PASS: Physically-Aware Secure Software Execution

PIs: Mark Tehranipoor (tehranipoor@ufl.edu) and Farimah Farahmandi (ffarahmandi@ufl.edu), U. of Florida
Primary Areas: Generation, protection, and establishment of trust models for hardware and firmware interacting with the software stack
Potential Industry Liaisons: Ro Cammarota and Sohrab Aftabjahani (Intel), Doug Gardner and Wendell Manley (Analog Devices), Peilin Song and Nitin Pundir (IBM), Dhwani Mehta (AMD), Harry Chen (MediaTek)

1. Problem Statement

With the growth of new and exciting applications such as mobile, smart cities, mobility and connected autonomous vehicles, security and privacy have emerged as major design challenges. These advancements mean that modern software applications running on vulnerable system-on-chips (SoCs) and FPGAs are facing heightened risks from both cyber and hardware attacks. These systems, integral to personal devices and critical infrastructure, must maintain secure and reliable operations. Traditionally, software has been protected against cyberattacks integrated across the software standardization, development, and compilation cycle [1]. However, physical attacks due to physical vulnerabilities, e.g., fault injection (FIA) and side channels (SCA), could effectively bypass these upper-level software protections, threatening the system's confidentiality, integrity, and availability.

Physical attacks, such as FIA, SCA, and data extraction, directly exploit the hardware's physical properties rather than the software's logical vulnerabilities. For example, fault injection can intentionally alter the state of a circuit, cause a variety of disruptions in a processor (such as memory value modifications, instruction skipping, or calculation errors) [2], and facilitate leaking secrets of a system. Fault-injection can be non-invasive (e.g., clock glitching, voltage glitching, optical), semi-invasive (e.g., local heating and laser), or invasive (e.g., focused ion beam and probing), which can be carried out by a variety of techniques and instruments with different cost and precision. Different forms of fault-injection attacks have been successfully demonstrated by researchers as well as practitioners in the industry on many applications, and almost all platforms, such as microcontrollers [2], SoCs, FPGA-based embedded systems, and IoT devices are vulnerable to such attacks. Side-channel attacks, on the other hand, can extract sensitive information by observing physical phenomena during circuits' operation such as power consumption, electromagnetic (EM) emissions, and optical emissions [7]. As an example, side-channel attacks targeting cache or timing could monitor and leak sensitive data or control flow of the software execution without altering the chip's function. Side-channel analysis during software executions can be also combined with fault-injection and achieve a higher attack precision. Unlike cyberattacks, which software developers are well-equipped to anticipate and defend against, physical attacks target aspects of the system that are often beyond the developer's control or awareness [3]. For example, optical probing and side-channel analysis could facilitate identifying the type and locations of hardware components such as flip-flops and buffers [14], as demonstrated in Figure 1, the stored values or the logical behavior could be extracted or changed. Instruction memory cells, cache, and cores are also vulnerable [15]. In today's landscape, such hardware attacks have become a significant threat to both system and software security. *Properly identifying and mitigating hardware vulnerabilities could reduce overall system vulnerabilities and software attacks by up to 43%* [16].

The challenge is further compounded by the fact that current software development practices and compilers are not typically designed to address hardware vulnerabilities. Software developers and compilers often lack detailed insights into the hardware's vulnerability to such attacks. This disconnect means that even software that is robustly protected against cyber threats can remain vulnerable to attacks that target weaknesses in the underlying hardware. For instance, GCC, a widely adopted compiler collection, provides effective features to harden software against stack overflow attacks—a powerful cyberattack method that could facilitate manipulating instructions. However, a simple fault-injection attack on a single instruction or its address can bypass it and enable unauthorized function access, even when GCC's protections are fully enabled [9]. Consequently, because the vulnerable hardware has already been fabricated, developers who intended to harden their software specifically against physical attacks must rely on unrealistic software-side predictions for security models and protections, exposing their code to risks from physical attacks [3]. *Since the hardware has already been fabricated, the cost of redesign is extremely high, hence hardening software/firmware against the identified physical vulnerabilities would be a recommended and less costly approach*.

