DeTrust: Defeating Hardware Trust Verification with Stealthy Implicitly-Triggered Hardware Trojans

Jie Zhang, Feng Yuan, and Qiang Xu

CUhk REliable Computing Laboratory (CURE)
Department of Computer Science & Engineering
The Chinese University of Hong Kong, Shatin, N.T., Hong Kong
Email: {jzhang, fyuan, qxu}@cse.cuhk.edu.hk

ABSTRACT

Hardware Trojans (HTs) inserted at design time by malicious insiders on the design team or third-party intellectual property (IP) providers pose a serious threat to the security of computing systems. Researchers have proposed several hardware trust verification techniques to mitigate such threats, and some of them are shown to be able to effectively flag all suspicious HTs implemented in the Trust-Hub hardware backdoor benchmark suite. No doubt to say, adversaries would adjust their tactics of attacks accordingly and it is hence essential to examine whether new types of HTs can be designed to defeat these hardware trust verification techniques.

In this paper, we present a systematic HT design methodology to achieve the above objective, namely *DeTrust*. Given an HT design, *DeTrust* keeps its original malicious behavior while making the HT resistant to state-of-the-art hardware trust verification techniques by manipulating its trigger designs. To be specific, *DeTrust* implements *stealthy implicit triggers* for HTs by carefully spreading the trigger logic into multiple sequential levels and combinational logic blocks and combining the trigger logic with the normal logic, so that they are not easily differentiable from normal logic. As shown in our experimental results, adversaries can easily employ *DeTrust* to evade hardware trust verification.

We close with a discussion on how to extend existing solutions to alleviate the threat posed by *DeTrust*. However, they generally suffer from high computational complexity, calling for more advanced techniques to ensure hardware trust.

Categories and Subject Descriptors

B.6.2 [**Hardware**]: Logic Design—Security and Trust

Keywords

hardware Trojan; hardware security; backdoors; implicit trigger

1. INTRODUCTION

With the ever-increasing hardware complexity and the large number of third-parties involved in the design and fabrication process of integrated circuits (ICs), today's IC products are vulnerable to a wide range of malicious alterations, namely *hardware Trojans* (HTs) [1–3]. For example, a hardware backdoor can be introduced into the design by simply writing a few lines of hardware description language (HDL) codes [4, 5], which leads to functional deviation from design specification and/or sensitive information leakages. Skorobogatov and Woods [7] found a "backdoor" in a military-grade FPGA device[1], which could be exploited by attackers to extract all the configuration data from the chip and access/-modify sensitive information. Liu *et al.* [28] demonstrated a silicon implementation of a wireless cryptographic chip with an embedded HT and showed it could leak secret keys. HTs thus pose a serious threat to the security of computing systems and have called upon the attention of several government agencies [8, 9].

HTs can be inserted into an IC product at any stage, e.g., specification, register-transfer level (RTL) design, IP integration, physical design, and fabrication. Generally speaking, the likelihood of HTs being inserted at design time is usually much higher than that being inserted at manufacturing stage, because adversaries do not need to access foundry facilities to implement HTs and it is also more flexible for them to implement various malicious functions.

In recent years, several techniques have been proposed to protect hardware designs against certain type of HTs inserted at design time [10–14]. Among them, some techniques [10–12] operate at runtime and try to de-activate suspicious circuitries. These techniques, however, require to modify the original design to include runtime protections, and hence incur runtime overhead and increase design complexity. Performing verification for hardware trust without necessarily modifying the design is therefore quite appealing. Hicks *et al.* [10] made the first attempt and formulated the HT detection problem as an *unused circuit identification* (UCI) problem. However, due to the relatively simple definition of "unused circuit", it could only cover a small set of HTs. Later, Zhang *et al.* [5] and Sturton *et al.* [15] presented how to automatically construct HTs that are able to evade UCI detection algorithm. Based on the observation that HT trigger inputs are redundant to circuit normal functions when HTs are not activated during functional verification, Zhang *et al.* [13] proposed a so-called *VeriTrust* technique, which focused on HT trigger identification. Recently, Waksman *et al.* [14] presented a static HT detection technique based on Boolean functional analysis, namely *FANCI*. Both [13] and [14] showed that they were able to flag all the suspicious HTs implemented in the Trust-Hub hardware backdoor benchmark suite [6].

HT design and HT identification techniques are like arms race, wherein designers update security measures to protect their system while attackers respond with more tricky HTs. With state-of-the-art hardware trust verification techniques such as VeriTrust and FANCI being able to effectively identify existing HTs, no doubt to say, adversaries would adjust their tactics of attacks accordingly and *it is hence essential to examine whether new types of HTs can be designed to defeat these hardware trust verification techniques.*

[1]The company responded that the hidden super key is used for failure analysis. However, it exactly matches the definition for backdoor given by the dictionary: "Backdoor - an undocumented way to get access to a computer system or the data it contains".

CCS'14, November 3–7, 2014, Scottsdale, Arizona, USA.

DOI string from ACM form confirmation .

1.1 Threat Model

We follow the same threat model and assumptions used in [10, 13]. That is, a hardware design can be covertly compromised by HTs inserted into the RTL source code or the gate-level netlist. These HTs are introduced by one or more rogue designers in the design team or integrated into the design with third-party IP cores. Attackers cannot control the suite of tests used for functional verification, but they can learn arbitrary information about the test cases. We assume the verification procedure is trustworthy and HTs manifest themselves as long as they are activated.

Consequently, from the adversaries' perspective, on the one hand, HTs should be resistant to functional verification with extremely low activation probability; on the other hand, they should be resistant to trust verification by hiding as normal logic circuit. This work aims to devise such an HT design methodology.

1.2 Contributions

In this work, we present a systematic HT design methodology that is resistant to hardware trust verification, namely *DeTrust*, by targeting the weakness of these solutions. To be specific, to make *DeTrust* evade FANCI, HT trigger logics are carefully spread among multiple combinational logic blocks so that Boolean functional analysis would not flag them as *nearly-unused* logics. To defeat VeriTrust, we combine HT triggers with circuit original functional logic and hide them into multiple sequential levels. Such *implicit triggers* would not be seen as redundant inputs under non-trigger condition. In addition, *DeTrust* also borrows existing stealthy HT design methodology (e.g., [5]) to hide from conventional functional verification and UCI techniques. With the above, HT designs can be performed in a one-off manner to be resistant to trust verification solutions while still passing functional verification.

To be specific, this work contributes to the field of hardware trust in the following ways:

- We show that VeriTrust [13] and FANCI [14] have limitations. We design and implement an attack on a processor that is able to evade these hardware trust verification techniques while still passing functional verification.
- We present a systematic HT design methodology called *DeTrust* that *automatically* equips HTs with stealthy implicit triggers to be resistant to all existing hardware trust verification techniques in a one-off manner. We analyze the stealthiness of the proposed HT designs and present heuristic algorithms to increase its stealthiness.
- We present how to extend VeriTrust and FANCI to alleviate the threat posed by *DeTrust*. However, there is no easy fix to this problem. We analyze why such defenses are not sufficient to defend against *DeTrust*, even though they do increase HT design effort.

The remainder of this paper is organized as follows. We first present the preliminaries related to hardware Trojan design and identification in Section 2. The proposed HT design methodology, i.e., *DeTrust*, is then detailed in Section 3. We validate the effectiveness of *DeTrust* by introducing practical attacks and analyzing their stealthiness in Section 4. Next, we present some potential defenses for *DeTrust* in Section 5. Finally, we survey related works in Section 6 and conclude this paper in Section 7.

2. PRELIMINARIES

In this section, we first introduce some terms used in this paper. Next, we describe state-of-the-art hardware trust verification techniques for HTs inserted at design time. Finally, we discuss the effectiveness of these techniques.

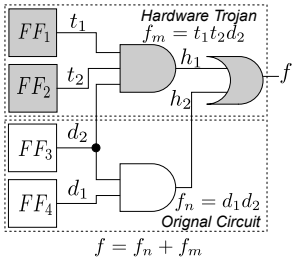


Figure 1: An HT-infected circuit with trigger inputs t_1 and t_2, wherein the original logic function $f_n = d_1 d_2$ is compromised by an HT with malicious function $f_m = t_1 t_2 d_2$ and the trigger condition is $\{t_1, t_2\} = \{1, 1\}$.

2.1 Definitions

Generally speaking, an HT is composed of its activation mechanism (referred to as *trigger*) and its malicious function (referred to as *payload*). For the ease of discussion, we have the following definitions:

DEFINITION 1. *An **HT-affected signal** is a signal that the HT targets to manipulate (e.g., f in Fig. 1).*

DEFINITION 2. *An **HT-related signal** is a signal that is driven by any part of the HT (e.g., t_1, t_2, h_1, f in Fig. 1).*

DEFINITION 3. *A **functional input** is an input that is used by the circuit's specified normal functionality (e.g., d_1, d_2 in Fig. 1).*

DEFINITION 4. *A **trigger input** is an input that is used in the condition under which the HT is activated (e.g., t_1, t_2 in Fig. 1).*

Note that, functional inputs can also serve as HT trigger inputs [15].

2.2 HT Classification

In [13], the authors classified HTs into *bug-based HTs* and *parasite-based HTs*, according to their impact on the normal functions of the circuit. A Bug-based HT alters the circuit and causes it to lose some of its normal functionalities while a parasite-based HT hides along with the original circuit and does not cause it to lose any normal functionalities. Generally speaking, when compared with bug-based HTs, parasite-based HTs are more difficult to be activated with functional verification tests, since its malicious behavior is not included in the specification. As a result, almost all HTs appeared in the literature belong to the parasite-based type.

Note that, as a small set of HTs are sufficient for attackers to compromise a hardware design, we do not attempt to define the class of all possible HTs that are likely to evade hardware trust verification techniques in this work, which is very difficult, if not impossible. In this paper, we consider that a hardware design is inserted with one or more parasite-based HTs whose inputs are separated into functional inputs and some *dedicated* trigger inputs. For the convenience of presentation, HTs mentioned in the rest of the paper means parasite-based HTs unless otherwise specified.

2.3 Verification for Hardware Trust

Various verification techniques can be employed for HT identification, as detailed in the following.

2.3.1 Formal Verification

Theoretically speaking, we can formally prove whether a hardware design contains HTs or not with a given trustworthy high-level system model (e.g., [16]). In practice, however, full formal verification of large circuits is still computationally infeasible. In addition, the golden model itself may not be available. Consequently, we usually have to resort to either functional verification or dedicated trust verification techniques for HT detection.

