A Golden-Free Formal Method for Trojan Detection in Non-Interfering Accelerators

Anna Lena Duque Antón*, Johannes Müller*, Lucas Deutschmann*, Mohammad Rahmani Fadiheh†, Dominik Stoffel*, Wolfgang Kunz*
*University of Kaiserslautern-Landau, Kaiserslautern, Germany †Stanford University, Stanford, USA
{anna.duqueanton, johannes.mueller, lucas.deutschmann, dominik.stoffel, wolfgang.kunz}@rptu.de, fadiheh@stanford.edu

***Abstract*—The threat of hardware Trojans (HTs) in security-critical IPs like cryptographic accelerators poses severe security risks. The HT detection methods available today mostly rely on golden models and detailed circuit specifications. Often they are specific to certain HT payload types, making pre-silicon verification difficult and leading to security gaps. We propose a novel formal verification method for HT detection in non-interfering accelerators at the Register Transfer Level (RTL), employing standard formal property checking. Our method guarantees the exhaustive detection of any sequential HT independently of its payload behavior, including physical side channels. It does not require a golden model or a functional specification of the design. The experimental results demonstrate efficient and effective detection of all sequential HTs in accelerators available on Trust-Hub, including those with complex triggers and payloads.**

***Index Terms*—Hardware Security, Formal Verification, Hardware Trojans**

I. Introduction

Many of today's Systems on Chip (SoCs) outsource complex computation tasks to specialized hardware accelerators. Such blocks of Intellectual Property (IP) can implement certain functions with higher performance and efficiency than software (SW) solutions using a CPU. The resulting need for specialized and cost-effective accelerators is catered to by a global supply chain. Accelerator IPs can be acquired and integrated as third-party IPs (3PIPs) or generated using third-party EDA tools. However, the flexibility and the opportunities of a diverse and global supply chain also introduce new security risks. Among these, hardware Trojans (HTs) are a prominent class of threats [1]. The threat is exacerbated by the trend to outsource even security-critical computations to 3PIPs, including implementations for encryption, which is the foundation of the overall system security. Encryption accelerators, therefore, need to undergo thorough verification for any malicious behavior. For reasons of cost, verification time and coverage, this process is increasingly moved to the pre-silicon design phase [2].

So far, however, classical verification methods often perform poorly in detecting HTs [3]. Intelligent adversaries construct *stealthy HTs* that are able to evade common detection techniques. A stealthy HT executes its malicious behavior, commonly referred to as *payload*, only after a *trigger* condition is met. We distinguish between combinational and sequential HTs. While there exist effective detection methods for the former type [4], detecting the latter type is still a hard problem [5].

This work was supported partly by Bundesministerium für Bildung und Forschung Scale4Edge, grant no. 16ME0122K-16ME0140+16ME0465, by Intel Corp., Scalable Assurance Program and by Siemens EDA.

For sequential HTs the trigger condition is created such that it requires a potentially long sequence of input events which has a very low probability of occurring during testing. While contemporary detection methods like functional validation can be quite effective against HTs with short trigger sequences, more complex triggers, especially those based on very long input sequences, can easily neutralize such methods [6]. Furthermore, many detection methods rely on golden models, i.e., an HT-free design, or detailed specifications [7], which may not be available. And finally, most methods are limited with respect to what payload types they can detect. Physical side channels, in particular, are largely ignored by previous formal pre-silicon HT detection methods.

We intend to overcome these challenges by providing a formal verification method for HT detection in accelerator IPs. We target *non-interfering* accelerators (cf. Sec. III). In fact, many loosely-coupled accelerators integrated in heterogeneous SoCs belong to this class [8]. Our method operates on RTL models and is based on standard formal property checking, which makes it easy to integrate it into existing verification flows. The proposed approach can produce formal guarantees for the absence of HTs in a design.

In summary, this paper makes the following contributions:

- We propose, for the first time, a formal methodology that allows us to *exhaustively* verify the absence of any sequential HT with an arbitrary long and complex trigger sequence in a non-interfering accelerator IP. (Sec. IV)
- Our method does not rely on a golden model or a functional specification of the design, by merit of the proposed 2-safety computational model. (Sec. IV-A)
- We guarantee the detection of any sequential HT independently of its payload. We exploit that sequential HTs have some RTL representation of their payload, even in the case of physical side channels. (Sec. IV-C)
- We effectively and soundly decompose the verification target into single-cycle properties allowing us to introduce a scalable iterative verification flow (Sec. V). We demonstrate the efficiency and effectiveness of our method by application to all accelerator IPs available on Trust-Hub [9], [10] (Sec. VI).

