T-TER: Defeating A2 Trojans with Targeted Tamper-Evident Routing

Timothy Trippel
Kang G. Shin
University of Michigan
Ann Arbor, Michigan, USA
{trippel,kgshin}@umich.edu

Kevin B. Bush
MIT Lincoln Laboratory
Lexington, Massachusetts, USA
kevin.bush@ll.mit.edu

Matthew Hicks
Virginia Tech
Blacksburg, Virginia, USA
mdhicks2@vt.edu

ABSTRACT

Since the inception of the Integrated Circuit (IC), the size of the transistors used to construct them has continually shrunk. While this advancement significantly improves computing capability, fabrication costs have skyrocketed. As a result, most IC designers must now outsource fabrication. Outsourcing, however, presents a security threat: comprehensive post-fabrication inspection is infeasible given the size of modern ICs, so it is nearly impossible to know if the foundry has altered the original design during fabrication (i.e., inserted a hardware Trojan). Defending against a foundry-side adversary is challenging because—even with as few as two gates—hardware Trojans can completely undermine software security. Researchers have attempted to both *detect* and *prevent* foundry-side attacks, but all existing defenses are ineffective against additive Trojans with footprints of a few gates or less.

We present Targeted Tamper-Evident Routing (T-TER), a layout-level defense against untrusted foundries, capable of thwarting the insertion of even the stealthiest hardware Trojans. T-TER is *directed* and *routing-centric*: it prevents foundry-side attackers from routing Trojan wires to, or directly adjacent to, security-critical wires by shielding them with guard wires. Unlike shield wires commonly deployed for cross-talk reduction, T-TER guard wires pose an additional technical challenge: they must be tamper-evident in both the digital (deletion attacks) and analog (move and jog attacks) domains. We address this challenge by developing a class of *designed-in* guard wires that are added to the design specifically to protect security-critical wires. T-TER's guard wires incur minimal overhead, scale with design complexity, and provide tamper-evidence against attacks. We implement automated tools (on top of commercial CAD tools) for deploying guard wires around targeted nets within an open-source System-on-Chip. Lastly, using an existing IC threat assessment toolchain, we show T-TER defeats even the stealthiest known hardware Trojan, with ≈ 1% overhead.

KEYWORDS

Hardware Security, Fabrication-time Attacks and Defenses, VLSI

1 INTRODUCTION

Integrated circuits (ICs) are the foundation of computing systems. Security vulnerabilities in silicon are devastating as they subvert even formally verified software. For almost 50 years, the transistors within ICs have continued to shrink, enhancing performance while reducing power and area usage. However, these advances that push the laws of physics come with a financial cost: the price to build a 3 *nm* fabrication facility capable of producing ICs at a commercial scale is estimated to be \$15–20B [30]. Even when entities can afford to make such an investment, they must continually run the IC fabrication line (approximately 40,000 wafers/month) as many fabrication processes cannot be readily stopped and restarted.

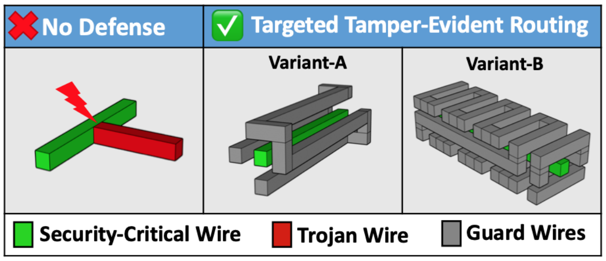

Figure 1: **T-TER is a *preventive* layout-level defense against fabrication-time Trojans. T-TER deploys tamper-evident guard wires around security-critical wires in a circuit layout—in a pattern similar to variant A or B—to prevent attackers from attaching Trojan wires to them.**

This extreme cost forces most semi-conductor companies, and even nation states, to become "fabless", i.e., they outsource fabrication. Today, only 3 companies in the world (Intel, Samsung, and TSMC) have capabilities to fabricate ICs at the 10/7 *nm* process nodes [31]. This presents a security threat: fabless semiconductor companies and nation states must trust these three manufacturers (and their partners) not to alter their designs at any point throughout the fabrication process (i.e., implant a hardware Trojan).

The most stealthy and controllable hardware Trojans involve inserting additional[1] circuit components designed to maliciously subvert the functionality of the chip (i.e., an additive hardware Trojan). Specifically, the A2 Trojan [65] utilizes only two additional cells—one analog capacitor and one digital logic gate— to provide a hardware foothold [26] within a microprocessor IC for an attacker to gain unauthorized supervisor privileges with user-mode code.

There are now only two ways of defending against hardware Trojans implanted at fabrication-time: post-fabrication *detection* [1, 14, 24, 32, 42, 69] and pre-fabrication *prevention* [3, 59, 60, 64]. The former tries to detect the presence of Trojan components after the chip has been fabricated, while the latter attempts to alter the IC's physical layout, at design time, in a way that makes foundry-side alterations challenging to an attacker.

[1] Additive hardware Trojans are a class of Trojan designs that require additional hardware to be added to a circuit design. We are unaware of any documented stealthy and controllable subtractive or substitution Trojans. Dopant-level Trojans [8, 28, 44] are the closest to such; however, they have limited controllability and are detectable [48].

Detection is more commonly studied than prevention and consists primarily of two techniques [51]: 1) side-channel analysis and 2) functional testing. Side-channel analysis attempts to detect noticeable deviations in power usage, electromagnetic (EM) emanations, performance (timing), etc. [1, 24, 38, 42]. It often requires a "golden" reference chip to be effective, and can only detect the side-channel signature deviations greater than those caused by process variation (i.e., the hardware Trojan must have a large physical footprint). Alternatively, functional testing attempts to inadvertently trigger the Trojan by activating as many logic paths through the circuit as possible. Functional testing does not require any "golden" reference chip, but it requires the Trojan's trigger to be activated by the IC's common mode operation, as exhaustive testing of even a moderately complex integrated circuit is infeasible.

Albeit less studied, *prevention* is another defense against fabrication-time hardware Trojans. To prevent such attacks, we advocate that the *placement and routing of security critical circuit elements should be a first-class part of an IC's back-end design*, on the level of performance, power, and cost. To the best of our knowledge, only three preventive fabrication-time defenses have been explored [3, 4, 64]. All of them are *placement*-centric, attempting to increase the device layer (core) density by filling empty spaces with with tamper-evident logic gates, thus making it challenging for an attacker to find open space in the design to insert their Trojan components (cells/gates). However, there are several problems with placement-centric defenses. As Ba *et al.* [4] point out, the BISA cell approach [64] is infeasible as it requires 100% placement density. Contrast this with the 60-80% density of current IC layouts that ensures routability. If 100% density were feasible, every IC design would be manufactured that way to save cost. Alternatively, Ba *et al.* [3, 4] suggest targeted filling: only filling placement sites that are located closest to "security-critical" logic. While prioritizing security-critical logic is a significant improvement, focusing on the device layer only impedes attacks due to inflated timing requirements, it does not prevent them, as §6.2.2 shows.

Unfortunately, no single technique is effective in detecting, and/or preventing the insertion of the stealthiest known additive hardware Trojan, the A2 Trojan [65], which requires only two additional cells. To fill this gap, we propose *Targeted Tamper-Evident Routing (T-TER)*, a *routing-centric* defense that *prevents* foundry-side attackers from routing Trojan wires to, or directly adjacent to, security-critical wires. We define T-TER as any routing method that protects security-critical wires from fabrication-time alterations. Specifically, we leverage concepts from the signal-integrity domain [18, 19] and apply them to a security domain (addressing several technical challenges along the way): we route "guard wires" around security-critical wires that make it infeasible for an attacker to tap any such wire without detection (i.e., tamper-evident), something characteristic of additive Trojans [54] (Fig. 1). Extending signal-integrity domain techniques to the security domain entails two technical challenges:

(1) *completely* shielding all surfaces of critical wires,
(2) and be tamper-evident.

Contrary to placement-centric defenses, which focus on preventing attack *implementation*, T-TER focuses on preventing attack *integration*, and thus, does *not* require filling *all* the empty space in an IC design to be effective.

We make the following contributions:

- Targeted Tamper-Evident Routing (T-TER): a routing-centric, preventative, defense against stealthy IC fabrication-time attacks. T-TER places *tamper-evident* guard wires alongside security-critical wires, making fabrication-time modifications to such wires infeasible and/or detectable post-fabrication.
- Characterization of possible guard wire bypass attacks.
- Attack-driven design of *designed-in guard wires*. Designed-in guard wires are added during the place-and-route phase of the IC design process for the sole purpose of defending security-critical wires. They have minimal routing constraints and can guard all surfaces of designer-targeted wires.
- Automated routing toolchain for deploying guard wires within an IC layout that integrates with commercial and open-source VLSI CAD tools.
- Evaluation of the effectiveness of T-TER compared to previous defenses against both digital and analog A2 Trojans embedded in a System-on-Chip intended to be a surrogate for DoD systems of interest [36], using a recently published fabrication-time threat assessment tool [54]. The results indicate T-TER is more effective than existing placement-centric defenses [3, 4], and is capable of thwarting even the stealthiest additive hardware Trojans, including A2 [65].[2]

2 BACKGROUND

2.1 IC Design Process

Creating an Integrated Circuit (IC) consisting of a billion transistors is a complex process that requires its decomposition into sub-processes and extensive use of automation via Computer Aided Design (CAD) tools. The IC design process consists of five main phases, as illustrated in Fig. 2. First, during RTL design, high-level descriptions of the IC are written in Hardware Description Languages (HDL) like Verilog or VHDL. Next, during synthesis, the HDL code is "compiled" into a gate-level netlist. The gate-level netlist is then placed-and-routed (PaR), and a physical geometric blueprint of the chip is encoded in a Graphics Database System II (GDSII) file. Lastly, the IC is fabricated, and packaged into a device for mounting on a printed circuit board. In line with prior work on untrusted foundry [3, 4, 8, 28, 34, 54, 64, 65], and economic forces, we assume all design phases—except fabrication—are trusted.

Defensive Routing is deployed at the physical level, i.e., the PaR design phase. During PaR, the gate-level netlist is physically arranged onto a 3-dimensional grid, shown in Fig. 3. The 3D grid consists of a device layer, where circuit components (e.g., digital logic gates) are placed, and several routing layers vertically stacked above, where wires are routed to connect the circuit components on the device layers. Each layer is separated by an insulating dielectric, and vias are used to connect wires on adjacent layers.

[2] It is important to note that routing-centric and placement-centric defenses are compatible (belt and suspenders). A designer would first apply T-TER, then fill open placement sites in a targeted manner.

