Toward Building and Optimizing Trustworthy Systems Using Untrusted Components: A Graph-Theoretic Perspective

Xiaotong Cui, Xing Zhang, Hao Yan, Liang Zhang, Kefei Cheng, Yu Wu, and Kaijie Wu

Abstract—**The globalization process for integrated circuits (ICs) raises serious concerns regarding hardware Trojans (HTs). Due to the stealth and variety of HTs, detecting them at test time can be very resource intensive. This situation becomes even worse when the device under test (DUT) contains untrusted third-party intellectual property (3PIP) cores. For systems on chip (SoCs) that rely on untrusted 3PIP cores, this article solves the online HT detection and recovery problem from a graph-theoretic perspective. The proposed graph-theoretic models minimize the implementation cost of the system in terms of both the cost of purchasing different IP cores and the area overhead. To further enhance the security of the system, we also propose schemes to locate and replace an HT-infected IP core. The feasibility and efficiency of the proposed techniques are verified by experiments.**

Index Terms—**Graph theoretic, hardware Trojan (HT), localization and replacement, third-party IP core.**

I. Introduction

MANY fabricationless design companies outsource their designs to third-party foundries for fabrication to achieve lower cost and faster iteration. Such an integrated circuit (IC) development process, however, raises serious concerns about security and trustworthiness. One concern is that an adversary may insert hardware Trojans (HTs) in order to cause logic errors, secret leakage, performance degradation and system crashes. According to [1]-[4], HTs can be inserted into ICs at any phase(s) of the IC development process, such as specification, design, fabrication, testing, assembly, and packaging, which makes it difficult to develop a comprehensive method to counter all kinds of HTs.

Manuscript received November 22, 2020; revised January 18, 2021 and April 11, 2021; accepted May 30, 2021. Date of publication June 7, 2021; date of current version April 21, 2022. This work was supported in part by the State Key Laboratory of Computer Architecture Research Fund under Grant CARCH201902; in part by the State Key Laboratory of Vehicle NVH and Safety Technology Research Fund under Grant NVHSKL-202114; in part by the Science and Technology Research Program of Chongqing Municipal Education Commission under Grant KJQN201900641 and Grant KJQN202000632; and in part by the Education Reform Project on course "Hardware Security" of CQUPT. This article was recommended by Associate Editor S. Ghosh. *(Corresponding author: Kefei Cheng.)*

Xiaotong Cui is with the School of Cyber Security and Information Law, Chongqing University of Posts and Telecommunications, Chongqing 400065, China, and also with the Smart Security Center, State Key Laboratory of Vehicle NVH and Safety Technology, Chongqing 401122, China (e-mail: xiaotong.sd@gmail.com).

Xing Zhang, Hao Yan, Liang Zhang, Kefei Cheng, and Yu Wu are with the School of Cyber Security and Information Law, Chongqing University of Posts and Telecommunications, Chongqing 400065, China (e-mail: chengkf@cqupt.edu.cn).

Kaijie Wu is with the Department of ECE, New York University, New York, NY 10003 USA.

Digital Object Identifier 10.1109/TCAD.2021.3086765

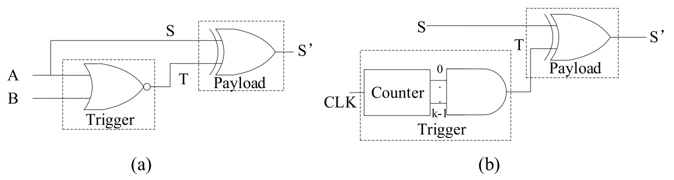


Fig. 1. HT with a trigger that uses (a) combinatorial logic and (b) sequential logic.

An HT typically consists of a trigger and a payload [5]. The trigger logic can be either combinatorial or sequential, and it continually monitors a set of inputs or signals and activates the payload under a certain condition. Once activated, the payload may cause malfunctions or system failures. Illustrations of combinatorial trigger logic and sequential trigger logic are shown in Fig. 1(a) and (b), respectively. In Fig. 1(a), the payload is activated under the condition that $A = 0, B = 0$. In Fig. 1(b), the payload is activated when the counter reaches $2^k - 1$.

The key feature of an HT is its stealth [6], [7]; i.e., 1) it can use as few as tens of gates while the host circuit could have millions and 2) it has extremely rare trigger conditions and remains silent or inactive most of the time. As a result, activating HTs during the test phase is quite challenging because it is impractical to generate exhaustive test vectors for traversing a large HT space.

Instead, one set of approaches attempts to activate HTs by identifying inactive nets in the host circuit and increasing their transition probabilities [8]–[11]. This works to some extent because most HTs are designed to minimize the possibility of accidental activation as the first priority, and hence, they are more likely to connect to inactive nets [12].

Another set of approaches attempts to observe exceptional side-channel characteristics [13], [14], as the extra circuitry introduced by HTs inevitably causes some side effects [15], including an extended path delay [16]–[18], increased current or power consumption [19]–[25], and abnormal electromagnetic emanations [26]–[28]. Although these exceptional characteristics are not very significant and have less of an effect than noise introduced by process variations and measurement noise, they will be present even if the HT is not activated.

In addition, built-in self-authentication and obfuscation techniques are used to detect HT insertion or to compromise the ability of HTs to generate targeted attacks [29]–[31].

A. Related Work and Motivation

Although there are various HT detection techniques at test time, most of them cannot guarantee the detection of all possible HTs. This situation worsens when employing third-party intellectual property (3PIP) cores for IC designs [32]. This is because an IP-core vendor will only provide IP cores with required functions, while no "golden" (i.e., Trojan-free) models are available for trustworthiness verification of these IP cores.

Rajendran *et al.* [33] classified HT detection techniques for 3PIPs into three categories: 1) code/structural analysis [34], [35]; 2) formal verification [36]; and 3) design for trust techniques [37]–[39].

Code analysis approaches may identify suspicious signals as a part of a Trojan [40], [41], but they cannot guarantee that the delivered code is fault free even with 100% code coverage. Structural analysis methods, such as FANCI [34] and VeriTrust [42] mark gates with low activation probabilities and gates that are not driven by functional inputs as suspicious, respectively. Then, these gates are manually analyzed to determine whether they are part of a Trojan. These two methods can both be defeated by DeTrust [35].

In the case of formal verification, the 3PIP should satisfy a set of predefined properties. Then, it can be converted into a proof check format for coincidence verification. Formal verification has been used to detect data leakage and the malicious modification of registers [36]. The limitations of formal verification include the following: 1) the satisfaction of predefined properties cannot guarantee the elimination of vulnerabilities and 2) there are equivalent conversion problems between the proof checking format and VHDL/Verilog [33]. As a result, code/structural analysis and formal verification techniques cannot guarantee the establishment of trustworthy systems. Design-for-trust techniques, on the other hand, which modify the target design to mute and/or detect the consequences of HTs, are promising in building trustworthy systems with untrusted components. The relevant design-for-trust techniques are as follows.

Amin *et al.* [37] proposed a majority voting technique that uses an odd number of untrusted IP cores from multiple vendors to protect against functional disruptions and availability (DoS) attacks. As at least three IP cores from different vendors are used at the same time and the outputs of the IP cores need to be compared at each cycle, this technique incurs a large computing resource overhead.

Based on IP core diversity and a redundant computing paradigm, Rajendran *et al.* [38] defined two task-to-IP core binding rules to detect HT attacks and prevent collusion among IP cores from the same vendor. The way that attacks are conducted by HTs (with collusion) within IP cores was illustrated in [33]. The shortcoming of their technique is that it only provides HT detection without recovery, which is not acceptable, especially for mission-critical real-time systems. Cui *et al.* extended their work by adding a recovery phase with extra recovery rules. They proposed an integer linear programming (ILP) model to optimize the system cost in terms of the number of different IP cores [39]. However, the ILP model performs exhaustive search and hence is inefficient in solving medium-to-large-scale problems.

TABLE I
COMPARISON OF RUN-TIME DESIGN-FOR-TRUST TECHNIQUES

Methods	Features	Overhead		Limitations
		Hardware	Power	
[37]	Direct majority voting	3x	3x	Limit types of HTs; Large power overhead
[38] [33]	Refined rules for HT collusion	2x	2x	No recovery for business continuity
[39]	Improved rules for recovery	3x	2x	Exponential complexity for large circuits
[44]	Continuous states monitoring	1.33x	1.03x	No recovery for Business continuity

This work	Precise threat model;
	Localization for fast recovery;
	Replacement for more HTs;
	Polynomial complexity
		3x	2x	Extra overhead for Loc.& Rep.;
				Cannot guarantee optimal solutions

To prevent accesses to confidential data from malicious IP cores, Gundabolu and Wang proposed a data protection technique based on access control mechanism. But it is unable to detect logic errors incurred by HTs [43]. Sayed-Ahmed *et al.* designed a subsystem to monitor a series of continuous states. Once behaviors of a IP core violate the security rules, then the network originated from the IP core will be disconnected to isolate it [44]. Obviously, this method cannot guarantee business continuity and is not applicable to mission-critical systems.

In addition to the shortcomings in the above studies to be addressed, there are two other realistic problems: 1) simply assuming all available IP cores are from untrusted vendors is overly pessimistic and will result in an overestimated system implementation cost and 2) locating and replacing an infected IP core when detecting a Trojan circuit in a multiprocessor system is necessary; this is not yet resolved but is of great importance for the further reliability and security enhancement of the system.

The comparison between the proposed technique and other run-time design-for-trust techniques, including their features and limitations, is summarized in Table I. It is observed that the proposed technique is preferred when there are potentially more than one HTs in a large-scale circuit and a better-performance recovery scheme is required.

With untrusted 3PIP cores, we propose to establish trustworthy systems and optimize systems' cost at system level, where graph coloring and bipartite graph matching models are used to ensure the security with suitable overhead. In addition, we propose a reconfigurable framework for HT-infected IP core localization and replacement to further enhance the security of the system.

The main contributions of this article are as follows.

1) We propose a precise threat model regarding systems on chip (SoCs) that rely on untrusted 3PIP cores. It is observed under this threat model, the implementation cost of the system decreases significantly when applying design-for-trust techniques.