II. Related Work

Several works leverage verification tests for hardware trojan detection. In [11], malicious circuit detection is based on Unused Circuit Identification (UCI). UCI identifies circuit parts

979-8-3503-4859-0/DATE24/© 2024 EDAA

that do not affect the outputs during verification tests and therefore may include malicious logic. However, as demonstrated by [12], the adversary can design HTs in such a way that there is a verification test affecting the HT's logic without fulfilling the trigger condition. VeriTrust [13] detects HTs in a design by analyzing system states that are not covered by verification tests to identify the trigger. In [14], circuit wires are analyzed regarding their probability to influence outputs. The lower their impact the more likely they belong to an HT. However, [13] and [14] do not detect HTs with more complex sequential trigger logic, as discussed in [15]. In contrast, this paper proposes an approach that guarantees the detection of HTs with arbitrary long and complex trigger sequences.

Other works apply formal verification for HT detection. The work of [16], for example, leverages Information Flow Tracking (IFT) to derive formal models that are used for checking security properties like confidentiality or integrity of sensitive data. While the approaches are sound w.r.t. to their target properties, they depend on modeling specific payloads and are, thus, not exhaustive. In particular, they cannot detect HTs based on non-functional design specifications like (physical) side channels. The same limitations exist in approaches that use Bounded Model Checking (BMC) to detect HTs that modify data in registers [17] or leak secret data to the IP outputs. Furthermore, due to the limitations of the bounded proof, the approaches are unable to detect trojans with very long trigger sequences. For the same reason, functional verification approaches based on BMC [8] are not effective for trojan detection. A SAT-based method is proposed in [18] to detect HTs that do not violate design specifications. The method detects HTs that modify unspecified design functionality. Our approach is not limited to a specific HT implementation, but detects arbitrary, possibly unknown, payload implementations.

The approaches in [19], [20] use equivalence checking to compare the IP under verification against a golden, HT-free design, which, however, is often unavailable. Therefore, we propose a golden-free detection method.

In recent years, pre-silicon HT detection based on Machine Learning (ML) has become popular. These approaches employ ML to generate test patterns that are more likely to trigger HTs [21] or to classify structural design features as HT-free [22], [23]. While these approaches are effective even for large designs, they are inherently not exhaustive and thus cannot provide security guarantees.

III. Trojans in Non-Interfering Accelerators

Our method focuses on the detection of HTs in so-called *non-interfering* accelerator IPs. This notion, introduced by [8], refers to the typical characteristic of accelerators that the computed result for a given input is independent of any inputs received earlier or later. It is important to note that this definition is not a restriction to combinational circuits but includes sequential ones. Fig. 1 illustrates such a design. It shows a cryptographic accelerator infested with an HT. The IP implements encryption of a plaintext with a key.

HTs can be classified by their trigger and their payload. The purpose of the trigger is to provide a reliable but hard-to-detect

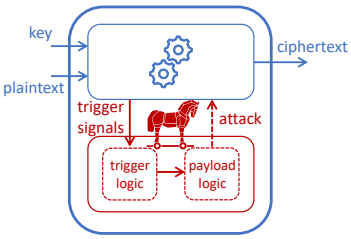

Fig. 1. HT in a cryptographic IP. The HT consists of a trigger and payload.

means for an attacker to activate the trojan. To this end, an HT trigger may rely on multiple logic signals of the circuit and can consist of arbitrary complex (sequential) logic. When an HT is activated, it executes its payload, which manifests itself as malicious behavior in various ways, such as leaking a key via output pins or using a power side channel.

It is common to distinguish between *combinational* and *sequential* HTs. There exist works that effectively detect combinational HTs, i.e., HTs that are triggered by a combinational circuit [5]. However, the detection of sequential HTs, i.e., HTs that are triggered by a sequential circuit is still an open challenge [24]. Sequential HTs are activated only after a possibly long sequence of regular, specification-conforming executions. The trigger does not necessarily depend on the input values. It may also simply be a counter activated by the reset signal.

