FuncTeller: How Well Does eFPGA Hide Functionality?

Zhaokun Han[†], Mohammed Shayan[*], Aneesh Dixit[†], Mustafa Shihab[*],
Yiorgos Makris[*], and Jeyavijayan (JV) Rajendran[†]
[†]*Texas A&M University,* [*]*The University of Texas at Dallas*
[†] {hzhk0618,aneeshdixit,jeyavijayan}@tamu.edu,
[*] {shayan.mohammed,mustafa.shihab,yiorgos.makris}@utdallas.edu

Abstract

Hardware intellectual property (IP) piracy is an emerging threat to the global supply chain. Correspondingly, various countermeasures aim to protect hardware IPs, such as logic locking, camouflaging, and split manufacturing. However, these countermeasures cannot always guarantee IP security. A malicious attacker can access the layout/netlist of the hardware IP protected by these countermeasures and further retrieve the design. To eliminate/bypass these vulnerabilities, a recent approach redacts the design's IP to an embedded field-programmable gate array (eFPGA), disabling the attacker's access to the layout/netlist. eFPGAs can be programmed with arbitrary functionality. Without the bitstream, the attacker cannot recover the functionality of the protected IP. Consequently, state-of-the-art attacks are inapplicable to pirate the redacted hardware IP. In this paper, we challenge the assumed security of eFPGA-based redaction. We present an attack to retrieve the hardware IP with only black-box access to a programmed eFPGA. We observe the effect of modern electronic design automation (EDA) tools on practical hardware circuits and leverage the observation to guide our attack. Thus, our proposed method *FuncTeller* selects minterms to query, recovering the circuit function within a reasonable time. We demonstrate the effectiveness and efficiency of *FuncTeller* on multiple circuits, including academic benchmark circuits, Stanford MIPS processor, IBEX processor, Common Evaluation Platform GPS, and Cybersecurity Awareness Worldwide competition circuits. Our results show that *FuncTeller* achieves an average accuracy greater than 85% over these tested circuits retrieving the design's functionality.

1 Introduction

1.1 Hardware IP: Threats and Defenses

The monetary loss associated with intellectual property (IP) theft is comparable to the amount of US exports to Asia [72]. This ongoing theft results in a loss of revenue for the IP developers and diminishes incentives for investment in research and development. Many leading semiconductor companies outsource design and manufacturing while owning the hardware IP. In 2020, such companies constituted 33% of the entire semiconductor market [37]. Consequently, there is an economic incentive for reverse engineering, unauthorized usage, and overproduction of hardware IPs. For example, according to a report from the US Department of Justice in 2018, the worldwide supply for dynamic random access memory (DRAM) is worth nearly $100 billion, and Micron controls 20-25% of the DRAM industry; however, IP theft in Micron caused an estimated loss of $8.75 billion [27]. These deleterious consequences underline the need for countermeasures against IP theft in the semiconductor industry.

Semiconductor fabrication facilities are concentrated in countries prone to geopolitical tensions and conflicts [78]. Except for Intel, IBM, and Samsung, nearly all chip fabrication is carried out in such foreign territories. Consequently, the supply chain of the semiconductor industry is vulnerable to threats [76]. Since untrusted foundries and testing facilities have access to the hardware IP, rogue employees from these entities may attempt to pirate the IP. Additionally, an untrusted end-user can purchase a chip from the market and use reverse engineering to extract the hardware IP netlist.

Numerous countermeasures have been proposed to prevent hardware IP theft: logic locking, camouflaging, and split manufacturing are three prominent examples [8, 38, 40, 62, 82]. Logic locking protects the design by adding extra logic controlled by additional key inputs [62, 82]; only the correct key restores the correct functionality. Logic locking can defend against reverse engineering as it *obscures* the IP. Camouflaging deters reverse engineering attacks from end-users by designing a chip that looks alike but performs different functionalities, which is known only to the owner [8]. Split manufacturing protects the IP by splitting the manufacturing process among different foundries, thereby preventing a single foundry from obtaining the complete design information [38, 40]. While these approaches increase trust in the supply chain, they are still vulnerable to several attacks.

The existing IP protection techniques become vulnerable

Table 1: The relationship between various hardware security threats and countermeasures. A ✓/✗ denotes the countermeasure is secure against/vulnerable to the threat, and "N.A." denotes the threat is not applicable to the countermeasure.

Threat	Countermeasure			
	Logic locking [62, 82]	Camouflaging [8]	Split manufacturing [38, 40]	FPGAs [10, 11, 30, 55, 73, 83]
IP piracy [14]	✓	✓	✓	✓
Reverse engineering [75]	✓	✓	✓	✓
Algorithmic attacks [29, 48, 49, 65, 67, 80]	✗	✗	✗	✓
Bitstream extraction [9, 45, 70, 84, 87]	N.A.	N.A.	N.A.	✓

when the attacker has oracle access, i.e., a functional chip sold on the market. For instance, some defenses are vulnerable to input-output (I/O) query-based attacks [29, 67], where selected input patterns are applied, and the output differences between the locked design and the oracle are observed to extract the key. In addition, the attacker can analyze the structural traces in the locked netlist and recover the original design; these attacks are referred to as structural attacks [32, 49, 65]. Collectively, I/O and structural attacks render most logic locking, camouflaging, and split manufacturing techniques vulnerable, as listed in Table 1.

1.2 FPGA-based IP Protection

Given the weakness of existing IP protection techniques, researchers have proposed to redact the complete design information from the untrusted supply chain using a field-programmable gate array (FPGA). An FPGA is a reprogrammable integrated circuit that can be programmed to have an arbitrary functionality by uploading a bitstream to the FPGA. In contrast, traditional application-specific integrated circuits (ASICs) are designed to perform a specified function and cannot be reconfigured. While ASICs comprise multiple gates derived from a specified technology library, FPGAs consist of multiple programmable logic blocks. These logic blocks, in turn, consist of multiple lookup tables (LUTs), which are programmable based on the desired functions. Therefore, an FPGA netlist appears as a cluster of LUTs and does not provide any meaningful information unless programmed with a bitstream. Thus, compared to the conventional integrated circuit (IC) design model, the hardware IP is the bitstream rather than the FPGA platform itself.

Xilinx FPGAs can protect the IP at the software level by encrypting the bitstream and preventing the attacker from using the *readback* functionality to extract the bitstream [9, 83, 87]. Recent attacks attempting to recover the bitstream have been published [9, 45, 70, 84, 87]—however, advanced bitstream protection mechanisms thwart these attacks [30, 73, 83]. These protections safeguard the bitstream at the software and hardware levels. Thus, due to their security and versatility (as shown in Table 1), FPGA-based countermeasures have gained interest in academia and industry [10, 11, 25, 39, 54, 55].

FPGAs can also be embedded into ASIC, referred to as embedded FPGAs (eFPGAs). A design implemented on an ASIC with an eFPGA offers the reconfigurability of FPGAs while mostly retaining the power, performance, and area (PPA) cost-effective benefits of ASICs. eFPGAs have also piqued interest due to their security capabilities. Particularly, it has led to the development of an IP piracy countermeasure known as eFPGA-based hardware redaction. This IP protection has captured significant interest [10, 11, 55]. Obfuscated Manufacturing for GPS (OMG) [54] and Structured Array Hardware for Automatically Realized Applications (SAHARA) [25] are two notable projects supported by the Defense Advanced Research Projects Agency (DARPA) utilizing this technology. The recent Cybersecurity Awareness Worldwide (CSAW) 2021 competition supported by Siemens *et al.* also included designs redacted by eFPGAs [43]. Importantly, the US military has selected Intel's structured ASIC technology, a hybrid of FPGA and ASIC, to protect hardware IPs [25, 39].

1.3 Our Goals and Contributions

In this work, we ask a question: *How secure are eFPGAs?* We answer it by attempting to recover an IP implemented on an eFPGA with only I/O (oracle) access. To this end, we develop a heuristic attack, *FuncTeller*, overcoming the following challenges of recovering hardware IPs on eFPGAs:

1. The size of the search space for an attacker is 2^n, where n is the number of inputs of the redacted design implemented on the eFPGA. Many practical hardware designs have a large number of inputs, e.g., the IBEX processor has 1,386 inputs [81]. This renders brute-force search impractical and forces the attacker to smartly choose input patterns for retrieving the redacted design (Section 2.2).

2. A heuristic algorithm may predict approximate functionality, but it sacrifices accuracy for efficiency. Thus any practical attack must ensure accuracy scaling, particularly for hardware IPs with a large number of inputs (Section 2.2).

3. eFPGAs have a generalized structure independent of the implemented design, and the number of I/O pins is design-agnostic. Generally, some output pins are unused by design. Such unused output pins are driven to a constant 0/1. For example, the IBEX processor [81] implemented on an FPGA with 18 K gates has 117 outputs with constant 0/1 functionality. Identifying these corner cases is crucial for reducing the execution time of a successful attack.

To overcome these challenges, we leverage several properties of hardware designs to develop a practical and effective attack, *FuncTeller*. The salient features of *FuncTeller* are: Firstly, *FuncTeller* predicts the Boolean function of a hardware IP by querying a small number of input patterns within the entire search space of size 2^n. The prediction is made by carefully selecting the input patterns, leveraging the fact that the input patterns with similar behavior are clustered together, and the distance between them is small (Section 4.2). Secondly, *FuncTeller* parameterizes the prediction algorithm, enabling the user to find the appropriate trade-off between accuracy and efficiency. These parameters are empirically derived to optimize the *FuncTeller* prediction (Section 4.2). Thirdly, *FuncTeller* predicts each output independently. Thus, it allows the attack to be executed parallelly on each output, thereby speeding up the overall prediction (Section 4.3). Finally, *FuncTeller* identifies special cases, such as constant logic 0/1 outputs, allowing it to reduce prediction time and improve accuracy (Section 4.4).