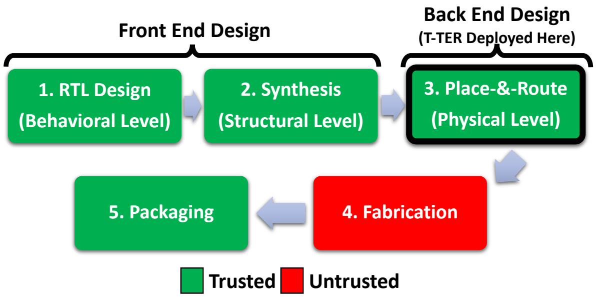

Figure 2: The IC design process consists of five main phases. We assume fabrication (phase 4) is the only untrusted phase, as this is often outsourced due to economic forces. T-TER is deployed at the place-&-route phase.

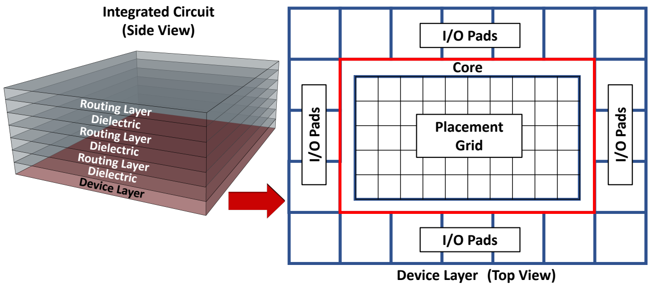

Figure 3: Typical 3D physical IC layout designed during the place-and-route IC design phase (Fig. 2). On the bottom is a device layer, and stacked above are several routing layers.

2.2 Hardware Trojans

A hardware Trojan is a malicious modification to a circuit designed to alter its operative functionality [7]. It consists of two main building blocks: a **trigger** and **payload** [12, 24, 54, 62]. Prior work provides hardware Trojan taxonomies based on the type of trigger and payload designs they employ [12, 24, 54, 55, 62]. Likewise, we adopt the same taxonomy.

2.2.1 *Trigger*.

The trigger is circuitry that initiates the delivery of the payload when it encounters a specific state. The goal of the trigger is to control payload deployment such that it is hidden from test cases (stealthy), but readily deployable by the attacker (controllable). Triggers are created by adding, removing, and/or manipulating existing circuit components [28, 44, 51, 65], and can be digital or analog [26, 43, 65]. The ideal trigger—e.g., A2 [65]—achieves stealth and controllability while being small (i.e., requiring few additional circuit components).

2.2.2 *Payload*.

The payload is circuitry that, upon being signaled by the trigger, alters the functionality of the victim (host) circuit. Like the trigger, the payload can be analog or digital, and has a variety of possible malicious effects. Prior work demonstrates Trojan payloads that leak information [34], alter the state of the IC [65], and render the IC inoperable [44]. *One attribute all documented controllable hardware Trojans have in common is that they must route a rogue wire to, or directly adjacent to, a security-critical wire within the victim IC [54].*

2.2.3 *Fabrication-Time Attacks*.

Inserting a hardware Trojan at fabrication time is different from inserting a Trojan during the front-end design. Unlike behavioral or structural-level attackers that maliciously modify the HDL or gate-level netlist, respectively [2, 23, 58], the fabrication-time attacker only has access to the *physical-level* representation of the IC design (i.e., output of phase 3 in Fig. 2). Specifically, they must edit the geometric representation of the *circuit layout*, e.g., the GDSII file. While this is more challenging than editing the design at the behavioral- (HDL) or structural-level (netlist), where design specific semantics are more readily interpretable, it is even more difficult to defend. The post-fabrication defender receives a literal black box from the foundry. Comprehensively inspecting each fabricated die to verify the absence of malicious perturbations is infeasible for the most advanced hardware Trojans [65].

As previous research reveals, **implanting a hardware Trojan into an IC layout requires three steps [54]: 1) *Trojan Placement*, 2) *Victim/Trojan Integration*, and 3) *Intra-Trojan Routing*.** *Trojan Placement* is the process of finding empty space on the IC's device layer to add additional circuit components, e.g., logic gates, to construct the Trojan trigger and payload. *Victim/Trojan Integration* requires attaching a rogue Trojan wire, or routing it directly adjacent, to an unblocked surface on a security-critical wire(s). Lastly, *Intra-Trojan Routing* involves routing the Trojan circuit components to the Victim/Trojan integration point—the unblocked security-critical wire segment.

2.2.4 *Layout-Level Defenses*.

Prior work attempts to thwart fabrication-time attacks by increasing the difficulty of *Trojan Placement*: filling empty space on the IC's device layer with temper-evident functional logic gates [3, 4, 64]. As shown in [54], this approach is only effective for Trojans with large footprints, as filling all placement sites is infeasible [4], and even targeting fill around security-critical logic [3] leaves the IC layout vulnerable to Trojans with small footprints [65]. Orthogonally, *T-TER targets Victim/Trojan Integration by directing protection, at the routing level, around wires Trojans want to attach to.*

2.3 Time-Domain Reflectometry (TDR)

Time-domain reflectometry (TDR) is an electrical analysis technique used to measure physical characteristics about a transmission line (i.e., a wire) such as length, number and distance between impedance discontinuities (e.g., bends), propagation delay, dielectric constant, etc. [15, 21]. Foundries already use TDR to perform root cause analysis on chips that fail post-fabrication testing—often during bring-up of a new process node. TDR works by characterizing a wire within a circuit by injecting a single rising pulse down that wire and analyzing its reflection(s).

2.3.1 *IC Interconnect Models*.

There are two ways to model IC interconnects: lumped and transmission-line models [5]. Lumped interconnect models approximate interconnects using networks of resistors and capacitors. Transmission-line models approximate interconnects as transmission lines with a characteristic impedance and propagation delay.

The choice of interconnect model is a function of maximum frequency component to wire length [49]. A common rule of thumb for IC interconnects is: *a wire is considered a transmission line if its length is greater than ≈10% of the wavelength of the maximum frequency component it transmits [49].* In digital electronics, it is common to think of signals in terms of rise and fall times, rather than maximum frequency component. Thus, one can modify the prior rule of thumb to: *a wire is considered as a transmission line if the transmitted signal rise time, T_{rise}, is less than twice the wire's propagation delay, T_{pd} [49].* Eq. (1) captures this rule of thumb.

$$\text{Model} = \begin{cases} \text{Transmission Line}, & T_{rise} < 2T_{pd} \\ \text{Lumped RC}, & \text{otherwise} \end{cases} \qquad (1)$$

Choosing the right model is vital to understanding operational limitations and ensuring signal integrity within an IC layout. For example, an interconnect that carries a high-speed signal transitions will observe signal reflections from impedance discontinuities that are destructive to the signal integrity of the overall system. Modeling such interconnects using a lumped RC model can hide these destructive effects, while a transmission-line does would not.

2.3.2 ***TDR for IC Fault Analysis.*** By Eq. (1), the faster the rising edge of TDR's incident pulse, the finer-grain of propagation delay changes are detectable. TDR was first developed as a fault-analysis technique for long transmission lines, such as telephone or optical communication lines [41, 46]. As commercial TDR systems became more advanced, TDR became a standard IC packaging fault analysis tool [13, 39, 45]. Researchers have now demonstrated terahertz-level TDR systems capable of locating faults in IC interconnects to nanometer-scale accuracies [11, 37, 50, 52]. With such fine-grain resolution, **TDR is an ideal tamper-analysis technique for ensuring the integrity of the guard wires** used in T-TER (§A).

3 THREAT MODEL

We adopt a threat model in which all phases of the IC design process are trusted *except* fabrication (Fig. 2). The untrusted foundry threat model stems from the extreme ramp-up costs associated with fabricating leading-edge silicon [30, 31] that make outsourcing IC fabrication a necessity—even for nation states. In line with previous untrusted foundry threat models [34, 43, 51, 54, 63, 65], we assume the worst case: that any fabrication-time modifications are carried out by a malicious actor within the foundry (or any foundry partners) that has access to the entire physical layout of the IC in the form of a GDSII file.

While there are many types of hardware Trojans [43] (§2.2), we focus on additive Trojans, rather than subtractive or substitution Trojans. Additive Trojans require implanting additional circuit components and wiring into the IC design. We focus on additive Trojans as there are no documented stealthy and controllable examples of subtractive or substitution Trojans that we are aware of. The closest example of such Trojans are dopant-level Trojans [8, 28, 44], all of which have limited controllability and are detectable with imaging [48].

Previous work shows that to successfully implement an *additive* hardware Trojan, the adversary must complete the three steps—*Trojan Placement*, *Victim/Trojan Integration*, and *Intra-Trojan Routing* [54]—without being exposed. Namely, they must 1) find empty space on the device layer to insert the Trojan's components (logic gates/cells), 2) locate an unblocked segment on a security-critical wire to attach the Trojan to, and 3) route the Trojan components to that unblocked wire segment. They are restricted from modifying the dimensions of the chip and/or violating manufacturing design rules that would risk their exposure. They are allowed to move components and/or existing wiring around, but are constrained by available resources (e.g., time) and correctness from making mass perturbations to the layout. As process technologies scale, manufacturing design rules become increasingly complex [47]. Thus, rearranging components and/or existing wiring comes at a substantial cost. The time to complete any layout modifications, and verify such modifications have not violated design correctness, cannot disrupt the fabrication turn-around time expected by their customers.[3] Additionally, the attacker avoids any modifications that are detectable using existing test-case or side-channel based defenses. While it would be trivial for an attacker with *infinite* time and resources to reverse-engineer the physical layout into HDL, add a Trojan, and re-run the design through the entire IC design process (Fig. 2) thus generating an entirely new layout, such an attack will be infeasible within the hard time limits of fabrication contracts, thus outside the scope of our threat model.