2) Based on the threat model, the graph-theoretic techniques we propose to address the underlying security problem are adaptable to different-scale applications.
3) We propose a framework for HT-infected IP core localization and replacement, which not only improves the system performance when HTs are detected but also enhances the reliability and security of the system.
4) We explore the tradeoff between area and performance overhead while satisfying security policies, which can be used for system resource optimization.

The remainder of this article is organized as follows. Section II presents the threat model. Section III illustrates the proposed graph-theoretic models. Section IV introduces the framework for infected IP core localization and replacement. Section V presents the experiments. Section VI concludes the paper.

II. Threat Model

Untrusted 3PIP cores may contain HTs. As the SoC designers or integrators, our goal is to use these untrusted components to build a trustworthy system, where the application is partitioned into a series of tasks and these tasks are mapped to hardware modules (IP cores from untrustworthy vendors in this scenario). Mapping mechanisms (including task scheduling and binding operations) need to be designed to ensure appropriate security and cost of the system.

Once IP cores from untrusted vendors are integrated into SoCs, HTs (if any) in these IP cores may destroy the functionality of the system during "run time." Thus in this article, we try to deploy run-time security schemes for SoCs that rely on untrusted 3PIPs and work in postfabrication phase such that the underlying systems could function properly and securely.

However, assuming that all IP cores are from untrusted vendors is overly pessimistic. A typical kind of scenario is using split manufacturing for IP protection of IC [45], [46], where the trusted side takes the tasks of low-tech backend of line while the untrusted side takes the tasks of high-tech frontend of line. In this scenario, we have a set of low-end trusted IP cores while the untrustworthy third party provides high-end untrusted IP cores.

HTs within IP cores can be further classified as functional or parametric. Once triggered, functional HTs are designed to modify the original outputs of some logics, i.e., by causing logic errors, while parametric HTs are able to make the system unstable, e.g., to result in a short lifetime. Compared to parametric HTs, functional HTs are more aggressive, and they are considered in this article.

Although an activated HT causes logic errors, it cannot be modeled by traditional fault models because its mechanism and impact are different from that of environment-induced soft/temperary or hard/permanent errors of hardware [47]. An environment-induced soft error of hardware, such as a single event upset that makes a bit reversed temporally, will disappear after a period of time. Hence, the affected unit remains healthy [48]. The soft error of hardware can either be recovered by re-executing the task in the same hardware with time redundancy or using another hardware to execute the task in parallel with space redundancy. An environment-induced hard error, such as a single event latch-up that makes a bit line of the hardware stuck-at-0, does not disappear. Hence, the affected unit is deemed faulty thereafter [49]. The hard error of hardware can only be recovered by using another hardware unit to execute the task in parallel with space redundancy.

With these time and/or space redundancies, it is easy to recover environment-induced soft or hard errors. A logic error caused by a HT, on the other hand, will persist if its trigger condition remains true and will disappear (deactivate) if its trigger condition becomes invalid. This means that re-executing the task in the same hardware with time redundancy will not work if the HT's trigger condition holds true. Using another hardware to execute the task in parallel with space redundancy may not work either since these hardware units from the same vendor will contain the same HT.

In this article, HTs are characterized by ACTIVATION MECHANISM and PAYLOAD. We consider HTs that will be activated under some rare conditions and will be deactivated under all other conditions. Once Triggered, the payload will cause logic errors, i.e., change the functionality of the host circuit.

III. Problem Formulation

In this section, we first summarize the rules for runtime HT detection and recovery. Then, we formulate the security problem.

A. Premise

The runtime HT detection and recovery scheme takes advantage of IP cores from diverse vendors. Although no "golden" IP core is available for comparison to ensure trust, the outputs of two IP cores of the same functionality from different vendors can be compared to discover abnormal behavior. This is based on the assumption that "IP cores from different vendors tend to have different implementations. It is unlikely that they are both Trojan infected. Even if they are, the chance that their Trojans are activated in the same way is assumed to be extremely rare"[1] [33], [37]–[39].

The secure scheme consists of a detection phase and a recovery phase. In the detection phase, there is a normal computing and a recomputing. Normal computing is defined as a continuous execution of subtasks that will eventually achieve goals of an application. For example, to obtain the final results of $f = a_1x_1 + a_2x_2$, we need a series of computations. Recomputing is defined as a re-execution of subtasks of the normal computing on the same or different hardware units. For mission-critical applications, normal computing and recomputing are necessary parts of fault-tolerant schemes. In HT detection scenario, normal computing and recomputing are adapted according to security policies, then the results of normal computing and recomputing are compared to check whether there are HTs. If the results are not consistent, the system will switch to the recovery phase to execute recovery computing. Taking the 5-node *polynom* benchmark in Fig. 2(a)

[1]The basic assumption, on which basis that the diversified duplication mechanisms can work.

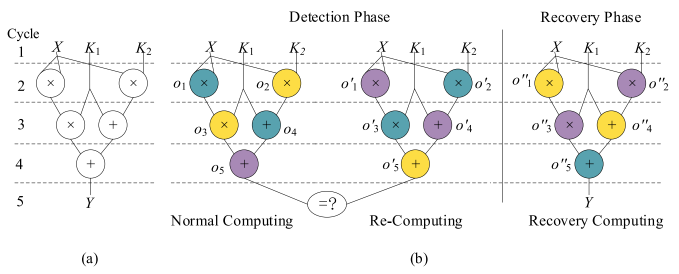


Fig. 2. Polynom benchmark. (a) DFG of *polynom*. (b) HT detection and recovery scheme of *polynom*.

TABLE II
AREA SETUP OF THE IP CORES USED IN THE BENCHMARKS

Cycle	Normal computing					Recomputing				
	o_1	o_2	o_3	o_4	o_5	o'_1	o'_2	o'_3	o'_4	o'_5
1	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
2	25	15	N/A	N/A	N/A	25	15	N/A	N/A	N/A
3	25	15	**30**	17	N/A	25	15	50	17	N/A
4	25	15	**30**	17	**47**	25	15	50	17	67
5	$o_5 \neq o'_5$ followed by recovery computing									
	o''_1	o''_2	o''_3	o''_4	o''_5					
6	25	15	N/A	N/A	N/A					
7	25	15	50	17	N/A					
8	25	15	50	17	67					

as an example, the security policies are summarized by the following two task-to-IP core diversity binding rules.

1) *Rule 1:* The corresponding task nodes in normal computing, recomputing and recovery computing are bound to IP cores from different vendors.
2) *Rule 2:* Task nodes that have a parent-child relation, provide inputs to the same child, or have closely related inputs[2] should be bound to IP cores from different vendors.

The resulting scheme for the *polynom* benchmark is shown in Fig. 2(b), where task nodes in different colors are bound to IP cores from different vendors.

Let us look into the example in Fig. 2 to check how security rules work. The function denoted by Fig. 2(a) is $f = K_1X^2 + K_2X + K_1$. Assume that $K_1 = 2$, $K_2 = 3$, $X = 5$ and there are no HTs in all IP cores, then the output of normal computing will be 62 as well as that of recomputing. In such a case, the recovery computing will not be executed. However, if a HT in IP core of o_3 is activated under input pattern (25, 2) during function mode and it changes o_3's output to 30. Consequently, the output of normal computing is different from that of recomputing and the recovery computing will be executed. Values of all intermediate outputs could be find in Table II, where values in bold are wrong outputs due to the activation of HT.

Although IP cores of o'_2, o'_3, and o_1 may contain the same HT as they are in the same type and from the same vendor, HTs in IP cores of o'_2 and o'_3 will not be activated due to different input conditions. On the other hand, the output of recovery computing will be correct since no HTs will be activated.

[2]Task nodes with close-related inputs typically have similar or the same inputs. In some computing paradigms like FFT and FIR filters, they form symmetric structures.

It should be noted that the security rules have a limit capability. For example, if IP cores of o_3 and o'_3 contain the same HT and their HTs are activated by the same input pattern. Then outputs of normal computing and recomputing will be the same wrong results, which hence are treated as correct results. We can use $\sum_{i=1}^{3} A_i^{HT} \leq 1$ to describe the situation where security rules are still effective, where A_i^{HT} represents whether there are *activated* HTs in the ith computing. $A_i^{HT} = 1$ if there exist, otherwise 0.

B. Problem Formulation

Designing with untrusted IP cores should follow the two rules presented in Section III-A. While these design rules impose requirements on the diversity of IP cores and ensure runtime Trojan detection and recovery, they also increase the purchasing cost—the license fees paid to IP vendors. Hence, the motivated optimization problem is formulated as given below.

Given an application represented by a data flow graph (DFG), which is also called normal computing, recomputing needs to be implemented for HT detection. Normal computing and recomputing together are referred to as the detection phase, as shown on the left side of Fig. 2(b). A mismatch between the output of normal computing and that of recomputing indicates the detection of an HT and triggers the recovery phase, in which the DFG will be reimplemented, as shown on the right side of Fig. 2(b).

In addition to the DFG to be implemented, the SoC designer or integrator will provide a maximal latency constraint for the detection phase and recovery phase as well as a maximal silicon area constraint. In addition, he/she will be provided with a list of vendors and the price and area of each IP core they offer. It is assumed that using multiple copies of the same IP core does not incur additional fees. The goal of the SoC designer or integrator is to find, if it exists, the optimal design that follows the rules for HT detection and recovery, meets the latency and area constraints, and incurs the minimal purchasing cost.

IV. System Cost Optimization With Graph Theory

In this section, we first transform the security problem to a set of graph-theoretic models, i.e., graph-coloring and bipartite matching models, which are then solved by heuristic algorithms. The proposed graph-theoretic models and heuristic algorithms are not limited by the problem scale and can achieve (nearly) optimal results for the problem. The notation and definitions used in the graph-theoretic models are shown in Table III.

A. Graph-Theoretic Models—Graph Coloring Problem

An application can be modeled by a DFG. A simple application $y = a_1x_1 + a_2x_2$ can be represented by the 3-node DFG shown in Fig. 3(a). The corresponding adjacency matrix of the DFG is shown in Fig. 3(b).