The paper's contributions can be summarized as follows:

1. We present a heuristic solution, *FuncTeller*, to extract redacted hardware IP with high accuracy, for *FuncTeller* achieves an accuracy of 91% on the IBEX processor [81] (Section 5).

2. *FuncTeller* can achieve 1.22× more accuracy than the existing state-of-the-art attack [18] for predicting a black-box circuit's functionality (Section 5).

3. We evaluate the performance of *FuncTeller* on various open-sourced and widely-used circuits, including seven academic benchmarks (ISCAS'85 [33] and ITC'99 [26]) and three industrial circuits, such as Stanford MIPS processor [34] (4 K gates), IBEX processor [81] (18 K gates), and Common Evaluation Platform (CEP) GPS circuit [35] (213 K gates) (Section 5.3).

4. We present an analysis of the trade-off between accuracy and efficiency while running *FuncTeller*. In a case study on IBEX [81], the trade-off curve describes the variation in accuracy from 81.2% to 88.5%. This analysis allows the attacker to ascertain the estimated accuracy for a given time limit before running *FuncTeller* (Section 5.5).

5. The paper discusses potential countermeasures to defeat *FuncTeller*, to motivate the development of better countermeasures to safeguard the hardware IP (Section 7.2).

In addition to the listed contributions, we plan to open-source our attack tool.

2 Background and Prior Work

This section describes an intellectual property (IP) piracy countermeasure using embedded field-programmable gate arrays (eFPGAs) and discusses why multiple attempts to circumvent the countermeasure have failed. Finally, we detail hardware design principles from the field of logic synthesis. These principles will form the backbone of our attack.

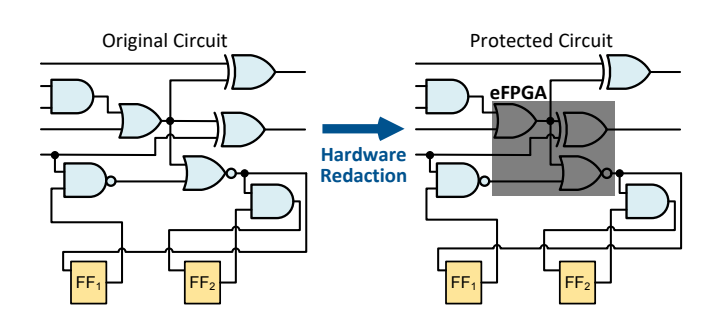

Figure 1: Design netlists before and after hardware redaction. Gates and wires inside the black box are eFPGA-redacted.

2.1 eFPGA-based Hardware Redaction

As described in Section 1.2, a field-programmable gate array (FPGA) netlist does not provide valuable information to an attacker. This fact is utilized to design a special class of circuits wherein a traditional application-specific integrated circuit (ASIC) is embedded with an FPGA. This eFPGA can implement a part of the hardware IP. Thus, *"hiding"* or redacting the said part of the circuit. Post-manufacturing, the eFPGA is loaded with the bitstream. Due to the lookup table (LUT) based structure of FPGAs, the netlist does not reveal the hardware IP that it has been programmed with. Using an eFPGA to *"hide"* the IP is termed as eFPGA-based hardware redaction [10, 11], as represented in Figure 1.

2.2 Prior Work

All digital circuits implement a Boolean function. Thereby, the problem of extracting hardware IP from a black-box design is similar to the problem of learning the Boolean function implemented by the design. Thus, we can observe the responses of black-box design to chosen queries and use the results to recover the Boolean function or its approximation.

Multiple works follow the same principle and provide solutions for learning a Boolean function [13, 19]. However, these algorithms are borrowed from the field of circuit design and are designed for traditional ASICs requiring a netlist. As discussed in Section 2.1, a meaningful netlist with design-specific information is not accessible for eFPGA-redacted hardware. Without the netlist, Angluin *et al.* [4] show that these learning algorithms require querying all 2^n input patterns, where n is the number of inputs, rendering these algorithms inefficient for practical scenarios.

A Satisfiability (SAT) based approach [67] can successfully retrieve IP for ASIC designs, even if the circuit is protected

by state-of-the-art IP protection mechanisms, by solving conjunctive normal forms (CNFs) with a SAT solver. However, this approach does not apply to FPGA designs because FPGAs consist of programmable LUTs. LUTs with large input sizes result in a computational complexity exponential to the number of inputs. Thus, applying SAT solvers to deduce the designs implemented on FPGAs is computationally infeasible. A recent work on deobfuscating eFPGAs, Rezaei *et al.* [61], proposes to apply a SAT-based approach that simplifies the CNF conditions (eliminating structural loops) by leveraging the usage information/parameters of the FPGA (such as the input size and the number of used LUTs) [61]. However, these parameters (the input size and the number of used LUTs) are not accessible to the attacker under our threat model, where the attacker has only black-box access.

Generic logic regression-based approaches [63] attempt to recreate a Boolean function by fitting a model on a collection of data points. These data points are obtained by querying the oracle with input patterns and observing outputs. However, this approach does not consider properties specific to practical hardware circuits and operates on the entire Boolean space. Further, it results in low efficiency and lack of scalability to larger circuits, as shown in Section 2.3.

Chen *et al.* [18] propose a circuit learning approach based on Boolean regression. To the best of our knowledge, this presents the state-of-the-art attack to retrieve black-box Boolean functions. Chen *et al.* [18] overcome the severe drawback of generic logic regression methods by considering properties specific to hardware circuits. However, this approach maps the outputs to a library of Boolean functions to speed up the algorithm, and this speed benefit may not be universally applicable to all circuits. Thus, its practicality is limited.

2.3 Limitations of Existing Approaches

The logic regression techniques mentioned in Section 2.2 face issues while scaling up to large circuits. More importantly, these logic regression techniques have low accuracy. For example, the state-of-the-art tool from Chen *et al.* [18] can recover IBEX with an accuracy of only 56.68%. Further, we observed that some IPs, such as Common Evaluation Platform (CEP) GPS [35], cannot be recovered even after three days of run-time. These techniques choose queries randomly without adapting them to the underlying practical hardware design properties. This hardware-agnostic approach results in low accuracy and efficiency and is a deficiency for all traditional logic regression techniques, including Chen *et al.* [18].

FuncTeller overcomes these drawbacks by focusing on: (i) specificity towards practical hardware designs, (ii) scalability, (iii) accuracy, and (iv) usability. *FuncTeller* specifically targets practical hardware IPs in contrast to previous works, which are mainly extensions of theoretical solutions. As a result, *FuncTeller* significantly improves accuracy and can scale to larger designs. Additionally, *FuncTeller*'s heuristic nature provides users flexibility through user-defined parameters and allows them to tune the attack to different settings.

2.4 Logic Synthesis

As mentioned in Section 2.3, *FuncTeller* considers hardware-specific properties which enable it to outperform previous attacks. These properties are rooted in fundamental logic synthesis principles, which are outlined in this subsection.

Consider a Boolean function f with single output and multiple inputs, a, b, and c. The Boolean function is written as $f = abc + ab\bar{c}$. This representation of the function is referred to as the *sum of products (SOP)*, and every individual product is referred to as a minterm. Here, each product can be regarded as an *ON-set minterm* if it returns $f = 1$ when it is an input; otherwise, an *OFF-set minterm* if the result is $f = 0$. Note that there are 2^n minterms for an n-input function, and each minterm is either ON-set or OFF-set.

For some cases, multiple ON-set minterms can be compressed/merged into an *implicant*. The implicant is represented by elements from $\{0, 1, \text{-}\}$, where "-" is a *don't care* covering both 0 and 1 cases. For example, $I_1 = ab\text{-}$ is an implicant with one don't care bit. I_1 covers two minterms: abc and $ab\bar{c}$. Therefore, the Boolean function of f can be simplified as $f = ab$.

Most logic synthesis algorithms aim to reduce the cost of logic implementation by removing redundant implicants. Implicants uncovered by other implicants are called *prime implicants (PIs)*. A set of PIs containing all ON-set minterms forms a *prime implicant table (PIT)*. For most practical designs, PIs are distributed close to each other. PIs are rarely far apart, according to Han *et al.* [32]: this is an attribute of the Boolean functions of these designs. In the modern hardware design cycle, powerful commercial tools, such as *Synopsys Design Compiler* [68], *Cadence Genus* [15], and *Siemens Precision RTL* [52], utilize this attribute of Boolean functions while performing synthesis processes to optimize the circuit, simplify the logic, and reduce hardware cost. For example, consider the Boolean function of $f = abc + ab\bar{c}$. If each of the function's minterm is implemented individually (without any optimization), the function can be realized using six logic gates, as shown in Figure 2(a). However, the same function can be simplified/synthesized to $f = I_1 = ab\text{-}$, so the circuit can be realized with one logic gate, as shown in Figure 2(b). This type of optimization is a hallmark of all synthesis tools that aggressively minimize the number of PIs in order to minimize the number of gates, which in turn minimizes the circuit's area and power. *FuncTeller* exploits the fact that most practical designs have this property (PIs are distributed close to each other) so that synthesis tools can utilize it and simplify PIT.