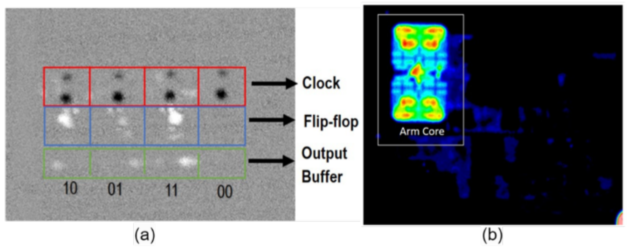
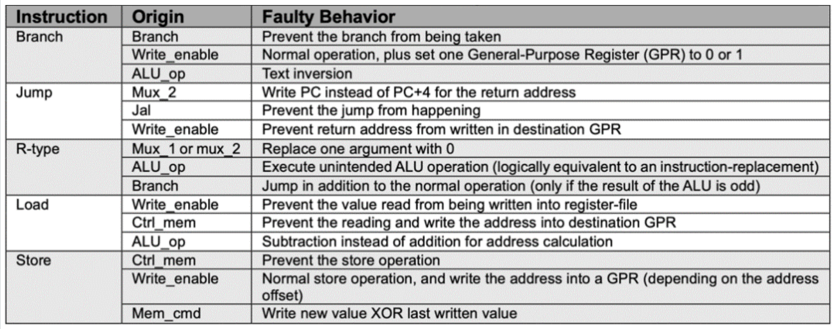


Figure 1: (a) The type, location, and stored value of hardware components can be probed through optical side-channel emissions [14]; (b) the ARM core in a commercially available chip localized from PEA [15];

2. Prior Work, Objectives and Novelty

Existing compilers such as GCC, Clang, and LLVM lack hardening techniques for physical attacks, and many conventional software-level countermeasures are proved ineffective or even harmful to system security [8], even well-established software countermeasures can be bypassed by simple fault-injection techniques, such as a single clock-glitch attack. In literature, some software-level physical-attack models have been designed to address the threat posed by physical attacks. For example, high-level abstractions such as the Common Criteria Methodology for smartcards, describe the potential effects of faults on processor behavior—such as instruction skipping, instruction replacement, or calculation errors. Also, fault-injection is modeled as bit-flips in some instruction-level or microarchitecture-level simulators, which can help observe the consequences of software executions with errors. These models have been widely adopted because of their simplicity and flexibility in studying hardware attacks' impacts on software/firmware programs.

Table 1: A summary of faulty behavior in many different types of RISC-V instruction shows a powerful potential for physical attackers to manipulate the program execution [8]. It is crucial to consider realistic physical-attack models and develop robust, physically-aware techniques for software hardening.



Instruction	Origin	Faulty Behavior
Branch	Branch	Prevent the branch from being taken
	Write_enable	Normal operation, plus set one General-Purpose Register (GPR) to 0 or 1
	ALU_op	Text inversion
Jump	Mux_2	Write PC instead of PC+4 for the return address
	Jal	Prevent the jump from happening
	Write_enable	Prevent return address from written in destination GPR
R-type	Mux_1 or mux_2	Replace one argument with 0
	ALU_op	Execute unintended ALU operation (logically equivalent to an instruction-replacement)
	Branch	Jump in addition to the normal operation (only if the result of the ALU is odd)
Load	Write_enable	Prevent the value read from being written into register-file
	Ctrl_mem	Prevent the reading and write the address into destination GPR
	ALU_op	Subtraction instead of addition for address calculation
Store	Ctrl_mem	Prevent the store operation
	Write_enable	Normal store operation, and write the address into a GPR (depending on the address offset)
	Mem_cmd	Write new value XOR last written value

However, research has shown that software models often lack the precision needed to accurately reflect the complexities of the real-world physical attack scenarios, particularly those observed at the lower abstraction levels. For fault-injection models, the software-level efforts often fail to account for microarchitectural states that are invisible to the programmer and do not consider faults that occur during instruction execution. For instance, extensive fault-injection campaigns [10] have shown that faults injected at lower levels of abstraction cannot always be effectively modeled by simple software-level approaches. Table 1 summarizes the potential faults within the RISC-V architecture that could lead to severe consequences, many of these faults target hardware components, indirectly causing faulty behavior in instructions, rather than directly targeting instruction-level machine codes. As a result, they fall beyond the scope of existing software-level models [8]. Further studies [11] have highlighted the intricate nature of fault propagation within a processor's pipeline and how a single fault can impact multiple architectural states, influencing program execution in unforeseen ways. Similarly, software-level side-channel leakage models typically rely on transforming cryptographic software into data flow graphs and performing static dataflow analysis to assess side-channel vulnerabilities [12]. However, real-life side-channel attacks depend heavily on measuring side-channel leakages (e.g., power, EM, optical) from hardware devices. Since the analysis and modeling of such leakages involve multi-physics considerations, software-only predictions and countermeasures are often questioned for their accuracy and overhead. To address side-channel vulnerabilities effectively at the software compilation stage, a physical-model-based hardening approach would be better suited to guide compilers. These findings emphasize the critical need to bridge the gap between software hardening and physical attack models, while also highlighting the importance of leveraging a more cost-effective and flexible solution to protect software executions, one that eliminates the need for call-outs of already fabricated chips.