In the graphs of normal computing, recomputing and recovery computing, we can add an edge between each pair of task nodes that cannot be bound to IP cores from the same vendor according to the security rules in Section III-A. The

TABLE III
NOTATION AND DEFINITIONS IN THE GRAPH-THEORETIC MODELS

Notation	Definition
G	A data flow graph that represents a function or application.
n	The total number of operations in the function (or normal computing) to be implemented, i.e., the number of vertices in G.
m	The total number of edges in G.
o_i	Operation i, $1 \leq i \leq n$.
$e(o_i, o_j)$	The edge/dependency between operations o_i and o_j in G. $e(o_i, o_j) = 1$ represents that the execution of o_j depends on the output of o_i. Otherwise, $e(o_i, o_j) = 0$.
ven	The set of vendors from which IP cores can be purchased.
$\|ven\|$	The number of vendors in the set *ven*.
τ	The set of different types of IP cores.
$\|\tau\|$	The number of different types of IP cores.
c(ven,τ)	A two-dimensional array, where *c(k,t)* is a known variable representing the cost of purchasing an IP core license for the t^{th} type of IP core from the k^{th} vendor. $1 \leq k \leq \|ven\|$, $1 \leq t \leq \|\tau\|$.
u(ven,τ)	A two-dimensional array, where *u(k,t)* is a unknown binary variable denoting whether the t^{th} type of IP core from the k^{th} vendor is used; i.e., if it is used, *u(k,t)=1*, and otherwise, *u(k,t)=0*.
λ	The upper bound of the scheduling length.
CN	A chromatic number, which denotes the number of colors output by a graph-coloring algorithm; these colors are represented by *clr*={CLR_1, CLR_2, ..., CLR_{CN}}.

Algorithm 1 Heuristic Graph-Coloring Algorithm

```
Require: A data flow graph $G$;
Ensure: The chromatic number and the set of vertices in each color;
1: Use matrix $M$ to represent $G$;
2: Extend matrix $M$ to $M'$ so that it includes 3 computings;
3: Add dependencies to matrix $M'$ according to the security rules;
4: Transform $M'$ into an undirected graph;
5: Color matrix $M'$ based on the saturation largest first (SLF) algorithm;
6: return The chromatic number $CN$ and the set of nodes in each color;
```

resulting graph is shown in Fig. 3(c). In this way, the system cost optimization regarding the number of different IP cores is transformed into a graph coloring problem (GCP): given a set of colors, find the smallest number of colors to use to color the vertices of a graph such that no two adjacent vertices have the same color. As the GCP is NP-hard, we can use the simple heuristic algorithm shown in Algorithm 1 to solve the problem.

Algorithm 1 takes a DFG as input and outputs a certain coloring solution, including both the *chromatic number* (the smallest number of colors needed to color the graph) and the set of vertices (task nodes) in each color CLR$_i$. Taking the DFG in Fig. 3(a) as an example, each step in the algorithm is illustrated in Fig. 4. In the first step, we use adjacency matrix M in Fig. 3(b) to represent the DFG. In the second step, we extend M to M', which contains the adjacency matrices of normal computing, recomputing and recovery computing that are shown in the rectangles in Fig. 4. In the third step, we add edges to M' according to the two security rules listed in Section III-A. In the fourth step, we transform M' into a symmetric matrix, which is also the adjacency matrix corresponding to the graph in Fig. 3(c). Then, we color the graph based on the saturation largest first (SLF) algorithm [50], which has a time complexity of $O((m+n)\log n)$ and yields optimal solutions for nearly all k-chromatic graphs.

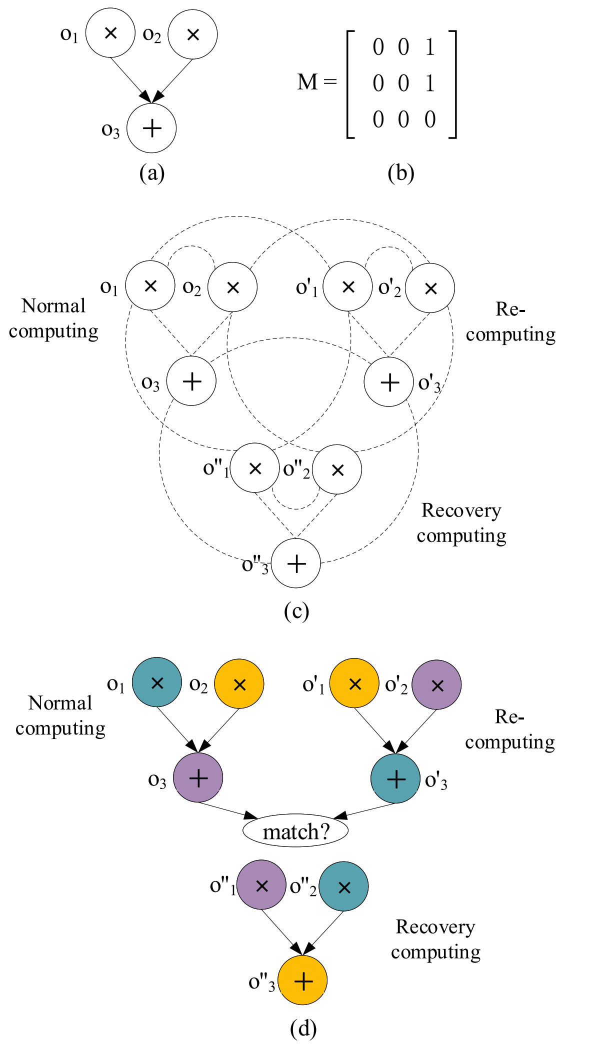

o1 × o2 × o3 +
(a)
$M = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix}$
(b)
Normal computing
Re-computing
Recovery computing
o'1 o'2 o'3 o''1 o''2 o''3
(c)
match?
(d)

Fig. 3. Transformed GCP. (a) 3-node DFG of $y = a_1x_1 + a_2x_2$. (b) DFG adjacency matrix. (c) Resulting graph after adding edges according to the security rules, where an edge between a pair of task nodes means that they cannot be bound to IP cores from the same vendor. (d) Final coloring solution.

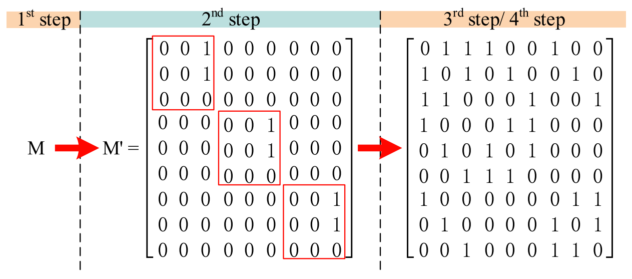

1st step
2nd step
3rd step/ 4th step
M
$M' = \begin{bmatrix} 0&0&1&0&0&0&0&0&0 \\ 0&0&1&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&1&0&0&0 \\ 0&0&0&0&0&1&0&0&0 \\ 0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&1 \\ 0&0&0&0&0&0&0&0&1 \\ 0&0&0&0&0&0&0&0&0 \end{bmatrix}$
$\begin{bmatrix} 0&1&1&1&0&0&1&0&0 \\ 1&0&1&0&1&0&0&1&0 \\ 1&1&0&0&0&1&0&0&1 \\ 1&0&0&0&1&1&0&0&0 \\ 0&1&0&1&0&1&0&0&0 \\ 0&0&1&1&1&0&0&0&0 \\ 1&0&0&0&0&0&0&1&1 \\ 0&1&0&0&0&0&1&0&1 \\ 0&0&1&0&0&0&1&1&0 \end{bmatrix}$

Fig. 4. Illustration of Algorithm 1.

TABLE IV
FOUR IP CORE VENDORS AND THEIR LICENSE FEES ($)

Vendor \ Type	multiply	add	SLL
VEN_1	1170	516	860
VEN_2	1108	567	882
VEN_3	1180	571	900
VEN_4	1149	578	889

Algorithm 2 Minimum-Weight Matching Algorithm

```
Require: A bipartite graph $G'$ and its weight matrix $D$;
Ensure: A minimum-weight matching $MT$;
1: Change all values in $D$ to the corresponding negatives;
2: Apply the Kuhn-Munkres maximum-weight matching algorithm;
3: return The minimum weight $W$ and a matching $M$;
```

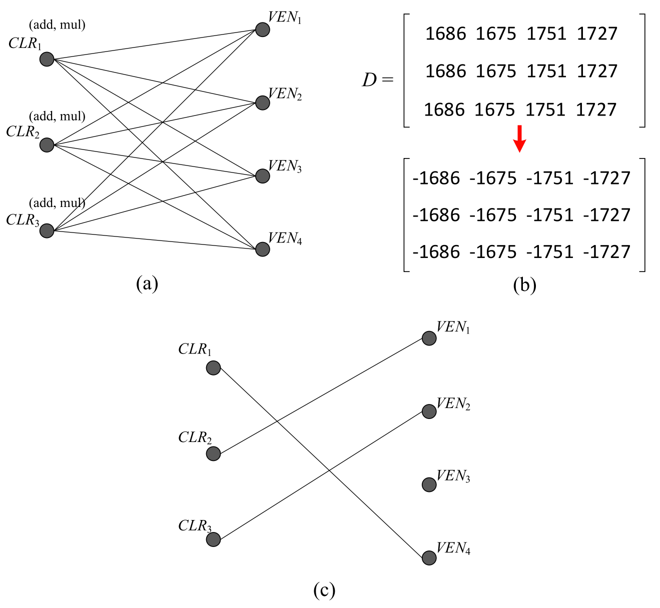
After obtaining the coloring solution [as shown in Fig. 3(d)], the next step is to optimize the system purchasing cost—the license fees paid to the IP core vendors.

B. Graph-Theoretic Models—Assignment Problem

The optimization of the system purchasing cost is based on the output of Algorithm 1. Consequentially the optimization of the system purchasing cost can be transformed into an assignment problem: find a minimum-weight matching in a bipartite graph. To be specific, the goal is to find a matching from the set $\text{clr} = \{\text{CLR}_1, \text{CLR}_2, \ldots, \text{CLR}_{\text{CN}}\}$ to the set $\text{ven} = \{\text{VEN}_1, \text{VEN}_2, \ldots, \text{VEN}_{|\text{ven}|}\}$ such that the system purchasing cost is minimal, where the weight of each edge $\text{CLR}_i \to \text{VEN}_j$ is $w_{ij} = \sum_{h=1}^{|\tau|} u(i,h) \times c(j,h)$, representing the purchasing cost when matching color CLR_i to vendor VEN_j. Definitions of both $u(i,h)$ and $c(j,h)$ can be found in Table III.