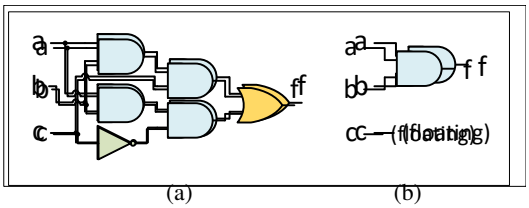

Figure 2: The circuits of Boolean function $f = abc + ab\bar{c}$ (a) before and (b) after synthesis.

3 Threat Model

We consider a threat model consistent with previous works on embedded field-programmable gate array (eFPGA) based hardware redaction [11], attacks on eFPGA [21, 61], and the Cybersecurity Awareness Worldwide competition [43].

Attacker's Locations: The attacker could be a collusion of an untrusted foundry/testing facility and an untrusted end-user. The attacker in the foundry/testing facility has access to a chip netlist with the unprogrammed eFPGA. The attacker can procure a functional chip from the market and can isolate eFPGA by analyzing the chip netlist and identifying the scan chains connected to the eFPGA using reverse engineering [75]. Note that the purchased functional chip is the application-specific integrated circuit (ASIC) integrated with the configured (loaded with a certain bitstream) eFPGA.

Attacker's Capabilities: In this threat model, the attacker has the following capabilities:

- The attacker can isolate the eFPGA from the rest of the design by accessing dedicated scan chains of eFPGA, a feature commonly supported by eFPGA vendors [21]. Alternatively, the attacker can perform a probing attack to isolate eFPGA by locating and accessing the ASIC's internal signals to control/observe eFPGA's inputs/outputs [21, 79].
- An attacker is unable to retrieve the hardware intellectual property (IP) by performing a side-channel attack or bitstream extraction. Over the years, multiple schemes have been proposed to mitigate these attacks [30, 45, 70, 73, 84].
- After isolating the eFPGA, the attacker can enable scan-chain access by stripping scan-chain protections [2, 3, 23, 51]. Further, the attacker can access input-output (I/O) pins through scan chains to query the hardware IP design. Appendix A provides more details on the mechanisms to enable or unlock scan-chain access, including attacks on scan-chain protections [2, 3, 23, 51].

Attacker's Goal: The attacker aims to accurately and efficiently recover the hardware IP's functionality by only querying the black-box combinational part of the hardware design.

4 *FuncTeller*: Technical Approach

In this section, we present our attack, *FuncTeller*, to recover a design implemented on an eFPGA under the threat model described in Section 3. Recall that logic synthesis on practical circuits optimizes power, performance, and area (PPA) by minimizing the number of prime implicants (PIs) and literals in the circuit's prime implicant table (PIT). As a consequence, logic synthesis clusters ON-set minterms into several PIs. This behavior is a consistent feature of practical hardware designs and has the following two implications:

- A single PI covers multiple ON-set minterms that are consistent with the literals in the PI representation. We use this property to predict each PI by expanding from a discovered ON-set minterm (seed): attempting to replace each literal with a don't care and heuristically verifying each replacement.
- The distance between any two PIs in a PIT is usually much smaller than the input size. We use this property to reduce the search space when updating the predicted PIT with the new PI. The generation of new PI requires the discovery of the next ON-set minterm. Therefore, we limit the search space for the next ON-set minterm to being in close proximity to the current PIs.

For the remainder of Section 4, we show *FuncTeller* exploits these implications. Further, we describe the mechanism of *FuncTeller*, using the example circuit in Figure 3. Consider a circuit cone[1] with n inputs $(a_1, a_2, \cdots, a_n)$ and one output; the circuit implements the Boolean function f with $n = 6$, as shown in Figure 3(a). The Boolean function is $f(a_1, a_2, \cdots, a_6) = a_1 a_2 \overline{a_4} + a_4 \overline{a_6} + \overline{a_1} a_6$, and its PIT is shown in Figure 3(b). Note that the expression of PIT for a certain Boolean function is non-canonical.[2]

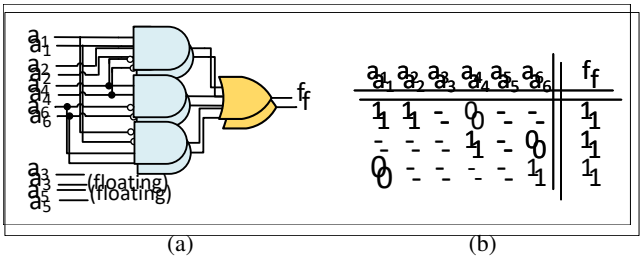

Figure 3: An example circuit: (a) the netlist and (b) the PIT.

4.1 Initialization

FuncTeller initializes by finding an ON-set minterm m_0. We search for m_0 by querying the oracle with random minterms,

[1] A multi-output circuit could be collapsed into several 1-output circuits. Such a 1-output circuit is referred to as a cone.

[2] A circuit representation, which is not unique, is considered non-canonical. Here, there may exist multiple valid PITs for the same output.

and it ends upon finding an ON-set minterm within r queries ($r \in \mathbb{N}^+$). If all the r queries are OFF-set minterms, then *FuncTeller* regards the output's function as constant 0 and terminates out of the purpose of efficiency.

4.2 Expanding from ON-set Minterm to PI

After finding an ON-set minterm m_0, *FuncTeller* predicts the PI containing m_0. This is referred to as "the expansion of the ON-set minterm m_0 to a predicted PI PI_{pred}". Algorithm 1 describes the corresponding routine. We first set a variable string "*cube*" equal to the binary pattern of the ON-set minterm m_0. We update each literal of *cube* to form predicted PI PI_{pred}.

For each (i^{th}) literal of the cube, we recursively construct a new group of minterms $\{m_1\}$ by inverting the i^{th} *cube* literal and replacing the don't cares (if any) with binary "1"/"0". After constructing each minterm m_1, we query it using the oracle O. If the query shows that m_1 is in the OFF-set, we stop generating new minterm m_1 corresponding to the i^{th} literal inversion. Instead, we keep the i^{th} literal as in m_0 and move to the next literal. On the other hand, if all the minterms formed by inverting the i^{th} *cube* literal belong to the ON-set, then the i^{th} *cube* literal is updated to a don't care "-" and then we move to the next literal. Once it has been through all literals in m_0, the algorithm returns the updated *cube* as PI_{pred}.

During the PI prediction, the number of minterms to be queried is exponential to the number of don't cares in the *cube*. Algorithm 1 uses a heuristic approach to avoid querying all covered minterms and can be inaccurate while predicting. The probability of prediction error depends on the number of minterms queried during the prediction routine. To control the error while keeping *FuncTeller* efficient, we introduce a "linear limitation parameter p." When updating each literal, the maximum number of queries is capped at p times the number of don't cares. Thus, the PI prediction for an n-input function requires querying at most $p \times n^2$ minterms. Thus, the algorithm takes time complexity $O(p \times n^2) = O(n^2)$. We further discuss the performance improvement of Algorithm 1 in Appendix B and Appendix C.

We now explain Algorithm 1 with the aid of an example. Consider the minterm $m_0 = 000110$, which belongs to f's ON-set, as shown in Figure 3. We expand this minterm to a PI in 6 stages, as shown in Figure 4 and described below:

1. Initially, the first literal a_1 is flipped to obtain a query **1**00110. The oracle returns "1", i.e., the queried minterm belongs to the ON-set. As a result, a_1 is replaced with a don't care "-" and the cube is updated as -00110.

2. Then, the second literal a_2 is flipped, and the don't care (a_1) is replaced by "1" and "0" to obtain the next set of queries. Recall that we use the linear limitation parameter ($p = 1.1$) to limit the number of queries.[3] This implies that we replace a_2 to obtain $1.1 \times d \approx 2$ queries: 1**1**0110 and 0**1**0110. Since both queried minterms belong to the ON-set, a_2 is replaced by "-" to update the cube as --0110.

3. Now, a_3 is flipped, and the two don't cares (a_1, a_2) are replaced randomly by "1/0" to obtain $1.1 \times d \approx 3$ queries, as shown in Figure 4. After observing the result, a_3 is replaced by a don't care, and the resultant cube is ---110.

4. Next, a_4 is flipped to "0", and the don't cares are replaced, as in the earlier stages. The first query 000**0**10 returns "0",

[3] Based on our tested designs, $p = 1.1$ is an empirically derived constant.