PASS, for the first time to our knowledge, proposes a novel, cost-effective, scalable, and adoptable approach for addressing physical threats at the software compilation stage. PASS assumes one or both of the following steps are performed first: (1) Physical attacks are modeled based on hardware characteristics at the spice/layout-level in order to facilitate secure software compilation using physically-aware hardening techniques. (2) One performs post-silicon physical attacks/test on the chip/SoC during post-silicon validation. If the chip passes the security test, then no further action is needed. When the chip fails the test, the collected data and response will help validate and fine-tune the pre-silicon models. In case of laser-induced faults as an example, the model is based current sources that force a cell's output voltage to shift to the opposite level. In case of clock glitching, the model represents delay variations, that could also be supplemented by Fault-to-Time Converter sensors output [13]. The models' output, in the form of analog variations, will facilitate establishing a criticality and attack feasibility ranking of hardware components under specific attack scenarios, thereby informing the software compiler to avoid such attacks. Using data from the above-mentioned approaches, we aim to develop PASS framework, as shown in Figure 2, to harden the software code execution against physical attacks (e.g., fault-injection and side-channel

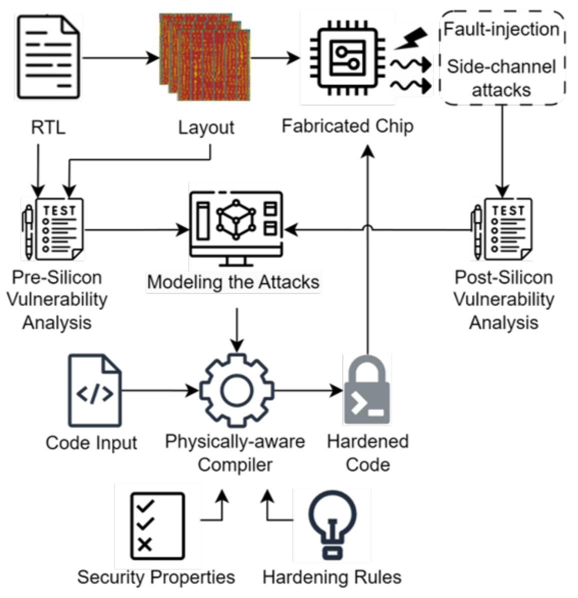


Figure 2: Overview of PASS: physically-aware secure software execution

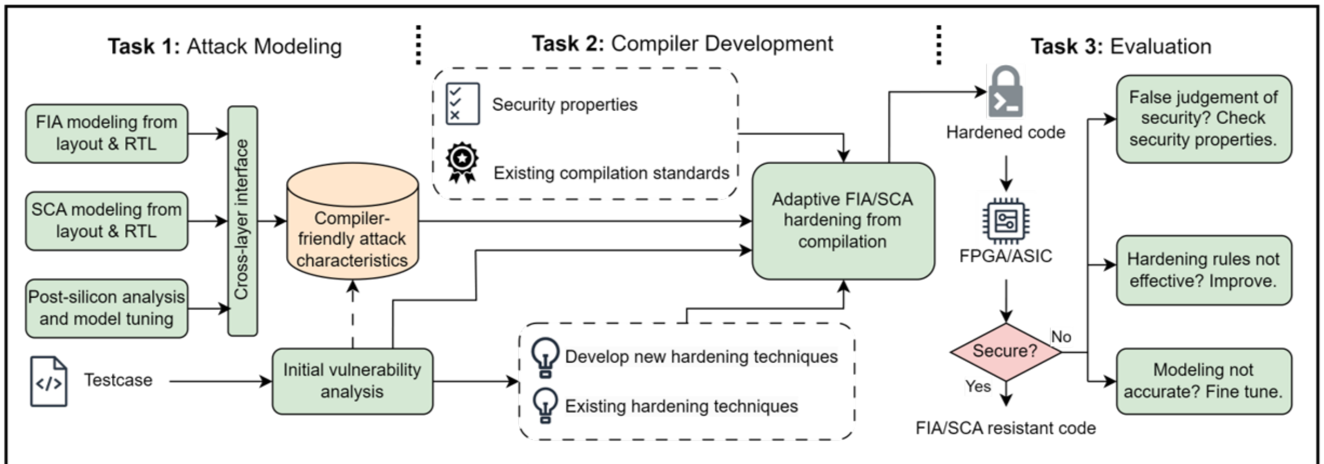
appeared) at the code compilation stage. To successfully carry out this project, the following three major tasks must be carefully executed: ***Task 1:*** *Effective physical attack characterizations and modeling,* ***Task 2:*** *Applying code hardening techniques to ensure software execution is resistant against physical vulnerabilities present in a fabricated SoC, and* ***Task 3:*** *Post-silicon evaluation on an FPGA platform and iterative refinements on attack modeling, security properties, and hardening strategies.*