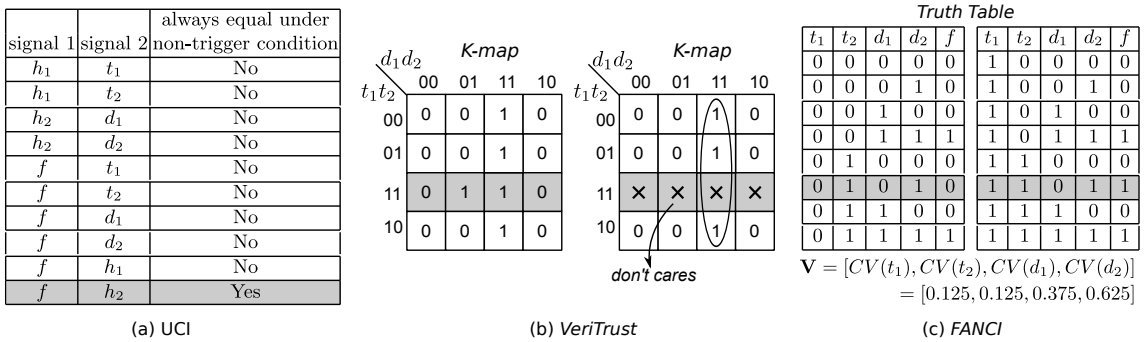


Figure 2: HT identification with trust verification techniques

2.3.2 Functional Verification (FV)

Simulation can be used for HT detection, by activating an HT and observing its malicious behavior. If we were able to walk through all possible system states, we can catch all HTs with exhaustive simulation. In practice, however, due to the sheer volume of states that exist in even a simple design, FV techniques only cover a small subset of the functional space of a hardware design. Considering the fact that attackers have full controllability for the location and the trigger condition of their HT designs at design time, which are secrets to functional verification engineers, it is usually very difficult, if not impossible, to directly activate an HT.

The above has motivated a number of dedicated hardware trust verification techniques, as discussed in the following.

2.3.3 Trust Verification

Trust verification techniques flag suspicious circuitries based on the observation that HTs are nearly always dormant (by design) in order to pass functional verification. Such analysis can be conducted in a *dynamic* manner by analyzing which part of the circuit is not sensitized during functional verification, as in *UCI* [10] and VeriTrust [13]. Alternatively, *static* Boolean functional analysis can be used to identify suspicious signals with *weakly-affecting* inputs, as in FANCI [14].

In the following, we discuss these trust verification techniques and use the circuit shown in Fig. 1 to demonstrate how they can be used for HT identification.

UCI: Hicks *et al.* [10] first addressed the problem of identifying HTs inserted at design time, by formulating it as an unused circuit identification problem. They defined "unused circuits" as follows. Consider a signal pair (s,t), where t is dependent on s. If $t = s$ throughout the entire functional verification procedure, the intermediate circuit between s and t is regarded as "unused circuit". With the above definition, the UCI algorithm in [10] traces all signal pairs during verification, and reports those ones for which the property $s = t$ holds throughout all test cases as suspicious circuitries. For our example circuit, the signal pair (h_2, f) will be always equal under non-trigger condition and hence it is guaranteed to be flagged as suspicious if the HT is not activated during functional verification. Whether the other signal pairs will be flagged as suspicious circuitries depend on the test cases applied during functional verification.

Another way to define unused circuit is based on the code coverage metrics used in verification (e.g., line coverage and branch coverage) such that those uncovered circuitries are flagged as suspicious malicious logic. Surprisingly, such simple analysis is able to catch a large number of HT designs in the Trust-Hub hardware backdoor benchmark suite, as demonstrated in [13].

One of the main limitations of UCI techniques is that they are sensitive to the implementation style of HTs. Later, [5, 15] presented how to exploit this weakness to defeat UCI detection algorithms.

VeriTrust: VeriTrust [13] flags suspicious circuitries by identifying potential trigger inputs used in HTs, based on the observation that these inputs keep dormant under non-trigger condition (otherwise HTs would have manifested themselves) and hence are redundant to the normal logic function of the circuit. For our example circuit whose K-map is shown in Fig. 2 (b), by setting all entries of the malicious function as "don't cares" (i.e., they can be assigned with logic '0' or logic '1' freely), the trigger inputs (i.e., t_1 and t_2) become redundant.

VeriTrust works as follows. Firstly, a *tracer* traces the activation history of the circuit in the form of simplified sum-of-products (SOP) and product-of-sums (POS) (instead of all entries) to save tracing overhead. Next, by setting all the un-activated entries as don't cares, a *checker* identifies redundant inputs by analyzing those unactivated SOPs and POSs. These redundant inputs are then flagged as potential HT trigger inputs for further examination. Note that, VeriTrust may incur *false positives* (a false positive would mean a flagged input is not a true HT trigger input), because functional simulation is not complete and there are some un-activated entires belonging to normal function. However, it would not miss any true trigger inputs.

VeriTrust is shown to be insensitive to the implementation style of HTs (at least for existing HT designs), and it is able to identify all HTs implemented in the Trust-Hub hardware backdoor benchmark suite.

FANCI: FANCI [14] identifies signals with weakly-affecting inputs by *static* Boolean function analysis, based on the observation that an HT trigger input generally has a *weak* impact on output signals. Without running verification tests, [14] proposed to use the so-called *control value (CV)* to estimate the impact between an input signal and an output that is driven by it:

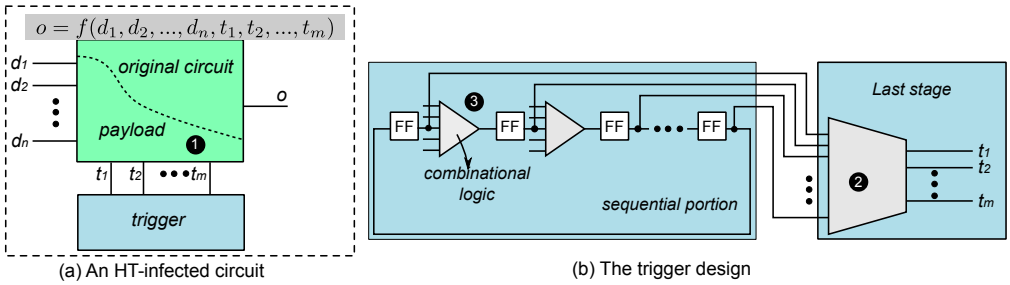
$$CV = \frac{counter}{size(T)}, \tag{1}$$

where *counter* denotes the total number of patterns under which flipping this input signal results in the change of the output value, while $size(T)$ denotes the size of the truth table. For example, as shown in Fig. 2 (c), there are only two input patterns under which flipping t_1 leads to the change of the output, and hence $CV(t_1) = 2/2^4 = 0.125$. FANCI examines each state element in the circuit and obtains a vector of control values $\mathbf{V}$ in its fan-in combinational logic cone. A heuristic metric (e.g., median or mean) on $\mathbf{V}$ is then calculated and compared against a *cut-off threshold* to determine whether this signal is HT-related.

Consider our example circuit, as shown in Fig. 2 (c), we have $\mathbf{V} = [0.125, 0.125, 0.375, 0.625]$ and 0.3125 for f with the mean metric. Therefore, FANCI would flag f as suspicious if the cut-off threshold is set to be larger than 0.3125. By setting a proper cut-off threshold value in [14], FANCI is able to identify all HTs from Trust-Hub [6].

	Static/Dynamic	Detection Method	Runtime	False Negatives	False Positives
Functional Verification	dynamic	activate the HT	good	HTs with rare trigger condition	none
UCI by Code Coverage	dynamic	identify uncovered parts	good	HTs in [5]	few with thorough verification
UCI by [10]	dynamic	identify equal signal pair	fair	HTs in [5, 15]	some with thorough verification
VeriTrust [13]	dynamic	identify HT trigger inputs	fair	unknown	some with thorough verification
FANCI [14]	static	identify weakly-affecting inputs	fair	possible with low threshold	many with high threshold

Table 1: HT Detection with Hardware Functional Verification and Trust Verification



(a) An HT-infected circuit (b) The trigger design

Figure 3: A structural overview of existing HT designs

2.4 Discussion

Table 1 summarizes the characteristics of existing solutions for HT detection. Since dynamic trust verification techniques (i.e., UCI and VeriTrust) analyze the corner cases of functional verification for HT detection, these two types of verification techniques somehow complement each other. Generally speaking, with more FV tests applied, the possibility for HTs being activated is higher while the number of suspicious circuitries reported by UCI and VeriTrust would decrease. As a static solution that does not depend on verification, one unique advantage of FANCI over the other solutions is that it does not require a trustworthy verification team. On the other hand, however, adversaries could also validate their HT designs using FANCI without necessarily speculating the unknown test cases used to catch them.

All trust verification techniques try to eliminate *false negatives* (a false negative would mean an HT is not detected) whilst keeping the number of *false positives* as few as possible in order not to waste too much effort on examining benign circuitries that are deemed as suspicious. However, their detection capability is related to some user-specified parameters and inputs during trust verification. For example, FANCI defines a *cut-off threshold* for what is suspicious and what is not during Boolean functional analysis. If this value is set to be quite large, it is likely to catch HT-related wires together with a large number of benign wires. This, however, is a serious burden for security engineers because they have to evaluate all suspicious wires by code inspection and/or extensive simulations. If this value is set to be quite small, on the contrary, it is likely to miss some HT-related wires. Similarly, if we apply a small number of FV tests, UCI and VeriTrust would flag a large number of suspicious wires (all wires in the extreme case when no FV tests are applied), which may contain HT-related signals but the large amount of false positives make the following examination procedure infeasible.

From the above, a successful HT design would behave similar to normal logic, such that a large number of false positives would be generated when HTs are detected during verification.

3. THE DETRUST METHODOLOGY

In this section, we detail the proposed *DeTrust* methodology to construct HTs that are resistant to state-of-the-art hardware trust verification techniques.

3.1 Overview

Fig. 3(a) presents the typical structure of an HT-infected design, which contains its original logic, the HT payload and the HT trigger. The HT payload implements certain malicious function while the HT trigger activates it under some trigger conditions. Generally speaking, the stealthiness of HT designs mainly depends on the HT trigger design and it usually comprises both a combinational portion and a sequential portion, as shown in Fig. 3(b).

Given an HT design, the objective of *DeTrust* is to revise its trigger design to be resistant to known trust verification techniques while maintaining its malicious function. The overall flow is as follows. For a given HT that is not stealthy enough to evade functional verification and/or UCI techniques, we first adopt the HT design methodology proposed in [5] to defeat them. To be specific, the trigger condition is carefully selected to be a rare value to resist FV and the HT is implemented with the code model in [5] to evade UCI techniques. Next, we carefully redesign the HT trigger to be resistant to both FANCI and VeriTrust, which is the focus of this work.

3.2 Defeating FANCI

Since FANCI identifies signals with weakly-affecting inputs within the combinational logic block, the key idea to defeat FANCI is to make the control values of all HT-related signals comparable to those of functional signals.

3.2.1 Motivational Case

Let us start with the case shown in Fig. 4 to illustrate the key idea of defeating FANCI. Fig. 4 (a) and Fig. 4 (b) present a regular multiplexer (MUX) and a malicious one with a rare trigger condition, respectively. FANCI would be able to differentiate the two types of MUXes and flag the malicious one since the trigger inputs, denoted as $t_0, t_1, \ldots, t_{63}$, have very small control values for the output o ($\frac{1}{2^{65}}$).

