and M. Platzner. “Malicious routing: Circumventing bitstream-level verification for FPGAs”. In: *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE. 2021, pp. 1490–1495 (cit. on pp. 20, 58, 59).
- [99] Z. Zhang et al. “Thwarting security threats from malicious FPGA tools with novel FPGA-oriented moving target defense”. In: *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 27.3 (2018), pp. 665–678 (cit. on pp. 20, 59).
- [100] MITRE Corporation. *CVE-2021-27208: Improper Access Control in Xilinx Vivado Hardware Server*. <https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2021-27208>. Access control vulnerability in Xilinx Vivado Hardware Server could allow an attacker to gain unauthorized access or cause denial of service through crafted requests. 2021 (cit. on pp. 20, 59).
- [101] P. Mishra, M. Tehranipoor, and S. Bhunia. “Security and trust vulnerabilities in third-party IPs”. In: *Hardware IP Security and Trust*. Springer, 2017, pp. 3–14 (cit. on pp. 20, 59).
- [102] D. R. Gnad, F. Oboril, and M. B. Tahoori. “Voltage drop-based fault attacks on FPGAs using valid bitstreams”. In: *2017 27th International Conference on Field Programmable Logic and Applications (FPL)*. IEEE. 2017, pp. 1–7 (cit. on pp. 21, 53, 60).
- [103] J. Krautter, D. R. Gnad, and M. B. Tahoori. “FPGAhammer: Remote voltage fault attacks on shared FPGAs, suitable for DFA on AES”. In: *IACR Transactions on Cryptographic Hardware and Embedded Systems* (2018), pp. 44–68 (cit. on pp. 21, 31, 53, 60, 63, 94, 96).
- [104] M. M. Alam et al. “RAM-Jam: Remote temperature and voltage fault attack on FPGAs using memory collisions”. In: *2019 Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC)*. IEEE. 2019, pp. 48–55 (cit. on pp. 21, 53, 60).
- [105] Y. Luo et al. “Deepstrike: Remotely-guided fault injection attacks on dnn accelerator in cloud-fpga”. In: *2021 58th ACM/IEEE Design Automation Conference (DAC)*. IEEE. 2021, pp. 295–300 (cit. on pp. 21, 53, 60).
- [106] J. Gravellier et al. “Remote side-channel attacks on heterogeneous SoC”. In: *International Conference on Smart Card Research and Advanced Applications*. Springer. 2019, pp. 109–125 (cit. on pp. 21, 53, 60, 96).
- [107] N. Jacob et al. “How to break secure boot on fpga socs through malicious hardware”. In: *International Conference on Cryptographic Hardware and Embedded Systems*. Springer. 2017, pp. 425–442 (cit. on pp. 21, 53, 60, 95, 98, 102).
- [108] Z. Weissman et al. “JackHammer: Efficient Rowhammer on Heterogeneous FPGA-CPU Platforms”. In: *IACR Transactions on Cryptographic Hardware and Embedded Systems* 2020.3 (2020), pp. 169–195. doi: 10.46586/tches.v2020.i3.169-195. url: <https://tches.iacr.org/index.php/TCHES/article/view/8595> (cit. on pp. 22, 53, 61).

- [109] Trustonic. *Trustonic Application Protection Delivers Comprehensive Security for Mobile Financial Services*. <https://www.trustonic.com/markets/financial-services/>. Accessed in 2025. 2018 (cit. on p. 22).
- [110] PallyCon. *About Google Widevine DRM*. <https://www.pallycon.com/google-widevine-drm/>. Accessed in 2025. 2019 (cit. on pp. 22, 26).
- [111] Android Developers. *Android Keystore System*. <https://developer.android.com/training/articles/keystore>. Accessed in 2025. 2020 (cit. on p. 22).
- [112] M. Schneider et al. "SoK: Hardware-Supported Trusted Execution Environments". In: *arXiv preprint arXiv:2205.12742* (2022). url: <https://arxiv.org/abs/2205.12742> (cit. on p. 23).
- [113] T. Alves and D. Felton. *TrustZone: Integrated Hardware and Software Security*. Tech. In-Depth. 2004 (cit. on pp. 23, 33).
- [114] Qualcomm. *Qualcomm Mobile Security*. <https://www.qualcomm.com/solutions/mobile-computing/features/security>. Accessed: 2025-10-07. 2018 (cit. on pp. 26, 34, 43).
- [115] Trustonic. *Trustonic Secured Platforms (TSP)*. <https://www.trustonic.com/solutions/trustonic-secured-platforms-tsp/>. Accessed: 2025-10-07. 2018 (cit. on pp. 26, 34, 43).
- [116] Huawei. *EMUI 10.1 Security Technical White Paper*. <https://consumer-img.huawei.com/content/dam/huawei-cbg-site/common/campaign/privacy/whitepaper/emui-10-security-technical-white-paper-v1.pdf>. Accessed: 2025-10-07. 2020 (cit. on pp. 26, 34, 43).
- [117] A. Tang, S. Sethumadhavan, and S. Stolfo. "CLKSCREW: Exposing the Perils of Security-Oblivious Energy Management". In: *Proc. in USENIX Security*. 2017 (cit. on pp. 28, 79, 107).
- [118] N. Zhang et al. *TruSpy: Cache Side-Channel Information Leakage from the Secure World on ARM Devices*. <https://eprint.iacr.org/2016/980>. IACR Cryptology ePrint Archive, Report 2016/980. 2016 (cit. on pp. 28, 79).