Algorithm 1: Expansion of an ON-set minterm to PI

Input: Oracle O, output index w, ON-set minterm m_0, linear limitation parameter p
Output: Predicted PI PI_{pred}

```
Function expand_minterm_to_PI(O, w, m0, p):
    cube := m0                                   ▷Initialize cube
    for index ∈ {1,2,···,len(m0)} do
        num_dc := cube.count("-")
        iter_limit := min{2^num_dc, p × num_dc}
        cube := update_cube (O, w, cube, index,
          iter_limit)
    PI_pred := cube
return PI_pred
Function update_cube(O, w, cube, index, iter_limit):
    flag := 1
    for i ∈ {1,2,···,iter_limit} do
        m1 := pick_random_minterm(cube, index)
        response := O.query_oracle(m1, w)
        if response == 0 then
            flag := 0          ▷Fail to replace with don't care
            break
    if flag == 1 then
        cube := replace_with_dc_bit(cube, index)
return cube
```

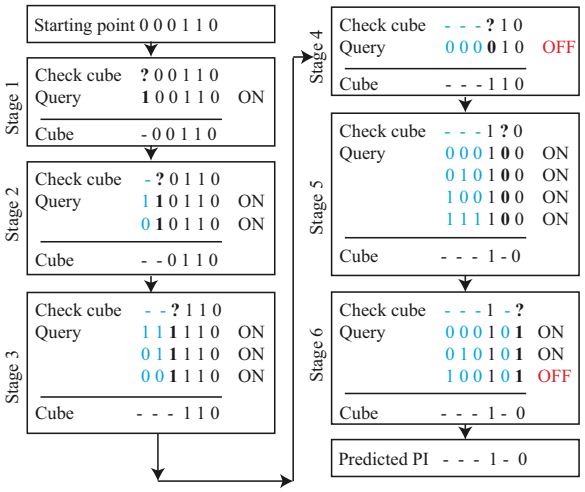

Figure 4: The PI expansion from the initial minterm (starting point/seed) $m_0 = 000110$ to $PI_{pred} = ---1\text{-}0$.

i.e., the queried minterm is OFF-set. Therefore, $a_4 = 1$, and cube (---110) is restored back, as shown in Figure 4.

5. Then, a_5 is flipped, and $1.1 \times d \approx 4$ queries are generated by replacing the three don't cares, as shown in Figure 4. As all queries belong to the ON-set, the cube is updated to ---1-0 by substituting a_5 with "-".

6. In the last step, a_6 is flipped to "1". The oracle is queried by replacing don't care bits as earlier. However, the third query 100101 returns "0". As a result, the step terminates by retaining the cube ---1-0, forming the predicted PI.

The time complexity of Algorithm 1 is $O(n^2)$. Thus, scalability is a concern when n is large. To reduce the worst-case time complexity, we rely on the following observation: In most practical circuits, not all inputs participate in the computation of a given output. Thus, effective inputs[4] of the output are usually much less than all inputs. If these effective inputs can be identified for each output, then the time taken for PI expansion can be reduced by querying fewer minterms. Appendix B provides an algorithm and more details on how to scalably expand from an ON-set minterm to a predicted PI.

4.3 Searching for the Next ON-set Minterm

After expanding an ON-set minterm to a predicted PI, *FuncTeller* searches for another ON-set minterm to be expanded as the next PI. Rather than randomly searching for the next ON-set minterm in the complete Boolean space ($\mathbb{B}^n$), *FuncTeller* uses a satisfiability (SAT) solver to reduce the search for the next ON-set minterm. Specifically, the search space is encoded with the following constraints:

[4]An effective input can propagate its value to the output.

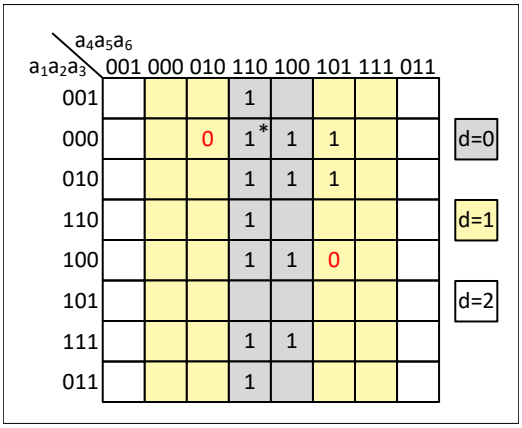

Figure 5: Search space for the next ON-set minterm when the current predicted PIT is {---1-0}, as shown in the example of Figure 4. Gray shade denotes the space covered by the current predicted PIT, yellow shade denotes space with $d = 1$ from PIT, and the entry with $*$ denotes the starting point m_0.

(1) The search space for the next ON-set minterm excludes the current predicted PIT. In other words, the next minterm is not within the distance of $d_0 = 0$ of any existing predicted PI. The gray shade in Figure 5 showcases the excluded space when the current predicted PIT is {---1-0}.

(2) The search space for the next ON-set minterm only includes the neighboring minterms of the current predicted PIT. More specifically, the search space is bounded within the distance of d_0 to the current predicted PIT.[5] The yellow shade in Figure 5 showcases this constraint when $d_0 = 1$.

For an n-input function, if the current predicted PIT has one PI with x don't care bits, the size of the search space for the next ON-set minterm is $(n-x) \times 2^x$ instead of 2^n. In Figure 5, $n = 6$ and $x = 4$.

We now elaborate on the encoding process for the next ON-set minterm search according to constraints (1) and (2).

First, we construct a conjunctive normal form (CNF) CNF_0 describing the inverse of the predicted PIT. In other words, the next candidate minterm m' belongs to the OFF-set of the current predicted function f_{pred}. Thus, CNF_0 is encoded as

$$\{m' \mid f_{pred}(m') = 0\}. \tag{1}$$

Next, we add constraint (2) to walk through the uncovered space incrementally. This is done by searching within the distance of d from the PIs in the predicted PIT. The distance d is incremented from 2 to n (the number of inputs) to ensure that the search is efficiently directed. Therefore, the candidate minterms belong to the following set

$$\{m' | \exists m_0 \in \mathbb{B}^n, \text{s.t.} f_{pred}(m_0) = 1 \text{ and } D(m', m_0) \leq d\}. \tag{2}$$

This CNF encoding is denoted as CNF^d_{cstr}. The constraint CNF for the next ON-set minterm search, CNF_{search}, is

$$CNF_{search} = CNF_0 \wedge CNF^d_{cstr}. \tag{3}$$

These constraints are fed to a SAT solver that returns a candidate minterm m'. This minterm (m') is queried using the oracle. If m' belongs to the ON-set, then the search is terminated, and the minterm m' is expanded to a predicted PI as described in Algorithm 1. On the other hand, if m' belongs to the OFF-set, the SAT solver is called again to return another candidate minterm. However, to avoid SAT solver returning the same solution, CNF_0 is updated to exclude the previous result. Assume m'_{OFF} is the previous OFF-set minterm, then the CNF constraint in Equation (1) is updated as

$$CNF_0 = CNF_0 \wedge (\neg m'_{OFF}). \tag{4}$$

The process of generating candidate minterms is recursive until either finding an ON-set minterm or consecutively

[5]Distance is the number of pairs of $(0,1)$ and $(1,0)$ between two PIs. The concept of distance is similar to "Hamming distance," except distance takes into account PIs containing don't cares, but Hamming distance does not.

Algorithm 2: Prediction on the w-th output

Input: Oracle O, output index w, distance parameter d_0, linear parameter p, convergence parameter p_{conv}, time limit T
Output: Predicted PIT PIT_{pred}

1 **Function** `predict_cone(`O, w, d_0, p, p_{conv}, T`)`**:**
2 $PIT_{pred} := \varnothing$
3 $m_0^{1st} :=$ search_for_1st_ON_set_minterm(O)
4 $PI_{pred}^{1st} :=$ expand_minterm_to_1stPI$(O, w, m_0^{1st},)$
5 $PIT_{pred} := PIT_{pred} \bigcup \{PI_{pred}^{1st}\}$
6 **while** $exe\_time < T$ **do**
7 $CNF_0 :=$ exclude_PIT(PIT_{pred})
8 $d := d_0$
9 $CNF_{search} := CNF_0 \wedge CNF_{cstr}^d$
10 $solution :=$ query_SAT_solver(CNF_{search})
11 $counter := 0$
12 **while** $solution \neq UNSAT$ & $exe\_time < T$ **do**
13 **if** $counter \geq p_{conv}$ **then**
14 **break** ▷Consecutively visit p_{conv} OFF-set minterms
15 $m' :=$ extract_minterm$(solution)$
16 **if** $O.query\_oracle(m_0, w) == 1$ **then**
17 $m_0 := m'$
18 $PI_{pred} :=$ expand_minterm_to_PI(O, w, m_0, p)
19 $PIT_{pred} := PIT_{pred} \bigcup \{PI_{pred}\}$
20 **goto** line 6
21 **else**
22 $counter := counter + 1$
23 $m'_{OFF} := m'$
24 $CNF_{search} := CNF_{search} \wedge (\neg m'_{OFF})$
25 $solution :=$ query_SAT_solver(CNF_{search})
26 **if** $exe\_time < T$ **then**
27 $d := d + 1$
28 **if** $d == n + 1$ **then**
29 **return** PIT_{pred}
30 **else**
31 **goto** line 9
32 **return** PIT_{pred}

visiting p_{conv} OFF-set minterms ($p_{conv} \in \mathbb{N}^+$). p_{conv} is a user-defined convergence parameter for balancing between efficiency and accuracy, as shown in Algorithm 2. Appendix C details the analysis on p_{conv}. If p_{conv} OFF-set minterms are consecutively visited, then the constraint CNF_{cstr}^d is updated by incrementing d in Equation (2). If $d = n+1$ (the first time that $d > n$), *FuncTeller* terminates.

4.4 Recovering Single Output

The process of searching for the next ON-set minterm and the subsequent PI expansion is repeated till one of the termination conditions is met. One termination condition is $d = n+1$, as described at the end of Section 4.3. Another termination condition is reaching a user-defined time limit T. When the time exceeds the specified time limit, the attack terminates

Algorithm 3: Predicting entire circuit's functionality

Input: Oracle O, distance parameter d_0, linear parameter p, convergence parameter p_{conv}, time limit T
Output: Entire predicted circuit C_{pred}

1 **Function** `predict_circuit(`O, d_0, p, p_{conv}, T`)`**:**
2 $output\_size :=$ count_number_of_outputs(O)
3 $Cones_{pred} := \varnothing$
4 **for** $w \in \{1, 2, \cdots, output\_size\}$ **do**
5 $PIT_{pred} :=$ `predict_cone(`O, w, d_0, p, p_{conv}, T`)`
6 $Cones_{pred}^w :=$ convert_PIT_to_netlist(PIT_{pred})
7 $C_{pred} :=$ merge_cones_to_circuit$(Cones_{pred})$
8 **return** C_{pred}

and returns the predicted PIT generated during attack execution, as described in Algorithm 2. The time limit termination condition is checked after each PI expansion and during the search for the next ON-set minterm.