The minimum-weight matching algorithm is shown in Algorithm 2. Its inputs are a bipartite graph G' and a corresponding weight matrix D, and its output is a minimum-weight matching MT. We use an example to illustrate the steps of the algorithm. Fig. 2(b) shows that three colors and two types of operation (addition and multiplication) in each color are used. Assuming that alternative IP core vendors in Table IV are used, the complete bipartite graph G', from clr to ven is shown in Fig. 5(a). According to the calculation of the weight of each edge $\text{CLR}_i \to \text{VEN}_j$, $w_{ij} = \sum_{h=1}^{|\tau|} u(i,h) \times c(j,h)$, the graph's weight matrix D and its negative matrix are as shown in Fig. 5(b) (the first step in Algorithm 2). Then, we can apply the Kuhn–Munkres maximum-weight matching algorithm to the negative weight matrix to obtain the minimum-weight solution (the second step in Algorithm 2). The final minimum-weight matching solution, with the minimum weight being 5088, shown in Fig. 5(c). The time complexity of Algorithm 2 is $O(m \times n)$.

In this section, the optimization process for the security problem is split into two phases. The first phase determines the minimum number of IP core vendors, and the second phase obtains the minimum system purchasing cost. Such a two-phase optimization process can gain (nearly) optimal results



$$D = \begin{bmatrix} 1686 & 1675 & 1751 & 1727 \\ 1686 & 1675 & 1751 & 1727 \\ 1686 & 1675 & 1751 & 1727 \end{bmatrix}$$

$$\begin{bmatrix} -1686 & -1675 & -1751 & -1727 \\ -1686 & -1675 & -1751 & -1727 \\ -1686 & -1675 & -1751 & -1727 \end{bmatrix}$$

Fig. 5. Bipartite graph matching. (a) Complete bipartite graph from the set CLR to ven. (b) Graph's weight matrix and its negative. (c) Minimum-weight matching of the bipartite graph.

in terms of the system purchasing cost and has a scaling advantage due to its polynomial time complexity.

As mentioned in Sections I-A and II, simply assuming that all IP core vendors are untrustworthy overestimates the system purchasing cost. As a result, we adapt the threat model by assigning some operations to the IP cores that we trust. In this way, we have a trusted list and an untrusted list of IP core vendors. The system purchasing cost is reduced by decreasing different types of IP cores that we need to pay for the license fee. Taking the example in Fig. 3, assuming we have trusted IP cores for the addition operation in the DFG, the addition-operation nodes as well as their dependencies can be removed in Fig. 3(c). The resulting dependency graph and coloring solution are shown in Fig. 6(a) and (b), respectively.

From Figs. 3(d) and 6(b), we can see that both coloring solutions require three colors, CLR_A, CLR_B and CLR_C. However, the set of IP cores that we need to pay for is changed from $\{\text{CLR}_A^{\text{add}}, \text{CLR}_A^{\text{mul}}, \text{CLR}_B^{\text{add}}, \text{CLR}_B^{\text{mul}}, \text{CLR}_C^{\text{add}}$ and $\text{CLR}_C^{\text{mul}}\}$ in Fig. 3(d) to $\{\text{Trust}^{\text{add}}, \text{CLR}_A^{\text{mul}}, \text{CLR}_B^{\text{mul}}$ and $\text{CLR}_C^{\text{mul}}\}$ in Fig. 6(b). That is, even if our trusted IP cores have the same cost as other untrusted IP cores of the same types, the system purchasing cost can be greatly reduced, which is verified by the experimental results in Section VI.

It should be noted that there exists a certain limitation for the proposed technique. That is, once the number of colors/vendors output by Algorithm 1 exceeds the number of alternative vendors, it is difficult to establish the trustworthy computing as expected. Even with a set of trusted vendors, the number of required vendors for some type of IP core cannot be reduced like system purchasing cost, as shown in Fig. 6. In such a situation, it is necessary to recheck security rules and deploy them in order of importance to reduce consequential losses. For example, guarantee security rules in the detection phase first and add a switch for mission-critical applications.

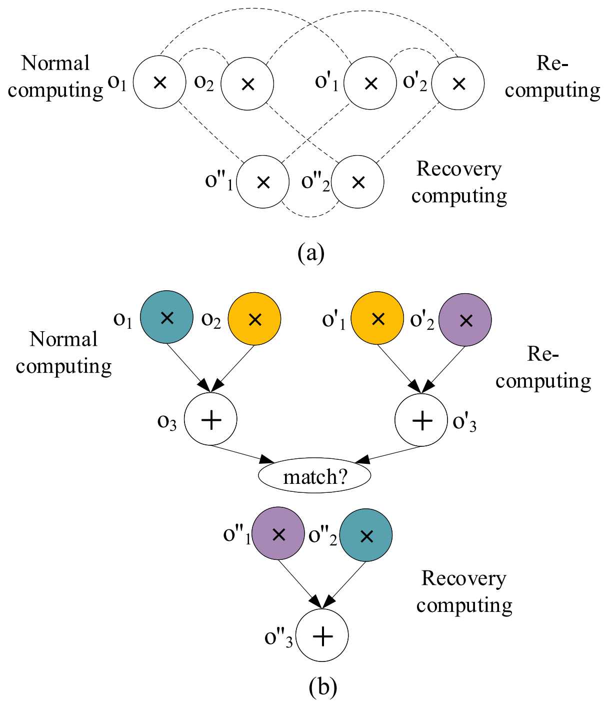


Fig. 6. Graph-coloring problem with trusted IP cores. (a) Dependency graph after assigning the addition nodes to trusted IP cores. (b) Coloring solution with trusted IP cores.

Based on the above optimized HT detection and recovery architecture, in the next section, we propose an architecture to locate and replace the infected IP cores for further performance improvement and security enhancement.

V. Infected IP Core Localization and Replacement

In the existing architecture, recovery computing is executed only if the results of normal computing and recomputing are not consistent, which could save computing resources compared to the method in [37]. However, recovery computing will always be executed when HTs are detected and before the infected IP core is removed. As a result, it is necessary to adapt the existing architecture to locate and replace the infected IP core for the sake of both saving computing resources and enhancing security.

A. Locating the Infected IP Core

1) Localization Architecture: Based on the above assumption made in the existing architecture, we can affirm that HTs exist in a system when the outputs of normal computing and recomputing are not consistent. For example, the output of operations o_3 and o'_3 will be different if the HT (if any) is activated in Fig. 3.

However, when such an inconsistency is detected, it is not clear which computing is correct, not to mention which IP cores are infected. Therefore, the first step is to identify the incorrect computing, which involves the HT. The simple way is to compare the final output of the recovery computing with that of any one of the computings in the detection phase; e.g., if the outputs of operations o_3 and o''_3 are the same, then the recomputing is wrong; otherwise, the normal computing is wrong. However, we still do not know where the infected IP core is, which motivates us to look into the intermediate results of the computings.

To find the infected IP cores efficiently, the steps of identifying the incorrect computing and locating the infected IP core are performed simultaneously. That is, we compare the intermediate results of recovery computing with those of normal computing and recomputing at the same time.

For the sake of saving time and computing resources, the intermediate results of the computings in the detection phase are stored in memory while the computings are running. These intermediate results are recovered for comparison when the recovery computing is running. Once an inconsistency is detected, the correct computing is identified, and hence, the recovery computing can be terminated. In this way, the computations in the detection phase will avoid running twice, and it will be unnecessary to wait for the final output of recovery computing.

According to the security rules and the basic assumption, an intermediate result of recovery computing is guaranteed to be correct once an inconsistency is detected in the corresponding detection phase. Assume that the normal computing is incorrect; then, we have the following theorem for the infected IP core.

Theorem 1: The infected IP core is the IP core that generates the first wrong intermediate result in the step-by-step comparison process.

Now, we see the changes we have to make to the existing architecture. First, we need memory to store the intermediate results that are generated in the detection phase. Second, we need more comparators to compare the intermediate results. Taking Fig. 2(b) as an example, for simplicity, we assign each operation an IP core. Therefore, there will be 15 IP cores for all the computings. The infected IP core localization architecture is shown in Fig. 7, where the left, middle and right groups of five IP cores are from normal computing, recovery computing and recomputing, respectively. When the system is running, the two main phases based on Fig. 7 are explained as follows.

1) In the detection phase, the intermediate result of each IP core in normal computing and recomputing is stored in memory. When normal computing and recomputing are finished, their final outputs are moved from memory for comparison; e.g., the comparator COMP1 in the figure takes the comparison task. If COMP1 outputs 0, it means that outputs of normal computing and recomputing are consistent and no HT is activated. Then, the output of normal computing is chosen by the MUX as the correct output. If COMP1 outputs 1, it means that the HT is activated in one of the computings; then, the system switches to the recovery phase.
2) In the recovery phase, the intermediate results of the detection phase are obtained from memory at each cycle to be compared with the corresponding intermediate outputs of recovery computing. Once an inconsistency is detected, the infected IP core can be inferred from the address where its intermediate result is stored. At the same time, recovery computing can be terminated since

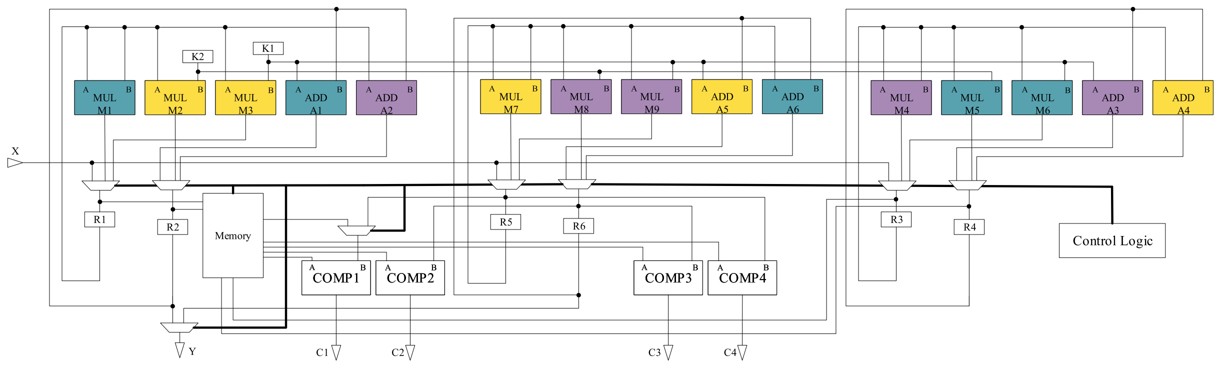


Fig. 7. Hardware architecture of *POLYNOM* for runtime HT detection, recovery and localization, where the left, right, and middle five IP cores represent normal computing, recomputing, and recovery computing, respectively.

the correct computing has been identified, and its final output will be accepted as the correct output.