IV. Method

In the following we present a formal property checking method that allows us to detect *all* sequential HTs in accelerator IPs. The properties are design-agnostic and do not require a golden model of the design. We describe the intuition behind our analysis and the key challenges in Sec. IV-A. In Sec. IV-B and IV-C, we then describe how the proposed methodology meets these challenges. In addition, we discuss the exhaustiveness of our method in Sec. IV-D.

A. Intuition

As discussed in Sec. III, an HT executes its payload after being activated by its trigger. The payload can be any malicious behavior that modifies the functionality of the IP or adds unwanted functionality to the IP. Instead of comparing the IP's behavior with a golden model, which is usually unavailable, our method compares *two identical instances* of the IP: We analyze the behavior of the two instances under the same inputs, but allow the solver to assume different, arbitrary *input histories*. These different input histories are captured by the different *symbolic starting states* in the two instances of our computational model, as elaborated below. In case the design is infested with a sequential HT, we can compare one instance with a triggered HT about to execute its payload to one with an untriggered HT. The method then verifies whether the two instances behave the same for all possible input sequences from that time point on. If the proof fails, it returns a counterexample pointing at the HT's payload. If the verification does not detect any divergence in the state or output of the two IP instances, then there is no HT with a malicious payload in the design. This is illustrated with the 2-safety *miter* setup shown in Fig. 2.

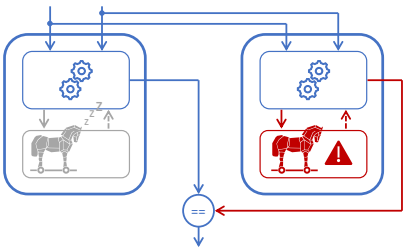

Fig. 2. Two instances of the same IP ("miter"), both containing an HT. The same inputs are fed to both instances. The HT in instance$_1$ is triggered while the HT in instance$_2$ is dormant. This results in an observable difference in the behavior of the two IPs.

There are two key challenges with this approach that our method addresses:

1) modeling arbitrary (unknown) trigger sequences, including ones of arbitrary length, and allowing the HT to be triggered in one of the instances while not in the other;
2) targeting an arbitrary payload.

In Secs. IV-B and IV-C respectively, we elaborate on how these two challenges are overcome.

B. Modelling trigger sequences

The key idea for dealing with HT triggers exploits the nature of our employed property checking technique. Our detection method is based on *Interval Property Checking (IPC)* [25], which uses bounded properties but achieves unbounded proof validity. IPC facilitates proving (interval) properties of the form *antecedent* $\Rightarrow$ *consequent* with a symbolic starting state. This means that a solver verifying an interval property is free to choose any state of the design as starting state for the proof as long as the antecedent is fulfilled. This symbolic starting state can thus *implicitly* model any history of previous states, including *any possible trigger sequence*. Essentially, our method leverages IPC to "fast-forward" to the time point at which a potential HT is activated.

The verification method presented in the following sections exploits an important characteristic of non-interfering accelerators. Such designs are often *data-driven*, i.e., they typically determine the internal states relevant for their computations only from the inputs and, essentially, independently of the accelerator's internal state at the start of the computation. This allows for the effective use of symbolic starting states in the properties of our verification flow without the risk of many false alarms.

C. Detecting payloads

Using a symbolic starting state, we can model an active HT in one instance together with a dormant HT in the other instance. This addresses the first challenge from Sec. IV-A. However, the second challenge – the variety of HT payloads – still remains. We aim to detect not just specific payloads, such as leaking a secret key, but any possible sequential HT visible at the RTL, without requiring a full specification.

Consider a basic IPC proof that enumerates all malicious HT behaviors without reference to a golden model. Some functionality might be overlooked, and HT payloads with additional behaviors or physical side channel manifestations could be missed. We overcome these problems by leveraging an important observation: Regardless of its nature, the effect of a sequential HT's payload has to manifest itself in at least one state signal or output of the design or it does not have any security implications. Instead of formulating all possible behaviors in the consequent, we use this observation together with our employed miter setup.

trojan_property:
assume:
  at t: $inputs\_instance_1 = inputs\_instance_2$
prove:
  at $t+1$: $fanouts\_CC_1\_instance_1 = fanouts\_CC_1\_instance_2$
  at $t+2$: $fanouts\_CC_2\_instance_1 = fanouts\_CC_2\_instance_2$
  ...
  at $t+n$: $fanouts\_CC_n\_instance_1 = fanouts\_CC_n\_instance_2$