From this example, we can clearly see that, the main reason for HT-related signals (e.g., o in Fig. 4 (b)) having weakly-affecting inputs is that it is driven by a number of trigger inputs in its fan-in combinational logic cone. Consider a signal driven by a combinational logic block with m trigger inputs and n functional inputs. The size of the truth table for this particular HT-related signal is given by:

$$size(T) = 2^{m+n}. \quad (2)$$

For any trigger input, denoted as t_i, those input patterns under which t_i influences the output should meet two requirements: (i) all trigger inputs other than t_i are driven by the trigger values[2]; (ii) flipping t_i

[2]Trigger values are logic values for trigger inputs to satisfy trigger condition.

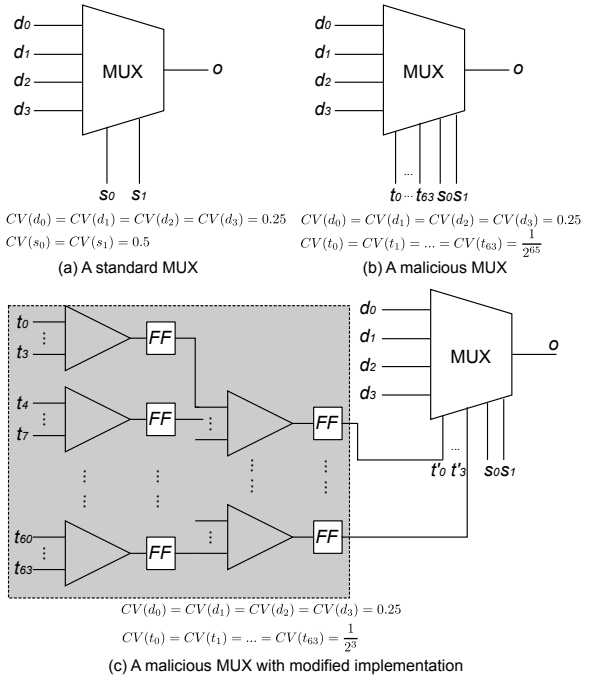


Figure 4: Motivational example for defeating FANCI

results in the change of the output value. There are in total 2^{n+1} input patterns meeting the first requirement. Among them, how many further satisfying the second requirement depends on the actual difference between the malicious function and the normal function, because they may output the same value under certain functional inputs. At the same time, they cannot always output the same value because otherwise there would be no malicious behavior. Therefore, the number of input patterns satisfying both requirements is bounded at:

$$2^1 \leq counter \leq 2^{n+1}. \tag{3}$$

With Eq. 2 and Eq. 3, the control value of t_i on the corresponding HT-related signal is bounded at:

$$\frac{1}{2^{n+m-1}} \leq CV(t_i) = \frac{counter}{size(T)} \leq \frac{1}{2^{m-1}}. \tag{4}$$

In order to make FANCI difficult to differentiate HT-related signals and function signals, we should make control values of HT-related signals to be comparable to those of functional signals. As indicated by Eq. 4, reducing m has an exponential impact on the increase of control values. Thus, we modify the implementation of the malicious MUX by balancing these trigger inputs into multiple sequential levels (see Fig. 4 (c)). In this way, the number of trigger inputs is controlled to be no more than four for any combinational block, rendering the control value of each trigger input comparable with those of functional inputs.

Motivated by the above, our approach of defeating FANCI is to reduce the number of trigger inputs in all the combinational logic blocks that drive HT-related signals, and it can be achieved by spreading HT trigger inputs among multiple sequential levels.

3.2.2 HT Design against FANCI

For the general HT design shown in Fig. 3, FANCI is likely to catch HT-related signals in the combinational logic of ❶, ❷ and ❸. Algorithm 1 presents the flow of our defeating method for FANCI. To reduce the number of trigger signals in a combinational logic, we consider the combinational logic of ❶ and ❷ together and then consider that of ❸, due to the fact that we need to adopt different methods to handle the extra signal delay of trigger inputs introduced by additional sequential levels.

Algorithm 1: The Flow to Defeat FANCI

```
   /* To increase HT stealthiness                    */
1  N_T = 2;
2  do
3  |  DefeatFANCI(N_T++);
4  while (The hardware cost is larger than a given constraint.);
   /* The procedure used to defeat FANCI             */
5  DefeatFANCI(N_T)
   |  /* For combinational logic of ❶ and ❷          */
6  |  foreach The fan-in cone of the input of the flip-flop do
7  |  |  if the number of trigger inputs > N_T then
8  |  |  |  Balance the trigger in the multiple sequential
   |  |  |  levels;
9  |  |  end if
10 |  end foreach
   |  /* For combinational logic of ❸                 */
11 |  Find out the maximum number of trigger signals, denoted
   |  as N_max, within a combinational logic cone;
12 |  if N_max > N_T then
13 |  |  Introduce multiple small FSMs until N_max ≤ N_T;
14 |  end if
15 end
```

For the combinational logic blocks in ❶ and ❷, our defeating method is similar to the one shown in Fig. 4 (c). As can be seen, the original trigger combinational logic with a large number of trigger inputs is spread among multiple combinational logic blocks, such that the number of trigger inputs in each combinational logic block is no more than N_T (a value used to tradeoff HT stealthiness and overhead). As shown in Algorithm 1 (Lines $6-10$), we only examine the inputs of flip-flops rather than all signals. This is because, as long as the number of trigger inputs for the inputs of flip-flops are smaller than N_T, the number of trigger inputs for all internal signals are also smaller than N_T.

The sequential part of an HT trigger can be represented by a finite-state machine (FSM) and the trigger inputs are used to control state transitions (see Fig. 5 (a)). For the combinational logic blocks in ❸ that sits inside the FSM, we cannot use the above defeating method because the additional pipeline delay introduced in this method would change trigger condition. Instead, we partition the original FSM into multiple small FSMs, e.g., as shown in Fig. 5 (b), the FSM with 64 trigger inputs is partitioned into eight small FSMs. By doing so, the number of trigger inputs in each small FSM is reduced to eight for this example, and it can be further reduced by introducing more FSMs. The HT is triggered when all the small FSMs reach certain states simultaneously.

Note that, the proposed defeating method against FANCI has no impact on both circuit's normal functionalities and HT's malicious behavior, because *DeTrust* only manipulates the HT trigger design, which is separate from the original circuit and the HT payload.

3.2.3 Stealthiness Optimization

Until now, we have described our defeating method against FANCI. However, as discussed in Section 2, whether an HT is able to evade FANCI is also influenced by the cut-off threshold that is used to trade-off false negatives and false positives.

DeTrust tries to maximize the stealthiness of the HT with respect to FANCI subject to a given constraint in terms of hardware cost. As the stealthiness of an HT is mainly determined by the number of trigger inputs in each combinational logic, this is achieved by finding the value of N_T in a greedy manner. That is, as shown in Algorithm 1, we start with $N_T = 2$ and gradually increase it until the cost of applying the defeating method is lower than the given constraint.

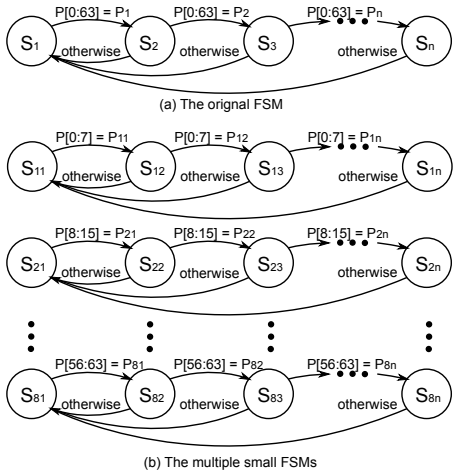


Figure 5: The proposed sequential trigger design to defeat FANCI

3.3 Defeating VeriTrust

As discussed earlier, VeriTrust flags suspicious HT trigger inputs by identifying those inputs that are redundant under verification. Consequently, the key idea to defeat VeriTrust is to make HT-affected signals driven by non-redundant inputs only under non-trigger condition.

3.3.1 Motivational Case

For any input that is not redundant under non-trigger condition, we have the following lemma.

LEMMA 1. *Consider an HT-affected signal whose Boolean function is $f(h_1, h_2, \ldots, h_k)$. Any input, h_i, is not redundant under non-trigger condition, as long as the normal function, denoted by f_n, cannot be completely represented without h_i.*

PROOF. Since f_n cannot be completely represented without h_i, there must exist at least one pattern for all inputs except h_i under which $f_n(h_i = 0) \neq f_n(h_i = 1)$. Therefore, h_i is not redundant. ∎

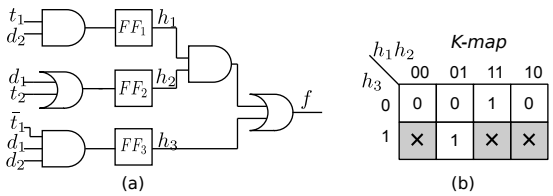


Figure 6: Motivational example for defeating VeriTrust

Inspired by Lemma 1, Fig. 6 (a) shows an HT-infected circuit that is revised according to the circuit shown in Fig. 1, wherein the HT is activated when $\{t_1, t_2\} = \{1, 1\}$. In this implementation, the malicious product $t_1t_2d_2$ is combined with the product $t_1d_1d_2$ from the normal function and hidden in the fan-in cones of h_1 and h_2, where $h_1 = t_1d_2$ and $h_2 = d_1 + t_2$. The K-map of f is shown in Fig. 6 (b), where entries that cannot be activated under non-trigger condition are marked as "don't cares". For this circuit, VeriTrust, focusing on the combinational logic, would verify four signals, f, h_1, h_2 and h_3. According to the K-map shown in Fig. 6(b), it is clear that h_1, h_2 and h_3 are not redundant for f under non-trigger condition. Moreover, h_1, h_2 and h_3, which are not HT-affected signals, have no redundant inputs as well, since all of their input patterns can be activated under non-trigger condition. Therefore, the HT in Fig. 6(a) is able to evade VeriTrust.

By examining the implementation of this motivational case, we find that the mixed design of the trigger and the original circuit makes trigger condition for the HT-affected signal not visible for VeriTrust. In order to differentiate existing HT designs and HTs like the one shown in Fig. 6 (a), we define two new terms: the *explicitly-triggered HT* and the *implicitly-triggered HT* as follows.

DEFINITION 5. *We say an HT is **explicitly-triggered** if in the HT-affected signal's fan-in logic cone, there exists an input pattern that uniquely represents the trigger condition.*

DEFINITION 6. *We say an HT is **implicitly-triggered** if in the HT-affected signal's fan-in logic cone, there does not exist any input pattern that uniquely represents the trigger condition.*

A careful examination for the HTs from Trust-Hub shows that they are all explicitly-triggered, and this is the reason why VeriTrust is able to flag all of them as suspicious. Interested readers can refer to the code model of HTs from Trust-Hub in [5]. On the contrary, the HT shown in Fig. 6 (a) is implicitly-triggered, since the trigger condition is hidden in the h_1h_2 which also contains certain circuit's normal functionalities.