- [119] M. Lipp et al. "ARMageddon: Cache Attacks on Mobile Devices". In: *25th USENIX Security Symposium (USENIX Security 16)*. Austin, TX, USA: USENIX Association, 2016, pp. 549–564. url: <https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/lipp> (cit. on pp. 28, 73, 76, 79).
- [120] S. Lee et al. "Inferring Fine-grained Control Flow Inside SGX Enclaves with Branch Shadowing". In: *Proc. in USENIX Security*. 2017 (cit. on pp. 28, 79, 107).
- [121] P. Carru. *Attack ARM TrustZone using Rowhammer*. [https://grehack.fr/data/2017/slides/GreHack17\\_Attack\\_TrustZone\\_with\\_Rowhammer.pdf](https://grehack.fr/data/2017/slides/GreHack17_Attack_TrustZone_with_Rowhammer.pdf) (cit. on pp. 28, 79, 107).

- [122] T. Hoque et al. "Hidden in plaintext: An obfuscation-based countermeasure against FPGA bit-stream tampering attacks". In: *ACM Transactions on Design Automation of Electronic Systems (TODAES)* 25.1 (2019), pp. 1–32 (cit. on pp. 30, 43, 54, 62, 96).
- [123] S. Mukherjee et al. "A counter measure to prevent timing-based side-channel attack on FPGA". In: *2021 6th International Conference on Communication and Electronics Systems (ICCES)*. IEEE. 2021, pp. 983–988 (cit. on pp. 30, 43, 54, 62).
- [124] A. Le Masle, G. C. Chow, and W. Luk. "Constant power reconfigurable computing". In: *2011 International Conference on Field-Programmable Technology*. IEEE. 2011, pp. 1–8 (cit. on pp. 30, 43, 54, 62).
- [125] A. Dubey, R. Cammarota, and A. Aysu. "Maskednet: The first hardware inference engine aiming power side-channel protection". In: *2020 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*. IEEE. 2020, pp. 197–208 (cit. on pp. 30, 43, 54, 62).
- [126] P. Sasdrich et al. "Achieving side-channel protection with dynamic logic reconfiguration on modern FPGAs". In: *2015 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*. IEEE. 2015, pp. 130–136 (cit. on pp. 30, 43, 54, 62, 70).
- [127] P. Maistri and R. Leveugle. "Double-Data-Rate Computation as a Countermeasure Against Fault Analysis". In: *IEEE Transactions on Computers* 57.11 (2008), pp. 1528–1539. doi: 10.1109/TC.2008.149 (cit. on pp. 30, 43, 54, 62).
- [128] N. Mentens, B. Gierlichs, and I. Verbauwhede. "Power and fault analysis resistance in hardware through dynamic reconfiguration". In: *International Workshop on Cryptographic Hardware and Embedded Systems*. Springer. 2008, pp. 346–362 (cit. on pp. 30, 43, 54, 62).
- [129] W. He, J. Breier, and S. Bhasin. "Cheap and cheerful: A low-cost digital sensor for detecting laser fault injection attacks". In: *International Conference on Security, Privacy, and Applied Cryptography Engineering*. Springer. 2016, pp. 27–46 (cit. on pp. 31, 43, 54, 62).
- [130] W. He et al. "Ring Oscillator under Laser: Potential of PLL-based Countermeasure against Laser Fault Injection". In: *2016 Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC)*. 2016, pp. 102–113. doi: 10.1109/FDTC.2016.13 (cit. on pp. 31, 43, 54, 62).
- [131] W. He et al. "An FPGA-compatible PLL-based sensor against fault injection attack". In: *2017 22nd Asia and South Pacific Design Automation Conference (ASP-DAC)*. 2017, pp. 39–40. doi: 10.1109/ASPDAC.2017.7858291 (cit. on pp. 31, 43, 54, 62).
- [132] L. L. Shen, I. Ahmed, and V. Betz. "Fast Voltage Transients on FPGAs: Impact and Mitigation Strategies". In: *2019 IEEE 27th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*. 2019, pp. 271–279. doi: 10.1109/FCCM.2019.00044 (cit. on pp. 31, 54, 62).

- [133] S. Tajik et al. “PUFMon: Security monitoring of FPGAs using physically unclonable functions”. In: *2017 IEEE 23rd International Symposium on On-Line Testing and Robust System Design (IOLTS)*. 2017, pp. 186–191. doi: 10.1109/IOLTS.2017.8046216 (cit. on pp. 31, 43, 54, 62).
- [134] S. Drimer and M. G. Kuhn. “A protocol for secure remote updates of FPGA configurations”. In: *International Workshop on Applied Reconfigurable Computing*. Springer. 2009, pp. 50–61 (cit. on pp. 31, 43, 54, 63).
- [135] J. Vliegen, N. Mentens, and I. Verbauwhede. “Secure, remote, dynamic reconfiguration of FPGAs”. In: *ACM Transactions on Reconfigurable Technology and Systems (TRETS)* 7.4 (2014), pp. 1–19 (cit. on pp. 31, 43, 54, 63).
- [136] H. Kashyap and R. Chaves. “Compact and on-the-fly secure dynamic reconfiguration for volatile FPGAs”. In: *ACM Transactions on Reconfigurable Technology and Systems (TRETS)* 9.2 (2016), pp. 1–22 (cit. on pp. 31, 43, 54, 63).
- [137] J. Vliegen et al. “SACHa: Self-attestation of configurable hardware”. In: *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE. 2019, pp. 746–751 (cit. on pp. 31, 43, 54, 63).