3. Proposed Research

As shown in Figure 3, PASS project is organized into three primary tasks (modeling, compiler development, and evaluation), each with multiple subtasks, as discussed in detail below.

3.1. Task 1: Physical Attack Modeling: Developing a comprehensive modeling methodology for existing physical attack techniques is crucial to ensure fast, reliable, and accurate software execution, even in the presence of sophisticated physical attacks. This task focuses on modeling a wide range of fault-injection techniques, such as laser, clock, and voltage glitching, as well as side-channel attacks, including power, EM, and optical attacks. The models' outputs, represented as analog variations, will help establish a criticality and attack feasibility ranking of hardware components under various attack scenarios. This information will guide the software compiler in prioritizing and applying appropriate hardening strategies, thereby mitigating such attacks and eliminating the need for call-outs of fabricated chips. By combining pre-silicon attack modeling and responses from post-silicon security checks, our objective is to correlate physical attack characteristics with software vulnerabilities. The following subtasks are required to accomplish this goal.

Initial vulnerability analysis: The main goal of this step is to identify both primary and secondary security assets. While the design team defines primary assets (e.g., AES keys) as the definitive targets for protection, we will also determine a set of intermediate design components as secondary security assets—these are infrastructures that closely interact with primary assets and could be exploited by attackers to compromise security (e.g., signals or registers). Given a software test case, we will enable cybersecurity features in existing compilers and assess whether these critical assets remain secure even when considering physical attacks. It is important to note that this step will


Task 1: Attack Modeling
Task 2: Compiler Development
Task 3: Evaluation
FIA modeling from layout & RTL
SCA modeling from layout & RTL
Post-silicon analysis and model tuning
Cross-layer interface
Compiler-friendly attack characteristics
Testcase
Initial vulnerability analysis
Security properties
Existing compilation standards
Adaptive FIA/SCA hardening from compilation
Develop new hardening techniques
Existing hardening techniques
Hardened code
FPGA/ASIC
Secure?
No
Yes
FIA/SCA resistant code
False judgement of security? Check security properties.
Hardening rules not effective? Improve.
Modeling not accurate? Fine tune.
Figure 3: Proposed workflow of implementing PASS: physical attack modeling, hardened compiler development, and evaluation

not develop exhaustive security properties or formal assertions, which will be done in Task 2. Instead, it facilitates understanding how existing compiler features impact hardware security and how they might impact our hardened compilation flow.

Fault-injection modeling: We aim to characterize the **criticality** of faults by examining factors such as fault controllability, timing, the number of concurrent faults (single vs. multiple), fault type (e.g., direct bit-flip, timing fault), and fault duration. In addition to fault criticality, we will also model fault **feasibility** by simulating attack parameters like current demand, IR drop, and propagation delay using SPICE and layout simulations. For example, laser fault-injection will be modeled by introducing two current sources to the impacted standard cells, enabling us to determine and rank the feasibility of laser-based attacks by observing variations in current demand at the VDD/VSS pins. Timing violation-based fault-injection attacks such as clock glitching can be modeled as a set of delay variations using Fault-to-Time Converter [13] at gate-level, so that the timing impact of clock glitching attacks can be quantitatively measured, subsequently, the attack feasibilities across different modules, critical paths, or standard cells can be ranked.