2) Localization Overhead: According to the above analysis, the overhead required to locate the infected IP core mainly results from the comparators and the multiplexers used with comparators. Fig. 7 shows that two comparators are needed at the same time to compare the output of each IP core in recovery computing with the intermediate results of normal computing and of recomputing. Assuming that there are N_c IP cores in one clock cycle, $2N_c$ comparators are needed for comparison. As comparators in different clock cycles can be multiplexed, we only need to care about the clock cycle with the maximum number of IP cores.

On the other hand, one of the inputs of a comparator receives the intermediate result of normal computing or recomputing that comes from memory. The other input receives the outputs of the IP cores from recovery computing. If this comparator is multiplexed at each cycle of the λ cycles, then the input to it that receives the results from the IP cores will need to be bound to a λ-input multiplexer.

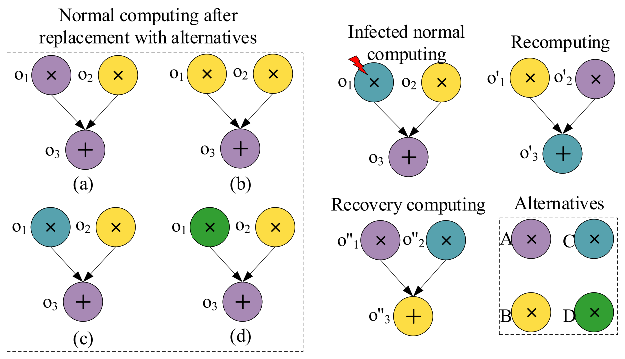
In summary, we can determine the number of comparators based on the maximum number of IP cores that will be placed in a single clock cycle. We can also determine the multiplexer scale of an input of a comparator according to the number of times it is multiplexed. By adding these factors, we will obtain the area overhead for the infected IP core localization.

B. Replacing the Infected IP Core

1) Replacement Process: Once the infected IP core is located, the next step is to replace it with alternatives. The replacement is accomplished by reprogramming the control logic, which can be done after the current application is finished and before the next runs. By this means, the outputs of the previous execution are guaranteed to be correct, while the security of the next execution is enhanced.

The alternatives for replacement should be carefully chosen and deployed for the following reasons.

First, an alternative using IP cores from the existing vendors may compromise the security rules. Assume there are 4 alternatives, and the IP core carrying o_1 is identified as the infected IP core, as shown in Fig. 8. We can see that there are four possible situations.



Normal computing after replacement with alternatives
(a)
(b)
(c)
(d)
Infected normal computing
Recomputing
Recovery computing
Alternatives
A
B
C
D

Fig. 8. Replace the infected IP core of o_1 in normal computing. (a) Replace the infected IP core with alternative *A*. (b) Replace the infected IP core with alternative *B*. (c) Replace the infected IP core with alternative *C*. (d) Replace the infected IP core with alternative *D*.

1) When replacing the infected IP core with alternative A in Fig. 8, we find that operation o_1, o_1'' and o_3 are bound to IP cores from the same vendor. This compromises both security rules described in Section III-A. The activation of HTs (if exist) in IP core of o_1 will make outputs of normal computing and recomputing different. When goes to the recovery phase, HTs in IP core of o_1'' will be activated under the same condition such that the recovery computing also outputs wrong results.
2) When replacing the infected IP core with alternative B in Fig. 8, we find that operation o_1, o_1', and o_2 are bound to IP cores from the same vendor. This also compromises both security rules. The activation of HTs (if exist) in IP core of o_1 and o_1' will generate the same wrong outputs of normal computing and recomputing. And the system will consider this as the correct outputs, which makes HTs avoid being detected.
3) Replacing the infected IP core with alternative C is the same as the original case, so nothing is done.
4) Replacing the infected IP core with alternative D obeys all rules and is the correct option.

As a result, we can conclude that the alternative IP core must come from a different vendor from all existing ones.

Second, as we do not know which IP core is the infected IP core in advance, the alternatives from the new vendor should include all types of IP cores in case replacement is

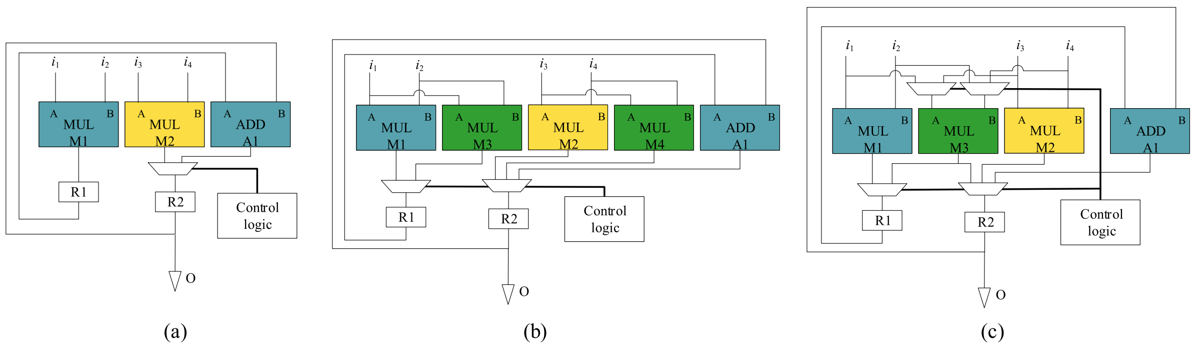


Fig. 9. Reducing the area overhead at the cost of logical complexity. (a) Computing architecture. (b) Using two alternative multipliers with a lower logical complexity. (c) Using one alternative multiplier with a higher logical complexity.

needed. Taking Fig. 8 as an example, there should also be an adder from the green vendor in case one of the two adders in the detection phase is identified as an infected IP core.

2) Replacement Overhead: Since we do not know which IP core is infected before an inconsistency occurs, we need to prepare an alternative for each IP core in the system. Taking the normal computing in Fig. 8 as an example, its hardware architecture is shown in Fig. 9(a), and it contains two multipliers and one adder. Assuming that one of the multipliers is infected and will output the wrong result when the system is running, we do not know exactly which one it is. In this situation, we can either 1) use two redundant multipliers as alternatives to two multipliers, as shown in Fig. 9(b) or 2) use one redundant multiplier as an alternative to two multipliers, as shown in Fig. 9(c). It is obvious that these two structures incur different area overheads—the former uses more multipliers but fewer multiplexers, while the latter uses fewer multipliers but more multiplexers.

To optimize the replacement architecture, different situations are analyzed.

Without loss of generality, we assume there are N_k original IP cores of the kth type in the system and that N_r redundant IP cores of the kth type act as alternatives. Meanwhile, we assume that each IP core of the kth type has two inputs and one output and that its area is a_k. According to [51], an h-input multiplexer can be implemented by $h-1$ 2-input multiplexers, and the area of a 2-input multiplexer is assumed to be a_m. The total area overhead of the replacement architecture is W.

At the input side of an alternative IP core, each input pin requires a multiplexer to receive the inputs of the IP cores that it will replace. Considering the balance of the structure, each alternative IP core needs to replace $\lceil (N_k/N_r) \rceil$ original IP cores. In addition, it is necessary to use a 2-input multiplexer to receive both the output of the original IP core and that of its alternative IP core, as shown in Fig. 9(b) and (c).

Thus, the area overhead can be divided into the following parts.

1) Each input of $N_r - 1$ of the alternative IP cores needs an $\lceil (N_k/N_r) \rceil$-input multiplexer, the area overhead of which is
$$W_1 = 2 \times (N_r - 1) \times (\lceil (N_k/N_r) \rceil - 1) \times a_m.$$
2) The remaining alternative IP core needs to replace $N_k - (N_r - 1) \times \lceil (N_k/N_r) \rceil$ original IP cores, the area overhead of which is
$$W_2 = 2 \times \left[N_k - (N_r - 1) \times \lceil (N_k/N_r) \rceil - 1 \right] \times a_m.$$
3) The area overhead of all the alternative IP cores is
$$W_3 = a_k \times N_r.$$
4) The area overhead of the 2-input multiplexers needed at the output side of the original IP cores is $W_4 = a_m \times N_k$.

Adding the above items, we obtain the overall overhead
$$\begin{aligned} W &= W_1 + W_2 + W_3 + W_4 \\ &= 3a_mN_k - 2a_mN_r + a_kN_r. \end{aligned} \tag{1}$$
This equation can be further transformed into
$$\frac{W}{a_m} = 3N_k + \left(\frac{a_k}{a_m} - 2\right)N_r.$$
From the monotonicity of the equation, we know that the overhead is minimal when $(a_k/a_m) - 2 < 0$, i.e., $a_k < 2a_m$. Otherwise, the overhead is minimal when $N_r = 1$.