- [138] Y. Yao et al. “Programmable ro (pro): A multipurpose countermeasure against side-channel and fault injection attack”. In: *Security of FPGA-Accelerated Cloud Computing Environments*. Springer, 2023, pp. 297–325 (cit. on pp. 31, 43, 54, 63).
- [139] A. Alanwar et al. “Dynamic fpga detection and protection of hardware trojan: A comparative analysis”. In: *arXiv preprint arXiv:1711.01010* (2017) (cit. on pp. 31, 43, 54, 63).
- [140] M. M. Ahmadi et al. “FPGA-Patch: Mitigating remote side-channel attacks on FPGAs using dynamic patch generation”. In: *2023 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*. IEEE. 2023, pp. 1–6 (cit. on pp. 31, 43, 54, 63).
- [141] F. Restuccia et al. “BASTION: A Framework for Secure Third-Party IP Integration in NoC-based SoC Platforms”. In: *IACR Transactions on Cryptographic Hardware and Embedded Systems* 2025.4 (2025), pp. 317–340 (cit. on pp. 32, 43, 63).
- [142] B. Tan et al. “Toward hardware-based IP vulnerability detection and post-deployment patching in systems-on-chip”. In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 40.6 (2020), pp. 1158–1171 (cit. on pp. 32, 43, 63).
- [143] A. Dharavathu and B. Tan. “Investigating the feasibility of efpga-based hardware patching”. In: *2024 IEEE 30th International Symposium on On-Line Testing and Robust System Design (IOLTS)*. IEEE. 2024, pp. 1–7 (cit. on pp. 32, 43, 64).
- [144] H. K.-H. So. *BORPH: An operating system for FPGA-based reconfigurable computers*. University of California, Berkeley, 2007 (cit. on pp. 32, 43, 54, 64).

- [145] A. Ismail and L. Shannon. “FUSE: Front-end user framework for O/S abstraction of hardware accelerators”. In: *2011 IEEE 19th annual international symposium on field-programmable custom computing machines*. IEEE. 2011, pp. 170–177 (cit. on pp. 32, 43, 54, 64).
- [146] A. Agne et al. “ReconOS: An operating system approach for reconfigurable computing”. In: *IEEE Micro* 34.1 (2013), pp. 60–71 (cit. on pp. 32, 43, 54, 64).
- [147] J. H. Kelm and S. S. Lumetta. “HybridOS: runtime support for reconfigurable accelerators”. In: *Proceedings of the 16th international ACM/SIGDA symposium on Field programmable gate arrays*. 2008, pp. 212–221 (cit. on pp. 32, 43, 54, 64).
- [148] J. Zhang et al. “The Feniks FPGA operating system for cloud computing”. In: *Proceedings of the 8th Asia-Pacific Workshop on Systems*. 2017, pp. 1–7 (cit. on pp. 32, 43, 54, 64).
- [149] K. Fleming and M. Adler. “The LEAP FPGA operating system”. In: *FPGAs for software programmers*. Springer, 2016, pp. 245–258 (cit. on pp. 32, 43, 54, 64).
- [150] A. Khawaja et al. “Sharing, protection, and compatibility for reconfigurable fabric with {AmorphOS}”. In: *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*. 2018, pp. 107–127 (cit. on pp. 32, 43, 54, 64).
- [151] A. Vaishnav et al. “FOS: A modular FPGA operating system for dynamic workloads”. In: *ACM Transactions on Reconfigurable Technology and Systems (TRETS)* 13.4 (2020), pp. 1–28 (cit. on pp. 32, 43, 54, 64).
- [152] N. Bartzoudis et al. “Agile FPGA Computing at the 5G Edge: Joint Management of Accelerated and Software Functions for Open Radio Access Technologies”. In: *Electronics* 13.4 (2024), p. 701 (cit. on pp. 32, 43, 54, 64).
- [153] P. D. Schiavone et al. “Arnold: An eFPGA-augmented RISC-V SoC for flexible and low-power IoT end nodes”. In: *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 29.4 (2021), pp. 677–690 (cit. on pp. 32, 43, 54, 64).
- [154] E. S. Chung, J. C. Hoe, and K. Mai. “CoRAM: an in-fabric memory architecture for FPGA-based computing”. In: *Proceedings of the 19th ACM/SIGDA international symposium on Field programmable gate arrays*. 2011, pp. 97–106 (cit. on pp. 32, 43, 54, 64).
- [155] F. Chen et al. “Enabling FPGAs in the cloud”. In: *Proceedings of the 11th ACM Conference on Computing Frontiers*. 2014, pp. 1–10 (cit. on pp. 32, 43, 54, 64).
- [156] S. A. Fahmy, K. Vipin, and S. Shreejith. “Virtualized FPGA accelerators for efficient cloud computing”. In: *2015 IEEE 7th International Conference on Cloud Computing Technology and Science (CloudCom)*. IEEE. 2015, pp. 430–435 (cit. on pp. 33, 43, 54, 64).
- [157] O. Knodel et al. “FPGAs and the cloud—An endless tale of virtualization, elasticity and efficiency”. In: *International Journal on Advances in Systems and Measurements* 11.3-4 (2018), pp. 230–249 (cit. on pp. 33, 43, 54, 64).

- [158] M. Asiatici et al. "Virtualized execution runtime for FPGA accelerators in the cloud". In: *Ieee Access* 5 (2017), pp. 1900–1910 (cit. on pp. 33, 43, 54, 64).