4 TARGETED TAMPER-EVIDENT ROUTING (T-TER)

T-TER aims to make the second step of Trojan insertion—*Victim/Trojan Integration* (§2.2.3)—intractable by shielding the surfaces of targeted wires (interconnects) with tamper-evident guard wires (§2.2.4), creating an additional obstacle for adversaries to overcome. Similar to prior work [3, 4, 35, 54], T-TER is made practical by leveraging the observation that, for most hardware designs, only a subset of the IC is security-critical [17, 23, 35, 53, 66, 67], or the target of a hardware Trojan. In designing T-TER, we pose three questions:

1. *Which wires in the design are security-critical (should be guarded)?*
2. *How can an attacker bypass T-TER guard wires?*
3. *How do we design guard wires that are tamper-evident with respect to bypass attacks?*

4.1 Identifying Security-Critical Nets to Guard

While identifying security-critical *features* in a design is an orthogonal problem—and an ongoing area of research [17, 23, 35, 53, 66, 67]—identifying the *nets (wires)* that comprise said features is the first step in deploying T-TER. Currently, there exist two techniques for identifying security-critical nets: 1) *manual* [17, 23, 35] or 2) *semi-autonomous* [66, 67]. In *manual* identification, a human expert analyzes the design's specification, and the corresponding HDL, and flags nets that implement features critical to the security of software or other hardware that interface to the design [17, 23, 35]. Alternatively, in *semi-autonomous* identification, a set of security-critical nets for a specific design are first manually identified [17, 23], or mined from a list of published errata [66], and either: 1) used to train a classifier that identifies similar nets in other designs [66], 2) expanded using information flow [53] or fan-in analyses [54], or 3) translated to an entirely different design [67]. In this paper, we adopt the most common approach in this area of semi-autonomous identification [3, 54].

[3] Typically, fabrication turn-around times are ≈3 months [29, 56].

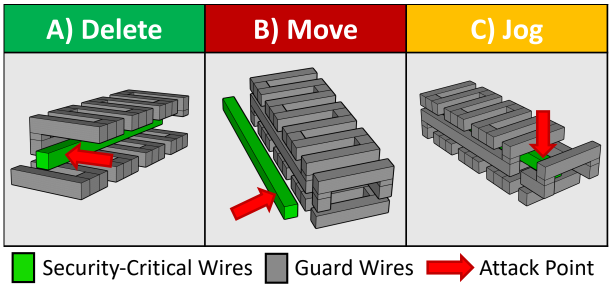

Figure 4: There are three ways an attacker could bypass T-TER guard wires to connect a Trojan wire to a security-critical wire, color-coded by attacker difficulty: A) *delete* guard wire(s), B) *move* an intact set of guard wires, or C) *jog* guard wires out of the way. We study the *jog* attack to assess defensive sensitivity, as it strikes a balance in attacker difficulty, and is the most difficult to detect.

4.2 Guard Wire Bypass Attacks

With T-TER deployed, attackers must *bypass* guard wires—by exposing the surface of a security-critical wire(s)—to complete *Victim/Trojan Integration*, i.e., connect a rogue Trojan wire to a security-critical wire(s) (§2.2.3). Given a set of interconnected guard wires (Fig. 1), there are three ways an attacker can bypass them, color-coded by attacker difficulty (Fig. 4): A) delete, B) move, or C) jog attacks. In a *deletion* attack (Fig. 4A), entire guard wire(s) are removed from the layout. While this attack is easy to implement, it is also easy to defend. A post-fabrication continuity check of a connected set of guard wires will detect a deletion attack. In a *move* attack (Fig. 4B), all interconnected guard wires are left intact, but translated to another location on the chip. Move attacks are the most difficult to implement: an attacker must find a contiguous group of unused routing tracks to translate each set of guard wires too. Even then, a post-fabrication cross-talk analysis between security-critical and guard wires would expose this attack [18, 42]. Lastly, in a *jog* attack, guard wires are *lengthened* to make room for a rogue Trojan wire to connect to a security-critical wire using a via. Jog attacks strike a compromise in terms of implementation difficulty, and are the stealthiest of all bypass attacks. They are easier to implement than move attacks, and are undetectable with post-fabrication continuity tests or cross-talk analyses. *The only artifacts of a jog attack are: 1) a change in the number of bends in the guard wire, i.e. number of impedance discontinuities, and/or 2) an increase in the guard wire's length.* However, nanometer scale TDR [37, 50] detects these changes (§A).

4.3 Tamper-Evident Guard Wires

While techniques for detecting all three bypass attacks exist, each of them requires the ability to measure physical characteristics (e.g., continuity, cross-talk, and length) about a guard wire post-fabrication. *How do we design guard wires whose physical characteristics are tamper-evident post-fabrication?* Based on these considerations, we take a straw-man approach in designing guard wires capable of preventing even the stealthiest of attacks.

4.3.1 *Naïve Approach: Re-purpose Existing Wires.*

One idea for constructing guard wires is to re-purpose existing non-security-critical wires, inherent to the host IC design, as guard wires. Such an approach creates hyper-local routing densities nearby security-critical wires, thus limiting or eliminating the locations where an attacker can attach rogue Trojan wires. By re-purposing pre-existing wires as guard wires, the guard wires incur no hardware overhead. Unfortunately, there are additional routing constraints (e.g., toggle frequency, length, layer, location, timing sensitive, and spacing) that limit the pool of candidate guard wires. Even when such constraints are met, the guard wires are only tamper-evident with respect to deletion and move attacks. For an existing wire to also be tamper-evident with respect to the more stealthy jog and bypass attacks, it must be timing-critical (i.e., if it is made longer, then it will cause timing violations that manifest as run-time errors). As Fig. 6 shows, deployment using existing guard wires is challenging. Namely, the lack of suitable wires in many designs makes it infeasible to block all surfaces of all security-critical wires.

4.3.2 *Designed-in Guard Wires.*

To fill the gaps of existing wires, we propose designed-in guard wires. Designed-in guard wires are **not** inherent to the host IC design. Rather, they are added to the design during the place-and-route IC design phase (Fig. 2). Since they do not implement any circuit functionality, they have fewer routing constraints. As we show in Fig. 6, completely blocking the accessible surface area of all security-critical wires is trivial. While designed-in guard wires incur hardware overhead, i.e., additional wires, they completely block an attacker from attaching a Trojan wire at fabrication time (Victim/Trojan Integration, §2.2.3), as shown in Fig. 7. Additionally, designed-in guard wires are tamper-evident with respect to ***all*** bypass attacks, when coupled with post-fabrication analysis techniques like continuity checking, cross-talk analysis, and time-domain reflectometry (§2.3 and §A), respectively.

There are several designed-in guard wire architectures that may be deployed, listed in order of increasing difficulty of deployment: 1) fully-disjoint, 2) partially-connected, and 3) fully-connected. Fully-disjoint designed-in guard wires are not connected between sides, i.e., the guard wires on each side of a security-critical wire are never connected to one another. Partially-connected guard wires allow for a single guard wire to be utilized on multiple sides. For example, a security-critical wire could be guarded on the north, east, and west sides by a single guard wire that wraps around the security-critical wire. Lastly, fully-connected guard wires are formed when a single guard wire is routed around all sides of all security-critical wires, as shown in Fig. 1.

To detect tampering of designed-in guard wires post-fabrication, their analog characteristics of must be observable. This can be implemented either on-chip, e.g., with internal sensors [25] or ring oscillators [68], or off-chip, e.g., with two I/O pins and a one-time programmable fabric [35]. If fully-joint or partially-connected designed-in guard-wires are deployed, the one-time programmable fabric could be randomly programmed to route both ends of a single (fully-disjoint) or single-set (partially-connected) of guard wire(s) to the two pins. If fully-connected designed-in guard wires are deployed, the one-time programmable fabric is not needed, as both ends of the guard wires set can be routed to the two pins.

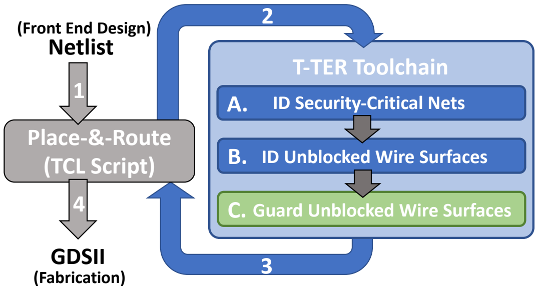
(Front End Design)
Netlist
1
Place-&-Route
(TCL Script)
4
GDSII
(Fabrication)
2
3
T-TER Toolchain
A. ID Security-Critical Nets
B. ID Unblocked Wire Surfaces
C. Guard Unblocked Wire Surfaces

Figure 5: T-TER is an automated toolchain consisting of three phases. Our toolchain first identifies which wires are security-critical, determines potential (unblocked) attachment points, and routes guard wires to block all attachment points. Identified components & wires are placed & routed *before* phase (A) of our toolchain is invoked. Before continuing with the traditional PaR flow, the protected nets and their guard wires are locked in-place to ensure they are untouched throughout the remainder of the layout process.

5 IMPLEMENTATION

We develop an automated toolchain for deploying T-TER in modern IC designs. Our toolchain integrates with existing IC design flows (Fig. 2) that utilize commercial VLSI CAD tools. Specifically, we implement the T-TER toolchain around the Cadence Innovus Implementation System [10], a commercial place-and-route (PaR) CAD tool. The toolchain is invoked by modifying a place-and-route TCL script,[4] as shown in Fig. 5.

5.1 Place-&-Route Process

The PaR design phase (Fig. 2) is typically automated by a CAD tool, programmatically driven by TCL script(s). There are several steps to PaR that are performed in the following order: 1) floor-planning, 2) placement, 3) clock tree synthesis, 4) routing, and 5) filling. *To ensure that all guard-wires are routed optimally, we modify the order of these PaR steps.* Specifically, after floor-planning (1), we use our automated toolchain to place identified components *and* route identified wires and their guard wires. Our toolchain then permanently fixes the locations of these components and wires to prevent the PaR CAD tool from modifying their positions and/or shapes throughout the remainder of the PaR process. Lastly, we utilize the PaR CAD tool to place all other components (2), synthesize the clock tree (3), route remaining wires(4) and fill the design with filler (capacitor) cells.

5.2 Automated Toolchain

The T-TER toolchain automates the insertion of either existing or designed-in guard wires around wires in need of protection. The toolchain consists of three main phases (Fig. 5). The first phase (A) identifies security-critical nets. The second phase (B) identifies the unblocked surfaces of all of these nets within a GDSII-encoded layout. The last phase (C) guards the nets and their influencer nets by routing guard wires nearby. We provide additional implementation details on all three stages of the T-TER toolchain below.

5.2.1 ***Identifying Nets.*** The first phase of T-TER requires identifying nets in the design to guard, i.e., nets that are security-critical. Phase A of our toolchain (Fig. 5A) utilizes a semi-autonomous approach to identifying such nets (§4.1). Specifically, our toolchain assumes the designer has *manually* flagged a set of *root* security-critical nets in the behavioral-level HDL by appending a unique prefix—*secure_*—to each signal (net) name. During PaR, our toolchain performs a data-flow analysis of the circuit netlist to locate the direct fan-in—to a configurable depth—of each root net. Since the netlist is often modified by PaR CAD tools to meet various design constraints (e.g., power, performance, and area), we disable the optimization of all root nets during PaR. Given the interconnected nature of nets within an IC design, an adversary may elect to target a net that influences a root net, rather than the root net itself. Our toolchain addresses this indirection, using an autonomous approach that widens the set of *targeted* nets to the root nets and those that influence root nets (to a designer configurable degree). The remainder of our tool flow focuses on protecting this set of targeted nets.