Fig. 3. Interval property for hardware trojan detection

We verify the equivalence of the two instances using a structural decomposition of the design. We compute all state and output signals that appear in the transitive fanout cone of the inputs and partition them according to the smallest number of clock cycles it takes for the inputs to affect their value. We denote the set of *state* and *output* signals affected after k clock cycles by $fanouts\_CC_k$. We then prove for each k the equivalence of $fanouts\_CC_k$ between the two instances for the corresponding time window. This results in the *trojan_property* in Fig. 3. The property checker is free to choose arbitrary starting states, as long as, at time point t, the same inputs are provided to the two IP instances. For each consecutive clock cycle, the equality of the corresponding $fanouts\_CC_k$ is checked. Proving this interval property makes any malicious behavior visible that manifests itself in either state or output signals in the fanout path of the inputs. We provide a detailed pseudo-code description for our formal verification flow in Sec. V.

D. Exhaustiveness

The question arises whether we can exhaustively detect *any* malicious behavior introduced by a sequential HT with this property. For this, we need to distinguish three cases:

1) The payload affects at least one state or output signal that lies on a fanout path of the inputs.
2) The payload affects only state or output signals that are not part of a fanout path.
3) The payload has no effect on any state or output signal.

Case 1: The state or output signal, say s, is covered by the consequent (*prove* part) of our property. Hence, a symbolic initial state exists where instance$_1$ meets the HT trigger condition while instance$_2$ does not. As a result, the triggered HT affects s in instance$_1$ but does not in instance$_2$. The property checker computes a counterexample (CEX) that demonstrates this difference, unveiling the payload's malicious behavior.

Case 2: The HT is activated independently of the inputs, e.g., a timer started by the system reset and its payload does not affect the fanout cone of the IP's inputs (cf. Sec. VI and design example AES-T1900). This case is not detectable by the property. However, it can be covered by a simple structural

analysis: We need to check whether all state and output signals of our IP appear in the *prove* part of our property. Those that do not may belong to a possible HT.

Case 3: If the HT's payload neither manifests itself in any state signal nor in any output signal then the HT does not implement security-critical behavior visible at the RTL.

Since cases 1 to 3 cover all possible cases of an HT's payload in the RTL design we conclude that we exhaustively detect all sequential HTs with this method.

V. Formal Detection Flow

For practical purposes, we seek to develop an HT analysis that is scalable and easy to use for the verification engineer. We decompose the property in Fig. 3 into a set of interval properties each covering exactly one clock cycle. These become elements of an iterative verification flow where individual proofs have short runtimes and counterexamples point to potentially malicious behavior with high accuracy.

We employ two types of properties. The first type is the *init_property* shown in Fig. 4. It verifies that there is no malicious interference with the propagation of the input signals to the first fanout signals, $fanouts\_CC_1$, reached in the IP. The property assumes equal inputs, under which the equality of the $fanouts\_CC_1$ is proven. The *init_property* is a cutout of the *trojan_property* of Fig. 2 until time point $t+1$. The second type of property, the fanout properties (*fanout_property_k*), shown in Fig. 5, covers the equality checks for all subsequent time points. As we show in Sec. V-A, the set of decomposed properties is equivalent to the *trojan_property*.

Alg. 1 implements the iterative HT detection flow based on these two property types and the method described in Sec. IV. In the first step, the $fanouts\_CC_1$, i.e., all state and output signals reachable within one clock cycle from the input signals, are computed: Get_Fanout() implements a simple structural analysis that traces syntactic dependencies of state-holding elements in the RTL design. With this information, the *init_property* is constructed (line 3). The property is then checked with IPC. In case the property fails, a counterexample, *CEX*, is returned that points to a possible hardware trojan and that must be inspected by the verification engineer. If the property holds, the procedure continues. In the next step the $fanouts\_CC_1$ become the starting points of the structural analysis. All state and output signals reachable within one clock cycle from the $fanouts\_CC_1$ are determined and the corresponding *fanout_property_1* is constructed for the next iteration. Any failing interval property will produce a counterexample (CEX) that shows the exact state signals where a potential trojan might be implemented. This process is repeated, verifying a *fanout_property_k* in each iteration, until no new state or output signals are added. In case all properties hold, we conclude the procedure by checking whether the property set covers all state and output signals of the IP under verification (cf. case 2 of Sec. IV-D). If there are any state or output signals left, the set of uncovered signals (*UCS*) is returned. It is important to note that the number of loop iterations (line 8-16) is limited by the structural, not the sequential, depth of the design.