- [159] J. M. Mbongue et al. "Architecture Support for FPGA Multi-tenancy in the Cloud". In: *2020 IEEE 31st International Conference on Application-specific Systems, Architectures and Processors (ASAP)*. 2020, pp. 125–132. doi: 10.1109/ASAP49362.2020.00030 (cit. on pp. 33, 43, 54, 64).
- [160] J. Ma et al. "A hypervisor for shared-memory FPGA platforms". In: *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*. 2020, pp. 827–844 (cit. on pp. 33, 43, 54, 64).
- [161] Y. Zha and J. Li. "Virtualizing FPGAs in the cloud". In: *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*. 2020, pp. 845–858 (cit. on pp. 33, 43, 54, 64).
- [162] Y. Zha and J. Li. "Hetero-ViTAL: A virtualization stack for heterogeneous FPGA clusters". In: *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*. IEEE. 2021, pp. 470–483 (cit. on pp. 33, 43, 54, 64).
- [163] Google. *Titan in depth: Security in plaintext*. <https://cloud.google.com/blog/products/gcp/titan-in-depth-security-in-plaintext>. 2023 (cit. on pp. 33, 35, 43, 73, 76, 97).
- [164] J. Jung et al. "A secure platform model based on ARM platform security architecture for IoT devices". In: *IEEE Internet of Things Journal* 9.7 (2021), pp. 5548–5560 (cit. on p. 33).
- [165] Samsung. *Samsung TEEGRIS*. <https://developer.samsung.com/teegris/overview.html>. Accessed: 2025-10-07. 2024 (cit. on pp. 34, 43).
- [166] SierraTEE. *SierraTEE: Open-Source ARM TrustZone*. <https://www.sierraware.com/open-source-ARM-TrustZone.html>. Accessed: 2025-10-07 (cit. on pp. 34, 43).
- [167] OP-TEE Documentation Contributors. *OP-TEE*. <https://optee.readthedocs.io/>. 2024 (cit. on pp. 34, 43, 116).
- [168] Google. *Trusty TEE*. (cit. on pp. 18, 27). 2024. url: <https://source.android.com/docs/security/features/trusty> (cit. on pp. 34, 43).
- [169] Arm Limited. *Trusted Firmware-A*. <https://git.trustedfirmware.org/TF-A/trusted-firmware-a.git>. (cit. on pp. 27, 77) (cit. on pp. 34, 43).
- [170] X. Yang et al. "Trust-E: A Trusted Embedded Operating System Based on the ARM Trustzone". In: *2014 IEEE 11th Intl Conf on Ubiquitous Intelligence and Computing and 2014 IEEE 11th Intl Conf on Autonomic and Trusted Computing and 2014 IEEE 14th Intl Conf on Scalable Computing and Communications and Its Associated Workshops*. 2014, pp. 495–501. doi: 10.1109/UIC-ATC-ScalCom.2014.15 (cit. on pp. 34, 43).

- [171] M. Paolino et al. “T-KVM: A trusted architecture for KVM ARM v7 and v8 virtual machines securing virtual machines by means of KVM, TrustZone, TEE and SELinux”. In: *The Sixth International Conference on Cloud Computing, GRIDs, and Virtualization (CLOUD COMPUTING 2015)*. IARIA. 2015, pp. 120–125 (cit. on pp. 34, 43).
- [172] B. McGillion et al. “Open-TEE – An Open Virtual Trusted Execution Environment”. In: *2015 IEEE Trustcom/BigDataSE/ISPA*. Vol. 1. 2015, pp. 400–407. doi: 10.1109/Trustcom.2015.400 (cit. on pp. 34, 43).
- [173] N. Feske. *Genode Operating System Framework*. (cit. on p. 27). 2015. url: <https://genode.org/files/53bcb8e33fe6602fed25edc3c7b922c5/manual-2015-04-27.pdf> (cit. on pp. 34, 43).
- [174] T. Mandt, M. Solnik, and D. Wang. “Demystifying the secure enclave processor”. In: *Black Hat* (2016) (cit. on pp. 34, 43, 97).
- [175] Qualcomm. *Qualcomm Secure Processing Unit SPU230 Core Security Target Lite*. [https://www.commoncriteriaportal.org/files/epfiles/1045b\\_pdf.pdf](https://www.commoncriteriaportal.org/files/epfiles/1045b_pdf.pdf). 2019 (cit. on pp. 34, 43, 73, 76).
- [176] Intel Corporation. *Intel® Active Management Technology: A Guide to the Intel® Management Engine*. <https://www.intel.com/content/dam/www/public/us/en/documents/white-papers/intel-active-management-technology-intel-me.pdf>. Accessed: 2025-06-30. 2013 (cit. on pp. 35, 43).
- [177] T. L. K. documentation community. *AMD-TEE (AMD’s Trusted Execution Environment)*. “next-20250918” version. Linux Kernel. 2025. url: <https://docs.kernel.org/next/tee/amd-tee.html> (cit. on pp. 35, 43).
- [178] M. Gross et al. “Enhancing the security of FPGA-SoCs via the usage of ARM TrustZone and a hybrid-TPM”. In: *ACM Transactions on Reconfigurable Technology and Systems (TRETs)* 15.1 (2021), pp. 1–26 (cit. on pp. 35, 43).
- [179] R. Perez, R. Sailer, L. Van Doorn, et al. “vTPM: virtualizing the trusted platform module”. In: *Proc. 15th Conf. on USENIX Security Symposium*. 2006, pp. 305–320 (cit. on pp. 35, 43).
- [180] K. Xia et al. “SGX-FPGA: Trusted Execution Environment for CPU-FPGA Heterogeneous Architecture”. In: *ACM/ IEEE DAC*. 2021 (cit. on pp. 35, 36, 43–45, 54, 66, 101, 105).