The above observation motivates us to implement implicitly-triggered HTs to defeat VeriTrust.

3.3.2 HT Design against VeriTrust

For existing HT designs as shown in Fig. 3, VeriTrust is able to detect HT trigger inputs in the combinational logic block in ❶, wherein the output is the HT-affected signal. Our proposed defeating method for VeriTrust therefore focuses on this combinational logic. According to Lemma 1, our approach to defeat VeriTrust for *DeTrust* implements the implicitly-triggered HT with the following two steps:

- combine all malicious on-set terms[3] with on-set terms from the normal function, and re-allocate sequential elements (e.g, flip-flops) to hide the trigger in multiple combinational logic blocks.
- simplify all remaining on-set terms and re-allocate sequential elements if the remaining on-set terms contain trigger inputs.

Note that, we select on-set terms only, since the circuit can be explicitly represented by the sum of all on-set terms.

Let us further illustrate our defeating method against VeriTurst as follows. Consider a circuit with an explicitly-triggered HT, and its Boolean function can be represented as,

$$f = \sum_{c_{n_i} \forall C_n,\, p_{n_j} \forall P_n,} c_{n_i} p_{n_j} + \sum_{c_{m_i} \forall C_m,\, p_{m_j} \forall P_m} c_{m_i} p_{m_j}. \tag{5}$$

where P_n and P_m driven by functional inputs denote the set of all patterns that make the normal function and malicious function output logic '1', while C_n and C_m driven by trigger inputs denote the set of non-trigger conditions and the trigger conditions, respectively.

For the sake of simplicity, let us first consider the case where the malicious function contains only one malicious on-set term, $c_{m_0}p_{m_0}$. Suppose $c_{n_0}p_{n_0}$ from the normal function is selected to combine with $c_{m_0}p_{m_0}$. Let f_n' be all the on-set terms from the normal function except $c_{n_0}p_{n_0}$. Then, f can be given as:

$$f = f_n' + (c_{n_0}p_{n_0} + c_{m_0}p_{m_0}). \tag{6}$$

Suppose $c_{n_0}p_{n_0}$ and $c_{m_0}p_{m_0}$ have the common literals, $c^c p^c$, and then we have

$$f = f_n' + c^c p^c (c^r_{n_0} p^r_{n_0} + c^r_{m_0} p^r_{m_0}), \tag{7}$$

[3]Malicious on-set term defined in [13] is the on-set term in the malicious function whose adjacent terms in the normal function are off-set. On-set term and off-set term are terms that make the function output logic '1' and logic '0', respectively.

where

$$c_{n_0} = c^c c^r_{n_0}; \quad p_{n_0} = p^c p^r_{n_0}; \\ c_{m_0} = c^c c^r_{m_0}; \quad p_{m_0} = p^c p^r_{m_0}. \tag{8}$$

After that, we re-synthesize the circuit and re-allocate flip-flops, making f become

$$f = h_1 h_2 + h_3, \tag{9}$$

where

$$h_1 = c^c p^c \\ h_2 = c^r_{n_0} p^r_{n_0} + c^r_{m_0} p^r_{m_0}. \\ h_3 = f'_n \tag{10}$$

In order to keep the time sequence, we re-allocate the flip-flops and h_1, h_2 and h_3 are outputs of the new flip-flops.

As can be observed in Eq. 9 and Eq. 10, the key of the defeating method is to extract common literals from the malicious on-set term and the on-set term from the normal function and hide the trigger into different combinational logic. With the above, we find that f, h_1, h_2 and h_3 have no redundant inputs under non-trigger condition. We detail this theoretical proof in Appendix.

When there are multiple malicious on-set terms, we can use the above method to combine each of them with one on-set term from the normal function and then hide the trigger in different combinational logic blocks. Finally, we have

$$f = \sum_{i=0}^{k-1} (h_{2i+1} h_{2i+2}) + h_{2k+1}, \tag{11}$$

where

$$h_{2i+1} = c^{c_i} p^{c_i} \\ h_{2i+2} = c^{r_i}_{n_i} p^{r_i}_{n_i} + c^{r_i}_{m_i} p^{r_i}_{m_i}. \\ h_{2k+1} = f'_n \tag{12}$$

It is easy to prove h_1, h_2, ..., h_{2k+1} have no redundant inputs under non-trigger condition as well with the theoretical proof in Appendix. Note that, the HT can be spreaded over multiple sequential levels by further combining the trigger loigc driving h_1, h_2, ..., h_{2k+1} with normal logic.

3.3.3 Stealthiness Analysis and Optimization

Until now, we have presented our defeating approach against VeriTrust. However, as discussed in Section 2, whether an HT is able to evade VeriTrust is directly related to the functional verification test cases applied to the circuit.

According to Appendix, the HT would evade VeriTrust provided that *statement 1*, *statement 2*, *statement 3* and *statement 4* (see Appendix for details) are satisfied during functional verification. Let P_{s_1}, P_{s_2}, P_{s_3} and P_{s_4} be the probabilities of *statement 1*, *statement 2*, *statement 3* and *statement 4* to be satisfied during functional verification. The probability for the HT to evade VeriTrust, denoted by $P_{DeVeriTrust}$, is given as,

$$P_{DeVeriTrust} = P_{s_1} P_{s_2} P_{s_3} P_{s_4}, \tag{13}$$

where the dependencies between each statement are ignored. Let $P(c_{n_i} p_{n_j})$ be the probability of an input pattern, $c_{n_i} p_{n_j}$, to be activated. According to Appendix, we approximates P_{s_1}, P_{s_2} and P_{s_3} as follows:

$$\begin{aligned} P_{s_1} =& P(c_{n_0} p_{n_0}) \times \{P(\{c^c_i c^r_{m_0} p^c p^r_{m_0} | c^c_i \in C^c, c^c_i \neq c^c\}) \\ &+ P(\{c^c_i c^r_{m_0} p^c_j p^r_{m_0} | c^c_i \in C^c, c^c_i \neq c^c; p^c_j \in F^c, p^c_i \neq p^c\}) \\ &+ P(\{c^c_i c^r_{n_0} p^c p^r_{n_0}) | c^c_i \in C^c, c^c_i \neq c^c\}) \\ &+ P(\{c^c c^r_{n_0} p^c_j p^r_{n_0}) | p^c_j \in F^c, p^c_j \neq p^c\}) \\ &+ P(\{c^c_i c^r_{n_0} p^c_j p^r_{n_0}) | c^c_i \in C^c, c^c_i \neq c^c; p^c_j \in F^c, p^c_j \neq p^c\})\}; \end{aligned} \tag{14}$$

$$\begin{aligned} P_{s_2} =& P(c_{n_0} p_{n_0}) \times \{P(\{c^c c^r_{m_i} p^c p^r_{m_0} | c^r_{m_i} \in C^r, c^r_{m_i} \neq c^r_{m_0}\}) + \\ & P(\{c^c c^r_{m_i} p^c p^r_{m_j} | c^r_{m_i} \in C^r, c^r_{m_i} \neq c^r_{m_0}; p^r_{m_j} \in F^r, p^r_{m_j} \neq p^r_{m_0}\})\}; \end{aligned} \tag{15}$$

$$\begin{aligned} P_{s_3} =& P(\{c_i p_j | c_i \in C_n, c_i \neq c_{n_0}; p_j \in (F - P_n), p_j \neq p_{m_0}\}) \times \\ & P(\{c_i p_j | c_i \in C_n, c_i \neq c_{n_0}; p_j \in P_n, p_j \neq p_{m_0}\}). \end{aligned} \tag{16}$$

P_{s_4} can be calculated if all inputs of h_1, h_2 and h_3 are specified.

By examining Eq. 13-16, we find that $P_{DeVeriTrust}$ is dominated by the probabilities of some terms, such as $c_{n_0} p_{n_0}$ that is used to combine with the malicious on-set term $c_{m_0} p_{m_0}$. We propose the following three methods to increase the stealthiness of HTs against VeriTrust:

- Combine simplified malicious products (rather than malicious on-set terms) with on-set terms from the normal function, so that fewer terms from the normal function are required to be activated to evade VeriTrust.
- Choose simplified products from the normal function to be combined with simplified malicious products, so that any of the terms in the product from the normal function being activated can make HT evade VeriTrust.
- Choose those simplified products from the normal function with high activation probabilities to be combined with malicious products. This method requires the knowledge about the probability of products from the normal function, which can be estimated by speculating on the test cases used in functional verification [5, 15].

Algorithm 2: The Flow to Defeat VeriTrust

```
/* Focus on combinational logic of ❶ */
Simplify Boolean function of this combinational logic;
Conduct the simulation with tests guessed by attackers to
  obtain the probability of each product;
foreach simplified malicious product do
    Greedily combine it with the product from the normal
    function with the largest activation probability and hide
    the trigger in different combinational logic blocks;
end foreach
Re-allocate flip-flops for the remaining products;
```

The flow to defeat VeriTrust is illustrated in Algorithm 2. It first simplifies the Boolean function of the combinational logic of HT-affected signals, and then conducts simulation with speculated test cases to obtain the probability of each product. After that, a loop is used to hide HT triggers whenever possible. In each iteration, one malicious product is combined with one product from the normal function with the largest activation probability and it is hidden in different combinational logic blocks. At last, flip-flops are re-allocated for the remaining products.

3.4 Discussion

The defeating approach against FANCI and that against VeriTrust in *DeTrust* do not interfere with each other. On the one hand, *DeTrust* for FANCI focuses on reducing the number of trigger inputs in the combinational logic blocks used in HT triggers without changing their logic functions; on the other hand, *DeTrust* for VeriTrust implements the implicitly-triggered HT without increasing the number of trigger inputs in any combinational logic.

Moreover, *DeTrust* for FANCI and VeriTrust would not influence the stealthiness of HT designs shown in [5] against FV and UCI techniques. Firstly, *DeTrust* does not change HT trigger condition and hence it has no impact on functional verification. For UCI techniques that analyze code coverages, *DeTrust* for FANCI splits the HT trigger among multiple sequential levels, which does not affect

```
always @ (posedge clk)
  instruct_prev <= instruct_curr;
always @ (posedge clk) begin
  if (instruct_prev == 'INSTRUCT )
    instruct_prev_t <= 1;          ⓐ
  else instruct_prev_t <= 0;
  if (instruct_curr == 'INSTRUCT)
    instruct_curr_t <= 1;          ⓑ
  else instruct_curr_t <= 0;
end
always @ (posedge clk)
  super <= (~ holden & super)  |  (holden & in.super)
    ⓒ    | ((~ resten) | (instruct_curr_t & (instruct_prev_t |
           ((~ holden & super) | (holden & in.super)))));
end
```

Figure 7: Supervisor transition foothold in [15] (Verilog HDL)

each part of the trigger being covered by verification test cases. *DeTrust* for VeriTrust could use basic AND, OR and NOT operators to implement the implicit trigger, avoiding the impact based on code coverage analysis. For the UCI technique in [10], on the one hand, the signals introduced by *DeTrust* for FANCI are within the HT trigger unit driven by different trigger inputs and they are unlikely to be always equal during functional verification; on the other hand, *DeTrust* for VeriTrust combines parts of the normal functionalities with the HT trigger and hence is also unlikely to create equal signal pairs during verification. Note that, the above is a brief discussion on the impact of *DeTrust* on earlier stealthy HT design techniques, please refer to [5] for more details.