Our fan-in analysis tool is a custom-backend to the Icarus Verilog (IVL) front-end Verilog compiler [61], and is implemented in C++. It performs a breadth-first search over the circuit-level data-flow graph generated by IVL. We release our fan-in analysis tool under an open-source license.

5.2.2 ***Identifying Unblocked Wire Surfaces.*** The second phase of T-TER is identifying the unblocked surfaces of targeted nets in a physical IC layout, i.e., potential locations of Trojan wire attachment. To do so, we implement, and open-source, a Python tool that analyzes the GDSII layout file containing only the placed-and-routed targeted components and wires. Our tool implements a 3-D scanning window approach to search the 3-D boundary surrounding each targeted wire, and compute the areas on each wire's surfaces that are not blocked by other wires or circuit components. While it is traditional for designers to only route wires on defined *routing tracks*, i.e., on a pre-defined routing grid, it may be possible for an attacker to route Trojan wires off this grid, so long as they maintain the minimum spacing requirements dictated by the manufacturing design rules. Thus, our tool takes a conservative approach when scanning for unblocked wire surfaces, only scanning the 3-D boundary surrounding each targeted wire that extends up to the minimum-spacing requirements defined for the given, and adjacent (top/bottom), routing layers. If and only if another component or wire overlaps a region of the 3-D boundary surrounding a targeted wire, that surface region will be considered blocked. The output of this stage of our toolchain is a list of coordinates within the 3-D place-and-route grid that must be filled with guard wires during the next phase in the T-TER toolchain.

5.2.3 ***Guard Unblocked Wire Surfaces.*** The last stage of the T-TER toolchain (Fig. 5) is a custom guard wire routing tool, also implemented in Python. It takes as input exact locations of targeted wires and their unblocked sides (output from Phase B, §5.2.2) and generates a TCL script that integrates with the Cadence Innovus Digital Implementation platform [10] to automatically route the guard wires. This TCL script is executed immediately after the targeted wires have been routed, but before placing the remaining

[4] Tool Command Language (TCL) scripts are the standard programmatic interface to commercial VLSI CAD tools. IC designers often develop a set of scripts for driving the CAD tools that automate most of the IC design process (Fig. 2).

components. Depending on the guard wires being deployed, existing or designed-in, different guard wire TCL scripts are generated (described below).[5] Note, in either case, our toolchain routes guard wires that are compliant with all manufacturing design rules.

There are numerous ways *existing guard wires* can be implemented. Since commercial PaR CAD tools do not offer an interface to enable fine-grain constraints between two unrelated signal wires, we develop an indirect method for implementing existing guard wires. We implement existing guard wires by constraining placement and routing resources nearby targeted wires. First, we identify all circuit components (i.e., logic gates) connected to all targeted wires, i.e., targeted components. Next, we draw a bounding box around these components and extend this boundary vertically by $Y\%$ of the overall box height, and horizontally by $X\%$ of the overall box width. Then, we set placement and routing density screens in the portion of the IC layout that lies *outside* the bounding box. These constraints limit the placement and routing resources outside the bounding box, thus forcing more components and wiring within the bounding box. With increased routing density nearby targeted wires, they are less accessible by Trojan payload delivery wires. The values of X, Y, and density screen configuration settings are optimized to maximize the net blockage metric computed by the GDS2Score metric.

Designed-in guard wires are more straightforward to implement. The automated guard wire deployment toolchain locates all unblocked surfaces (north, south, east, west, top, and bottom) of all targeted wires and routes guard wires in these regions. After all guard wire segments are routed, they are connected according to the architecture chosen (§4.3.2).

6 EVALUATION

We evaluate T-TER in three areas. First, we explore the effectiveness of T-TER at closing the fabrication-time attack surface of three security-critical features within an open-source System-on-Chip (SoC), with regard to the stealthiest additive Trojan known: the A2 Trojan [65]. We compare the capabilities of T-TER with existing state-of-the-art layout-level defenses [3, 4, 64]. Next, we demonstrate the practicality of T-TER, analyzing its power, performance, and area overheads. Finally, in Appendix A, we perform a threat assessment, demonstrating how guard wires are tamper-evident.

6.1 Experimental Setup

6.1.1 ***Surrogate SoC.*** We utilize the open-source Common Evaluation Platform (CEP) SoC design [36] for our evaluation. The CEP platform is designed as a surrogate SoC system for testing a variety of DoD-oriented IC technologies. It contains a general-purpose processor core, five cryptographic cores, four digital signal processing cores, and a GPS core. We focus on three cores from in the SoC: the *processor* core, the *DFT* core, and the *AES* core. The OR1200 processor[6] is a 5-stage pipelined CPU that implements a 32- bit OR1K instruction set and Wishbone bus interface [40], and is the same design used in previous fabrication-time attack studies [54, 65]. It supports Linux via BusyBox [57]. The AES core supports 128-bit key sizes. The DFT accelerator implements a Discrete Fourier Transform algorithm, a common component of radar and other sensing systems.

Table 1: A2 Trojans used in T-TER effectiveness assessment.

Trojan	# Std Cells	# Placement Sites	Timing Critical?
A2 Analog [65]	2	20	✗
A2 Digital [65]	91	1444	✓

We target a 45 *nm* Silicon-On-Insulator (SOI) process technology with 10 available routing layers. We synthesize our design with Cadence Genus (v16.23), and placed-and-route it using Cadence Innovus (v17.1). All layout variations of our SoC target a 100 *MHz* clock frequency and a core density of 60–80%. All CAD tools are run on a server with 2.5 *GHz* Intel Xeon E5-2640 CPU and 64GB of memory, running Red Hat Enterprise Linux (v6.9).

6.1.2 ***A2 Trojan.*** The goal of T-TER is to protect security-critical features within SoCs from the stealthiest additive Trojan currently known, the A2 Trojan [65]. The A2 Trojan is stealthy, i.e., evades current *prevention* and *detection* defenses, due to its small size and complex triggering mechanism. When implemented within our surrogate SoC, in a 45 *nm* process, the analog variant of the A2 Trojan [65] requires only two additional cells that occupy 20 placements sites, while the entirely digital variant of the same attack requires 91 additional cells that occupy 1,444 placement sites. The analog A2 attack is *not* timing critical: the Trojan components may be placed anywhere on the placement grid, at any distance from the Victim/Trojan integration point. Conversely, the digital A2 attack *is* timing-critical: the length of the interconnect between the Trojan components and the Victim/Trojan integration point must be within three standard deviations from the mean net length in the overall SoC (this is an entirely worst-case estimate borrowed from [54]). We summarize the placement and routing resource requirements for the two variants of the A2 Trojan in Table 1.

6.1.3 ***Exemplar Nets of Interest.*** For this evaluation, we need to protect nets that our example Trojan might want to use as integration points. Leveraging existing hardware Trojan payloads, we select three reference integration targets within our SoC design to protect with T-TER:

1. processor supervisor bit (supv),
2. DFT computation ready interrupt (next_out),
3. cryptographic key bits (key [0:127]).

The most popular hardware Trojans leverage the supervisor (supv) net as part of privilege escalation attacks [16, 26, 65]. Alternatively, hardware Trojans can also hide specific computations or state transitions, e.g., a Trojan that disables the DFT computation-ready interrupt signal (or *next_out* signal) that informs the CPU when a DFT computation is ready. Lastly, another popular hardware Trojan seeks to leak cryptographic key bits via side channels [34]. The A2 trigger can be attached to any of the nets that carry these signals

[5] While existing guard wires fail to defend against all types of guard wire attacks (§4.3.1), we implement a tool to deploy them in order to empirically show they are also inferior to designed-in guard wires in terms of surface-are coverage (Figs. 6 & 7), and thus should not be used in a security context.

[6] We use the OR1200 version of the CEP rather RISC-V version since the OR1200 is the processor used in the A2 Trojan [65]. We are not aware of similar Trojans available in the RISC-V. We expect similar results for the RISC-V version of the CEP since both processors are RISC-based, in-order, scalar, pipelined, capable of running Linux, and operate at similar clock frequencies. Thus, from an IC layout perspective, they have similar features (e.g., wire lengths) and will have similar hardware overheads.

to mount an attack, so we protect the interconnects that comprise these nets.

The initial stage (Fig. 5A) of our automated T-TER toolchain assumes the designer has manually annotated the root nets they have chosen to target with T-TER (§5.2.1). Thus, we manually annotate the above net (signal) definitions with the prefix *secure_* within our SoC design's RTL. We then synthesize and place-and-route our design prior to generating a final, optimized, netlist for which our toolchain computes the fan-in to each manually annotated net—to a depth of two layers of logic gates—thereby expanding the final set of all targeted nets (i.e., those guarded by T-TER). Fig. 8 (far right) shows the number of interconnect wires that comprise each set of nets that implement the aforementioned features within our surrogate SoC.

6.2 Effectiveness

We first evaluate the effectiveness of T-TER in thwarting the insertion of hardware Trojans at fabrication time. We compare the degree of protection provided by T-TER with that provided by deploying the current state-of-the-art *preventive* defense suggested by Ba *et al.* [3, 4]. This placement-based defense involves filling as many empty placement sites as possible (they show filling 95% of all placement sites is the max feasible), prioritizing empty sites nearest security-critical nets. We use our automated toolchain (§5.2) to deploy both types of guard wires (existing and designed-in). We assume the best case scenario for Ba *et al.'s* placement defense [3, 4] by filling 95% of the device layer with inverter cells—the smallest cells in our 45 *nm* cell library, for fine grain filling.