- [181] D. Hwang, S. Yeleuov, and et. al. “Ambassy: A Runtime Framework to Delegate Trusted Applications in an ARM/FPGA Hybrid System”. In: *IEEE Trans. on Mobile Computing* (2021) (cit. on pp. 36, 43–45, 54, 66, 96–98, 124, 125).
- [182] Y. Wang et al. “Towards Secure Runtime Customizable Trusted Execution Environment on FPGA-SoC”. In: *IEEE Transactions on Computers* 73.4 (2024), pp. 1138–1151. doi: 10.1109/TC.2024.3355772 (cit. on pp. 36, 37, 43–45, 54, 66, 96–98).

- [183] K. Xia and S. Wei. “DF-TEE: Trusted Execution Environment for Disaggregated Multi-FPGA Cloud Systems”. In: *Proceedings of the Asian Hardware Oriented Security and Trust Symposium (Asian-HOST)*. 2023, pp. 1–6. doi: 10.1109/ASIANHOST59942.2023.10409376 (cit. on pp. 37, 38, 43, 45, 54, 65–67, 96, 98).
- [184] H. Oh et al. “MeetGo: A Trusted Execution Environment for Remote Applications on FPGA”. In: *IEEE Access* (2021) (cit. on pp. 37–39, 43, 45, 54, 67, 74, 96–98, 102, 124, 125).
- [185] W. Ren, J. Pan, and D. Chen. “AccGuard: Secure and Trusted Computation on Remote FPGA Accelerators”. In: *2021 IEEE International Symposium on Smart Electronic Systems (iSES)*. 2021, pp. 378–383. doi: 10.1109/iSES52644.2021.00093 (cit. on pp. 37, 38, 43, 54, 68).
- [186] M. Armanuzzaman and Z. Zhao. “BYOTee: Towards Building Your Own Trusted Execution Environments Using FPGA”. In: *AsiaCCS*. 2024 (cit. on pp. 39, 43, 45, 54, 68, 96–98, 102, 124, 125).
- [187] R. Bahmani et al. “CURE: A Security Architecture with CUsTomizable and Resilient Enclaves”. In: *Proc. in USENIX Security*. 2021 (cit. on pp. 40, 43, 54, 67, 107).
- [188] S. Yuan et al. “Analyzing Secure Memory Architecture for GPUs”. In: *2021 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. 2021, pp. 59–69. doi: 10.1109/ISPASS51385.2021.00017 (cit. on p. 40).
- [189] S. Volos, K. Vaswani, and R. Bruno. “Graviton: Trusted Execution Environments on GPUs”. In: *USENIX OSDI*. 2018. url: <https://www.usenix.org/conference/osdi18/presentation/volos> (cit. on pp. 40, 43, 96, 97, 102).
- [190] I. Jang et al. “Heterogeneous Isolated Execution for Commodity GPUs”. In: *Proc. in ASPLOS*. ACM, 2019. url: <https://doi.org/10.1145/3297858.3304021> (cit. on pp. 40, 43, 102).
- [191] Y. Deng et al. “StrongBox: A GPU TEE on Arm Endpoints”. In: *Proceedings of CCS*. New York, NY, USA: ACM, 2022. doi: 10.1145/3548606.3560627 (cit. on pp. 40, 43, 96, 97, 102).
- [192] S. Fan et al. “XpuTEE: A High-Performance and Practical Heterogeneous Trusted Execution Environment for GPUs”. In: *ACM Transactions on Computer Systems* (2025) (cit. on pp. 41, 43).
- [193] A. W. B. Yudha et al. “LITE: a low-cost practical inter-operable GPU TEE”. In: *Proceedings of the 36th ACM International Conference on Supercomputing*. ICS ’22. New York, NY, USA: ACM, 2022. isbn: 9781450392815. doi: 10.1145/3524059.3532361. url: <https://doi.org/10.1145/3524059.3532361> (cit. on pp. 41, 43).
- [194] X. Wu, D. J. Tian, and C. H. Kim. “Building GPU TEEs using CPU Secure Enclaves with GEVisor”. In: *Proceedings of the 2023 ACM Symposium on Cloud Computing*. SoCC ’23. New York, NY, USA: ACM, 2023, pp. 249–264. isbn: 9798400703874. doi: 10.1145/3620678.3624659. url: <https://doi.org/10.1145/3620678.3624659> (cit. on pp. 41, 43).

- [195] T. Hunt et al. “Telekine: Secure Computing with Cloud GPUs”. In: *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*. Santa Clara, CA: USENIX Association, 2020, pp. 817–833. isbn: 978-1-939133-13-7. url: <https://www.usenix.org/conference/nsdi20/presentation/hunt> (cit. on pp. 41, 43).
- [196] W. Ren et al. “AccShield: a New Trusted Execution Environment with Machine-Learning Accelerators”. In: *2023 60th ACM/IEEE Design Automation Conference (DAC)*. 2023, pp. 1–6. doi: 10.1109/DAC56929.2023.10247768 (cit. on pp. 41, 43).
- [197] K. Vaswani et al. “Confidential Computing within an AI Accelerator”. In: *2023 USENIX Annual Technical Conference (USENIX ATC 23)*. Boston, MA: USENIX Association, 2023, pp. 501–518. isbn: 978-1-939133-35-9. url: <https://www.usenix.org/conference/atc23/presentation/vaswani> (cit. on pp. 41, 43).