With the above, *DeTrust* is a one-off HT design methodology to be resistant to all known trust verification techniques while still passing functional verification.

4. VALIDATION AND DISCUSSION

In this section, we first design and implement a practical attack to illustrate how to apply *DeTrust* to construct an HT that is resistant to FANCI and VeriTrust. Next, we study the stealthiness of HTs constructed with *DeTrust* in detail.

4.1 Practical Attack

We adopt the malicious HT used to defeat UCI technique shown in [15] as the input to *DeTrust*. This HT, called *supervisor transition foothold*, is detailed in Fig. 7, and we implement it on the OpenRISC processor [17]. Whenever a specific instruction repeats twice, the HT is triggered and allows attackers to gain the full control of the system. As shown by ⓐ and ⓑ in Fig. 7, *instruct_prev_t* and *instruct_curr_t* are the two trigger inputs used to indicate whether previous and current instructions (denoted as *instruct_prev* and *instruct_curr*) are the trigger instructions (denoted as '*INSTRUCT*). The trigger inputs, the payload and the original circuit are then carefully combined to resist UCI (see ⓒ in Fig. 7).

FANCI is likely to catch *instruct_prev_t* and *instruct_curr_t* considering the small control values of *instruct_prev* and *instruct_curr*; while VeriTrust guarantees to flag the HT-affected signal *super* (the supervisor-mode bit), since the inputs of *instruct_prev_t* and *instruct_curr_t* are redundant under non-trigger condition.

The HT implementation shown in Fig. 7 is not stealthy enough. *DeTrust* revises the HT design indicated by ⓐ, ⓑ and ⓒ to resist FANCI and VeriTrust, and the revised HT design is shown in Fig. 8. To resist FANCI at ⓐ and ⓑ, we limit the number of trigger inputs in each combinational logic to be no more than four by introducing multiple sequential levels, as shown by Fig. 8 (a). By doing so, the control value of each trigger input is increased to a level that is close to the control value of functional inputs. Finally, *instruct_prev_t*[10] and *instruct_curr_t*[10] are used to indicate the occurrence of the previous and current trigger instructions. We do not apply *DeTrust* for FANCI at ⓒ, since there are only two trigger inputs for signal *super*.

Input	$[t_1, t_2, d_1]$	$[h_1, h_2, h_3]$	*super*
h_1	$[0,1,0]$	$[0,1,0]$	0
	$[1,0,1]$	$[1,1,0]$	1
h_2	$[1,0,0]$	$[1,0,0]$	0
	$[1,0,1]$	$[1,1,0]$	1
h_3	$[0,1,0]$	$[0,1,0]$	0
	$[0,1,1]$	$[0,1,1]$	1

$t_1 = instruct\_prev\_t[10]$; $\quad h_1 = t_1$;
$t_2 = instruct\_curr\_t[10]$; $\quad h_2 = t_2 + \bar{t}_2 d_1$;
$d_1 = holdn\_in \& in.super\_in$; $\quad h_3 = (\bar{t}_1 + t_2) d_1$;
$super = \overline{resetn} + \overline{holdn} \cdot super + h_1 \cdot h_2 + h_3$.

Figure 9: Patterns used to prove that h_1, h_2 and h_3 are not redundant for *super*, where $[resetn, holdn, super] = [1,1,0]$

To resist VeriTrust at ⓒ, we carefully re-design the fan-in logic cone of *super* whose original Boolean function is shown in Fig. 8(b) and hide the trigger in the fan-in logic cones of three introduced signals, h_1, h_2 and h_3, as shown in Fig. 8(c). Note that, in order to keep the timing of all signals unchanged, h_2 and h_3 are driven by previous values of *holden* and *in.super*, denoted as *holden_in* and *in.super_in*. We can prove that *super*, h_1, h_2 and h_3 have no redundant inputs under non-trigger condition, according to the theoretical proof shown in Appendix. For example, as shown in Fig. 9, h_1 can be identified as non-redundant by VeriTrust, since the patterns of $[0,1,0]$ and $[1,1,0]$ for $[h_1, h_2, h_3]$ can be activated under non-trigger condition, leading to different output values for *super*.

The HT revised by *DeTrust* introduces about 80 code lines in the design file, which is comparable to HTs of [10]. We use the same environment as in the original experimental setup in FANCI [14] and VeriTrust [13] to validate the stealthiness of this HT in the OpenRISC processor, and we find that it successfully evades both HT identification techniques, which is further discussed in the following subsection.

4.2 The Stealthiness of the HT

Next, we study the stealthiness of the HTs designed with *DeTrust* with respect to all known verification techniques for hardware trust, including the static analysis with FANCI and various dynamic solutions (i.e., FV, UCI and VeriTrust).

4.2.1 Stealthiness in terms of FANCI

The experimental setup to evaluate the stealthiness of HT in terms of FANCI is set as follows. Experiments are conducted on benchmark circuits with various sizes, *s15850*, *s38417*, *s38584*, *wb_conmax* and *OpenRISC*. Only one HT designed by *DeTrust* is embedded in each benchmark circuit and it is similar to the one shown in Fig. 8. All the HTs introduce fewer than 200 gates, as shown in Table 2. For some of the benchmarks with RTL source code provided, we insert the HT directly into the RTL source code and then synthesize the whole design with Synposys Design Complier; for some of the benchmarks with the netlist provided, we synthesize the HT first and then insert it into the design netlist. Similar to the experimental setup in [14], we choose $2^{15} = 32,768$ items in the truth table to calculate the control value. Then, given a cut-off threshold, FANCI flags those signals with lower control values as suspicious ones.

As shown in Fig. 10 (a), the values of all HT-related signals calculated by the heuristic metric for the benchmark circuits are controlled at around 0.1. Consequently, if the cut-off threshold value is set as 0.001 as suggested in [14], FANCI would not flag these HTs designed with *DeTrust*. On the other hand, if we raise the cut-off threshold to be 0.1, FANCI is able to catch some HT-related signals, but it would suffer from a large number of false positives. Column 5 in Table 2 lists the concrete false positive rate when setting the cut-off threshold to be the lowest value where there is no false negative.

```
always @ (posedge clk) begin
  if (instruct_prev[3:0] == 'INSTRUCT_0 )
    instruct_prev_t[0] <= 1;
  else instruct_prev_t[0] <= 0;
    :          :          :
  if (instruct_prev[31:28] == 'INSTRUCT_7 )
    instruct_prev_t[7] <= 1;
  else instruct_prev_t[7] <= 0;
end
always @ (posedge clk) begin
  if (instruct_curr[3:0] == 'INSTRUCT_0 )
    instruct_curr_t[0] <= 1;
  else instruct_curr_t[0] <= 0;
    :          :          :
  if (instruct_curr[31:28] == 'INSTRUCT_7 )
    instruct_curr_t[7] <= 1;
  else instruct_curr_t[7] <= 0;
end
```

```
always @ (posedge clk) begin
  if (instruct_prev_t[3:0] == 4'b1111 )
    instruct_prev_t[8] <= 1;
  else instruct_prev_t[8] <= 0;
  if (instruct_prev_t[7:4] == 4'b1111 )
    instruct_prev_t[9] <= 1;
  else instruct_prev_t[9] <= 0;
  if (instruct_prev_t[8:9] == 2'b11 )
    instruct_prev_t[10] <= 1;
  else instruct_prev_t[10] <= 0;
end
always @ (posedge clk) begin
  if (instruct_curr_t[3:0] == 4'b1111 )
    instruct_curr_t[8] <= 1;
  else instruct_curr_t[8] <= 0;
  if (instruct_curr_t[7:4] == 4'b1111 )
    instruct_curr_t[9] <= 1;
  else instruct_curr_t[9] <= 0;
  if (instruct_curr_t[8:9] == 2'b11 )
    instruct_curr_t[10] <= 1;
  else instruct_curr_t[10] <= 0;
end
```

(a) Design of ⓐ and ⓑ to defeat FANCI

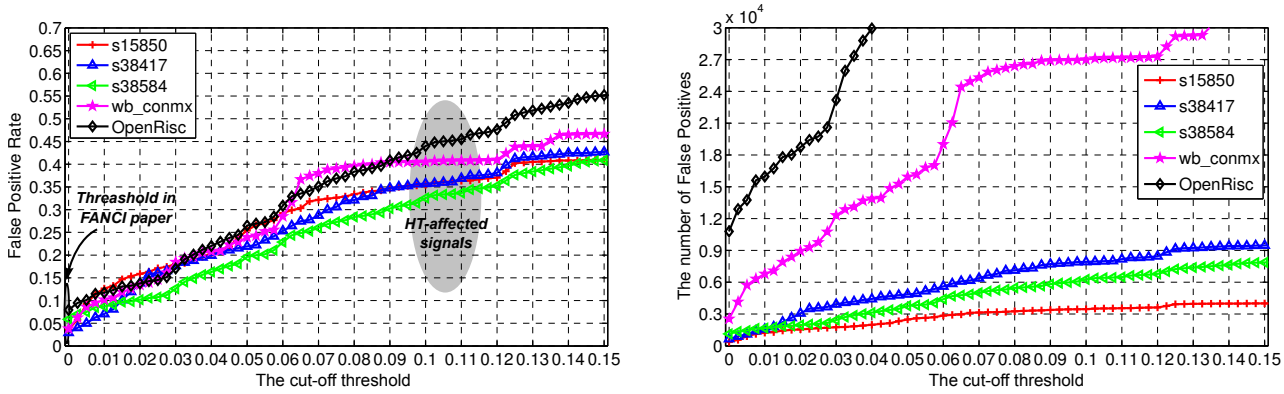
$$super = \overline{resetn} + holdn \cdot in.super + \overline{holdn} \cdot super + instruct\_prev\_t[10] \cdot instruct\_curr\_t[10];$$

(b) Boolean function of *super*

```
always @ (posedge clk) begin
  holdn <= holdn_in;
  in.super <= in.super_in;
  h1 <= instruct_prev_t[10];
  h2 <= ~instruct_curr_t[10]&holdn_in&in.super_in
        |instruct_curr_t[10];
  h3 <= (~instruct_prev_t[10] | instruct_curr_t[10])
        &(holdn_in&in.super_in);
end
always @ (posedge clk) begin
  super <=~resetn | ~holdn&super | h1&h2 | h3;
end
```

(c) Design of ⓒ to defeat VeriTrust

Figure 8: Supervisor transition foothold implemented by *DeTrust* (Verilog HDL)



(a) The false positive rate under different cut-off thresholds

(b) The number of false positives under different cut-off thresholds

Figure 10: The result of FANCI

Circuit	Circuit Size (No. of gates)	HT Size (No. of gates)	HT Size (Ratio %)	False Positive Rate %
s15850	10369	148	1.43	34.8
s38417	24370	148	0.61	34.2
s38584	21066	148	0.70	30.8
wb_conmax	75352	178	0.24	40.1
OpenRISC	143172	184	0.13	44.1

Table 2: The size of HTs and false positive rates under the cut-off threshold where HTs are just detected by FANCI.