Side-channel modeling: We aim to initiate this process by collecting power, EM, and optical side-channel traces from software running on RISC-V SoCs. These traces will be gathered across various abstraction levels. At the RTL and gate levels, experiments will focus on logical switching activities, fan-out signals of critical assets, and their driving load capacities. These factors will help establish a relationship between chip activity and its corresponding power consumption, EM, or optical emissions. At the layout level, trace collection will involve additional physical design parameters, such as placement and routing information from DEF files and parasitic data from SPEF files. Furthermore, we will account for the current distribution and IR drop across the layout when calculating side-channel leakage signals for targeted modules or cells. The modeling effort will involve using various physical variation metrics such as T-score, Measurements-to-Disclosure (MTD), and Pearson correlation coefficient for Correlation Power Analysis (CPA). By calculating and ranking these metrics for critical assets under different attack scenarios, we can create a comprehensive database that details the criticality and feasibility of side-channel attacks on software execution.

Post-silicon analysis and model tuning: In addition to pre-silicon modeling, we plan to perform real-life fault-injection and side-channel attacks on chip/SoC, and test if it could pass the security check while executing software. The data and responses collected from these post-silicon tests will be crucial in ensuring the accuracy and precision of our physical-attack models. For example, while laser fault-injection effects can be modeled during the pre-silicon phase, some side effects, such as thermal variations, may cause certain regions of the chip to be more vulnerable than initially expected. On the other hand, real-life power/EM side-channel attacks may introduce more noise into the measured traces, potentially making the system less vulnerable than predicted by pre-silicon models. Therefore, combining post-silicon findings with pre-silicon experiments is essential for developing realistic and robust models. The steps of post-silicon tests are outlined in Task 3.

Cross-layer interface for compiler-friendly attack characteristics: Since the raw responses from physical attack models are collected as analog values (e.g., current demand fluctuations under laser fault-injection) from lower abstraction levels, it becomes imperative to develop a cross-layer interface that allows the compiler to quickly determine priorities—such as which hardening techniques to use, and when and where to harden specific software instructions. We plan to achieve this goal by building a *compiler-friendly model interface via Look-up-Table (LUT) or Machine Learning (ML).* While coarse-grained models for various attack behaviors can be established in the initial steps, continuously refining model accuracy manually would be tedious and time-consuming. Once the accuracy reaches an acceptable level, we plan to store all input-response pairs from the models into LUTs and apply polynomial interpolations. Alternatively, we will leverage ML approaches to fit the output curves from our attack models. For example, laser-induced current changes under different laser intensities can be captured during SPICE simulations, and it is very efficient to use an ML approach, such as the K-Nearest Neighbors (KNN) algorithm, to fit the cell's or module's VDD/VSS current demand trend and avoid sweeping through all possible inputs. These ML models can then be integrated into the compiler's interface.

3.2. Task 2: Hardened Compiler Development: This task is focused on integrating the attack models developed in Task 1 into the software compilation process, thereby hardening software against physical vulnerabilities. The task is structured into three key steps as follows.

Code feature extraction: The vulnerability of different code segments can vary significantly based on their functions and the contexts in which they operate. For example, conditional checks and branched sections play a pivotal role in password authentication systems, as shown in Figure 4, the *secret_function* should be called only if the password checking is passed, i.e., the *bne* line in compiled assembly codes are correctly executed and did not branch to the code section for *incorrect_password*. However, this authentication could be bypassed if the *bne* line is faulty or bypassed. Similarly, loop bounds are critical in cryptographic algorithms like AES, where even a minor manipulation could compromise the entire encryption process. To identify these critical code snippets, we will employ ML techniques, such as pattern-matching algorithms and neural networks, which can analyze and detect vulnerability patterns within the code. Additionally, Large Language Models (LLMs) will be utilized to further refine this identification process by providing context-aware insights into code behavior. After identifying these critical sections, we will assess their corresponding hardware resources based on selected target hardware library. For a given Instruction Set Architecture (ISA), different instruction types activate different components or submodules of the hardware design. We will develop a mapping technique that connects certain instruction types with specific cells or submodules based on observed switching activity during functional simulations, enabling us to identify potentially vulnerable hardware resources during software executions.