Algorithm 1 Formal HT Detection Flow

```
1:  procedure HT-Detection(IP, inputs)
2:      fanouts_CC1 ← Get_Fanout(IP, inputs)
3:      init_property ← Create_Init_Property(inputs, fanouts_CC1)
4:      CEX ← IPC(init_property)
5:      if CEX ≠ ∅ then return CEX
6:      fanouts_all ← ∅
7:      k ← 1
8:      repeat
9:          fanouts_all ← fanouts_all ∪ fanouts_CCk
10:         fanouts_CCk+1 ← Get_Fanout(IP, fanouts_CCk)
11:         fanout_property_k ← Create_Property(fanouts_CCk,
                                                fanouts_CCk+1)
12:         CEX ← IPC(fanout_property_k)
13:         if CEX ≠ ∅ then return CEX
14:         fanouts_CCk ← fanouts_CCk+1
15:         k ← k + 1
16:     until fanouts_all ∪ fanouts_CCk == fanouts_all
17:     UCS ← Check_Signal_Coverage(IP, fanouts_all)
18:     if UCS ≠ ∅ then return UCS
19:     return "SECURE"
```

init_property:
assume:
at t: $inputs\_instance_1 = inputs\_instance_2$;
prove:
at $t+1$: $fanouts\_CC_1\_instance_1 = fanouts\_CC_1\_instance_2$;

Fig. 4. Interval property for Init Check

fanout_property_k:
assume:
at t: $fanouts\_CC_k\_instance_1 = fanouts\_CC_k\_instance_2$;
prove:
at $t+1$: $fanouts\_CC_{k+1}\_instance_1 = fanouts\_CC_{k+1}\_instance_2$;

Fig. 5. Interval property for Fanout Check

A. Soundness of Property Decomposition

In Sec. IV-D we have discussed the exhaustiveness of the *trojan_property*. Verifying this property for a design guarantees detection of any sequential HT that it might be infected with. In the following, we prove that the same guarantee is given by verifying the *init_property* and all computed fanout properties (*fanout_property_k*).

Theorem 1. *At least one of the fanout properties (fanout_property_k) or the init_property fails (1) iff the trojan_property fails (2).* □

Proof. A key observation to keep in mind for the proof is that each set of fanout signals $fanouts\_CC_k$ considered at time point k *is identical* in both property formulations: the aggregate *trojan_property* of Fig. 3 and the decomposition into *init_property* (Fig. 4) and the fanout properties (Fig. 5).

We decompose the theorem into two implications and prove them individually.

(2) ⇒ (1): Assume the *trojan_property* fails because there is a state or output signal z with different values in the two instances at clock cycle $t+k+1$ with $0 < k < n$. We further assume, w.l.o.g., that the proof commitments of all preceding clock cycles are proven to hold, i.e., in particular, $fanouts\_CC_k$ are equal between the two instances for the considered k. Hence, there must exist a state signal x in the fanin of z

with $x \notin$ *fanouts_CC*$_k$ which holds different values in the two instances. (Remember that x cannot be a primary input because all fanout signals of primary inputs are covered by the *init_property* and commitment $t+1$ of the *trojan_property*.) Now consider the corresponding *fanout_property_k*. Since it considers the same set of signals *fanouts_CC*$_k$ in its assumption as the aggregate property, x is missing in this set as well, causing the property to fail.

(1) ⇒ (2): We prove this implication by contradiction. Assume, w.l.o.g., *fanout_property_k* fails because of inequality of state signal z, but the *trojan_property* holds. This means that z must depend on at least one state signal x where $x \notin$ *fanouts_CC*$_k$. Furthermore, since the *trojan_property* holds, z must be proven equal at clock cycle $t+k+1$. But this requires that the *trojan_property* must also prove the equality of x at the previous clock cycle $t+k$, because the equality of z depends on the equality of x. However, if this is true, then x must be an element of *fanouts_CC*$_k$. This contradicts the assumption from above and proves the claim. □