VI. Experiments

A. Experimental Setup

The experiments are conducted on some known benchmarks. Most of them are chosen from 1992 high-level synthesis benchmarks and are converted from the C language to control DFGs (CDFGs) using the GAUT tool [52]. The operations used in these benchmarks include addition (add), subtraction (sub), comparison (comp), multiplication (mul), division (div), shift-right arithmetic (sra), the shift-left logical operator (sll), and absolute value (abs). Each operation is assigned an IP core in the experiments. The area costs of the IP cores that carry out these operations are shown in Table V. These data are also collected from GAUT. Additionally, the setup of the available IP core vendors and their license fees for different types of IP cores are shown in Table VI. The experiments are performed on an HP PC with an Intel Core i7-9750H CPU @2.60 GHz and 32.00-GB memory. All the experimental implementations in this section and a visual platform of graph-theory for HT detection in SoCs can

TABLE V
AREA SETUP OF THE IP CORES USED IN THE BENCHMARKS

Unit type	$add/sub/comp$	$mul/div/sra$	sll	abs	$comp\_mux$	$2\text{-}MUX$
area/unit	8	83	44	18	17	3

TABLE VI
SETUP OF THE IP CORE VENDORS AND THEIR LICENSE FEES ($)

Types / Vendor	$add/sub/comp$	$mul/div/sra$	sll	$comp\_mux/abs$
VEN_1	539	1128	877	393
VEN_2	548	1192	863	354
VEN_3	585	1111	886	311
VEN_4	565	1127	894	385
VEN_5	535	1124	849	378
VEN_6	581	1149	882	315

TABLE VII
SYSTEM COST OPTIMIZATION BY THE ILP AND GRAPH-THEORETIC MODELS

Benchmarks	N_u	N_t	Graph-theoretic		ILP	
			SC	*vendors*	*SC*	*vendors*
eq_2degre	3	2	1622	1,2,5	1622	1,2,5
polynom	5	2	7634	1,3,5	5951	1,2,3,4,5
ellipticicass	29	1	2187	1,2,4,5	2187	1,2,3,4,5
fir16	31	2	5018	1,4,5	4984	1,2,3,4,5
volterra	46	2	5018	1,4,5	4984*	1,2,3,4,5
lms	66	3	5018	1,3,4,5	-	-
sobel	349	5	5305	2,3,5,6	-	-
*conv3*3*	456	3	5018	1,3,4,5	-	-
dct32	467	5	7634	1,3,4,5	-	-

be found in *https://github.com/Light-City/Graph-theoretic* and *http://light-city.club:7879/gcp#/*, respectively.

B. Experimental Results

In the first experiment, we compare the results of our graph-theoretic models with those of the ILP model [39] regarding the system (purchasing) cost. The ILP model is solved in Python+Gurobi with an academic license [53]. Table VI is the experimental setup of IP cores, where there are 6 alternative vendors, each vendor has a set of different type of IP cores and each type of IP core is associated with its license fee. Based on Table VI, the optimization results are as shown in Table VII. The columns "N_u" and "N_t" represent the number of nodes/IP cores and the number of their types used for each benchmark circuit, respectively. The columns "*SC*" and "*vendors*" in "graph-theoretic" and "ILP" represent the system purchasing cost and chosen vendors calculated by the graph-theoretic and ILP models, respectively.

As the ILP model does not scale well with the problem size, the results of only five benchmarks are obtained, and most of them required hours of time. In the "ILP" column of Table VII, some of the results are not optimal but are the best results we could obtain in 24 h; these results are marked with "*." Table VII shows that the graph-theoretic models have a much better scalability—all the results can be collected within seconds. In addition, it is observed that the two-phase optimization process obtained by the graph-theoretic models

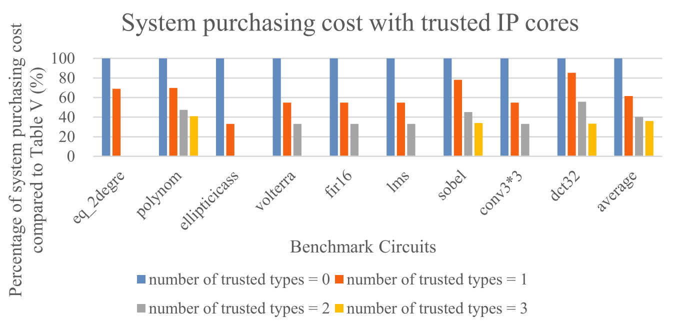


Fig. 10. System purchasing cost is determined by assigning a value to the number of trusted types of IP cores.

incurs small additional system purchasing costs, while using fewer vendors than the ILP. The average percentage of extra system purchasing costs for the first five benchmarks is only 5.9%, which is negligible.

We note that in Table VII that the number of vendors output by Algorithm 1 is smaller than that of ILP while the ILP has relatively smaller system purchasing cost. This is because the ILP model optimizes the system cost in a global, exhaustive way. To avoid scaling problems, the proposed technique splits the problem into two subproblems. In the graph coloring subproblem, we try to minimize the number of vendors required. In the assignment subproblem, we optimize system cost based on the number of vendors required. We have verified that even we use more colors to color the graph in the first subproblem, the system purchasing cost output by Algorithm 2 becomes more worse than the proposed technique, not to speak of the ILP. This is because we are adding colors with no specific optimization purpose and it is difficult to reconnect the two subprocesses for a global optimization.

In the second experiment, we apply the practical threat model by establishing a trusted list of IP cores. For simplicity, IP cores with the same price are treated as a single type of IP core, referring to Table VI. By setting the number of trusted types of IP cores to 0 (Table VII), 1, 2, ..., the corresponding results for the system purchasing cost obtained by our graph-theoretic models are as shown in Fig 10. With a few trusted IP cores, the system purchasing cost can be reduced greatly. On average, the system purchasing cost is reduced to 60% by assuming only one type of IP core to be trusted.

Although the operations in a DFG are allocated to IP cores from different vendors, the same type of IP core from the same vendor can be multiplexed at different cycles to reduce the area overhead of the system. To explore the tradeoff between the schedule length and area overhead, we conduct a third experiment. Based on the results of the graph-theoretic models, we first allocate each operation an IP core and schedule them as early as possible (referred to as the *baseline*, so the area overhead of *baseline* architecture will be the accumulation area of all IP cores used), then we relax the timing constraint to multiplex the IP cores until no additional IP cores can be multiplexed (referred to as the *final stage*). The schedule length and area overhead of the *baseline* and *final stage* are shown in Fig. 11. There exist very large adjustable

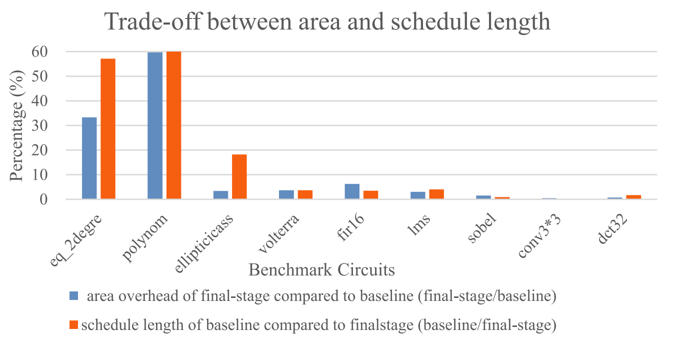


Fig. 11. Tradeoff between the area and schedule length.

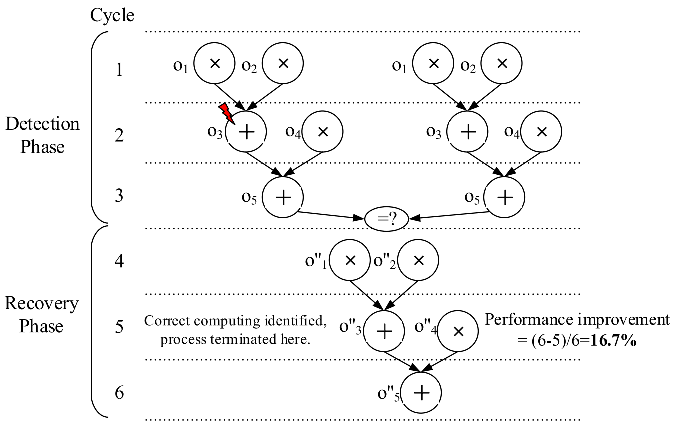

Cycle
Detection Phase
Recovery Phase
Correct computing identified, process terminated here.
Performance improvement = (6-5)/6=16.7%
=?

Fig. 12. Infected IP core localization process can improve the system performance.

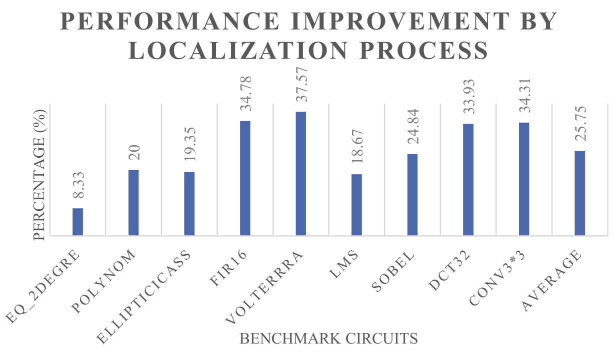


Fig. 13. Performance improvement obtained by the infected IP core localization process compared to the existing architecture when HTs are detected.

TABLE VIII
AREA OVERHEAD OF INFECTED IP CORE LOCALIZATION AND REPLACEMENT

Benchmarks	area of the *baseline*	Loc (%)	Loc&Rep (%)
polynom	226	17.6	89.4
fir16	1448	108.3	133
volterra	2453	57.2	77.2
lms	2998	65.1	87.5
*conv3*3*	21528	62.4	81.8
dct32	19360	28.1	50.4

ranges for the area and schedule length, especially for large-scale benchmark circuits. This means that the SoC designer or integrator can explore these tradeoffs for system resource optimization.

As mentioned in Section V-A, the infected IP core localization process can improve the system performance when HTs are detected. This is because we have stored intermediate results of normal computing and recomputing. When a HT is detected, i.e., the final outputs of normal computing and recomputing are different, the system will switch to the recovery computing. Outputs of each step in the recovery computing will be compared to previously stored results to check which IP core is infected. Once the infected IP core of one computing is identified, we can conclude that the other computing will be the correct computing, where its final output is already available. At the same time, the recovery computing is terminated, which saves computing resources in terms of time consumed. The original architecture, on the other hand, needs to wait for the recovery computing to end before it can identify the correct computing. This situation is illustrated in Fig. 12. In the figure, assume the IP core that carries out o_3 is infected and leads to an inconsistency between the final outputs of the normal computing and recomputing; then, the system switches to the recovery phase. At the fifth clock cycle, the localization process identifies the correct and incorrect computing and terminates the recovery computing, which saves 1 cycle compared to the original architecture. Hence, the performance improvement in this example is 16.7%. As a result, in the fourth experiment, we evaluate the average performance improvement by randomly assigning IP cores to be infected in the detection phase.