Even under such optimistic assumptions, FANCI would have more than 30% false positive rates, which means that designers need to manually examine more than 30% of signal wires in the entire circuit to finally catch the HT from the candidate list. Fig. 10(b) present the number of wires for further examination with different cut-off threshold values for various benchmark circuits.

Based on the above, we conclude that *DeTrust* is resistant to FANCI.

4.2.2 *Stealthiness in terms of FV, UCI and VeriTrust*

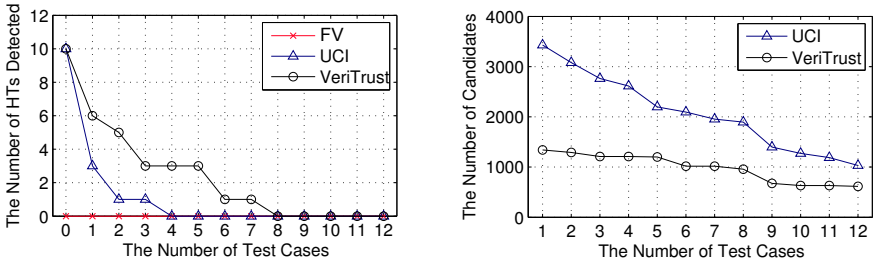
We adopt the same experimental environment of [13] to study the stealthiness of the HT with respect to dynamic verification solutions for HT identification. We conduct the experiment on the OpenRISC processor, considering the test cases required by these solutions. We adopt the 17 test cases bundled with the OpenRISC design for verification. Similar to [5], we use 5 test cases when implementing the HT and adopt the remaining 12 test cases to validate its stealthiness. The HTs used are listed in Table 3, wherein the first seven HTs (T1-T7) are from Trust-Hub[4] [6] while the last three HTs (T8-T10) are from some related papers [4, 10, 15]. All of the HTs are carefully transplanted from the original circuit to the OpenRISC design, keeping their trigger conditions and malicious behavior. Then, we implement them according to *DeTrust* to resist FV, UCI and VeriTrust.

The effectiveness of these dynamic verification techniques is closely related to the quantity and quality of verification test cases. We therefore show the number of detected HTs and the number of candidates reported for further examination with the increasing number of test cases to illustrate the stealthiness of the HTs designed with *DeTrust*. We consider that FV detects an HT if the trigger condition is satisfied, while UCI and VeriTrust detect an HT if any part of the HT is reported in the candidate list. Results are shown in Fig. 11.

Fig. 11 (a) shows the number of HTs detected with application of test cases. As can be observed, all HTs are able to evade FV, UCI and VeriTrust after all verification test cases are applied. For FV, all HTs evade it because none of these HTs has been activated. For UCI, all HTs evade it since all parts of the HT are treated as "useful circuit". Finally, all HTs evade VeriTrust because none of trigger inputs are identified as redundant inputs.

By examining the details in Fig. 11 (a), we have the following interesting observation. UCI and VeriTrust in fact are able to flag

[4]Trust-Hub HT benchmark suite contains various known HT triggers and payloads contributed by researchers in the hardware trust domain. For a detailed description of the code model, please refer to [5].



(a) The number of HTs detected with test cases

(b) The number of candidates with test cases

Figure 11: The result of FV, UCI and VeriTrust

Index	Circuit	Trigger	Payload
T1	MC8051	idle mode state	activate timer
T2	MC8051	a sequence of instructions	disable interrupt
T3	MC8051	a sequence of instructions	compromise data
T4	MC8051	a sequence of data	compromise stack pointer
T5	RISC	the number of instructions	compromise memory address
T6	RISC	the number of instructions	compromise the instruction
T7	RISC	the number of instructions	manipulate the address
T8	Leon3	a sequence of bus data	access any memories
T9	Leon3	a sequence of instructions	switch to administrator
T10	Leon3	a sequence of bus data	execute malicious codes

Table 3: The summary of HTs used in the experiment for FV, UCI and VeriTrust

some HTs with fewer test cases (e.g., less then 4 test cases shown in Fig. 11 (a)). This phenomenon, however, is not useful for HT identification in practice. This is because, fewer test cases would result in much more false positives, as shown in Fig. 11 (b), which leads to much more manual effort for further examination.

With the above, we conclude that *DeTrust* is also resistant to all known dynamic verification techniques used for HT identification.

5. POTENTIAL DEFENSES

In this section, we discuss potential defenses against *DeTrust*. We first show that, by extending state-of-the-art HT detection techniques such as FANCI and VeriTrust, we can alleviate the threat posed by *DeTrust*, but it is not an easy fix due to their associated computational complexity. Next, we discuss how to defend against *DeTrust* from a more practical viewpoint.

5.1 Extending FANCI and VeriTrust

One of the main reasons for *DeTrust* to be resistant to FANCI and VeriTrust is that these techniques focus on analyzing the combinational logics only. Thus, to evade FANCI and VeriTrust, *DeTrust* spreads HT trigger logic intro multiple sequential levels and combine it with normal logic to make HT become implicitly-triggered. From this perspective, it is possible to extend FANCI to verify signals across multiple sequential levels or extend VeriTrust to trace and verify signals across multiple sequential levels to defend against *DeTrust*, wherein the intermediate state elements can be simply treated as logic wires. The time-frame expansion [18] can be adopted to construct the combinational circuit that mimics the functionality of the sequential circuit.

The extensions for FANCI and VeriTrust may facilitate to identify HTs that could not be caught previously. For example, for the HT-infected circuit shown in Fig. 4 (c), if the extended FANCI analysis is conducted at the entire circuit level directly, the output signal o would be reported to be suspicious with the small control value of each trigger input as small as about $\frac{1}{2^{63}}$. For the example circuit shown in Fig. 6 (b), the extension for VeriTrust would catch the HT by verifying the entire circuit, because the implicit trigger condition is now made explicit and the corresponding logic entries keep dormant during verification.

The main limitation of the above extension is the associated computational complexity due to the exponential increase space where HT could be inserted. Without knowing the number of sequential levels that HT trigger logics cover, both extensions have to enumerate all the possibilities (e.g., one-level, two-level, ...), which is computationally-infeasible for large circuits. Specifically, for the extension of FANCI, the total number of analysis for HT-related signals increases dramatically, and Boolean functional analysis itself becomes much more difficult due to the exponential increase of the truth table size in the multi-level logics; for the extension of VeriTrust, the number of products and sums to be traced and checked increases exponentially with the size of the sequential logic. In the worst case, the above extension becomes equivalence to verifying all states of the circuit, which is therefore inapplicable to the large-scaled IC design.

Moreover, the above extension would introduce a large number of false positives. On the one hand, when performing analysis across multiple sequential levels for FANCI, the control values of those functional inputs would be also rather small, rendering either false negatives with small cut-off threshold or a large number of false positives with relatively high cut-off threshold. On the other hand, for the extension of VeriTrust, as simulation usually only covers a small functional space, with the increase of the functional space for potential HTs to hide with *DeTrust*, VeriTrust would encounter many false positives and flag many functional inputs as suspicious HT trigger inputs.

5.2 Discussion

From the above, we can conclude that simple extensions of existing trust verification techniques are not effective to defend against *DeTrust*. The main reason is that, with *DeTrust*, the problem space for trust verification of the entire circuit is at the same level as verifying its entire functional space, which is prohibitive for large circuits.

Consequently, for a specific design, a more practical solution to alleviate the threat posed by *DeTrust* is to reduce the problem space by conducting security analysis and protecting its main assets, based on our knowledge about the design. That is, with a given design, we first identify the critical components in the system, e.g., the cryptographic module. Next, we can adopt information flow checking techniques (e.g., [19]) to identify those circuitries that may affect these critical components. Finally, we run the extended trust verification techniques as discussed earlier for HT identification. Note that, if the problem space is still too large, we can further partition the critical components and focus on each functional block at a time (e.g., random number generator and key generator in a cryptographic module). However, care must be taken to verify the interface between these blocks to ensure the completeness of trust verification.

No doubt to say, the above design-aware HT identification solution significantly reduces computational complexity. However, how to perform security analysis in terms of HTs is still an open question.

6. RELATED WORK

In this section, we survey related work in the field of hardware security and trust.

6.1 Hardware Trust Challenges

Traditionally, the hardware layer of a secure computing system (e.g., [20–23]) is often implicitly regarded as trustworthy. This is a rather "naive" assumption, and various hardware Trojans have been presented in the literature.

King *et al.* [4] implemented two HTs in general-purpose processor, which grants privileged access to the memory elements of the system. Skorobogatov and Woods [7] found a backdoor in a military-grade FPGA device. Various HT designs that are able to to compromise cryptographic device were presented in [24,25]. These HTs are inserted at the design stage, and *DeTrust* can be used to enhance their stealthiness in terms of trust verification techniques.

HTs can be also inserted at the manufacturing stage. Lin *et al.* [26–28] proposed the so-called *Trojan side-channels*, which are HTs that can support side-channel attacks. In [29], Wei *et al.* presented three types of one-gate HT triggers based on switching power, leakage power, and delay measurements, respectively. Recently, Becker *et al.* [30] implemented a stealthy HT by changing the dopant polarity of transistors during the manufacturing process.

6.2 Side-Channel Analysis for HT Identification

Early works in hardware trust field are mainly concerned about HTs being inserted by a third-party foundry during the manufacturing process, and they rely on *side-channel analysis* (SCA) for HT identification. The idea behind is that an HT will affect some side-channel signatures (e.g., path delay, power consumption or supply current), even when it is not functionally activated. According to the signatures, they can be classified into timing-based analysis (e.g., [31]), current-based analysis (e.g., [32]), and power-based analysis (e.g., [33, 34]). Process variation has a significant impact on the effectiveness of early works on SCA analysis. Recently, gate-level characterization [35], multimodal analysis [36], and outlier analysis [37] are shown to be resistant to process variation effects and hence are quite promising.

One common assumption of the above HT detection techniques is the existence of HT-free golden ICs used as reference, and hence they are not applicable for identifying HTs inserted at design time.

6.3 Design for Hardware Trust

Ideally, we would like to prevent HTs from ever being inserted into circuits or ever being triggered at runtime. Some design-for-trust techniques presented in the literature tried to achieve the above objectives.

6.3.1 Design Time Prevention

Chakraborty and Bhunia [38] proposed to employ design obfuscation such that the circuit operates in two distinct modes, which dramatically increases the difficulty of HT insertion for attackers. Potkonjak [39] showed how to prevent untrusted CAD tool to compromise the design by checking at every synthesis step. For FPGA-based design, Huffmire *et al.* [40] proposed to physically isolate untrusted IP cores and trusted ones and restrict their communication, while Dutt and Li [41] adopted error correction coding (ECC) to detect design tampers that try to change, delete or add logic into the design.