B. Analyzing Counterexamples

Although this occurs rarely, as explained in Sec. IV-B, for some IPs the property checker may produce false alarms, i.e., the *init_property* or a *fanout_property_k* fails for some signal z although there is no HT in the design. For understanding such cases, assume *fanout_property_k* fails for z. This means that z is affected by some other signal x in the fanin of z, but $x \notin$ *fanouts_CC*$_k$. In other words, x is not proven to be equal between the two instances by the predecessor *fanout_property_k-1*. This may happen in two scenarios:
(1) We do not necessarily prove the fanout properties in topological order. Therefore, *fanout_property_k* may fail even though x is proven to be equal in another fanout property or the *init_property*. This scenario can be solved by changing the proof order of the fanout properties and adding equality for x to the assumptions of *fanout_property_k*. We omitted this procedure in Alg. 1 to keep the presentation simple.
(2) x depends on values of previous computations but is not part of an HT. In this scenario the verification engineer receives a CEX pinpointing the exact behavior that demonstrates the dependency between x and z. This greatly helps in disqualifying the behavior as an HT. Similarly to the first scenario, equality for x can then be assumed in *fanout_property_k*.

Fortunately, the characteristics of non-interfering accelerators allow for effective use of a symbolic initial state in our computational model. As we demonstrate in our experimental results, we encountered false counterexamples only in few cases which were easy to diagnose.

VI. Experiments

We applied our method to the accelerator IPs available on Trust-Hub [10]. We evaluated all accelerators, except for three with simple, combinational HTs which are not considered in this work. All are non-interfering. As can be seen in Tab. I, they implement different crypto algorithms and HTs with a variety of payloads and triggers. For the payloads, secret data is leaked via output pins (OUT), different implementations of power side channels (*PSC*), through leakage currents (*LC*) or by modulating the output signal on an unused pin creating a radio frequency (*RF*) signal. In four cases, the payload consists of denial-of-service attacks (*DoS*) that aim at rapidly draining the battery. Other payloads interfere with the encryption via bit flips of the ciphertext output. The triggers in the experiments depend either on a predefined plaintext sequence or on implementations of counters that count certain events.

TABLE I

Benchmark	Payload	Trigger	Detected by
AES-T100	PSC	plaintext seq.	init_property
AES-T1000	PSC	plaintext seq.	init_property
AES-T1100	PSC	plaintext seq.	init_property
AES-T1200	PSC	# encryptions	init_property
AES-T1300	PSC	plaintext seq.	init_property
AES-T1400	PSC	plaintext seq.	init_property
AES-T1500	PSC	# encryptions	init_property
AES-T1600	RF	plaintext seq.	init_property
AES-T1700	RF	# encryptions	init_property
AES-T1800	DoS	plaintext seq.	init_property
AES-T1900	DoS	# encryptions	coverage check
AES-T2000	LC	plaintext seq.	init_property
AES-T2100	LC	# encryptions	init_property
AES-T2500	bit flip	# clock cycles	fanout_property_21
AES-T2600	bit flip	# values	fanout_property_7
AES-T2700	bit flip	# clock cycles	fanout_property_21
AES-T2800	bit flip	# values	fanout_property_11
AES-T200	PSC	plaintext seq.	init_property
AES-T300	PSC	plaintext seq.	init_property
AES-T400	RF	plaintext seq.	init_property
AES-T500	DoS	plaintext seq.	init_property
AES-T600	LC	plaintext seq.	init_property
AES-T700	PSC	plaintext seq.	init_property
AES-T800	PSC	plaintext seq.	init_property
AES-T900	PSC	# encryptions	init_property
BasicRSA-T200	DoS	plaintext seq.	init_property
BasicRSA-T300	OUT	# encryptions	init_property
BasicRSA-T400	OUT	# encryptions	init_property

We successfully detected all HTs. Our method, as discussed in Sec. IV, is independent of the specific characteristics of the HT's implementation. The HTs were detected by a failed init or fanout property or through the coverage check. Each AES benchmark also includes an HT-free version. We successfully applied our method to the HT-free designs and verified them to be secure with respect to sequential HTs. They required the proof of the init and fanout properties. We did not encounter any spurious counterexamples. For the RSA designs no HT-free version was available on Trust-Hub. We manually removed the HTs from the designs and afterwards could verify the absence of any HTs. During the proof we encountered 2 spurious CEXs that we handled according to the proposed methods in Sec. V-B. The proof runtime of each property was within 1 to 3 seconds and the memory usage was less than 1 GB. All experiments were conducted on an Intel i7-8700 @ 3.2 GHz with 64 GB RAM running Linux and the commercial property checker OneSpin 360 DV by Siemens EDA.