We use the ICAS framework [54] to quantify the effectiveness of each defense. ICAS analyzes the physical layout of an IC (encoded in a GSDII file), and computes security metrics detailing the IC layout's fabrication-time attack surface. Namely, it computes three metrics: 1) *trigger space*, 2) *net blockage*, and 3) *route distance*. The ***trigger space*** metric characterizes the open space on the device layer (empty placement sites) available for an attacker to add their Trojan components. The ***net blockage*** metric computes the percentage of surface area of identified nets that are blocked by other circuit components or wiring. Lastly, the ***route distance*** metric computes the minimal distance between unblocked identified nets and unused placement sites that an adversary would have to route a rogue Trojan wire to "connect" the hardware Trojan to the host IC. The trigger space metric quantifies the difficulty of performing *Trojan Placement*, the net blockage quantifies the difficulty of performing *Trojan/Victim Integration*, and the route distance metric quantifies the difficulty of performing *Intra-Trojan Routing* (§2.2.3). Of the three ICAS metrics, the *net blockage* metric is most adept to quantifying the deployability of each guard wire type (existing and designed-in), i.e., how effective each guard wire type is at shielding all targeted nets. Alternatively, the *route distance* metric is the adept at comparing T-TER with Ba *et al.*'s placement defense, as it is essentially a combination of the *trigger space* metric—an entirely placement-focused metric—and the *net-blockage* metric—an entirely routing-focused metric. Therefore, we utilize these two ICAS metrics in the following evaluation.

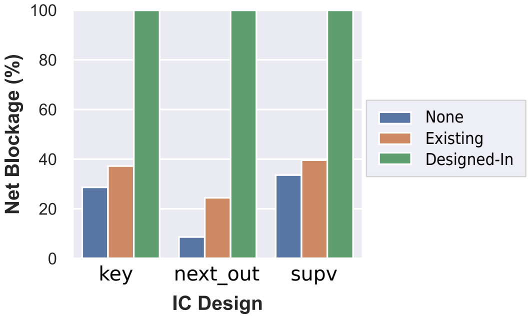

Figure 6: Plot of the *net blockage* [54] computed across three different sets of targeted nets within our SoC layout, with and without guard wires.

6.2.1 *Net Blockage Results.*

Both existing and designed-in guard wires attempt to block targeted nets to prevent attackers from attaching rogue wires to them, thus minimizing/eliminating possible *Victim/Trojan Integration* points (§2.2.3). We use the *net blockage* metric to compute the surface-area-coverage differences between existing and designed-in guard wires. Fig. 6 compares the net blockage computed across three total IC layouts of the same SoC design, including: three guard wires variations—without guard wires, with existing guard wires, and with designed-in guard wires—across three different sets of targeted nets. All net-blockage results are with respect to each set of targeted nets in the SoC.

Across all three sets of targeted nets, designed-in guard wire provide more protection than existing guard wires, as expected. Specifically, for all nets, designed-in guard wires achieve 100% net blockage. This means that there is no place on any targeted net within the SoC where an attacker can attach a rogue wire. Existing guard wires are unable to achieve 100% coverage due mainly to having to meet their own routing constraints which prevents our tool from locating enough nets to block all surfaces of all targeted nets, making them ineffective at thwarting attacks.

6.2.2 *Route Distance Results.*

Since T-TER only limits the routing resources needed to insert a Trojan at fabrication time, it is vital to understand how T-TER reduces the overall fabrication-time attack surface, i.e., both Trojan routing *and* placement resources. We use the *route distance* metric to locate all possible combinations of unused placement sites and unblocked targeted nets—i.e., all possible Trojan attack configurations [54]. We use the *route distance* metric to illustrate the attack surface across each core within our SoC where that contains the root net of interest. We analyze the *route distance* metric with respect to each containing core, as it is common practice for IC layout engineers to lay out each core separately, before integrating them, plus this increases the clarity of presentation.

Fig. 7 shows the *route distance* metric as computed across all three containing cores, with and without layout-level defenses including: 1) T-TER (both existing and designed-in guard wires) and 2) defensive placement. Each heatmap is intended to be analyzed column-wise, where each column is a histogram of the distances

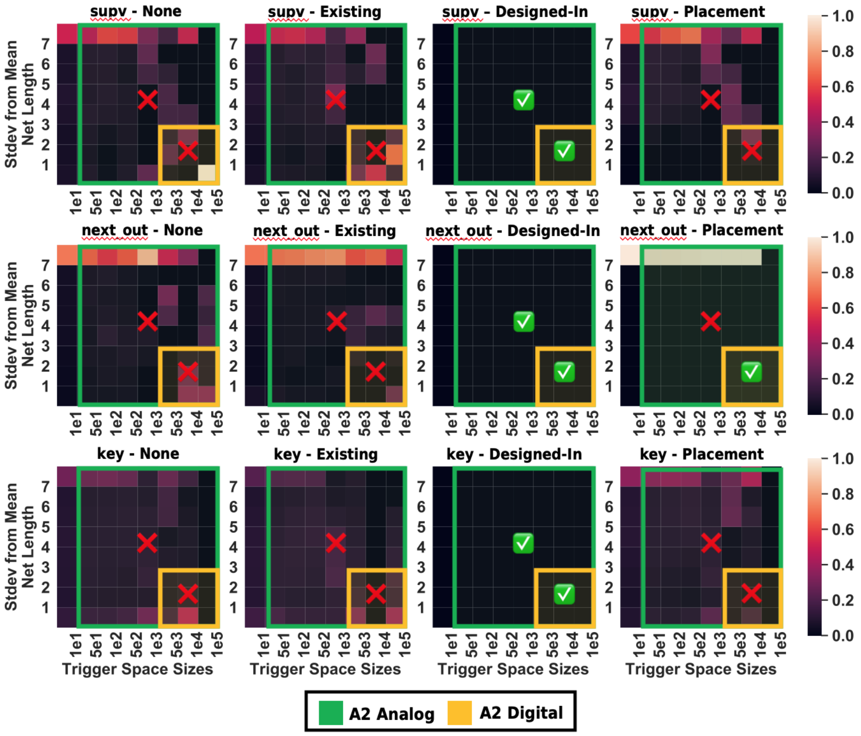

Figure 7: Plot of the ICAS *route distance* metric [54] computed across four different layouts of each core within our surrogate SoC, with and without guard wires and Ba *et al.*'s defensive placement [3, 4]. Each heatmap illustrates the percentage of (targeted net, trigger-space) pairs (possible Trojan layout implementations) of varying distances apart. The heatmaps are intended to be analyzed by column, as each column encodes a histogram of possible attack configurations with trigger-spaces of a given size range (X-axis). Route distances (Y-axis) are displayed in terms of standard deviations from mean net length in each respective design. Heatmaps that are completely dark indicate no possible attack configurations exist, i.e., no placement/routing resources to insert any Trojan. Overlaid on each heatmap are rectangles indicating regions on the heatmap a given A2 Trojan (Tab. 1) may exploit, and markers (checks and x-marks) indicating if a non-zero number of specific Trojan layout implementations are possible.

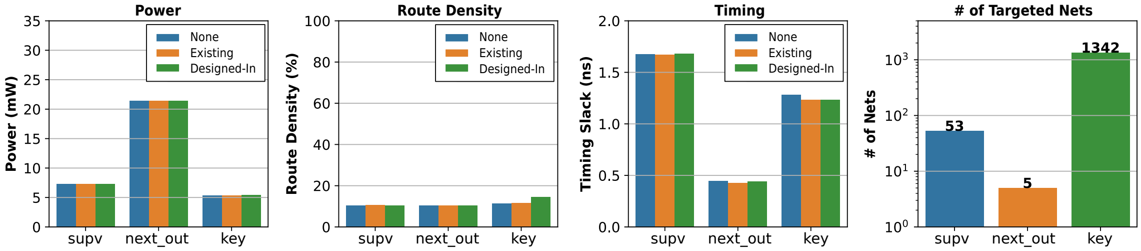

Figure 8: T-TER hardware overheads. The far right plot shows the number of wire (route) segments that implement the labeled security-critical feature (set of nets) in our surrogate SoC.

between unblocked targeted nets and trigger-spaces[7] within a size range. Namely, each heatmap illustrates the fabrication-time attack surface of each IC layout. If a circuit has no attack configurations, i.e., all targeted nets are blocked or there are no trigger-spaces, the route distance heatmap is completely dark (column ratios of 0). If it is impossible to eradicate all attack configurations, the most secure layout for such a circuit would have maximum distances between unblocked targeted net and trigger-spaces, i.e., a heatmap with the top row the lightest color (top row ratios of 1). This is because larger distances increase the signal delay for the hardware Trojan; increasing the challenge of the attacker to meet timing constraints for their attack. Overlaid on each heatmap are rectangles indicating the region of the attack surface that is exploitable by the color-coded Trojan, and check- or x-marks indicating whether any possible attack configurations exist for that attack. A check-mark indicates there are zero possible Trojan layouts (success)), where an x-mark indicates the opposite (vulnerable).

[7] Trigger spaces are contiguous groups of placement sites that are empty, or contain (removable) capacitive fill cells [54]

Designed-in guard wires outperform existing guard wires and placement-centric defenses. For all three example attack payloads, designed-in guard wires were able to close the fabrication-time attack-surface by *completely* blocking all targeted nets (Fig. 6). Therefore, even the stealthiest A2 Trojan [65] cannot be utilized to attack the features-of-interest within our SoC.

6.3 Practicality

T-TER is effective, but is it practical? We evaluate the cost of deploying T-TER across three exemplar security-critical features within our SoC that have been subject to attack. Specifically, we analyze the power, route density, and performance (timing) overheads incurred by deploying both existing and designed-in guard wires from §6.2. Note, while T-TER guard-wires can be deployed on any routing layer, we chose to prioritize routing security-critical nets on metal layers *three* and *four* (out of 10 total layers) to measure overheads in the worst case, i.e., guard wires routed on layers 2–5. Measurements are taken with respect to each feature's containing core, similar to the route distance measurement. While it is common to analyze power, performance, and area, of an IC design, we instead analyze power, performance and *route density*. Area measurements refer to the device-layer area, i.e., width and length, since the height (number of routing layers) is fixed for a given process technology. Since T-TER does not require additional logic gates, we do not increase the width and height (area) of the core area, rather T-TER alters the total wire length in the design. Thus, measuring routing density overhead is more meaningful. We use the built-in features of Cadence tools to compute these overheads.

Fig. 8 shows our results. Power and timing overheads were both less than 1%. In some cases, the timing was better for the guard wire designs. This is expected as T-TER does not require any additional logic gates, nor lengthen existing wires. Rather, the guard wires increase routing constraints that can push the PaR CAD tool to produce more optimal routing solutions. The route density overhead was less than 1% for all existing guard wires, and similar for designed-in guard wires when the number of targeted nets to guard is small, namely the *supv* and *next_out* nets. Intuitively, the more guard wires inserted, the higher the routing density increase. Keeping route density low is important to ensure automated CAD tools can route each design. However, even though all layouts targeted a placement density (density of logic gates on the device layer) of 60–80%, route density was relatively low even with guard wires. This was due to the characteristics of the designs and process technology (i.e., back-end-of-line metal stack option).