- [198] E. Feng et al. “sNPU: Trusted Execution Environments on Integrated NPUs”. In: *2024 ACM/IEEE 51st Annual International Symposium on Computer Architecture (ISCA)*. 2024, pp. 708–723. doi: 10.1109/ISCA59077.2024.00057 (cit. on pp. 42, 43).
- [199] T. Sorensen and H. Khlaaf. *LeftoverLocals: Listening to LLM Responses Through Leaked GPU Local Memory*. 2024. arXiv: 2401.16603 [cs.CR]. url: <https://arxiv.org/abs/2401.16603> (cit. on p. 42).
- [200] S. Lee et al. “TNPU: Supporting Trusted Execution with Tree-less Integrity Protection for Neural Processing Unit”. In: *2022 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. 2022, pp. 229–243. doi: 10.1109/HPCA53966.2022.00025 (cit. on pp. 42, 43).
- [201] H. Han et al. “TensorTEE: Unifying Heterogeneous TEE Granularity for Efficient Secure Collaborative Tensor Computing”. In: *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 4*. ASPLOS ’24. New York, NY, USA: ACM, 2025, pp. 282–297. isbn: 9798400703911. doi: 10.1145/3622781.3674168. url: <https://doi.org/10.1145/3622781.3674168> (cit. on pp. 42, 43).
- [202] P.-E. Gaillardon et al. “The programmable logic-in-memory (PLiM) computer”. In: *Proc. of the Conference DATE*. 2016 (cit. on p. 48).
- [203] I. Kuon and J. Rose. “Measuring the gap between FPGAs and ASICs”. In: *Proceedings of the 2006 ACM/SIGDA 14th international symposium on Field programmable gate arrays*. 2006, pp. 21–30 (cit. on p. 53).
- [204] P. Babu and E. Parthasarathy. “Reconfigurable FPGA architectures: A survey and applications”. In: *Journal of The Institution of Engineers (India): Series B* 102.1 (2021), pp. 143–156 (cit. on p. 53).

- 
- [205] J. Lant et al. "Toward FPGA-based HPC: Advancing interconnect technologies". In: *IEEE Micro* 40.1 (2019), pp. 25–34 (cit. on p. 53).
  - [206] M. Elnawawy et al. "Role of FPGA in internet of things applications". In: *2019 IEEE International Symposium on Signal Processing and Information Technology (ISSPIT)*. IEEE. 2019, pp. 1–6 (cit. on p. 53).
  - [207] M. Gazziro et al. "Design and Evaluation of Open-Source Soft-Core Processors". In: *Electronics* 13.4 (2024). issn: 2079-9292. doi: 10.3390/electronics13040781. url: <https://www.mdpi.com/2079-9292/13/4/781> (cit. on p. 53).
  - [208] S. Pereira et al. "Towards a Trusted Execution Environment via Reconfigurable FPGA". In: *CoRR* abs/2107.03781 (2021). arXiv: 2107.03781. url: <https://arxiv.org/abs/2107.03781> (cit. on pp. 54, 68, 102, 124, 125).
  - [209] A. Proulx et al. "A Survey on FPGA Cybersecurity Design Strategies". In: *ACM Trans. Reconfigurable Technol. Syst.* 16.2 (2023). issn: 1936-7406. url: <https://doi.org/10.1145/3561515> (cit. on p. 54).
  - [210] S. Duan et al. "A Survey of Recent Attacks and Mitigation on FPGA Systems". In: *ISVLSI*. IEEE, 2021. isbn: 978-1-6654-3946-6. url: <https://ieeexplore.ieee.org/document/9516757/> (cit. on p. 54).
  - [211] M. Kawser Ahmed et al. "Multi-Tenant Cloud FPGA: A Survey on Security, Trust, and Privacy". In: *ACM Trans. Reconfigurable Technol. Syst.* 18.2 (2025). issn: 1936-7406. doi: 10.1145/3713078. url: <https://doi.org/10.1145/3713078> (cit. on p. 54).
  - [212] P. D. Rosero-Montalvo, Z. István, and W. Hernandez. "A Survey of Trusted Computing Solutions Using FPGAs". In: *IEEE Access* (2023). url: <https://ieeexplore.ieee.org/document/10080920/> (cit. on pp. 54, 74, 105).
  - [213] S. Pinto and N. Santos. "Demystifying arm trustzone: A comprehensive survey". In: *ACM computing surveys (CSUR)* 51.6 (2019), pp. 1–36 (cit. on p. 54).
  - [214] S. M. Trimberger. *Field-programmable gate array technology*. Springer Science & Business Media, 2012 (cit. on pp. 56, 104).
  - [215] Microchip Technology Inc. *PolarFire SoC FPGA Icicle Kit User Guide*. DS60001679D. User Guide. Accessed July 2025. Chandler, AZ, USA, 2024. url: [https://ww1.microchip.com/downloads/aemDocuments/documents/FPGA/ProductDocuments/UserGuides/microchip\\_polarfire\\_soc\\_fpga\\_icicle\\_kit\\_user\\_guide\\_vb.pdf](https://ww1.microchip.com/downloads/aemDocuments/documents/FPGA/ProductDocuments/UserGuides/microchip_polarfire_soc_fpga_icicle_kit_user_guide_vb.pdf) (cit. on p. 57).
  - [216] W. Lie and W. Feng-yan. "Dynamic Partial Reconfiguration in FPGAs". In: *Intelligent Information Technology Application*. 2009. doi: 10.1109/IITA.2009.334 (cit. on pp. 57, 104).