6.3.2 Run Time Prevention

In [10], Hicks *et al.* also presented the so-called *BlueChip* concept to emulate the behavior of the suspicious circuitries at runtime. However, BlueChip identify suspicious circuitries with UCI algorithm only, and hence cannot detect HTs designed with *DeTrust*. Waksman and Sethumadhavan [11] proposed *TrustNet* and *DataWatch* to detect suspicious malicious behavior in the pipeline of the processor at runtime. However, they are only effective to certain pre-defined malicious behavior and their capabilities are limited by the amount of information to be checked at runtime. Later, the same authors [12] proposed to disable HTs at runtime by scrambling inputs of the hardware units. While effective for computational units, this technique would fail to disable HTs by *DeTrust* embedded in control logic. Dai *et al.* [42] proposed a specific HT detection method for Response-Computing Authentication module, but their approach cannot solve the general HTs designed by *DeTrust*.

7. CONCLUSION

IC products are the core components of electronic systems being used in daily life, and it is essential to ensure that they faithfully perform their specified functionalities. Hardware Trojans implemented by adversaries, being able to subvert or augment the normal operation of infected devices, are thus serious threats.

Recently, state-of-the-art hardware trust verification solutions such as FANCI and VeriTrust are shown to be able to effectively defend against existing HT designs presented in the literature. Unfortunately, this is not enough because it is expected that adversaries would adjust their tactics of attacks accordingly. Therefore, we need to examine whether new types of HTs can be designed to defeat these hardware trust verification techniques. In this paper, we present a so-called *DeTrust* HT design methodology that is able to be resistant to all known HT identification techniques, and its stealthiness has been validated with practical attacks performed on an OpenRISC processor. Finally, we show that there is no easy fix to existing solutions against the threat posed by *DeTrust*, calling for more advanced future works to ensure hardware trust.

8. ACKNOWLEDGMENTS

This work was supported in part by the Hong Kong SAR Research Grants Council (RGC) under General Research Fund No. CUHK418111 and No. CUHK418112.

9. REFERENCES

[1] M. Tehranipoor and F. Koushanfar. A survey of hardware trojan taxonomy and detection. *Design and Test of Computers*, 27(1):10–25, 2010.

[2] J. Markoff. Old trick threatens the newest weapons. *The New York Times*, 27, 2009.

[3] S. Adee. The hunt for the kill switch. *Spectrum, IEEE*, 45(5):34–39, 2008.

[4] S. T. King, J. Tucek, A. Cozzie, C. Grier, W. Jiang, and Y. Zhou. Designing and implementing malicious hardware. *LEET*, 8:1–8, 2008.

[5] J. Zhang and Q. Xu. On hardware trojan design and implementation at register-transfer level. In *Proc. IEEE International Symposium on Hardware-Oriented Security and Trust (HOST)*, pp. 107–112, 2013.

[6] *Trust-Hub Website*. http://www.trust-hub.org/resources/benchmarks.

[7] S. Skorobogatov and C. Woods. Breakthrough silicon scanning discovers backdoor in military chip. In *Proc. International Conference on Cryptographic Hardware and Embedded Systems (CHES)*, pp. 23–40, 2012.

[8] M. Beaumont, B. Hopkins, and T. Newby. Hardware trojans-prevention, detection, countermeasures (a literature review). Technical report, 2011.

[9] U.S.A. Department of Defense. Defense science board task force on high performance microchip supply. *Washington, DC*, pp. 2005–02, 2005.

[10] M. Hicks, M. Finnicum, S. T. King, M. K. Martin, and J. M. Smith. Overcoming an untrusted computing base: Detecting and removing malicious hardware automatically. In *Proc. IEEE Symposium on Security and Privacy (SP)*, pp. 159–172, 2010.

[11] A. Waksman and S. Sethumadhavan. Tamper evident microprocessors. In *Proc. IEEE Symposium on Security and Privacy (SP)*, pp. 173–188, 2010.

[12] A. Waksman and S. Sethumadhavan. Silencing hardware backdoors. In *Proc. IEEE Symposium on Security and Privacy (SP)*, pp. 49–63, 2011.

[13] J. Zhang, F. Yuan, L. Wei, Z. Sun, and Q. Xu. VeriTrust: verification for hardware trust. In *Proc. IEEE/ACM Design Automation Conference (DAC)*, Article No. 61, 2013.

[14] A. Waksman, M. Suozzo, and S. Sethumadhavan. FANCI: Identification of stealthy malicious logic using boolean functional analysis. In *Proc. ACM Conference on Computer and Communication Security (CCS)*, pp. 697–708, 2013.

[15] C. Sturton, M. Hicks, D. Wagner, and S. T King. Defeating UCI: Building stealthy and malicious hardware. In *Proc. IEEE International Symposium on Security and Privacy (SP)*, pp. 64–77, 2011.

[16] J. Bormann, et al. Complete formal verification of tricore2 and other processors. In *Design and Verification Conference*, 2007.

[17] *OpenCores Website*. http://opencores.org/.

[18] F. Fallah. Binary time-frame expansion. In *Proc. IEEE/ACM International Conference on Computer Aided Design (ICCAD)*, pp. 458–464, 2002.

[19] M. Tiwari, H. Wassel, B. Mazloom, S. Mysore, F. T. Chong, and T. Sherwood. Complete information flow tracking from the gates up. In *Proc. International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, pp. 109–120, 2009.

[20] J. Szefer and R. B. Lee. Architectural support for hypervisor-secure virtualization. 40(1):437–450, 2012.

[21] D. Champagne and R. B. Lee. Scalable architectural support for trusted software. In *Proc. Symposium on High Performance Computer Architecture (HPCA)*, pp. 1–12. IEEE, 2010.

[22] C. W. Fletcher, M. van Dijk, and S. Devadas. A secure processor architecture for encrypted computation on untrusted programs. In *Proc. ACM workshop on Scalable Trusted Computing (STC)*, pp. 3–8, 2012.

[23] *TPM Specification Architecture Overview*. http://www.trustedcomputinggroup.org/.

[24] A. Baumgarten, M. Steffen, M. Clausman, and J. Zambreno. A case study in hardware trojan design and implementation. *International Journal of Information Security*, 10:1–14, 2011.

[25] Y. Jin, N. Kupp, and Y. Makris. Experiences in hardware trojan design and implementation. In *Proc. IEEE International Workshop on Hardware-Oriented Security and Trust (HOST)*, pp. 50–57, 2009.

[26] L. Lin, M. Kasper, T. Güneysu, C. Paar, and W. Burleson. Trojan side-channels: lightweight hardware trojans through side-channel engineering. In *Proc. International Conference on Cryptographic Hardware and Embedded Systems (CHES)*, pp. 382–395, 2009.

[27] L. Lin, W. Burleson, and C. Paar. Moles: malicious off-chip leakage enabled by side-channels. In *Proc. IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pp. 117–122, 2009.

[28] Y. Liu, Y. Jin, and Y. Makris. Hardware Trojans in wireless cryptographic ICs: silicon demonstration & detection method evaluation. In *Proc. IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pp. 399–404, 2013.

[29] S. Wei, K. Li, F. Koushanfar, and M. Potkonjak. Hardware trojan horse benchmark via optimal creation and placement of malicious circuitry. In *Proc. IEEE/ACM Design Automation Conference (DAC)*, pp. 90–95, 2012.

[30] G. T. Becker, F. Regazzoni, C. Paar, and W. P. Burleson. Stealthy dopant-level hardware trojans. In *Proc. International Conference on Cryptographic Hardware and Embedded Systems (CHES)*, pp. 197–214, 2013.

[31] Y. Jin and Y. Makris. Hardware trojan detection using path delay fingerprint. In *Proc. IEEE International Workshop on Hardware-Oriented Security and Trust (HOST)*, pp. 51–57, 2008.

[32] D. Du, S. Narasimhan, R. S. Chakraborty, and S. Bhunia. Self-referencing: a scalable side-channel approach for hardware trojan detection. In *Proc. International Conference on Cryptographic Hardware and Embedded Systems (CHES)*, pp. 173–187, 2010.

[33] D. Agrawal, S. Baktir, D. Karakoyunlu, P. Rohatgi, and B. Sunar. Trojan detection using IC fingerprinting. In *Proc. IEEE Symposium on Security and Privacy (SP)*, pp. 296–310, 2007.

[34] R. M. Rad, X. Wang, M. Tehranipoor, and J. Plusquellic. Power supply signal calibration techniques for improving detection resolution to hardware trojans. In *Proc. IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pp. 632–639, 2008.

[35] S. Wei, S. Meguerdichian, and M. Potkonjak. Gate-level characterization: foundations and hardware security applications. In *Proc. IEEE/ACM Design Automation Conference (DAC)*, pp. 222–227, 2010.

[36] K. Hu, A. N. Nowroz, S. Reda, and F. Koushanfar. High-sensitivity hardware trojan detection using multimodal characterization. In *Proc. IEEE/ACM Design, Automation and Test in Europe (DATE)*, pp. 1271–1276, 2013.

[37] J. Zhang, H. Yu and Q. Xu. HTOutlier: hardware Trojan detection with side-channel signature outlier identification. In *Proc. IEEE International Symposium on Hardware-Oriented Security and Trust (HOST)*, pp. 55–58, 2012.

[38] R. S. Chakraborty and S. Bhunia. Security against hardware trojan through a novel application of design obfuscation. In *Proc. IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pp. 113–116, 2009.

[39] M. Potkonjak. Synthesis of trustable ICs using untrusted CAD tools. In *Proc. IEEE/ACM Design Automation Conference (DAC)*, pp. 633–634, 2010.

[40] T. Huffmire, B. Brotherton, G. Wang, T. Sherwood, R. Kastner, T. Levin, T. Nguyen, and C. Irvine. Moats and drawbridges: An isolation primitive for reconfigurable hardware based systems. In *Proc. IEEE Symposium on Security and Privacy (SP)*, pp. 281–295, 2007.

[41] S. Dutt and L. Li. Trust-based design and check of FPGA circuits using two-level randomized ECC structures. *Transcations on Reconfigurable Technology and System*, 2(1):1–36, 2009.

[42] S. Dai, T. Wei, C. Zhang, T. Wang, Y. Ding, Z. Liang, and W. Zou. A framework to eliminate backdoors from response-computable authentication. In *Proc. IEEE Symposium on Security and Privacy (SP)*, pp. 3–17, 2012.

APPENDIX

In this appendix, we prove that HT designed with *DeTrust* is able to evade VeriTrust under all non-trigger conditions. All symbols used in the following are listed in Tabel 4.