It is worth noting that in addition to low power, performance, and area overheads, deploying T-TER guard wires has minimal impact on the run-time of layout CAD tools. Without DR, the tools lay out each SoC core in less than 10 minutes, and with DR they lay out each core in less than 11 minutes. Tool run-time overheads are more impacted by the magnitude of features requiring protection than on circuit complexity.

7 DISCUSSION

T-TER aims to prevent fabrication-time Trojan attacks that target specific security-critical features in an IC design. Experiments on real circuit layouts of a SoC containing show that T-TER is effective, deployable, and tamper-evident. Discussed below are the limitations, scalability, signal integrity impact, flexibility, and extensibility of T-TER.

Limitations. T-TER is a mitigation strategy for hardware designs where only a subset of the design is security-critical [17, 53]. As our evaluation results show, the deployability and performance overhead of T-TER is low when the overall security-critical wire length is low. If *every* wire in a design is security-critical, then T-TER is not a good defensive strategy; in fact, the motive for outsourcing fabrication in such scenarios is tenuous. If fabrication must be outsourced, we recommend alternative mitigation strategies such as those proposed in [3, 4, 22, 35, 64]. The tradeoff is that these strategies have limited deployability, and a large, fixed, performance overhead that make them impractical for designs that require only a subset of security-critical functionality be protected.

Scalability. There are two notions of scalability to address. The first is scalability with regard to *routability*. Routing guard wires alongside security-critical wires can impact the routability of a layout, if the 1) ***percentage of overall wire length*** to guard, and 2) ***route-density without guard wires*** are both large. By placing and routing security-critical components and wires first, before any other portions of the circuit (§5.1), we are able to minimize security-critical wire length. This makes security-critical wire length scale with the total length of security-critical wires, as opposed to the size of the overall design. As we see when going from OR1200 and RISC-V class processor to modern x86-64 processors, the proportion of security-critical functionality (hence wires) decreases as relatively more transistors are spent on performance. Moreover, by deploying T-TER within advanced process nodes—which is the motivating threat model—route density is minimized since these nodes provide multiple metallization options[8] with 10 (or more) routing layers. To demonstrate this empirically, we highlight the AES core (Fig. 8–Route Density), where we guard over 1000 nets with little impact on power or performance. In fact, the reason we select the AES as a benchmark—even though it is arguably entirely security-critical—is because ***its key-bit nets exhibit a unique quality that stress tests T-TER***. Specifically, they are global, highly-connected routes that are orders-of-magnitude longer than any other nets in the layout.

The second notion of scalability is with regard to the detection of bypass attacks. Although Moore's law is near its limit, transistors continue to shrink. Only three companies in the world are capable of manufacturing 7–10 *nm* transistors [31]. It is, therefore, vital for T-TER to scale with process technology. With respect to deletion attacks (Fig. 4A), T-TER scales with process technology advances as measuring interconnect continuity does not differ across process technologies. With respect to move attacks (Fig. 4B), T-TER scales with process technology advances as cross-talk is amplified when interconnects are smaller and more densely packed. Lastly, with respect to jog attacks, T-TER also scales, as TDR capabilities directly scale with microelectronic feature sizes, i.e., faster transistors translates to faster TDR rise times.

[8] The *metallization option* defines the total number (and physical characteristics) of available routing (metal) layers defined within an IC's process technology.

Signal Integrity Impact. Routing long wires parallel to targeted nets increases coupling capacitance, thus creating cross-talk between the guard wires and the targeted nets they protect. However, designed-in guard wires are not actively driven during normal chip operation, and can be permanently grounded (using a one-time programmable fabric) after TDR analysis. Thus, cross-talk is not an issue—in fact, designed-in guard wires decrease cross-talk by acting as shields between targeted nets and the rest of the circuit.

Extensibility of CAD Tools. Our T-TER deployment framework (§5) is built on top of a commercial IC CAD tool [10] and an open-source VLSI analysis tool [54]. Extending T-TER to work across other commercial IC layout CAD tools involves incorporating support for each vendor's CAD tool APIs. In the future, we see T-TER deployed as an integrated component of commercial VLSI CAD tools.

8 RELATED WORK

Fabrication-time attacks and defenses have been extensively studied. Attacks have spanned the trade-space of footprint size, stealth, and controllability. Specifically, some attacks have demonstrated stealth and controllability, at the cost of large footprints [8, 26, 34], while others have demonstrated small (or non-existent) footprints, at the cost of controllability and stealth [28, 44]. The most formidable attack—the A2 attack [65]—has demonstrated all three: small footprint, stealth, and controllability.

On the defensive side, there are two main strategies: detective or preventive. Most prior work has focused on detective strategies, while few works have focused on preventive strategies. Detective strategies involve side-channel analysis [1, 6, 24, 38], imaging [69], and on-chip sensors [14, 20, 32]. Until T-TER, preventive measures have been placement-focused [3, 4, 64].

Fabrication-time Attacks. The first fabrication-time insertion of a hardware Trojan was developed by Lin *et al.* [34] who proposed a Trojan designed to leak information over a deliberately created side channel. Specifically, they designed and implemented a hardware Trojan, with a footprint of approximately 100 logic gates, to create an artificial power side channel for leaking cryptographic keys. Albeit unique at the time, today such a large footprint makes the attack detectable via side channel defenses [1, 6, 14].

The most lethal fabrication-time attack is the A2 Trojan, developed by Yang *et al.* [65]. The A2 Trojan utilizes analog components to build a counter-based trigger circuit with a footprint of less than the size of one flip-flop. Its complex triggering mechanism makes it stealthy, i.e., unlikely to accidentally deploy during post-fabrication functional testing or under normal chip operation, yet is controllable from user-level software. Its unique design makes it the only Trojan to evade all detection schemes, except T-TER.

Fabrication-time Defenses. The first side-channel detection scheme was proposed by Agrawal *et al.* [1]. They used power, temperature, and electromagnetic (EM) side-channel measurements to record a fingerprint of a "golden" IC during normal, and compared this fingerprint to one acquired from an untrusted IC. Similarly, Jin *et al.* [24] create a timing-based fingerprint obtained by measuring the output delays resulting from applying various input combinations to a given IC. While side-channel detection schemes are effective against hardware Trojans with large footprints, they fail at detecting Trojans like A2 [65], whose side-channel signatures are well below operational noise margins.

Like side-channel detection, imaging is another detective defense. Specifically, backside imaging is a non-destructive technique that can resolve device-layer components (Fig. 3) as this layer isn't blocked by any wires. Zhou *et al.* [69] propose filling the placement grid with highly reflective fill cells, as opposed to the standard fill cells used, to encode a watermark that can be captured using backside imaging. Thus, if the watermark has been perturbed during fabrication, an attack has occurred. Unfortunately, this technique requires hours to image a single IC, provides no visibility in metal layers, and its resolution seems capped at 45 nm processes.

Of all fabrication-time Trojan defenses, R2D2 [20] is the only one that claims to detect the A2 Trojan. R2D2 works by using on-chip sensors to monitor the toggling frequency of a select few security-critical signals within the design. If the toggling rate of any security-critical signals exceed a pre-determined threshold, then an alarm signal is activated to indicate an A2 Trojan may have been triggered. The crux of this approach is that, unlike T-TER guard wires, the hardware used to construct the toggle frequency monitors *is not tamper-evident*. There is no way to tell if a foundry-side attacker disabled the R2D2 hardware while inserting her Trojan.

9 CONCLUSION

T-TER is a routing-centric preventive defense against additive fabrication -time Trojans that target security-critical hardware features. It makes routing Trojan wires to, or directly adjacent to, attacker-targeted wires in a victim IC intractable by surrounding their surfaces with tamper-evident guard wires. We propose the use of designed-in guard wires in conjunction with post-fabrication terahertz time-domain reflectometry (TDR) analysis to detect *all* bypass attacks we contrive (*deletion*, *move*, and *jog* attacks). We develop an automated toolchain for deploying T-TER guard wire. Lastly, we evaluate the effectiveness, deployability, and tamper-evidence of T-TER at securing multiple security-critical features within an SoC that have been subject to attack by existing hardware Trojans. Our results show that T-TER thwarts the insertion of even the stealthiest known additive hardware Trojan—the A2 Trojan—with power, timing, and area overheads of ≈ 1%.

ACKNOWLEDGMENTS

We thank Brian Tyrrell, Matt Guyton, and other members of the MIT Lincoln Laboratory community for their thoughtful feedback that enhanced the quality of our work.

DISTRIBUTION STATEMENT A. Approved for public release. Distribution is unlimited. This material is based upon work supported by the Under Secretary of Defense for Research and Engineering under Air Force Contract No. FA8702-15-D-0001, and by the National Science Foundation Graduate Research Fellowship Program under Grant No. DGE 1256260. Any opinions, findings, conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the funding agencies.

REFERENCES

[1] Dakshi Agrawal, Selcuk Baktir, Deniz Karakoyunlu, Pankaj Rohatgi, and Berk Sunar. 2007. Trojan Detection using IC fingerprinting. In *IEEE Symposium on*

Security and Privacy (S&P).

[2] Yousra Alkabani and Farinaz Koushanfar. 2008. Designer's hardware Trojan horse. In *IEEE International Workshop on Hardware-Oriented Security and Trust (HOST).*

[3] Papa-Sidy Ba, Sophie Dupuis, Manikandan Palanichamy, Giorgio Di Natale, Bruno Rouzeyre, et al. 2016. Hardware Trust through Layout Filling: a Hardware Trojan Prevention Technique. In *IEEE Computer Society Annual Symposium on VLSI (ISVLSI).*

[4] Papa-Sidy Ba, Manikandan Palanichamy, Sophie Dupuis, Marie-Lise Flottes, Giorgio Di Natale, and Bruno Rouzeyre. 2015. Hardware Trojan prevention using layout-level design approach. In *European Conference on Circuit Theory and Design (ECCTD).*

[5] Halil B Bakoglu. 1990. Circuits, Interconnections, and Packaging for VLSI.

[6] Josep Balasch, Benedikt Gierlichs, and Ingrid Verbauwhede. 2015. Electromagnetic circuit fingerprints for hardware trojan detection. In *IEEE International Symposium on Electromagnetic Compatibility (EMC).*

[7] Mark Beaumont, Bradley Hopkins, and Tristan Newby. 2011. *Hardware trojans-prevention, detection, countermeasures (a literature review).* Technical Report. Defence Science and Technology Organization Edinburgh (Australia).

[8] Georg T Becker, Francesco Regazzoni, Christof Paar, and Wayne P Burleson. 2013. Stealthy dopant-level hardware trojans. In *International Workshop on Cryptographic Hardware and Embedded Systems (CHES).*

[9] Duane Boning and Sani Nassif. 2000. Models of process variations in device and interconnect. *Design of high performance microprocessor circuits* (2000).