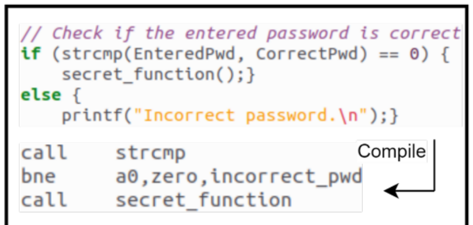

```
// Check if the entered password is correct
if (strcmp(EnteredPwd, CorrectPwd) == 0) {
    secret_function();}
else {
    printf("Incorrect password.\n");}
```
```
call    strcmp
bne     a0,zero,incorrect_pwd
call    secret_function
```

Figure 4: Critical conditional branches should be identified as a code feature.

Security property development: To effectively harden the identified vulnerable sections of the code, we will explicitly define security properties using SystemVerilog. These properties will aim to minimize the overhead of protection while ensuring that the most critical software components are adequately protected from attacks. For instance, in the context of AES encryption, although all intermediate states and keys during the encryption process might be considered critical, we will tailor security properties to address specific vulnerabilities. For example, rounds 8-10 are particularly susceptible to Differential Fault Analysis (DFA) attacks in AES-128 encryption. Therefore, we will focus on securing the intermediate states and round keys from these rounds by pinpointing their precise timing and corresponding hardware resources and defining these properties in an executable manner. Additionally, for side-channel attacks such as CPA, which often target the SubByte output values in the last two rounds of AES, we will develop separate security properties specifically. By focusing on such most at-risk stages of the encryption process, we can optimize the security measures to provide robust protection without incurring excessive performance costs.

Integration of hardening techniques: This step focuses on creating a comprehensive and scalable database of hardening rules to be integrated into the software compilation process. To accomplish this, we will undertake two key actions: *(1) evaluate and test existing hardening techniques*, such as redundant conditional checks, random execution delays, and redundant data storage and memory operations. However, these techniques are proven to be inadequate for many critical physical-attack scenarios [8], hence, we will *(2) develop innovative hardening rules* informed by insights from our hardware simulations and security property database. This approach aims to provide more targeted and effective countermeasures with reduced overhead. The above-mentioned hardening rules will be combined with the physical modeling data obtained in Task 1 to guide the compiler's hardening strategy, ensuring that they are grounded in realistic attack scenarios and can address the specific vulnerabilities identified in the earlier steps. We will implement these hardening rules in various forms, such as GCC plugins or LLVM, enabling seamless integration into standard software compilation workflows. As a result, the compilation process will automatically produce physically hardened assembly codes that are better equipped to withstand physical attacks.

3.3. Task 3: PASS Evaluation: The final phase of this project involves the extensive evaluation of the PASS flow on SoCs implemented on an FPGA platform. We aim to perform iterative tests of hardened software compilations, broaden the attack scenarios by including more security properties, and fine-tune the physical attack models. This evaluation will be conducted in two primary stages. First, we will implement three groups of designs on the FPGA: (1) a general-purpose core/SoC executing unprotected software test cases, (2) the same hardware executing software hardened by existing cybersecurity features in compilers like GCC, and (3) the same hardware executing the physically hardened software developed in Task 2. We will then check if the software execution could pass all security checks as defined in security properties, and evaluate the hardening overhead. This evaluation will involve both post-silicon physical attacks and emulated attacks, which will be performed by inserting attack points and probes into the FPGA emulation platform, such as Synopsys Zebu. The experiments will be conducted on a variety of cores, including general-purpose cores (e.g., RISC-V, microcontroller units, arithmetic cores) and cryptographic cores (e.g., AES, DES, RSA). The evaluation phase will not only measure the effectiveness of the hardened software but also serve as a validation and refinement step for the models developed in Task 1. Through this rigorous testing and validation process, we aim to ensure that the PASS flow provides robust protection against physical attacks while maintaining acceptable performance overheads.

4. Deliverables

We will deliver tools and methodologies developed through the course of this project. In particular, the outcome of Tasks 1 through 3 will be delivered to SRC member companies.

Year 1 (1/1/25-12/31/2025): Q1-3. SPICE-level and layout-level modeling of the physical characteristics of fault-injection and side-channel attacks; Q4. Post-silicon attack test on chip/SoC, perform vulnerability analysis of software executions, and improve the attack models.