Special Case: Outputs with Constant Logic 0/1. The numbers of input and output pins utilized in an eFPGA design are defined by the function being implemented. The remaining pins are constant 0/1. If we can identify these constant 0/1 outputs, then it allows us to allocate more time to predict other non-constant outputs. If the first predicted PI (PI_{pred}^1) consists of all don't cares ("-") and with no literal ("0"/"1"), we regard the output's functionality as constant 1. On the other hand, if the algorithm cannot find the first ON-set minterm, we consider the output's functionality as constant 0. If the output is determined as constant 0/1, then the prediction terminates.

4.5 Recovering the Entire Circuit

Having described how to recover a single output, we now discuss how to recover the entire circuit. The prediction on each output can be made in parallel, as shown in Algorithm 3. Once the predictions on all outputs finish, we collect all predicted PITs. Electronic design automation (EDA) tools can synthesize the design, such as *Synopsys Design Compiler* [68], *Cadence Genus* [15], and *Siemens Precision RTL* [52], to satisfy the desired PPA constraints for the predicted circuit.

4.6 *FuncTeller* in IC and FPGA Design Flows

Since the predicted logic format is PIT, it raises the question of *how to utilize the predicted design in the IC design flow*. We address this by discussing the steps the attacker can take after using *FuncTeller*. After recovering an eFPGA-redacted design, the attacker can flexibly choose to either upload the bitstream with predicted design information on the FPGA or replace the redacted components with ASIC netlist and generate its layout, as shown in Figure 6. To generate the bitstream of the FPGA or the layout of the ASIC, the attacker could utilize academic tools (e.g., *Berkeley ABC* [12]) and industry-standard commercial tools (e.g., *Xilinx Vivado* [83], *Synopsys*

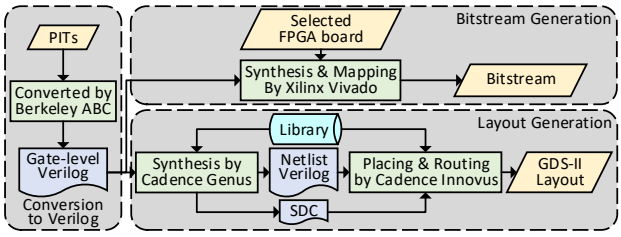

Figure 6: The IC design flow with *FuncTeller*-predicted PITs.

Design Compiler [68], *Cadence Genus* [15], and *Cadence Innovus* [17]. Next, we describe in detail the bitstream and layout generation.

Bitstream Generation. *FuncTeller* predicts PITs of the target design with input-output query access. After collecting the predicted PITs of all outputs, we convert them into a structural netlist by converting the logic of each output's PIT into a Verilog file using *Berkeley ABC* [12]. This gate-level design is then passed on to *Xilinx Vivado* [83] for synthesis, implementation, and bitstream generation. Thus, the attacker can upload this bitstream of the predicted design on the FPGA.

Layout Generation. Similar to the process of generating bitstream, the attacker can convert the predicted PITs to the Verilog design, as shown in Figure 6. Then, we use *Cadence Genus* [15] to generate the synthesized netlist Verilog design with the selected library, as shown in Figure 6. In this synthesis process, *Cadence Genus* also merges/optimizes the logic and generates an industry-standard SDC file containing design constraints and timing assignments. Further, to generate the layout, we provide the netlist Verilog file, SDC file, and library as inputs to *Cadence Innovus* [17], the physical implementation tool. *Cadence Innovus* optimizes the placing and routing processes to generate the optimal layout. The timing, power, and size characteristics of the *FuncTeller*-recovered design will vary and depend on what synthesis tools and constraints are used by the attacker and the defender. eFPGAs are purported to protect functionality [11, 55], which is the challenge we target in this work.

5 Results

In this section, we run *FuncTeller* on select practical circuits and discuss the efficiency and effectiveness of *FuncTeller* compared to the state-of-the-art technique, Chen *et al.* [18]. Later, we verify the validity of the underlying assumptions of our attack and corroborate these assumptions with our results. Finally, we further analyze *FuncTeller*'s performance on a few selected circuits and present them as case studies for more in-depth understanding.

5.1 Simulation Setup

FuncTeller targets practical circuits considering only black-box access to the combinational part of a design. Specifically, *FuncTeller* attacks hardware designs redacted by embedded field-programmable gate arrays (eFPGAs). To reduce the impact of inherent variation among multiple vendors due to factors such as field-programmable gate array (FPGA) board size and clock speed, we evaluate *FuncTeller* by testing it against circuits implemented on the OpenFPGA framework [71].

Environment: We run the attack simulations on a 32-core Intel Xeon processor at 2.6 GHz with 512 GB RAM. We use *Verilator* [66] for generating executable binary oracles of tested circuits, *Synopsys Design Compiler* [68] for synthesis, and the *Berkeley ABC* tool [12] for converting to Verilog.

Tested Circuits: We select a wide range of test circuits at different scales, including seven circuits from ISCAS'85 and ITC'99 benchmark suites, two processor circuits (Stanford MIPS and IBEX), and the Common Evaluation Platform (CEP) GPS circuit [26, 33–35, 81]. Further, we evaluate *FuncTeller*'s performance on circuits from the Cybersecurity Awareness Worldwide (CSAW) 2021 competition [43].[6]

5.2 Evaluation Metrics

To assess the performance of *FuncTeller*, we employ the following two metrics: (i) equivalence checking and (ii) an accuracy metric quantifying the functional similarity between the predicted and original circuits.

Equivalence Checking: This metric considers the ratio of functionally equivalent outputs to all outputs. Commercial tools such as *Synopsys Formality* [69] and *Cadence Conformal* [16] are used to perform the equivalence check. The equivalence checking tool identifies whether a predicted output is equivalent to the original circuit. Naturally, the tool does not quantify the error rate or functional similarity. Thus, this metric may be misleading as a design may have a very low error rate (<0.01%) but may have a low equivalence check score. Thus, we use another metric to quantify the error rate/accuracy and provide an in-depth evaluation.

Simulation-based Accuracy: Due to the drawback of the equivalence checking-based metric, we need to formalize an output accuracy-based metric to ensure a fair comparison with previous work and correctly measure attack effectiveness. To this end, we employ a simulation-based method to quantify the functional similarity between the original and predicted designs. For every i^{th} output ($i \in \{1,2,\cdots,m\}$, where there are m outputs), we test the original and predicted circuits with a set of randomly chosen minterms and record the number of minterms with matched output responses from both circuits. Thus, we calculate the accuracy of the i^{th} output as

$$AC_i = \frac{\text{\# minterms with matched responses}}{\text{\# tested minterms}} \times 100\%.$$

[6] CSAW is the largest student-run cybersecurity event in the world [43].

Table 2: A performance comparison between Chen *et al.* [18] and *FuncTeller* on multiple test circuits using different metrics. Some cells are marked as "*erroneous case*," which implies the failure of Chen *et al.* [18] to run on the tested circuit.

Circuit		# inputs	# outputs	# gates	Success rate				Efficiency
					Formal (%)		Simulation (%)		Attack time (hour)
					Chen *et al.* [18]	*FuncTeller*	Chen *et al.* [18]	*FuncTeller*	*FuncTeller*
Benchmark	c432	36	7	160	0	28.57	82.51	99.86	0.09
	c880	60	26	383	0	53.85	82.01	96.12	1.06
	c1355	41	32	546	0	0	61.02	50.77	1.01
	c1908	33	25	880	0	0	63.73	81.82	1.00
	c7552	207	107	3,512	0	53.27	72.05	86.66	1.25
	b14	277	299	9,821	*erroneous case*	20.40	*erroneous case*	92.46	1.39
	b20	522	512	6,787	0	11.91	64.26	84.10	9.92
Others	MIPS	466	330	3,902	0	63.39	81.97	95.49	1.84
	IBEX	1,386	1,385	18,087	0	16.46	56.68	90.96	72.85
	GPS	9,707	9,731	213,125	*erroneous case*	25.36	*erroneous case*	68.89	44.44

We repeat this process on all outputs and take the average value $AC = \sum_{i=1}^{m} AC_i / m$ as the accuracy of the entire design.

5.3 Performance of *FuncTeller*

We use the following parameter values to run *FuncTeller*: distance parameter $d_0 = 2$, linear parameter $p = 1.1$, constant limitation parameter $p_0 = 8$, and convergence parameter $p_{conv} = 50$. We obtain these values empirically.