[10] Cadence Design Systems. [n. d.]. Innovus Implementation System. https://www.cadence.com/content/cadence-www/global/en_US/home.html.

[11] Yongming Cai, Zhiyong Wang, Rajen Dias, and Deepak Goyal. 2010. Electro Optical Terahertz Pulse Reflectometry—an innovative fault isolation tool. In *Electronic Components and Technology Conference (ECTC), 2010 Proceedings 60th.*

[12] Rajat Subhra Chakraborty, Seetharam Narasimhan, and Swarup Bhunia. 2009. Hardware Trojan: Threats and emerging solutions. In *IEEE International High Level Design Validation and Test Workshop (HLDVT).* IEEE.

[13] Ming-Kun Chen, Cheng-Chi Tai, and Yu-Jung Huang. 2006. Nondestructive analysis of interconnection in two-die BGA using TDR. *IEEE Transactions on Instrumentation and Measurement* (2006).

[14] Domenic Forte, Chongxi Bao, and Ankur Srivastava. 2013. Temperature tracking: An innovative run-time approach for hardware Trojan detection. In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD).*

[15] Leonard A Hayden and Vijai K Tripathi. 1994. Characterization and modeling of multiple line interconnections from time domain measurements. *IEEE Transactions on Microwave Theory and Techniques* (1994).

[16] Matthew Hicks, Murph Finnicum, Samuel T. King, Milo M. K. Martin, and Jonathan M. Smith. 2010. Overcoming an Untrusted Computing Base: Detecting and Removing Malicious Hardware Automatically. In *IEEE Symposium on Security and Privacy (S&P).*

[17] Matthew Hicks, Cynthia Sturton, Samuel T. King, and Jonathan M. Smith. 2015. SPECS: A Lightweight Runtime Mechanism for Protecting Software from Security-Critical Processor Bugs. In *International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS).*

[18] Simon Hollis and Simon W Moore. 2006. RasP: an area-efficient, on-chip network. In *2006 International Conference on Computer Design.* IEEE, 63–69.

[19] Simon J Hollis. 2009. Pulse generation for on-chip data transmission. In *2009 12th Euromicro Conference on Digital System Design, Architectures, Methods and Tools.* IEEE, 303–310.

[20] Yumin Hou, Hu He, Kaveh Shamsi, Yier Jin, Dong Wu, and Huaqiang Wu. 2018. R2D2: Runtime reassurance and detection of A2 trojan. In *International Symposium on Hardware Oriented Security and Trust (HOST).* IEEE.

[21] Ching-Wen Hsue and Te-Wen Pan. 1997. Reconstruction of nonuniform transmission lines from time-domain reflectometry. *IEEE Transactions on Microwave Theory and Techniques* (1997).

[22] Frank Imeson, Ariq Emtenan, Siddharth Garg, and Mahesh Tripunitara. 2013. Securing Computer Hardware Using 3D Integrated Circuit (IC) Technology and Split Manufacturing for Obfuscation. In *USENIX Security Symposium.*

[23] Yier Jin, Nathan Kupp, and Yiorgos Makris. 2010. DFTT: Design for Trojan test. In *IEEE International Conference on Electronics, Circuits, and Systems (ICECS).*

[24] Yier Jin and Yiorgos Makris. 2008. Hardware Trojan detection using path delay fingerprint. In *IEEE International Workshop on Hardware-Oriented Security and Trust (HOST).*

[25] Shane Kelly, Xuehui Zhang, Mohammed Tehranipoor, and Andrew Ferraiuolo. 2015. Detecting hardware trojans using on-chip sensors in an ASIC design. *Journal of Electronic Testing* 31, 1 (2015), 11–26.

[26] Samuel T. King, Joseph Tucek, Anthony Cozzie, Chris Grier, Weihang Jiang, and Yuanyuan Zhou. 2008. Designing and Implementing Malicious Hardware. In *Proceedings of the Usenix Workshop on Large-Scale Exploits and Emergent Threats (LEET).*

[27] Angus I Kingon, Jon-Paul Maria, and SK Streiffer. 2000. Alternative dielectrics to silicon dioxide for memory and logic devices. *Nature* (2000).

[28] Raghavan Kumar, Philipp Jovanovic, Wayne Burleson, and Ilia Polian. 2014. Parametric trojans for fault-injection attacks on cryptographic hardware. In *Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC).*

[29] Mark Lapedus. 2017. Battling Fab Cycle Times. https://semiengineering.com/battling-fab-cycle-times/.

[30] Mark Lapedus. 2018. Big Trouble At 3nm. https://semiengineering.com/big-trouble-at-3nm/.

[31] Mark Lapedus. 2018. GF Puts 7nm On Hold. https://semiengineering.com/gf-puts-7nm-on-hold/.

[32] Jie Li and John Lach. 2008. At-speed delay characterization for IC authentication and Trojan horse detection. In *IEEE International Workshop on Hardware-Oriented Security and Trust (HOST).*

[33] Jun Jun Lim, Nor Adila Johari, Subhash C Rustagi, and Narain D Arora. 2014. Characterization of Interconnect Process Variation in CMOS Using Electrical Measurements and Field Solver. *IEEE Transactions on Electron Devices* (2014).

[34] Lang Lin, Markus Kasper, Tim Güneysu, Christof Paar, and Wayne Burleson. 2009. Trojan Side-Channels: Lightweight Hardware Trojans through Side-Channel Engineering.. In *International Workshop on Cryptographic Hardware and Embedded Systems (CHES).*

[35] Timothy Linscott, Pete Ehrett, Valeria Bertacco, and Todd Austin. 2018. SWAN: mitigating hardware trojans with design ambiguity. In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD).* IEEE.

[36] MIT Lincoln Laboratory. [n. d.]. Common Evaluation Platform. https://github.com/mit-ll/CEP.

[37] Michael Nagel, Alexander Michalski, and Heinrich Kurz. 2011. Contact-free fault location and imaging with on-chip terahertz time-domain reflectometry. *Optics Express* (2011).

[38] Seetharam Narasimhan, Xinmu Wang, Dongdong Du, Rajat Subhra Chakraborty, and Swarup Bhunia. 2011. TeSR: A robust temporal self-referencing approach for hardware Trojan detection. In *IEEE International Symposium on Hardware-Oriented Security and Trust (HOST).*

[39] C Odegard and C Lambert. 1999. Comparative TDR analysis as a packaging FA tool. In *ISTFA 1999: 25 th International Symposium for Testing and Failure Analysis.*

[40] OpenCores.org. [n. d.]. OpenRISC OR1200 Processor. https://github.com/openrisc/or1200.

[41] Dan L Philen, Ian A White, Jane F Kuhl, and Stephen C Mettler. 1982. Single-mode fiber OTDR: Experiment and theory. *IEEE Transactions on Microwave Theory and Techniques* (1982).

[42] Miodrag Potkonjak, Ani Nahapetian, Michael Nelson, and Tammara Massey. 2009. Hardware Trojan horse detection using gate-level characterization. In *Proceedings of ACM/IEEE Design Automation Conference (DAC).*

[43] Masoud Rostami, Farinaz Koushanfar, Jeyavijayan Rajendran, and Ramesh Karri. 2013. Hardware Security: Threat Models and Metrics. In *Proceedings of the International Conference on Computer-Aided Design (ICCD).*

[44] Yuriy Shiyanovskii, F Wolff, Aravind Rajendran, C Papachristou, D Weyer, and W Clay. 2010. Process reliability based trojans through NBTI and HCI effects. In *NASA/ESA Conference on Adaptive Hardware and Systems (AHS).*

[45] D Smolyansky. 2004. Electronic Package Fault Isolation Using TDR. *ASM International* (2004).

[46] PI Somlo and DL Hollway. 1969. Microwave Locating Reflectometer. *Electronics Letters* (1969).

[47] Ed Sperling. 2018. Design Rule Complexity Rising. https://semiengineering.com/design-rule-complexity-rising/.

[48] Takeshi Sugawara, Daisuke Suzuki, Ryoichi Fujii, Shigeaki Tawa, Ryohei Hori, Mitsuru Shiozaki, and Takeshi Fujino. 2014. Reversing stealthy dopant-level circuits. In *International Workshop on Cryptographic Hardware and Embedded Systems (CHES).*

[49] James Sutherland. 1999. As Edge speeds increase, wires become transmission lines. *EDN* (1999).

[50] MY Tay, L Cao, M Venkata, L Tran, W Donna, W Qiu, J Alton, PF Taday, and M Lin. 2012. Advanced fault isolation technique using electro-optical terahertz pulse reflectometry. In *Physical and Failure Analysis of Integrated Circuits (IPFA), 2012 19th IEEE International Symposium on the.*

[51] Mohammad Tehranipoor and Farinaz Koushanfar. 2010. A survey of hardware trojan taxonomy and detection. *IEEE Design & Test of Computers* 27, 1 (2010).

[52] TeraView. [n. d.]. *Electro Optical Terahertz Pulse Reflectometry: The world's fastest and most accurate fault isolation system.*

[53] Mohit Tiwari, Hassan M.G. Wassel, Bita Mazloom, Shashidhar Mysore, Frederic T. Chong, and Timothy Sherwood. 2009. Complete Information Flow Tracking from the Gates Up. In *International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS).* 109–120.

[54] Timothy Trippel, Kang G. Shin, Kevin B. Bush, and Matthew Hicks. 2020. ICAS: an Extensible Framework for Estimating the Susceptibility of IC Layouts to Additive Trojans. In *IEEE Symposium on Security and Privacy (S&P).*

[55] Timothy Trippel, Kang G. Shin, Kevin B. Bush, and Matthew Hicks. 2021. Bomberman: Defining and Defeating Hardware Ticking Timebombs at Design-time. In *To appear in the IEEE Symposium on Security and Privacy (S&P).*

[56] TSMC. 2019. TSMC Fabrication Schedule — 2019. https://www.mosis.com/db/pubf/fsched?ORG=TSMC.

[57] Denys Vlasenko. [n. d.]. BusyBox. https://www.busybox.net/.

[58] Adam Waksman, Matthew Suozzo, and Simha Sethumadhavan. 2013. FANCI: identification of stealthy malicious logic using boolean functional analysis. In *Proceedings of the ACM SIGSAC Conference on Computer & Communications Security (CCS)*.

[59] Huanyu Wang, Qihang Shi, Adib Nahiyan, Domenic Forte, and Mark M Tehranipoor. 2019. A physical design flow against front-side probing attacks by internal shielding. *Transactions on Computer-Aided Design of Integrated Circuits and Systems* (2019).