In the following, we explain our experiments in more depth using two of the benchmarks as examples.

Example 1: AES-T1400. The trojan in this example features a

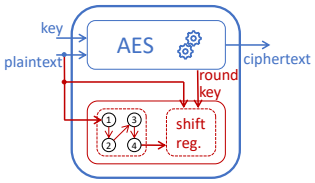

Fig. 6. AES-T1400

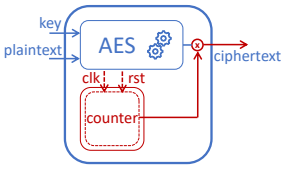

Fig. 7. AES-T2500

4-state FSM as its trigger and is illustrated in Fig. 6. The trojan is triggered when four specific plaintext inputs are observed in a specific order. Once activated, the trojan leaks, for each round of encryption, certain bits of the round key via a power side channel. The key bits are combined with known input bits and shifted into a register, thereby increasing power consumption. We detected the HT with a failed init property. The CEX provided by the property checker shows different values in the shift registers of the two instances.

Example 2: AES-T2500. In the second example, the trojan is triggered by the fourth bit of a 4-bit synchronous counter. The counter itself does not depend on the IP inputs but starts counting from reset. After activation, the trojan flips the least significant bit (LSB) of the ciphertext output. Fig. 7 illustrates this behavior. The HT is detected with *fanout_property_21* which proves equal ciphertext outputs. The property fails and the CEX shows the difference in the LSB of the ciphertext outputs due to a triggered HT in only one of the two instances.

Even though it is not in the focus of this work, we conclude our experiments with demonstrating the potential of our method for HW IPs with more complex control behavior. As an additional case study, we successfully applied the method also to a UART *(RS232-T2400)* from the same benchmark suite. We detected the HT by a failed fanout property. During the proof we encountered 3 spurious CEXs that we could resolve by property re-verification (cf. Sec. V-B (1)), and by disqualifying the behaviors as non-malicious (cf. Sec. V-B (2)).

VII. Conclusion

We introduce an exhaustive formal detection methodology for sequential HTs with arbitrarily long and complex trigger sequences. The proposed method is equally effective for any payload ranging from direct data leakage to power side channels. It is a golden-free approach by merit of a 2-safety verification model. The method is based on property checking and easy to integrate into pre-silicon verification. The application of the method to a representative set of accelerators demonstrates the efficiency and effectiveness of our proof method. Future work will explore the class of interfering IPs such as (special-purpose) processors.

References

[1] S. Bhunia, M. S. Hsiao, M. Banga, and S. Narasimhan, "Hardware trojan attacks: Threat analysis and countermeasures," *Proceedings of the IEEE*, vol. 102, no. 8, pp. 1229–1247, 2014.

[2] N. Jacob, D. Merli, J. Heyszl, and G. Sigl, "Hardware trojans: current challenges and approaches," *IET Computers & Digital Techniques*, vol. 8, no. 6, pp. 264–273, 2014.

[3] K. Xiao, D. Forte, Y. Jin, R. Karri, S. Bhunia *et al.*, "Hardware trojans: Lessons learned after one decade of research," *ACM Transactions on Design Automation of Electronic Systems*, vol. 22, no. 1, pp. 1–23, 2016.

[4] Z. Zhou, U. Guin, and V. D. Agrawal, "Modeling and test generation for combinational hardware trojans," in *IEEE VLSI Test Symposium (VTS)*. IEEE, 2018, pp. 1–6.

[5] A. Jain, Z. Zhou, and U. Guin, "Survey of recent developments for hardware trojan detection," in *IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, 2021, pp. 1–5.

[6] M. Xue, C. Gu, W. Liu, S. Yu, and M. O'Neill, "Ten years of hardware trojans: a survey from the attacker's perspective," *IET Computers & Digital Techniques*, vol. 14, no. 6, pp. 231–246, 2020.

[7] M. Rathmair, F. Schupfer, and C. Krieg, "Applied formal methods for hardware trojan detection," in *IEEE International Symposium on Circuits and Systems (ISCAS)*, 2014, pp. 169–172.