  - [217] M. A. Bouazzouni, E. Conchon, and F. Peyrard. "Trusted mobile computing: An overview of existing solutions". In: *FGCS* (2018) (cit. on p. 73).

- [218] O. Ohrimenko et al. "Oblivious Multi-Party Machine Learning on Trusted Processors". In: *Proc. in USENIX Security*. 2016 (cit. on p. 73).
- [219] F. Schuster et al. "VC3: Trustworthy Data Analytics in the Cloud Using SGX". In: *Proc. in S&P*. 2015 (cit. on p. 73).
- [220] W. Zheng et al. "Opaque: An Oblivious and Encrypted Distributed Analytics Platform". In: *Proc. in USENIX NSDI*. 2017 (cit. on p. 73).
- [221] I. Corporation. *Intel 64 and IA-32 Architectures Software Developer's Manual*. <https://www.intel.com/content/www/us/en/developer/articles/technical/intel-sdm.html>. 2023 (cit. on p. 73).
- [222] G. Chen et al. "SgxPectre Attacks: Leaking Enclave Secrets via Speculative Execution". In: *Proc. in IEEE EuroS&P*. 2019 (cit. on p. 73).
- [223] J. V. Bulck et al. "LVI: Hijacking Transient Execution through Microarchitectural Load Value Injection". In: *Proc. in IEEE S&P*. 2020 (cit. on p. 73).
- [224] N. Zhang et al. "CacheKit: Evading Memory Introspection Using Cache Incoherence". In: *Proc. in IEEE EuroS&P*. 2016 (cit. on pp. 73, 79).
- [225] J. V. Bulck et al. "Foreshadow: Extracting the Keys to the Intel SGX Kingdom with Transient Out-of-Order Execution". In: *Proc. in USENIX Security*. 2018 (cit. on p. 73).
- [226] J. Götzfried et al. "Cache Attacks on Intel SGX". In: *Proc. in ACM EuroSec*. 2017 (cit. on p. 73).
- [227] R. Guanciale et al. "Cache Storage Channels: Alias-Driven Attacks and Verified Countermeasures". In: *Proc. in IEEE S&P*. 2016 (cit. on pp. 73, 79).
- [228] M. Schwarz et al. "Malware Guard Extension: Using SGX to Conceal Cache Attacks". In: *Springer* (2017) (cit. on p. 73).
- [229] A. Prout et al. "Measuring the Impact of Spectre and Meltdown". In: *Proc. in IEEE HPEC*. 2018 (cit. on pp. 73, 125).
- [230] K. Kostiainen, A. Dhar, and S. Capkun. "Dedicated Security Chips in the Age of Secure Enclaves". In: *Proc. in IEEE S&P*. 2020 (cit. on pp. 74, 76).
- [231] Y. Chen et al. "DITES: A Lightweight and Flexible Dual-Core Isolated Trusted Execution SoC Based on RISC-V". In: *Sensors* (2022) (cit. on p. 74).
- [232] H. Oh et al. "TRUSTORE: Side-Channel Resistant Storage for SGX Using Intel Hybrid CPU-FPGA". In: *Proc. in ACM CCS*. 2020 (cit. on p. 74).
- [233] GlobalPlatform. *GlobalPlatform TEE System Architecture, TEE Internal Core API Specification, and TEE Client API Specification*. <https://globalplatform.org/specs-library/> (cit. on pp. 74, 75, 78, 84, 102, 105, 107).

- [234] GlobalPlatform. *Introduction to TEEs*. <https://globalplatform.org/resource-publication/introduction-to-trusted-execution-environments/>. 2018 (cit. on pp. 75, 78, 105, 107).
- [235] Z. Hua et al. "VTZ: Virtualizing ARM Trustzone". In: *Proc. in USENIX Security*. 2017 (cit. on p. 76).
- [236] D. Oliveira, T. Gomes, and S. Pinto. "uTango: An Open-Source TEE for IoT Devices". In: *IEEE Access* (2022) (cit. on p. 76).
- [237] D. Melotti, M. Rossi-Bellom, and A. Continella. "Reversing and Fuzzing the Google Titan M Chip". In: *Proc. in ACM*. 2022 (cit. on p. 76).
- [238] T. Roche et al. "A Side Journey To Titan". In: *Proc. in USENIX Security*. 2021 (cit. on p. 76).
- [239] E. Peterson. *Developing tamper-resistant designs with ultrascale and ultrascale+ FPGAs*. [https://www.xilinx.com/support/documentation/application\\_notes/xapp1098-tamper-resist-designs.pdf](https://www.xilinx.com/support/documentation/application_notes/xapp1098-tamper-resist-designs.pdf) (cit. on pp. 77, 95, 104).
- [240] H. Cho et al. "Prime+Count: Novel Cross-world Covert Channels on ARM TrustZone". In: *Proc. in ACM ACSAC*. 2018 (cit. on p. 79).
- [241] K. Ryan. *Hardware-BackedHeist: Extracting ECDSA Keys from Qualcomm's TrustZone*. <https://www.nccgroup.trust/globalassets/our-research/us/whitepapers/2019/hardwarebackedhesit.pdf>. 2019 (cit. on pp. 79, 107).
- [242] Quarkslab. *A Deep Dive Into Samsung's TrustZone*. <https://blog.quarkslab.com/a-deep-dive-into-samsungs-trustzone-part-3.html> (cit. on p. 79).
- [243] D. Shen. *Attacking your "Trusted Core" Exploiting TrustZone on Android*. BlackHat USA. 2015 (cit. on p. 79).