Table 2 shows that *FuncTeller* has a higher accuracy compared to the state-of-the-art technique, Chen *et al.* [18], on all tested circuits except for c1355. The average accuracy (simulation-based) of Chen *et al.* [18] and *FuncTeller* is 70.5% (excluding b14 and GPS) and 84.7%, respectively. Thus, *FuncTeller* has an improvement of 14.2% compared to Chen *et al.* [18]. *FuncTeller* achieves an accuracy of 70% while attacking GPS, with an attack time of 44.4 hours. Meanwhile, Chen *et al.* [18] fails to run on b14 and GPS, while it runs for more than 5 days on IBEX and achieves an accuracy of only 56.68%. We believe these anomalies are due to mismatches between the circuits and the tool's presumptions. Chen *et al.* [18] relies on similarities in port names and the presence of few selected linear operators, whereas *FuncTeller* makes no such presumptions. Therefore, *FuncTeller* achieves better prediction performance compared to Chen *et al.* [18] and recovers more designs.

The attack time for *FuncTeller* is the sum of two parts: (i) the maximum execution time while predicting single outputs and (ii) the time taken to merge all predicted single output cones into the entire circuit. For an attacker unconstrained by a lack of computational resources, it may be possible for them to run all single output predictions in parallel to speed up (i). However, considering computational limits, we run our attack on a maximum of 50 outputs in parallel.

In addition to the tested circuits in Table 2, we also run Chen *et al.* [18] and *FuncTeller* on the circuits featured in the CSAW 2021 competition [43]. The logic locking event included circuits redacted with eFPGA based on the OpenFPGA framework. For details, please check Appendix E.

5.4 Survey on Distance Between PIs

As mentioned in Section 5.3, we choose the distance parameter $d_0 = 2$ in *FuncTeller*. In this subsection, we explain the reason behind this decision. The choice of $d_0 = 2$ is due to the distances between prime implicants (PIs) on most practical hardware intellectual properties (IPs). To understand these distance properties on practical circuits, we conduct a study on IBEX [81] and investigate the prime implicant table (PIT) of each output using the *Berkeley ABC* tool. First, for each PIT, we calculate the distance between any two PIs within the PIT. Then, we collect the distance distribution among all extracted outputs PITs, as shown in Figure 7. In each extracted PIT, all pairs of PIs are with a distance of ≤ 19. This result gives a preliminary idea of the distance distribution for each pair of PIs. Further, to show how close each PI is to the rest of the PIT, we investigate the minimum distance of each PI to all other PIs in the same PIT, as shown in Table 3. Table 3 shows that the minimum distance between each PI to other PIs in the same PIT is either 0, 1, or 2. In other words, in the same PIT, for an arbitrary PI PI_i, there always exists another PI PI_j in the same PIT, such that $D(PI_i, PI_j) \leq 2$. Thus, to make the process of searching for the next ON-set minterm efficient and effective, we choose $d_0 = 2$. The findings of this analysis are generalizable to most practical circuits as the findings are an outcome of electronic design automation (EDA) algorithms clustering minterms together during the design encoding stage. Thus, the choice of $d_0 = 2$ is valid for most practical hardware IPs.

5.5 Performance Trade-off

To observe the trade-off between the time limit and the accuracy, we repeat *FuncTeller* on IBEX [81] 10 times. We choose IBEX as the design to be tested as IBEX has 1,386

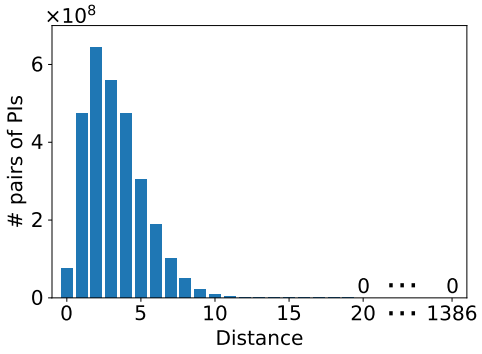

Figure 7: The distribution of the distance between two PIs of the extracted PIT for the case study of IBEX [81].

Table 3: A case study of IBEX [81]: the distribution of the minimum distance for PIs.

Distance	0	1	2	3	···	1,386
# PIs	7.8×10^5	1.4×10^3	4	0	···	0

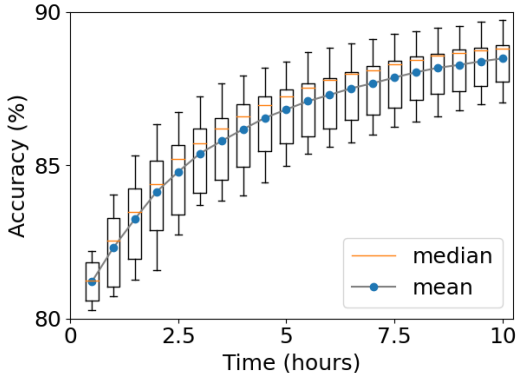

Figure 8: The performance trade-off between accuracy and the attack time of *FuncTeller* on IBEX [81].

inputs and 1,385 outputs and can be considered a large-scale design. Further, we calculate the accuracy with different time limits. The time limit estimates the total attack time. Note that *FuncTeller* terminates PIT predictions within the time limit; however, the actual attack time may exceed the time limit due to the last PI expansion still taking place and the PIT merging processes for individual cones.

Figure 8 shows the box plot considering 10 sets of attacks on IBEX when the time limit (T) ranges from 0.5 hours to 10 hours with $\Delta T = 0.5$ hours as the time step. When $T = 0.5$ hours, the average accuracy is 81.2%; when $T = 10$ hours, the average accuracy is 88.5%. Among the repeated 10 attacks on IBEX, the standard deviations of accuracy with different time limits are always within 1.5%. Thus, the accuracy results are stable since the random processes in *FuncTeller* do not result in significant accuracy deviations among repeated attacks. Further, by observing the tendency of the accuracy vs. time curve, the attacker can fit the collected data in a mathematical model, such as a logarithmic model, and use it to estimate the accuracy for a desired time limit T, and vice versa.

5.6 *FuncTeller* on a Real-World Application

Recall that we evaluate the performance of *FuncTeller* by calculating the accuracy metrics discussed in Section 5.2. We now show that the design recovered by *FuncTeller* can meaningfully work by evaluating the performance of *FuncTeller* at the application level.

We consider a hardware-software co-design application for image processing [28]. The core of this application is a hardware design in Verilog, `image_processing.v`. This application reads an image and generates an output image based on Gaussian blur, an image processing function. There are a total of eighteen 16-bit adders in the image processing circuit. We assume that this circuit is protected by redacting all the 16-bit adders to an eFPGA. We then use *FuncTeller* to recover the circuit by predicting and replacing the redacted modules. Thus, the recovered image processing circuit contains eighteen *FuncTeller*-recovered 16-bit adders. In our evaluation, we compare the output images of: (i) the golden image processing circuit with programmed eFPGA, (ii) the protected image processing unit with un-programmed eFPGA, and (iii) the recovered image processing circuit with *FuncTeller* prediction.

We use a 128×128-pixel image of Peppers for our evaluation, as shown in Figure 9(a). We perform Gaussian blur image processing on this input. Figure 9(b) shows the output

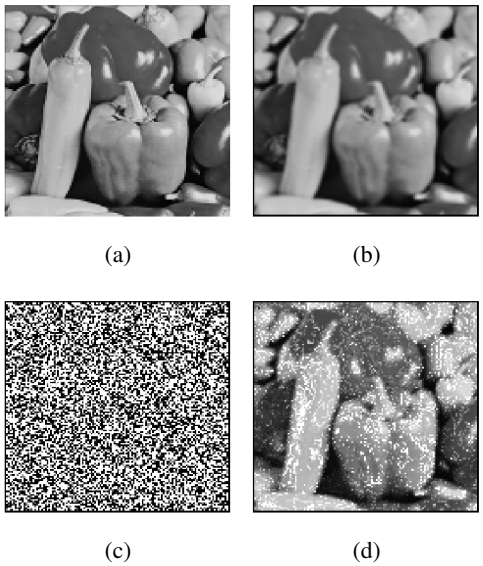

Figure 9: (a) Input image, (b) original output image, (c) output image when all the 16-bit adders are redacted by eFPGA, and (d) output image after the recovery of *FuncTeller*.

image of the original hardware, which is the golden reference. Figure 9(c) shows the output of the image processing circuit with un-programmed eFPGA. Recall that all eighteen 16-bit adders are redacted to an eFPGA. To mimic the behavior of the un-programmed eFPGA, we replaced all 16-bit adders with random number generators. The result in Figure 9(c) shows that the image content is not recognizable. Finally, Figure 9(d) shows the output image of the *FuncTeller*-recovered image processing circuit. A comparison of Figure 9(b) and Figure 9(d) images shows that the output image of recovered hardware is similar to that of the golden one.

6 Related Works

Application-specific integrated circuits (ASICs) and field-programmable gate arrays (FPGAs) allow for a variety of intellectual property (IP) protection mechanisms. Since there has been a significant interest in developing IP protection mechanisms, it is essential to determine the impact of *FuncTeller* in the field. To this end, we present related works in IP protection to understand *FuncTeller*'s relation to the current literature.

6.1 IP Protections and Attacks on ASIC

Preventing piracy of ASIC designs led to the genesis of initial IP protection countermeasures. Concurrent efforts by the research community led to the development of multiple defensive countermeasures, where most countermeasures protected against threats emerging from untrusted entities with minor differences in their threat models and use-case scenarios. Notably, almost all developed defenses were also successfully broken by innovative attacks.

Logic Locking inserts additional circuitry along with additional key inputs. The functionality is retrieved only with the correct key [62, 82]. Thus, logic locking can defend against threats such as reverse engineering and IP piracy. Logic locking has gained significant interest as an IP piracy countermeasure. For instance, the Siemens Security TrustChain platform integrates logic locking as a countermeasure [53]. However, powerful input-output (I/O) query-based and structural attacks [49, 65, 67] break many logic locking techniques.