Let us first consider the Boolean function of the HT-affected signal without *DeTrust*, wherein there is only one malicious on-set term $c_{m_0}p_{m_0}$, given by Eq. 6 (see Section 3.3.2):

$$\begin{aligned} f &= f'_n + (c_{n_0}p_{n_0} + c_{m_0}p_{m_0}) \\ &= f'_n + c^c p^c (c^r_{n_0} p^r_{n_0} + c^r_{m_0} p^r_{m_0}), \end{aligned}$$

where

$$\begin{aligned} &c_{n_0} = c^c c^r_{n_0}, \;\; p_{n_0} = p^c p^r_{n_0}, \\ &c_{m_0} = c^c c^r_{m_0}, \;\; p_{m_0} = p^c p^r_{m_0}, \\ &c_{n_0} \in C_n, \;\; p_{n_0} \in P_n, \\ &c_{m_0} \in C_m, \;\; p_{m_0} \in P_m, \end{aligned}$$

and $c_{m_0}p_{m_0}$ denotes a malicious on-set term/product, $c_{n_0}p_{n_0}$ denotes a on-set term/product from the normal function, $c^c p^c$ denotes the common literals of $c_{n_0}p_{n_0}$ and $c_{m_0}p_{m_0}$, and f'_n denotes all on-set terms/products from the normal function except $c_{n_0}p_{n_0}$. With *DeTrust*, we have

$$f = h_1 h_2 + h_3,$$

where

$$\begin{aligned} h_1 &= c^c p^c \\ h_2 &= c^r_{n_0} p^r_{n_0} + c^r_{m_0} p^r_{m_0}. \\ h_3 &= f'_n \end{aligned}$$

To prove that the above HT implementation is able to evade VeriTrust, we need to prove that f, h_1, h_2 and h_3 have no redundant inputs under non-trigger conditions. To be specific, we first prove that h_1, h_2 and h_3 are not redundant for f under non-trigger condition in *Statement 1*, *Statement 2* and *Statement 3*, and then prove that h_1, h_2 and h_3 have no redundant inputs under non-trigger condition in *Statement 4*.

Symbol	Meaning
T	the set of trigger inputs, $T = \{t_1, t_2, \ldots, t_m\}$
T^c/T^r	the subset of T, $T^c \subset T$,$T^r \subset T$
D	the set of functional inputs, $D = \{d_1, d_2, \ldots, d_n\}$
D^c/D^r	the subsets of D, $D^c \subset D$ and $D^r \subset D$
C	the set of all conditions driven by all trigger inputs T, $C = \{c_1, c_2, \ldots, c_{2^m}\}$
C_n	the set of non-trigger conditions, $C_n = \{c_{n_1}, c_{n_2}, \ldots, c_{n_x}\}$, $C_n \subset C$
C_m	the set of the trigger conditions, $C_m = \{c_{m_1}, c_{m_2}, \ldots, c_{m_y}\}$, $C_m \subset C$, $C_m \cup C_n = C$ $C_m \cap C_n = \varnothing$
C^c/C^r	the set of all conditions driven by T^c/T^r
F	the set of all input patterns driven by all functional inputs D
P_n	the subset of F that let the normal function output logic '1', $P_n = \{p_{n_1}, p_{n_2}, \ldots, p_{n_y}\}$, $P_n \subset F$
P_m	the subset of F that let the malicious function output logic '1', $P_m = \{p_{m_1}, p_{m_2}, \ldots, p_{m_z}\}$, $P_m \subset F$
F^c/F^r	the set of all input patterns driven by D^c/D^r

Table 4: List of Notations

To prove that an input is not redundant under non-trigger condition, our idea is to find one pair of input patterns that meet the following three requirements:

- they are different only at the value of this input;
- they would generate different output values;
- they can be activated under non-trigger condition.

Next, let us present proofs of *Statement 1, Statement 2* and *Statement 3* in the following, respectively.

Statement 1: h_1 is not redundant for f under non-trigger condition.

PROOF. We use input patterns, $\{1,1,0\}$ and $\{0,1,0\}$ for $\{h_1,h_2,h_3\}$, to prove *statement 1*. These two input patterns meet the first two requirements, and next we prove that they can be activated under non-trigger condition as follows.

The input pattern, $\{1,1,0\}$, can be generated by activating $c_{n_0}p_{n_0}$ that belongs to the normal function. With $c_{n_0}p_{n_0}$, $h_1 = 1$, as $c^c p^c$ is activated; $h_2 = 1$, as $c^r_{n_0}p^r_{n_0}$ is activated; $h_3 = 0$, as $h_3 = f'_n$ does not include $c_{n_0}p_{n_0}$. Note that only $c_{n_0}p_{n_0}$ can generate $\{1,1,0\}$.

The input pattern, $\{0,1,0\}$, can be generated by activating a set of input patterns, $\{c^c_i c^r_{m_0} p^c p^r_{m_0} | c^c_i \in C^c, c^c_i \neq c^c\}$ which are controlled by non-trigger condition. With such input patterns, $h_1 = 0$, as $c^c_i p^c$ where $c^c_i \neq c^c$ is activated; $h_2 = 1$, as $c^r_{m_0}p^r_{m_0}$ is activated; $h_3 = 0$, as $\{c^c_i c^r_{m_0} p^c p^r_{m_0} | c^c_i \in C^c, c^c_i \neq c^c\}$, that is the neighbor of the malicious on-set term/product of $c^c c^r_{m_0} p^c p^r_{m_0}$, must be equal to logic '0', according to [13]. Note that $\{c^c_i c^r_{m_0} p^c_j p^r_{m_0} | c^c_i \in C^c, c^c_i \neq c^c; p^c_j \in F^c, p^c_j \neq p^c\}$, $\{c^c_i c^r_{n_0} p^c p^r_{n_0} | c^c_i \in C^c, c^c_i \neq c^c\}$, $\{c^c c^r_{n_0} p^c_j p^r_{n_0} | p^c_j \in F^c, p^c_j \neq p^c\}$, and $\{c^c_i c^r_{n_0} p^c_j p^r_{n_0} | c^c_i \in C^c, c^c_i \neq c^c; p^c_j \in F^c, p^c_j \neq p^c\}$ are another four sets whose elements are possible to generate $\{0,1,0\}$, as long as they are not included into f'_n.

Since $f = h_1 h_2 + h_3$, only the pair of input patterns, $\{1,1,0\}$ and $\{0,1,0\}$, can be used to prove *Statement 1*. ∎

Statement 2: h_2 is not redundant for f under non-trigger condition.

PROOF. We use input patterns, $\{1,1,0\}$ and $\{1,0,0\}$ for $\{h_1,h_2,h_3\}$, to prove *statement 2*. These two input patterns meet the first two requirements, and next we prove that they can be activated under non-trigger condition as follows.

The input pattern, $\{1,1,0\}$, can be generated by $c_{n_0}p_{n_0}$, which has been proved in the *statement 1*.

The input pattern, $\{1,0,0\}$, can be generated by activating a set of input patterns, $\{c^c c^r_{m_i} p^c p^r_{m_0} | c^r_{m_i} \in C^r, c^r_{m_i} \neq c^r_{m_0}\}$ which are controlled by non-trigger condition. With such input patterns, $h_1 = 1$, as $c^c p^c$ is activated; $h_2 = 0$, as both $c^r_{m_0}p^r_{m_0}$ and $c^r_{n_0}p^r_{n_0}$ are not activated; $h_3 = 0$, as the activated $\{c^c c^r_{m_i} p^c p^r_{m_0} | c^r_{m_i} \in C^r, c^r_{m_i} \neq c^r_{m_0}\}$, that is the neighbor of malicious on-set term/product of $c^c c^r_{m_0} p^c p^r_{m_0}$, must be equal to logic '0'. Note that $\{c^c c^r_{m_i} p^c p^r_{m_j} | c^r_{m_i} \in C^r, c^r_{m_i} \neq c^r_{m_0}; p^r_{m_j} \in F^r, p^r_{m_j} \neq p^r_{m_0}\}$ is another set of input patterns to possibly generate $\{1,0,0\}$ as long as $c^r_{n_0}p^r_{n_0} = 0$ and $f'_n = 0$.

Since $f = h_1 h_2 + h_3$, only the pair of input patterns, $\{1,1,0\}$ and $\{1,0,0\}$, can be used to prove *Statement 2*. ∎

Statement 3: h_3 is not redundant for f under non-trigger condition.

PROOF. We use input patterns, $\{0,0,0\}$ and $\{0,0,1\}$ for $\{h_1,h_2,h_3\}$, to prove *statement 3*. These two input patterns meet the first two requirements, and next we prove that they can be activated under non-trigger condition as follows.

The input pattern, $\{0,0,0\}$, can be generated by activating a set of input patterns, $\{c_i p_i | c_i \in C_n, c_i \neq c_{n_0}; p_i \in (F - P_n), p_i \neq p_{m_0}\}$ which are controlled by non-trigger conditions. With such input patterns, $h_1 = 0$ and $h_2 = 0$, as $c^c p^c$, $c^r_{n_0}p^r_{n_0}$ and $c^r_{m_0}p^r_{m_0}$ are not activated; $h_3 = 0$, as f'_n outputs logic '0' due to $\{p_i | p_i \in (F - P_n), p_i \neq p_{m_0}\}$.

The input pattern, $\{0,0,1\}$, can be generated by activating a set of patterns, $\{c_i p_i | c_i \in C_n, c_i \neq c_{n_0}; p_i \in P_n, p_i \neq p_{m_0}\}$ which are controlled by non-trigger condition. With such input patterns, $h_1 = 0$ and $h_2 = 0$, as it is easy to find an element in this set where $c^c p^c$, $c^r_{n_0}p^r_{n_0}$ and $c^r_{m_0}p^r_{m_0}$ are not activated; $h_3 = 1$, as f'_n outputs logic '1' due to $\{c_i | c_i \in C_n, c_i \neq c_{n_0}\}$ and $\{p_i | p_i \in P_n, p_i \neq p_{m_0}\}$.

Moreover, *Statement 3* can be proven by the $\{0,1,0\}$ and $\{0,1,1\}$, and $\{1,0,0\}$ and $\{1,0,1\}$ as well. We do not list all those input patterns, since above is enough to prove *Statement 3*. ∎

With the above, *Statement 1*, *Statement 2* and *Statement 3* together prove that f has no redundant inputs under non-trigger condition.

Statement 4: h_1, h_2 and h_3 have no redundant inputs under non-trigger condition.

PROOF. All input patterns of h_1, h_2 and h_3 could be activated without triggering the HT, since the complete trigger condition does not exist in their fan-in cones. Therefore, all their inputs are not redundant under non-trigger condition. ∎

If the malicious function contains more than one on-set terms, with *DeTrust*, we have

$$f = \sum_{i=0}^{k-1} (h_{2i+1} h_{2i+2}) + h_{2k+1},$$

where

$$h_{2i+1} = c^{c_i} p^{c_i}$$
$$h_{2i+2} = c^{r_i}_{n_i} p^{r_i}_{n_i} + c^{r_i}_{m_i} p^{r_i}_{m_i},$$
$$h_{2k+1} = f'_n$$

as shown in Eq. 11 (see Section 3.3.2). We are able to follow the above procedure to prove that h_{2i+1}, h_{2i+2} and h_{2k+1} are not redundant for f, and meanwhile $h_1, h_2, \ldots, h_{2k+1}$ have no redundant inputs. Consequently, *DeTrust* is able to successfully evade VeriTrust.