- [244] SatoshiLabs. *Trezor-crypto*. <https://github.com/trezor/trezor-crypto>. 2021 (cit. on p. 85).
- [245] *7 Series Product Tables and Product Selection Guide (XMP101)*. AMD. 2022. url: <https://www.xilinx.com/support/documentation/selection-guides/7-series-product-selection-guide.pdf> (cit. on p. 93).
- [246] AMD Xilinx. *Bootgen User Guide*. Available at <https://docs.xilinx.com/r/en-US/ug1283-bootgen-user-guide>. 2023 (cit. on p. 95).
- [247] V. Kharchenko et al. "Security assessment of FPGA-based safety-critical systems: US NRC requirements context". In: *Proc. in ICIDT*. 2015 (cit. on p. 96).
- [248] H. Kareem and D. Dunaev. "Xilinx FPGA-based Ring Oscillator PUFs: Design Challenges and Solutions". In: *2022 11th Mediterranean Conference on Embedded Computing (MECO)*. 2022, pp. 1–5. doi: 10.1109/MECO55406.2022.9797077 (cit. on p. 96).

- [249] F. E. Potestad-Ordóñez, A. Casado-Galán, and E. Tena-Sánchez. “Protecting FPGA-Based Cryptohardware Implementations from Fault Attacks Using ADCs”. In: *Sensors* 24.5 (2024). issn: 1424-8220. doi: 10.3390/s24051598. url: <https://www.mdpi.com/1424-8220/24/5/1598> (cit. on p. 96).
- [250] S. Hayashi, J. Sakamoto, and T. Matsumoto. “Design methodology of digital sensors for detecting laser fault injection attacks in FPGAs”. In: *Journal of Cryptographic Engineering* 15.2 (2025), p. 12. issn: 2190-8516. doi: 10.1007/s13389-025-00378-4. url: <https://doi.org/10.1007/s13389-025-00378-4> (cit. on p. 96).
- [251] H. Lohrke et al. “Key extraction using thermal laser stimulation: A case study on Xilinx ultra-scale FPGAs”. In: *IACR Transactions on Cryptographic Hardware and Embedded Systems* 2018.3 (2018), pp. 573–595. doi: 10.13154/tches.v2018.i3.573-595 (cit. on p. 96).
- [252] S. Tajik et al. “On the power of optical contactless probing: Attacking bitstream encryption of FPGAs”. In: *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security (CCS’17)*. 2017, pp. 1661–1674 (cit. on p. 96).
- [253] K.-S. Chong et al. “Dual-hiding side-channel-attack resistant FPGA-based asynchronous-logic AES: Design, countermeasures and evaluation”. In: *IEEE Journal on Emerging and Selected Topics in Circuits and Systems* 11.2 (2021), pp. 343–356. doi: 10.1109/JETCAS.2021.3077887 (cit. on p. 96).
- [254] P. Nasahl et al. “HECTOR-V: A Heterogeneous CPU Architecture for a Secure RISC-V Execution Environment”. In: *Proc. in ACM ASIA CCS*. 2021 (cit. on pp. 96, 97).
- [255] V. Costan, I. Lebedev, and S. Devadas. “Sanctum: Minimal Hardware Extensions for Strong Software Isolation”. In: *Proc. in USENIX Security*. 2016 (cit. on pp. 96, 97).
- [256] F. Brasser et al. “SANCTUARY: ARMing TrustZone with User-space Enclaves”. In: *Proc. in NDSS Symposium*. 2019 (cit. on pp. 96, 97).
- [257] D. Lee et al. “Keystone: An Open Framework for Architecting Trusted Execution Environments”. In: *Proc. in ACM EuroSys*. 2020 (cit. on p. 97).
- [258] M. Sabt, M. Achemlal, and A. Bouabdallah. “Trusted Execution Environment: What It is, and What It is Not”. In: *IEEE Trustcom/ BigDataSE/ ISPA*. 2015. doi: 10.1109/Trustcom.2015.357 (cit. on pp. 102, 105).
- [259] R. Liu et al. “SecDeep: Secure and Performant On-device Deep Learning Inference Framework for Mobile and IoT Devices”. In: *ACM*, 2021. url: <https://doi.org/10.1145/3450268.3453524> (cit. on p. 102).
- [260] T. Huffmire et al. “Managing Security in FPGA-Based Embedded Systems”. In: *IEEE Design & Test of Computers* (2008). doi: 10.1109/MDT.2008.166 (cit. on p. 105).

- [261] R. Shu et al. "A Study of Security Isolation Techniques". In: *Comput. Surv.* (2016). url: <https://doi.org/10.1145/2988545> (cit. on p. 107).
- [262] Q. Chen et al. "Privacy-preserving searchable encryption in the intelligent edge computing". In: *Computer Communications* (2020). doi: <https://doi.org/10.1016/j.comcom.2020.09.012> (cit. on p. 107).
- [263] A. M. Shuvo et al. *A Comprehensive Survey on Non-Invasive Fault Injection Attacks*. Cryptology ePrint Archive. 2023. url: <https://eprint.iacr.org/2023/1769> (cit. on p. 122).
- [264] K. Lata and L. Cenkeramaddi. "FPGA-Based PUF Designs: A Comprehensive Review and Comparative Analysis". In: *Cryptography* (2023). doi: [10.3390/cryptography7040055](https://doi.org/10.3390/cryptography7040055) (cit. on p. 122).
- [265] D. Cerdeira et al. "ReZone: Disarming TrustZone with TEE Privilege Reduction". In: *USENIX Security*. Boston, MA, 2022 (cit. on pp. 123, 125).