[8] E. Singh, F. Lonsing, S. Chattopadhyay, M. Strange, P. Wei *et al.*, "A-qed verification of hardware accelerators," in *ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2020, pp. 1–6.

[9] H. Salmani, M. Tehranipoor, and R. Karri, "On design vulnerability analysis and trust benchmarks development," in *IEEE International Conference on Computer Design (ICCD)*. IEEE, 2013, pp. 471–474.

[10] B. Shakya, T. He, H. Salmani, D. Forte, S. Bhunia *et al.*, "Benchmarking of hardware trojans and maliciously affected circuits," *Journal of Hardware and Systems Security*, vol. 1, pp. 85–102, 2017.

[11] M. Hicks, M. Finnicum, S. T. King, M. M. Martin, and J. M. Smith, "Overcoming an untrusted computing base: Detecting and removing malicious hardware automatically," in *IEEE Symposium on Security and Privacy*. IEEE, 2010, pp. 159–172.

[12] C. Sturton, M. Hicks, D. Wagner, and S. T. King, "Defeating uci: Building stealthy and malicious hardware," in *IEEE Symposium on Security and Privacy*. IEEE, 2011, pp. 64–77.

[13] J. Zhang, F. Yuan, L. Wei, Z. Sun, and Q. Xu, "Veritrust: Verification for hardware trust," in *ACM/EDAC/IEEE Design Automation Conference (DAC)*, 2013, pp. 1–8.

[14] A. Waksman, M. Suozzo, and S. Sethumadhavan, "Fanci: identification of stealthy malicious logic using boolean functional analysis," in *ACM SIGSAC Conference on Computer & Communications Security*, 2013, pp. 697–708.

[15] J. Zhang, F. Yuan, and Q. Xu, "Detrust: Defeating hardware trust verification with stealthy implicitly-triggered hardware trojans," in *ACM SIGSAC Conference on Computer and Communications Security*, 2014, pp. 153–166.

[16] W. Hu, A. Ardeshiricham, M. S. Gobulukoglu, X. Wang, and R. Kastner, "Property specific information flow analysis for hardware security verification," in *IEEE/ACM International Conference on Computer-Aided Design*, 2018, pp. 1–8.

[17] J. Rajendran, V. Vedula, and R. Karri, "Detecting malicious modifications of data in third-party intellectual property cores," in *ACM/EDAC/IEEE Design Automation Conference (DAC)*, 2015, pp. 1–6.

[18] N. Fern, I. San, and K.-T. T. Cheng, "Detecting hardware trojans in unspecified functionality through solving satisfiability problems," in *Asia and South Pacific Design Automation Conference (ASP-DAC)*. IEEE, 2017, pp. 598–504.

[19] A. Ito, R. Ueno, and N. Homma, "A formal approach to identifying hardware trojans in cryptographic hardware," in *International Symposium on Multiple-Valued Logic (ISMVL)*. IEEE, 2021, pp. 154–159.

[20] F. Farahmandi, Y. Huang, and P. Mishra, "Trojan localization using symbolic algebra," in *Asia and South Pacific Design Automation Conference (ASP-DAC)*. IEEE, 2017, pp. 591–597.

[21] Z. Pan and P. Mishra, "Automated test generation for hardware trojan detection using reinforcement learning," in *Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2021, pp. 408–413.

[22] S. Yu, C. Gu, W. Liu, and M. O'Neill, "A novel feature extraction strategy for hardware trojan detection," in *IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, 2020, pp. 1–5.

[23] A. Hepp, J. Baehr, and G. Sigl, "Golden model-free hardware trojan detection by classification of netlist module graphs," in *Design, Automation & Test in Europe Conference (DATE)*. IEEE, 2022, pp. 1317–1322.

[24] X. Wang, S. Narasimhan, A. Krishna, T. Mal-Sarkar, and S. Bhunia, "Sequential hardware trojan: Side-channel aware design and placement," in *IEEE International Conference on Computer Design (ICCD)*. IEEE, 2011, pp. 297–300.

[25] M. D. Nguyen, M. Thalmaier, M. Wedler, J. Bormann, D. Stoffel *et al.*, "Unbounded Protocol Compliance Verification using Interval Property Checking with Invariants," *IEEE Transactions on Computer-Aided Design*, vol. 27, no. 11, pp. 2068–2082, November 2008.