[60] Yujie Wang, Pu Chen, Jiang Hu, and Jeyavijayan JV Rajendran. 2017. Routing perturbation for enhanced security in split manufacturing. In *22nd Asia and South Pacific Design Automation Conference (ASP-DAC)*. IEEE.

[61] Stephen Williams. [n. d.]. Icarus Verilog. http://iverilog.icarus.com/.

[62] Francis Wolff, Chris Papachristou, Swarup Bhunia, and Rajat S Chakraborty. 2008. Towards Trojan-free trusted ICs: Problem analysis and detection scheme. In *Proceedings of the ACM Conference on Design, Automation and Test in Europe (DATE)*.

[63] Kan Xiao, Domenic Forte, Yier Jin, Ramesh Karri, Swarup Bhunia, and Mohammad Tehranipoor. 2016. Hardware trojans: Lessons learned after one decade of research. *Transactions on Design Automation of Electronic Systems (TODAES)* (2016).

[64] Kan Xiao and Mohammed Tehranipoor. 2013. BISA: Built-in self-authentication for preventing hardware Trojan insertion. In *IEEE International Symposium on Hardware-Oriented Security and Trust (HOST)*.

[65] Kaiyuan Yang, Matthew Hicks, Qing Dong, Todd Austin, and Dennis Sylvester. 2016. A2: Analog malicious hardware. In *IEEE Symposium on Security and Privacy (S&P)*.

[66] Rui Zhang, Natalie Stanley, Christopher Griggs, Andrew Chi, and Cynthia Sturton. 2017. Identifying Security Critical Properties for the Dynamic Verification of a Processor. In *International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*.

[67] Rui Zhang and Cynthia Sturton. 2020. Transys: Leveraging Common Security Properties Across Hardware Designs. In *IEEE Symposium on Security and Privacy (S&P)*.

[68] Xuehui Zhang and Mohammad Tehranipoor. 2011. RON: An on-chip ring oscillator network for hardware Trojan detection. In *2011 Design, Automation & Test in Europe*. IEEE, 1–6.

[69] Boyou Zhou, Ronen Adato, Mahmoud Zangeneh, Tianyu Yang, Aydan Uyar, Bennett Goldberg, Selim Unlu, and Ajay Joshi. 2015. Detecting hardware trojans using backside optical imaging of embedded watermarks. In *Proceedings of IEEE Design Automation Conference (DAC)*.

A THREAT ANALYSIS OF BYPASS ATTACKS

Recall, of the three ways an attacker can bypass T-TER guard wires to carry out a fabrication-time attack (Fig. 4 and §4.2), the *jog* attack is the stealthiest. An attacker mounts a jog attack by *jogging*, or moving, a portion of a guard wire to a nearby routing track, in order to make room for a rogue Trojan wire to attach to a targeted net (Fig. 4C). In such an attack, the guard wire is *lengthened*, or *bends* are added/removed. To evaluate the detectability of such an attack, we ask three questions:

(1) *What is the smallest jog attack, i.e., the minimum alteration in a guard wire's length and/or number of bends?*
(2) *Is the smallest jog attack masked by process variation?*
(3) *Can modern TDR detect the smallest jog attacks?*

A.0.1 ***Smallest Jog Attack.*** The minimum jog attack is to jog a top (or bottom) guard wire to an adjacent routing track, and attach to the targeted net from above (or below) with a via, as illustrated in Fig. 4C. This edit either increases the length of the guard wire, or adds/removes bends—impedance discontinuities—in the guard wire to keep its overall length unchanged. This edit is minimal because the minimal metal pitch (MMP), or (horizontal) distance between the centers of adjacent routing tracks on the *same routing layer*, is much smaller than the (vertical) distance between overlapping routing tracks on *adjacent routing layers*. Specifically, the smallest jog attack would either: 1) increase a guard wire's length by: $L_{attack} = 2*MMP_r$, where MMP_r is the MMP on layer r, as defined in the design rules of a given process technology, or 2) add/remove bend(s) in the guard wire that are at least a distance of L_{attack} apart from existing bends. *In either case, a feature resolution—of overall length or length between bends—of* L_{attack} *is required to detect the smallest jog attack.* Table 2 summarizes the *minimal-attack-edits* (L_{attack} distances), to a guard wire's features an attacker must make to bypass T-TER, according to the 45 *nm* process technology we target in this study.

Table 2: Minimum guard wire jog attack (Fig. 4C) edit–distances for each routing layer in the IBM 45 *nm* SOI process technology.

Routing Layer	Min Wire Spacing (*um*)	Min Metal Pitch (*um*)	Min Attack Edit (*um*)	TDR Detect?
1	0.07	0.14	0.28	✓
2	0.07	0.14	0.28	✓
3	0.07	0.14	0.28	✓
4	0.09	0.19	0.38	✓
5	0.09	0.19	0.38	✓
6	0.14	0.28	0.56	✓
7	0.14	0.28	0.56	✓
8	0.80	1.60	3.20	✓
9	0.80	1.60	3.20	✓
10	2.00	4.00	8.00	✓

A.0.2 ***Process Variation vs. Smallest Jog Attack.*** Assume for a moment that we can measure the of overall length, or length between bends, of a guard wire to infinite accuracy. Even then, detecting the smallest jog attack requires the minimal attack edit distance, L_{attack}, be discernable from deviations between simulated and fabricated guard wire lengths due to process variation. Fortunately, L_{attack} **is larger than the <u>worst-case</u> manufacturing process variation** in a guard wire's length. Namely, with L_{design} as the designed length of the guard wire, and L_{wc_error}, as the worst-case manufacturing error in the actual guard wire's length (+ or −):

$$L_{design} - L_{wc\_error} + L_{attack} > L_{design} + L_{wc\_error} \tag{2}$$

For a guard wire on routing layer r, the *worst-case* manufacturing error, L_{wc_error}, can be deduced from the manufacturing design rules as:

$$L_{wc\_error} = 2 * \frac{min\_spacing_r}{2} = min\_spacing_r \tag{3}$$

where $min\_spacing_r$ is the minimum required spacing surrounding a wire routed on metal layer, r.

We illustrate the *worst-case* manufacturing error, L_{wc_error} in Eqs. 2 & 3, in Fig. 9, where we plot the minimum length differences between unmodified (un-attacked) and minimally-jogged (attacked) guard wires, overlaid with error bars indicating the worst-case range of variation in a guard wires fabricated length caused by process variation. Across all routing layers in the process we study, unmodified vs. attacked guard wires are discernible.

A.0.3 ***Attack Detection with TDR.*** When IC interconnects are injected with a pulsed waveform with a rise time less than twice the

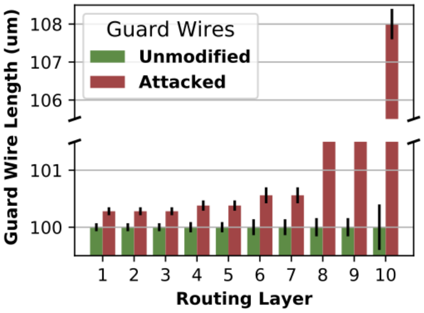

Figure 9: Worst-case manufacturing process variation (error bars) effect on unmodified and minimal jog attacks on 100-micron guard-wires.

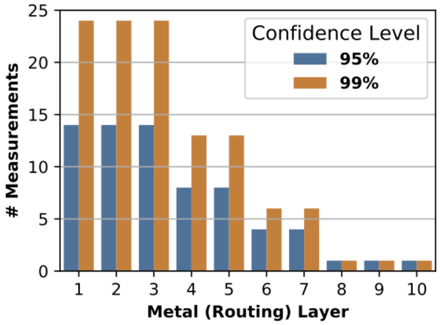

Figure 10: Number of TDR measurements required to detect the smallest jog attacks (Table 2) with 95% and 99% confidence, per layer.

propagation delay of the interconnect, they behave like transmission lines (Eq. (1)). Hence, time-domain reflectometry (TDR) can be used to measure several characteristics of designed-in guard wires to ensure they have not been tampered with (§2.3). Specifically, the *lengths* of each guard wire, or *lengths between bends* on each guard wire, are computed by measuring the reflection time(s) of a single incident rising pulse applied to the guard wires under test. Once measured, the lengths can be compared with that predicted by a 3D electromagnetic field solver [33] to detect if they have been altered. While modeling *all* interconnects within a large complex IC using a field solver is computationally impractical, it is *practical* to analyze only a small subset of interconnects, e.g., the guard wires and surrounding circuit structures [37].

Prior work demonstrates terahertz TDR systems [11, 37, 50, 52] capable of measuring the propagation delay of an interconnect to a resolution of ±2.6 femptoseconds (fs). Such systems utilize laser-driven optoelectronic measurement techniques to achieve such high resolutions. According to the ideal transmission line model [49], the propagation delay, T_{pd}, is a function of the dielectric constant, D_k, speed of light, C, and **length of the transmission line (guard wire)**, L_{gw}, as shown in Eq. (4).

$$T_{pd} = L_{gw} * \frac{\sqrt{D_k}}{C} \tag{4}$$

TDR is the ideal tamper detection tool as process variation has no impact on its accuracy. Knowing the dielectric constant, D_k, of the insulating material surrounding the guard wires—the inter-layer dielectric (ILD)—is *all* that is required to compute their lengths, or the lengths between their bends (Eq. (4)). Since, the dielectric constant of the ILD is **not** dependent on its geometric properties, it is well controlled [9].

Using the TDR propagation delay model described in Eq. (4), and the previously studied resolution of optoelectrical terahertz TDR [11, 37, 50, 52], we simulate the detection of the smallest jog attacks on guard wires across every routing layer in our target 45 nm process. Namely, we simulate the difference in reflection times observed for single pulse TDR waveforms applied to (unmodified) guard wires that are 100 microns long, compared to the reflection time observed from similar guard wires that have been lengthened by the minimal attack edit distances, L_{attack}, across each routing layer (Table 2). We assume a dielectric constant of 3.9, the nominal dielectric constant of silicon dioxide [27]. Taking into account a (Gaussian) standard error (across reflection time measurements) of ± 2.6 fs, as reported by [37], we compute the minimum number of TDR measurements required to discriminate an unmodified guard wire from an attacked guard wire with confidence levels of 95% and 99%. We plot these results in Figure 10. Our results demonstrate that existing terahertz TDR systems are capable of detecting the smallest jog attacks across all routing layers (Table 2) in our target 45 nm process, requiring at most 14 and 24 TDR measurements to achieve confidence levels of 95% and 99%, respectively.