Camouflaging [8] adds additional dummy layout structures to hide the IP layout details. Notably, Rambus utilized camouflaging and developed its proprietary camouflaging technology [59]. However, a design protected by camouflaging can be mapped to a logic-locked design [86]. Thus, pirating the hardware IP protected by camouflaging is equivalent to breaking the corresponding logic-locked design [29].

Split Manufacturing is an IP protection technique applied to IP layout. The layout layers are split into the back-end-of-line (BEOL) and front-end-of-line (FEOL) layers. Then, the FEOL layers are sent to an untrusted foundry, and a trusted foundry fabricates the BEOL layers. Without the BEOL connections, the attacker (in the FEOL foundry) cannot directly pirate the design. The Intelligence Advanced Research Projects Activity (IARPA) proposed split manufacturing to protect chip fabrication [36]. However, there have been successful algorithmic attacks against split manufacturing, such as proximity attacks on split-manufactured designs [48, 80].

Multiple proposed attacks defeat IP protection techniques on ASICs. As stated in Table 1, algorithmic attacks can circumvent the previously mentioned countermeasures (logic locking, camouflaging, and split manufacturing) on ASICs. However, algorithmic attacks cannot successfully adapt to eFPGAs. I/O attacks, such as Satisfiability (SAT) attack [67] and its variants, cannot scale to attack eFPGAs. This incompatibility is due to the exponentially increased complexity of attacking large lookup table (LUT) based structures for SAT solvers. Structural attacks [49, 65] are not applicable to eFPGAs due to the lack of meaningful structural traces since each LUT's function and routing information are not readily available. Thus collectively, algorithmic attacks, one of the most popular forms of IP theft attacks, fail to break eFPGAs.

6.2 IP Protection Techniques on FPGA

Bitstream Protection. Previous works have considered extracting the design from the bitstream [9, 87]. The bitstream of an FPGA is encoded with the design functionality. Thus, bitstream extraction would enable reverse engineering of the design implemented on the eFPGA. Thus, it is important to protect the eFPGA bitstream by developing countermeasures. Bitstream extraction attacks can be circumvented by disabling read-back capabilities, ensuring strong encryption on the bitstream, and storing the bitstream in a tamper-proof memory [83]. Furthermore, other bitstream protections from a hardware perspective [30, 45, 70, 73, 84] defend against side-channel attacks. The increased interest in developing bitstream extraction countermeasures calls for newer attack methods to recover the design redacted by eFPGA.

6.3 Alternative Attacks on eFPGA

Chhotaray *et al.* [20] stated that, for a protected design, the main objective should be function recovery rather than recovering the correct key when facing a strong attack model [20], such as the threat model in this paper. Functional recovery considers the ratio of input patterns that result in the correct output and total input patterns. The concept of function recovery is useful in quantifying attacks for eFPGA redacted hardware. This paper uses this concept to quantify the performance of *FuncTeller*, as shown in Section 5.2.

Recently, attacks following this principle have been proposed. Chowdhury *et al.* [21] use a predictive model which aims to map the eFPGA redacted design to a previously known circuit. This approach requires obtaining data from a large pool of circuits [21]; however, creating such a dataset is challenging, especially for proprietary designs, such as indus-

trial processors, as they are not open-source. *FuncTeller* does not need such a dataset and, thus, can be applied to proprietary designs. Recent research also investigates the percentage of redacted fabric in the design. Ulabideen *et al.* [77] found that an obfuscation rate of 80% on SHA-256 prevents most template-based attacks. Further, they could synthesize the stated design to match current state-of-the-art constraints. *FuncTeller*, due to its heuristic approach, remains effective as the attack is independent of the design's obfuscation rate and is not dependent on a dataset of previous designs, which is a drawback of predictive and template-matching approaches.

7 Discussion

Even if *FuncTeller* broadly breaks most practical hardware designs efficiently and effectively, some limitations exist.

7.1 Limitations of *FuncTeller*

Table 2 shows that the *FuncTeller* predicts circuit functionality with an average accuracy of 85%. However, the accuracy drops to close to 50% on c1355 and Advanced Encryption Standard (AES) circuits. Analyzing these corner cases helps develop countermeasures against *FuncTeller*.

c1355 is a 32-bit single-error-correcting design [33]. This implies that, for any two adjacent minterms keeping a distance of 1, one of the minterms belongs to the ON-set, and the other belongs to the OFF-set [31]. In other words, on each output of c1355, each ON-set minterm is a prime implicant (PI), and there are 2^{32-1} PIs in each prime implicant table (PIT). The PIT of each c1355's output contains the exponential number of PIs to the input size, which is not scalable. As a result, *FuncTeller* predicts c1355 with low accuracy, even though c1355 is small-scale with 41 inputs, 32 outputs, and 546 gates.

AES, a popular and well-researched cryptographic core, is also seemingly secure against *FuncTeller*. To verify this hypothesis, we choose to test *FuncTeller* on an AES design for 10 rounds with an unknown key. We run *FuncTeller* on the AES circuit with the time limit set at 24 hours per output. As a result, *FuncTeller* achieves an accuracy of 49.99% on AES. When the key is unknown, the AES design is a pseudo-random function. However, we demonstrate *FuncTeller* performance on most practical circuits, such as processor IPs, which constitute a major share of the hardware IP market, according to a recent semiconductor market analysis [44].

7.2 Potential Countermeasures

The threat model of *FuncTeller* assumes that the attacker can obtain the unauthorized scan-chain access by performing attacks [1–3, 6, 7, 22, 23, 51], as described in Appendix A. *Dishonest oracle (DisORC)* [50] protects the black-box design (oracle) and remains unbroken against various attacks. If the attacker tries to enable the scan access, the functionality of the design is corrupted, and hence, the outputs are incorrect. However, DisORC may not sufficiently protect a circuit from recovery by *FuncTeller* because DisORC's effectiveness is limited. The Hamming distance (HD) between the correct output and the output on applying a random key introduced by DisORC is insufficient [50, 58], so the attacker may still use *FuncTeller* to recover functionality, as described in Appendix A. Therefore, a potential countermeasure to defeat *FuncTeller* should consider both strong security of scan-chain access and sufficient HD. Further, a defender can harden the recovery of the redacted designs against *FuncTeller* by carefully selecting and redacting the logic module whose PI distribution profile is similar to that of the AES or c1355. In other words, if the redacted logic PIT has a large number of PIs and literals (e.g., exponential to the input size), the redacted logic is secure against *FuncTeller*, as discussed in Section 7.1. Yet, this may also exponentially increase the hardware size.

8 Conclusion and Ramifications

This paper proposes a heuristic approach, *FuncTeller*, to extract the approximate functionality of the design implemented on an embedded field-programmable gate array (eFPGA). *FuncTeller* exploits the attribute of most practical hardware Boolean functions, where the prime implicants (PIs) are close to each other. *FuncTeller* can effectively recover most practical designs, including benchmark circuits and processors. Significantly, *FuncTeller* achieves $> 90\%$ accuracy on the IBEX processor. However, *FuncTeller* cannot recover pseudo-random functions, such as Advanced Encryption Standard (AES). In this paper, we have focused our discussion solely on the eFPGA platform, as our goal is to challenge the security of eFPGA-based hardware redaction; we plan to extend *FuncTeller* to other hardware platforms such as ASICs, FPGAs, and cloud FPGAs.

Acknowledgement

We thank Cybersecurity Awareness Worldwide (CSAW) 2021 logic locking competition organizers for providing the competition circuits and platform [43]. We thank the members of the TAMU SETH lab for their help in improving the paper. Moreover, we thank the reviewers and Shepherd for providing valuable comments during the reviewing process. The work was supported in part by the National Science Foundation (NSF CNS–1822848 and # 2016650) and the Defense Advanced Research Projects Agency (DARPA) grants (HR0011-20-9-0043 and M2102069) from the Automatic Implementation of Secure Silicon (AISS) program [24] and the Structured Array Hardware for Automatically Realized Applications (SAHARA) program [25]. Any opinions, findings, conclusions, or recommendations expressed herein are those of the authors, and do not necessarily reflect those of the US Government.

References

[1] M. Agrawal, S. Karmakar, et al. Scan Based Side Channel Attacks on Stream Ciphers and Their Counter-Measures. *Cryptology-INDOCRYPT International Conference on Cryptology in India*, pages 226–238, 2008.

[2] S. S. Ali, S. M. Saeed, et al. Novel Test-Mode-Only Scan Attack and Countermeasure for Compression-Based Scan Architectures. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, pages 808–821, 2015.

[3] L. Alrahis, M. Yasin, et al. ScanSAT: Unlocking Obfuscated Scan Chains. *IEEE/ACM Asia and South Pacific Design Automation Conference*, pages 352–357, 2019.

[4] D. Angluin, J. Aspnes, et al. Learning a Circuit by Injecting Values. *Journal of Computer and System Sciences*, pages 60–77, 2009.

[5] Y. Atobe, Y. Shi, et al. Secure Scan Design with Dynamically Configurable Connection. *IEEE Pacific Rim International Symposium on Dependable Computing*, pages 256–262, 2013.

[6] S. Banik, A. Chattopadhyay, et al. Cryptanalysis of the Double-Feedback XOR-Chain Scheme Proposed in Indocrypt 2013. *Cryptology–INDOCRYPT International Conference on Cryptology in India*, pages 179–196, 2014.