The experimental results are shown in Fig. 13. The average improved performance is 25.75%, which indicates that the proposed infected IP core localization process improves the system performance significantly when HTs are detected compared to the existing architecture. In addition, the percentage of performance improvement is related to the topology of the benchmark circuits. For example, the fir16 and volterra benchmark circuits have the highest performance improvements. This is because their topology follows a cone structure with more inputs than outputs, which schedules more IP cores at earlier clock cycles. If HTs are detected in these IP cores, the proposed scheme identifies the correct computing earlier and hence saves time. As a result, scheduling IP cores as early as possible is more desirable when designing energy- and timing-aware on-chip systems.

In the fifth experiment, we evaluate the area overhead of the localization and replacement process. The experimental results are shown in Table VIII. The columns “Loc” and “Loc&Rep” in the table represent the area overhead of the localization process and that of both the localization and replacement processes, respectively. Compared to *baseline* architecture, the localization and replacement process incurs a comparatively large area overhead. This is because for each benchmark circuit, we schedule the IP cores as early as possible, which results in more IP cores in each cycle and hence requires more comparators and multiplexers according to the analysis in Section V.

At the cost of performance, one way to reduce the area overhead of the localization and replacement processes is

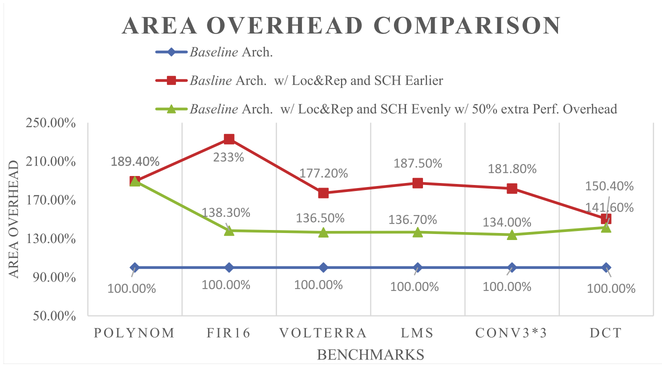


Fig. 14. Extra area overhead of localization and replacement processes when scheduling IP cores earlier compared with scheduling them evenly.

to schedule all the IP cores in each clock cycle as evenly as possible while satisfying all security and scheduling constraints. With a tolerance of 50% extra performance overhead, the reduced area overhead of the localization and replacement process is as shown in Fig. 14. In the figure, with 50% extra performance overhead to schedule the IP cores as evenly as possible, the area overhead needed to locate and replace the infected IP core decreased significantly compared to scheduling all IP cores as early as possible.

VII. Conclusion

In this article, we analyze the process of building trustworthy systems from a graph-theoretic perspective, i.e., using untrusted third-party IP cores to detect HTs and recover from their errors at runtime. The HT detection and system cost optimization problems in SoCs can be transformed into graph coloring and assignment problems, which can be solved efficiently using heuristic algorithms. Moreover, to further enhance the security of the system, we propose a method of locating and replacing the infected IP core when HTs are detected. Detailed evaluations are conducted in terms of the system purchasing cost, system performance and area overhead.

Appendix
Proof of Theorem 1

Proof (Case 1): An IP core has no parent nodes in normal computing. In this case, the IP core's inputs are from among the primary inputs and constants, which are also the same as the inputs of the IP core in recovery computing. If this IP core in normal computing outputs an incorrect result (i.e., one that is different from the corresponding result in recovery computing), it means that the activated HT in this IP core creates malicious logic functions. Thus, this IP core is infected.

Case 2: An IP core has parent nodes in normal computing. In this case, the IP core's inputs are from its parent nodes' outputs, which were verified to be correct in previous steps. If this IP core outputs an incorrect result, it means that this IP core contains activated HTs and hence is the infected IP core. Therefore, the theorem is proved. ■

References

[1] R. Karri, J. Rajendran, K. Rosenfeld, and M. Tehranipoor, "Trustworthy hardware: Identifying and classifying hardware trojans," *IEEE Comput.*, vol. 43, no. 10, pp. 39–46, Oct. 2010.
[2] M. Tehranipoor and F. Koushanfar, "A survey of hardware trojan taxonomy and detection," *IEEE Design Test Comput.*, vol. 27, no. 1, pp. 10–25, Jan./Feb. 2010.
[3] H. Li, Q. Liu, and J. Zhang, "A survey of hardware trojan threat and defense," *Integration*, vol. 55, pp. 426–437, Sep. 2016.
[4] 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 Comput. Digit. Techn.*, vol. 14, no. 6, pp. 231–246, 2020.
[5] J. Zhang and G. Qu, "Recent attacks and defenses on FPGA-based systems," *ACM Trans. Reconfigurable Technol. Syst.*, vol. 12, no. 3, pp. 1–24, Aug. 2019. [Online]. Available: https://doi.org/10.1145/3340557
[6] B. Cha and S. K. Gupta, "A resizing method to minimize effects of hardware trojans," in *Proc. IEEE 23rd Asian Test Symp.*, 2014, pp. 192–199.
[7] N. G. Tsoutsos and M. Maniatakos, "Fabrication attacks: Zero-overhead malicious modifications enabling modern microprocessor privilege escalation," *IEEE Trans. Emerg. Topics Comput.*, vol. 2, no. 1, pp. 81–93, Mar. 2014.
[8] H. Salmani, "COTD: Reference-free hardware trojan detection and recovery based on controllability and observability in gate-level netlist," *IEEE Trans. Inf. Forensics Security*, vol. 12, pp. 338–350, 2017.
[9] M. Zou, X. Cui, L. Shi, and K. Wu, "Potential trigger detection for hardware trojans," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 37, no. 7, pp. 1384–1395, Jul. 2018.
[10] A. Shabani and B. Alizadeh, "PMTP: A MAX-SAT-based approach to detect hardware trojan using propagation of maximum transition probability," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 39, no. 1, pp. 25–33, Jan. 2020.
[11] T. Dhar, S. K. Roy, and C. Giri, "Hardware trojan detection by stimulating transitions in rare nets," in *Proc. 32nd Int. Conf. VLSI Design 18th Int. Conf. Embedded Syst. (VLSID)*, Delhi, India, 2019, pp. 537–538.
[12] C. Liu, P. Cronin, and C. Yang, "Securing cyber-physical systems from hardware trojan collusion," *IEEE Trans. Emerg. Topics Comput.*, vol. 8, no. 3, pp. 655–667, Jul.–Sep. 2020.
[13] D. Agrawal, S. Baktir, D. Karakoyunlu, P. Rohatgi, and B. Sunar, "Trojan detection using IC fingerprinting," in *Proc. IEEE Symp. Security Privacy (SP)*, Berkeley, CA, USA, May 2007, pp. 296–310.
[14] Y. Lyu and P. Mishra, "Efficient test generation for trojan detection using side channel analysis," in *Proc. Design Autom. Test Eur. Conf. Exhibit. (DATE)*, Florence, Italy, 2019, pp. 408–413.
[15] K. Xiao, D. Forte, Y. Jin, R. Karri, S. Bhunia, and M. Tehranipoor, "Hardware trojans: Lessons learned after one decade of research," *ACM Trans. Design Autom. Electron. Syst.*, vol. 22, no. 1, p. 6, 2016.
[16] Y. Jin and Y. Makris, "Hardware trojan detection using path delay fingerprint," in *Proc. IEEE Int. Workshop Hardw. Orient. Security Trust*, Anaheim, CA, USA, 2008, pp. 51–57.
[17] X. Cui, E. Koopahi, K. Wu, and R. Karri, "Hardware trojan detection using the order of path delay," *ACM J. Emerg. Technol. Comput. Syst.*, vol. 14, no. 3, p. 33, 2018.
[18] A. Amelian and S. E. Borujeni, "A side-channel analysis for hardware trojan detection based on path delay measurement," *J. Circuits Syst. Comput.*, vol. 27, no. 9, 2018, Art. no. 1850138.
[19] S. Narasimhan *et al.*, "Hardware trojan detection by multiple-parameter side-channel analysis," *IEEE Trans. Comput.*, vol. 62, no. 11, pp. 2183–2195, Nov. 2013.
[20] S. K. Rao, D. Krishnankutty, R. Robucci, N. Banerjee, and C. Patel, "Post-layout estimation of side-channel power supply signatures," in *Proc. IEEE Int. Symp. Hardw. Orient. Security Trust (HOST)*, Washington, DC, USA, 2015, pp. 92–95.
[21] F. K. Lodhi, S. R. Hasan, O. Hasan, and F. Awwadl, "Power profiling of microcontroller's instruction set for runtime hardware trojans detection without golden circuit models," in *Proc. Design Autom. Test Eur. Conf. Exhibit. (DATE)*, Lausanne, Switzerland, 2017, pp. 294–297.
[22] M. Yan, H. Wei, and M. Onabajo, "Modeling of thermal coupling and temperature sensor circuit design considerations for hardware trojan detection," in *Proc. IEEE 61st Int. Midwest Symp. Circuits Syst. (MWSCAS)*, Windsor, ON, Canada, 2018, pp. 857–860.
[23] N. Karimi, J.-L. Danger, and S. Guilley, "On the effect of aging in detecting hardware trojan horses with template analysis," in *Proc. IEEE 24th Int. Symp. On-Line Test. Robust Syst. Design (IOLTS)*, Platja d'Aro, Spain, Jul. 2018, pp. 281–286.

[24] M. Xue, R. Bian, W. Liu, and J. Wang, "Defeating untrustworthy testing parties: A novel hybrid clustering ensemble based golden models-free hardware trojan detection method," *IEEE Access*, vol. 7, pp. 5124–5140, 2018.

[25] M. Xue, R. Bian, J. Wang, and W. Liu, "Building an accurate hardware trojan detection technique from inaccurate simulation models and unlabelled ICs," *IET Comput. Digit. Techn.*, vol. 13, no. 4, pp. 348–359, 2019.

[26] O. Söll, T. Korak, M. Muehlberghuber, and M. Hutter, "EM-based detection of hardware trojans on FPGAS," in *Proc. IEEE Int. Symp. Hardw. Orient. Security Trust (HOST)*, Arlington, VA, USA, May 2014, pp. 84–87.

[27] J. Balasch, B. Gierlichs, and I. Verbauwhede, "Electromagnetic circuit fingerprints for hardware trojan detection," in *Proc. IEEE Int. Symp. Electromagn. Compat. (EMC)*, Dresden, Germany, 2015, pp. 246–251.