Year 2 (1/1/2026-12/31/2026): Q1. Investigation and development of physically-aware hardening techniques; Q2. Developing executable security properties and LUT/ML-based model interfaces to guide hardening strategies. Q3-4. Integration of hardening techniques into software compilation in the form of GCC plug-in/LLVM/Clang;

Year 3 (1/1/2027-12/31/2027): 1-2. Extensive validation of the PASS flow on FPGA platform with iterative test of hardened software compilations; Q3-4. Broadening the attack scenarios by including more security properties, and fine-tuning of physical attack models.

5. Collaboration and Student Support

This project will support one graduate student. The requested funding is $105K per year. The PIs have an excellent record of collaboration with SRC member companies over the past several years, and a number of his technologies have been transferred to practice. PI Tehranipoor has been leading this effort for the past several years, including modeling and CAD frameworks for fault-injection [6] and side-channel attacks [7], vulnerability analysis and protections on firmware and software [4] against physical attacks. Co-PI Farahmandi's core research expertise includes formal approaches in hardware security applications as well as developing CAD frameworks for security such as analyzing vulnerabilities in FSMs [5]. The PASS framework will be developed through interaction with the member companies. We will augment these interactions with internships, joint pilot projects, and the distribution of software platforms and design techniques.

6. References

[1] GCC, the GNU Compiler Collection, "Program Instrumentation Options", https://gcc.gnu.org/onlinedocs/gcc/Instrumentation-Options.html, accessed 2024. [2] N. Moro et al., "Electromagnetic Fault Injection: Towards a Fault Model on a 32-bit Microcontroller," 2013 Workshop on Fault Diagnosis and Tolerance in Cryptography, 2013. [3] D. Pozza and R. Sisto, "A Lightweight Security Analyzer inside GCC," International Conference on Availability, Reliability and Security, Barcelona, 2008. [4] M. M. Hossain, F. Rahman, F. Farahmandi, and M. Tehranipoor, Software Security with Hardware in Mind, in Emerging Topics in Hardware Security, Springer, 2021. [5] A. Nahiyan, F. Farahmandi, M. Tehranipoor et al., "Security-Aware FSM Design Flow for Identifying and Mitigating Vulnerabilities to Fault Attacks," IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 38, no. 6, pp. 1003- 1016, June 2019. [6] H. Wang, M. Tehranipoor, F. Farahmandi et al., "SoFI: Security Property-Driven Vulnerability Assessments of ICs Against Fault-Injection Attacks," 2020 IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD), 2020. [7] H. Li, L. Lin, N. Chang, M. Tehranipoor et al., "Photon Emission Modeling and MachineLearning Assisted Pre-Silicon Optical Side-channel Simulation," IEEE International Symposium on Hardware Oriented Security and Trust (HOST), 2024. [8] J. Laurent et al., "Cross-layer analysis of software fault models and countermeasures against hardware fault attacks in a RISC-V processor", Microprocessors and Microsystems, Volume 71, 2019. [9] A. Höller et al., "QEMU-Based Fault Injection for a System-Level Analysis of Software Countermeasures Against Fault Attacks," 2015 Euromicro Conference on Digital System Design, Madeira, Portugal, 2015, pp. 530-533. [10] H. Cho et al., "Quantitative evaluation of soft error injection techniques for robust system design", 2013 50th ACM/EDAC/IEEE Design Automation Conference (DAC), 2013, pp. 1–10. [11] N. J. Wang et al., "Characterizing the effects of transient faults on a high-performance processor pipeline, in: International Conference on Dependable Systems and Networks", 2004, pp. 61–70. [12] G. Agosta et al., "Compiler-Based Techniques to Secure Cryptographic Embedded Software Against Side-Channel Attacks," IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 39, no. 8, pp. 1550-1554, Aug. 2020. [13] M. R. Muttaki, T. Zhang, M. Tehranipoor and F. Farahmandi, "FTC: A Universal Sensor for Fault Injection Attack Detection," 2022 IEEE International Symposium on Hardware Oriented Security and Trust (HOST), McLean, VA, USA, 2022, pp. 117-120. [14] N. Vashistha et al., "Is Backside the New Backdoor in Modern SoCs?: Invited Paper," 2019 IEEE International Test Conference (ITC), Washington, DC, USA, 2019, pp. 1-10. [15] M. Dhwani, M. Tehranipoor et al., "On the physical security of ai accelerators," International Conference on Physical Assurance and Inspection of Electronics, 2019. [16] The MITRE Corporation, "CVE Program Mission", https://www.cve.org, accessed 2024.