[28] J. He, Y. Zhao, X. Guo, and Y. Jin, "Hardware trojan detection through chip-free electromagnetic side-channel statistical analysis," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 25, no. 10, pp. 2939–2948, Oct. 2017.

[29] K. Xiao, D. Forte, and M. Tehranipoor, "A novel built-in self-authentication technique to prevent inserting hardware trojans," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 33, no. 12, pp. 1778–1791, Dec. 2014.

[30] Q. Shi, M. M. Tehranipoor, and D. Forte, "Obfuscated built-in self-authentication with secure and efficient wire-lifting," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 38, no. 11, pp. 1981–1994, Nov. 2019.

[31] J. Zhang, C. Shen, H. Su, M. T. Arafin, and G. Qu, "Voltage over-scaling-based lightweight authentication for IoT security," *IEEE Trans. Comput.*, early access, Jan. 6, 2021, doi: 10.1109/TC.2021.3049543.

[32] H. Li, A. Abdelhadi, R. Shi, J. Zhang, and Q. Liu, "Adversarial hardware with functional and topological camouflage," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 68, no. 5, pp. 1685–1689, May 2021.

[33] J. J. Rajendran, O. Sinanoglu, and R. Karri, "Building trustworthy systems using untrusted components: A high-level synthesis approach," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 24, no. 9, pp. 2946–2959, Sep. 2016.

[34] A. Waksman, M. Suozzo, and S. Sethumadhavan, "FANCI: Identification of stealthy malicious logic using Boolean functional analysis," in *Proc. ACM SIGSAC Conf. Comput. Commun. Security*, 2013, pp. 697–708.

[35] J. Zhang, F. Yuan, and Q. Xu, "DeTrust: Defeating hardware trust verification with stealthy implicitly-triggered hardware trojans," in *Proc. ACM SIGSAC Conf. Comput. Commun. Security*, 2014, pp. 153–166.

[36] J. Rajendran, A. M. Dhandayuthapany, V. Vedula, and R. Karri, "Formal security verification of third party intellectual property cores for information leakage," in *Proc. 29th Int. Conf. VLSI Design 15th Int. Conf. Embedded Syst.*, Kolkata, India, 2016, pp. 547–552.

[37] H. A. M. Amin, Y. Alkabani, and G. M. I. Selim, "System-level protection and hardware trojan detection using weighted voting," *J. Adv. Res.*, vol. 5, no. 4, pp. 499–505, 2014.

[38] J. Rajendran, H. Zhang, O. Sinanoglu, and R. Karri, "High-level synthesis for security and trust," in *Proc. IEEE 19th Int. On-Line Testing Symp. (IOLTS)*, Chania, Greece, Jul. 2013, pp. 232–233.

[39] X. Cui, K. Ma, L. Shi, and K. Wu, "High-level synthesis for run-time hardware trojan detection and recovery," in *Proc. 51st Annu. Design Autom. Conf.*, San Francisco, CA, USA, 2014, pp. 1–6.

[40] M. Banga and M. S. Hsiao, "Trusted RTL: Trojan detection methodology in pre-silicon designs," in *Proc. IEEE Int. Symp. Hardw. Orient. Security Trust (HOST)*, Anaheim, CA, USA, Jun. 2010, pp. 56–59.

[41] X. Zhang and M. Tehranipoor, "Case study: Detecting hardware trojans in third-party digital IP cores," in *Proc. IEEE Int. Symp. Hardw. Orient. Security Trust*, San Diego, CA, USA, Jun. 2011, pp. 67–70.

[42] J. Zhang, F. Yuan, L. Wei, Z. Sun, and Q. Xu, "VeriTrust: Verification for hardware trust," in *Proc. 50th ACM/EDAC/IEEE Design Autom. Conf. (DAC)*, Austin, TX, USA, 2013, pp. 1–8.

[43] S. Gundabolu and X. Wang, "On-chip data security against untrustworthy software and hardware IPs in embedded systems," in *Proc. IEEE Comput. Soc. Annu. Symp. VLSI (ISVLSI)*, Hong Kong, 2018, pp. 644–649.

[44] A. Sayed-Ahmed, J. Haj-Yahya, and A. Chattopadhyay, "SoCINT: Resilient system-on-chip via dynamic intrusion detection," in *Proc. 32nd Int. Conf. VLSI Design 18th Int. Conf. Embedded Syst. (VLSID)*, Delhi, India, 2019, pp. 359–364.

[45] A. Sengupta, S. Patnaik, J. Knechtel, M. Ashraf, S. Garg, and O. Sinanoglu, "Rethinking split manufacturing: An information-theoretic approach with secure layout techniques," in *Proc. Int. Conf. Comput.-Aided Design*, Irvine, CA, USA, 2017, pp. 329–336.

[46] X. Cui, J. J. Zhang, K. Wu, S. Garg, and R. Karri, "Split manufacturing-based register transfer-level obfuscation," *ACM J. Emerg. Technol. Comput. Syst.*, vol. 15, no. 1, p. 11, 2019.

[47] X. Cui *et al.*, "On the difficulty of inserting trojans in reversible computing architectures," *IEEE Trans. Emerg. Topics Comput.*, vol. 8, no. 4, pp. 960–972, Oct.–Dec. 2020.

[48] T. Karnik and P. Hazucha, "Characterization of soft errors caused by single event upsets in CMOS processes," *IEEE Trans. Dependable Secure Comput.*, vol. 1, no. 2, pp. 128–143, Apr./Jun. 2004.

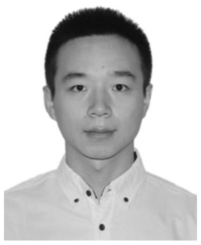
[49] R. Gaillard, "Single event effects: Mechanisms and classification," in *Soft Errors in Modern Electronic Systems*. Boston, MA, USA: Springer, 2011, pp. 27–54.

[50] A. Kosowski and K. Manuszewski, "Classical coloring of graphs," in *Contemporary Mathematics*, vol. 352. Providence, RI, USA: Amer. Math. Soc., 2004, pp. 1–20.

[51] Y. Thakur, R. Mehra, and A. Sharma, "CMOS design of area and power efficient multiplexer using tree topology," *Int. J. Comput. Appl.*, vol. 112, no. 11, pp. 32–36, 2015.

[52] E. Martin, O. Sentieys, H. Dubois, and J.-L. Philippe, "GAUT: An architectural synthesis tool for dedicated signal processors," in *Proc. Design Autom. Conf. EURO-DAC EURO-VHDL Eur. Design Autom.*, Hamburg, Germany, 1993, pp. 14–19.

[53] *Gurobi 9.1*, Gurobi, Beaverton, OR, USA. [Online]. Available: https://www.gurobi.com/



Xiaotong Cui received the B.S. and the Ph.D. degrees from Chongqing University, Chongqing, China, in 2013 and 2018, respectively.

He went to New York University for one-year Visiting Scholar in 2016. He joined the Chongqing University of Posts and Telecommunications, Chongqing, as an Assistant Professor in 2018. He has also serves as a Visiting Scholar with the State Key Laboratory of Vehicle NVH and Safety Technology since 2021. His current research interests include real-time task scheduling in embedded systems, hardware Trojan detection, and IP protection of integrated circuits.

Xing Zhang received the M.S. degree from the Chongqing University of Posts and Telecommunications, Chongqing, China, in 2021.

His research interests include intrusion detection and hardware Trojan detection.

Hao Yan received the B.S. degree from the Chongqing University of Posts and Telecommunications, Chongqing, China, in 2021.

His current research interests include network intrusion detection and nonline-of-sight image reconstruction.

Liang Zhang received the Ph.D. degree from the National University of Defense Technology, Changsha, China, in 2011.

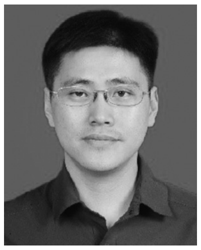
He joined the Chongqing University of Posts and Telecommunications, Chongqing, China, as an Engineer. His current research interests include cyber security and IoT security.

Kefei Cheng received the B.E. degree from Xi'an Technological University, Xi'an, China, in 1997, the M.E. degree from the Chongqing University of Posts and Telecommunications, Chongqing, China, in 2000, and the Ph.D. degree from Chongqing University, Chongqing, in 2005.

He is currently serving as the Director of the Institute of Network Technology, Chongqing University of Posts and Telecommunications, Chongqing, China, and also the Network Intelligence and Network Technology Innovation Team. He serves as a Distinguished Expert of Chongqing CIO Association and Chongqing Software Promotion Center, Chongqing, an Expert Consultant of Chongqing Productivity Promotion Center, Chongqing. He presided over multiple projects from National Natural Science Foundation of China and Chongqing Science and Technology Bureau.

Yu Wu received the Ph.D. degree in automation theory from Chongqing University, Chongqing, China, in 1997.

In 1998, she joined the Chongqing University of Posts and Telecommunications, Chongqing, where she is currently a Professor and the Director of the School of Cyber Security and Information Law. She has participated in and undertaken some research projects, such as National Natural Science Foundation, 863 Plan and Climbing S&T Heights Plan of China, Key Science and Technology Research Foundation by the State Education Ministry of China, Application Science Foundation of Chongqing of China, and Science and Technology Research Program of the Municipal Education Committee of Chongqing of China. Her research interests focus on multimedia communication based on wavelets, intelligent information system based on the artificial neural network and rough sets, and information security, including network invasion detection and anti-spam email filter.



Kaijie Wu received the B.E. degree from Xidian University, Xi'an, China, in 1996, the M.S. degree from the University of Science and Technology of China, Hefei, China, in 1999, and the Ph.D. degree in electrical engineering from Polytechnic University (currently Polytechnic Institute of New York University), Brooklyn, NY, USA, in 2004.

He then joined the University of Illinois at Chicago, Chicago, IL, USA, as an Assistant Professor. In 2013, he became a Professor with the College of Computer Science, Chongqing University, Chongqing, China. His research is on the big area of trustworthy computing with special interest on dependable computing and hardware security.

Prof. Wu is the recipient of the 2004 EDAA Outstanding Dissertation Award for new directions in circuit and system test and the Most Significant Paper Award from the International Test Conference in 2014.