[7] S. Banik and A. Chowdhury. Improved Scan-Chain Based Attacks and Related Countermeasures. *Cryptology–INDOCRYPT International Conference on Cryptology in India*, pages 78–97, 2013.

[8] J. Baukus, L. Chow, et al. Method and Apparatus for Camouflaging a Standard Cell based Integrated Circuit with Micro Circuits and Post Processing. *US Patent no. 20120139582*, 2012.

[9] F. Benz, A. Seffrin, et al. Bil: A Tool-Chain for Bitstream Reverse-Engineering. *International Conference on Field Programmable Logic and Applications*, pages 735–738, 2012.

[10] J. Bhandari, A. K. T. Moosa, et al. Not All Fabrics Are Created Equal: Exploring eFPGA Parameters For IP Redaction. *arXiv preprint arXiv:2111.04222*, 2021.

[11] J. Bhandari, A. K. Thalakkattu Moosa, et al. Exploring eFPGA-based Redaction for IP Protection. *IEEE/ACM International Conference On Computer Aided Design*, pages 1–9, 2021.

[12] R. Brayton and A. Mishchenko. ABC: An Academic Industrial-Strength Verification Tool. *International Conference on Computer Aided Verification*, pages 24–40, 2010.

[13] N. H. Bshouty. Exact Learning Boolean Functions via the Monotone Theory. *Information and Computation*, pages 146–153, 1995.

[14] CAD For Assurance. IP Piracy. https://cadforassurance.org/trust-issues/ip-piracy/, 2020. Last accessed on 10/04/2022.

[15] Cadence. Genus Synthesis Solution. https://www.cadence.com/en_US/home/tools/digital-design-and-signoff/synthesis/genus-synthesis-solution.html, 2022. Last accessed on 10/04/2022.

[16] Cadence. Logic Equivalence Checking. https://www.cadence.com/en_US/home/tools/digital-design-and-signoff/logic-equivalence-checking.html, 2022. Last accessed on 10/04/2022.

[17] Cadence. Innovus Implementation System. https://www.cadence.com/en_US/home/tools/digital-design-and-signoff/soc-implementation-and-floorplanning/innovus-implementation-system.html, 2023. Last accessed on 04/18/2023.

[18] P. Chen, Y. Huang, et al. Circuit Learning for Logic Regression on High Dimensional Boolean Space. *ACM/IEEE Design Automation Conference*, pages 1–6, 2020.

[19] Y. Chen and B. Wang. Learning Boolean Functions Incrementally. *International Conference on Computer Aided Verification*, page 55–70, 2012.

[20] A. Chhotaray and T. Shrimpton. Hardening Circuit-Design IP Against Reverse-Engineering Attacks. *Cryptology ePrint Archive*, 2021.

[21] P. Chowdhury, C. Sathe, et al. Predictive Model Attack for Embedded FPGA Logic Locking. *ACM/IEEE International Symposium on Low Power Electronics and Design*, pages 1–6, 2022.

[22] A. Cui, Y. Luo, et al. Why current secure scan designs fail and how to fix them? *Integration*, 56:105–114, 2017.

[23] J. DaRolt, G. Di Natale, et al. Scan attacks and countermeasures in presence of scan response compactors. *IEEE European Test Symposium*, pages 19–24, 2011.

[24] DARPA. DARPA Selects Teams to Increase Security of Semiconductor Supply Chain. https://www.darpa.mil/news-events/2020-05-27, 2020. Last accessed on 10/20/2022.

[25] DARPA. Expanding Domestic Manufacturing of Secure, Custom Chips for Defense Needs. https://www.darpa.mil/news-events/2021-03-18, 2022. Last accessed on 10/20/2022.

[26] S. Davidson. Characteristics of the ITC'99 Benchmark Circuits. *IEEE International Test Synthesis Workshop*, 1999.

[27] Department of Justice. Attorney General Jeff Sessions Announces New Initiative to Combat Chinese Economic Espionage. https://www.justice.gov/opa/speech/attorney-general-jeff-sessions-announces-new-initiative-combat-chinese-economic-espionage, 2018. Last accessed on 10/04/2022.

[28] D. Doy. FPGA Image Processing . https://github.com/damdoy/fpga_image_processing, 2019. Last accessed on 04/18/2023.

[29] M. El Massad, S. Garg, et al. The SAT Attack on IC Camouflaging: Impact and Potential Countermeasures. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, pages 1577–1590, 2019.

[30] O. Glamočanin, D. G. Mahmoud, et al. Shared FPGAs and the Holy Grail: Protections against Side-Channel and Fault Attacks. *IEEE Design, Automation & Test in Europe Conference & Exhibition*, pages 1645–1650, 2021.

[31] R. W. Hamming. Error Detecting and Error Correcting Codes. *The Bell System Technical Journal*, pages 147–160, 1950.

[32] Z. Han, M. Yasin, et al. Does logic locking work with EDA tools? *USENIX Security Symposium*, pages 1055–1072, 2021.

[33] M. C. Hansen, H. Yalcin, et al. Unveiling the ISCAS-85 Benchmarks: A Case Study in Reverse Engineering. *IEEE Design & Test of Computers*, pages 72–80, 1999.

[34] J. Hennessy, N. Jouppi, et al. MIPS: A VLSI Processor Architecture. In *VLSI Systems and Computations*, pages 337–346. Springer, 1981.

[35] M. Hicks, P. Fiscarelli, et al. Common Evaluation Platform v4.2. https://github.com/mit-ll/CEP, 2022. Last accessed on 05/25/2023.

[36] IARPA. Trust Integrated Chips. https://www.iarpa.gov/research-programs/tic, 2011. Last accessed on 10/04/2022.

[37] IC Insights. McClean Report. https://www.icinsights.com/services/mcclean-report/, 2021. Last accessed on 10/04/2022.

[38] F. Imeson, A. Emtenan, et al. Securing Computer Hardware Using 3D Integrated Circuit (IC) Technology and Split Manufacturing for Obfuscation. *USENIX Security Symposium*, pages 495–510, 2013.

[39] Intel. Intel eASICs. https://www.intel.com/content/www/us/en/products/details/easic.html, 2020. Last accessed on 10/04/2022.

[40] R. Jarvis and M. McIntyre. Split Manufacturing Method for Advanced Semiconductor Circuits. *US Patent no. 7,195,931*, 2007.

[41] R. Karmakar, S. Chatopadhyay, et al. Encrypt Flip-Flop: A Novel Logic Encryption Technique For Sequential Circuits. *arXiv preprint arXiv:1801.04961*, 2018.

[42] R. Karmakar, H. Kumar, et al. Efficient Key-Gate Placement and Dynamic Scan Obfuscation Towards Robust Logic Encryption. *IEEE Transactions on Emerging Topics in Computing*, pages 2109–2124, 2019.

[43] R. Karri, B. Tan, et al. CSAW'21 Logic Locking. https://www.csaw.io/logic-locking, 2021. Last accessed on 10/04/2022.

[44] Kenneth Research. Semiconductor Intellectual Property (IP) Market Size, Share, Report-Global Forecase to 2031. https://www.kennethresearch.com/report-details/semiconductor-intellectual-property-ip-market/10070796, 2022. Last accessed on 03/19/2023.

[45] F. Koeune and F. Standaert. A Tutorial on Physical Security and Side-channel Attacks. In *Foundations of Security Analysis and Design III*, pages 78–108. Springer, 2005.

[46] J. Lee, M. Tehranipoor, et al. Securing Scan Design Using Lock & Key Technique. *IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems*, pages 51–62, 2005.

[47] J. Lee, M. Tehranipoor, et al. Securing Designs against Scan-Based Side-Channel Attacks. *IEEE transactions on dependable and secure computing*, 4(4):325–336, 2007.

[48] H. Li, S. Patnaik, et al. Attacking Split Manufacturing from a Deep Learning Perspective. *ACM/IEEE Design Automation Conference (DAC)*, pages 1–6, 2019.

[49] M. Li, K. Shamsi, et al. Provably Secure Camouflaging Strategy for IC Protection. *IEEE transactions on computer-aided design of integrated circuits and systems*, 38(8):1399–1412, 2017.

[50] N. Limaye, E. Kalligeros, et al. Thwarting All Logic Locking Attacks: Dishonest Oracle With Truly Random Logic Locking. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, pages 1740–1753, 2020.

[51] N. Limaye and O. Sinanoglu. Dynunlock: Unlocking scan chains obfuscated using dynamic keys. *IEEE Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 270–273, 2020.

[52] Mentor Graphics. Precision RTL. https://eda.sw.siemens.com/en-US/ic/precision/rtl/, 2022. Last accessed on 10/04/2022.

[53] Mentor Graphics. TrustChain Security Platform. https://eda.sw.siemens.com/en-US/ic/trustchain/, 2022. Last accessed on 10/04/2022.

[54] P. Mohan, O. Atli, et al. Hardware Redaction via Fine-Grained eFPGA. https://eri-summit.darpa.mil/docs/ERIsummit2020/posters/35_OMG_Plaks_CMU_Poster.pdf, 2020. Last accessed on 10/04/2022.

[55] P. Mohan, O. Atli, et al. Hardware Redaction via Designer-Directed Fine-Grained eFPGA Insertion. *IEEE Design, Automation & Test in Europe Conference & Exhibition*, pages 1186–1191, 2021.

[56] M. Oya, Y. Atobe, et al. Secure scan design using improved random order and its evaluations. *IEEE Asia Pacific Conference on Circuits and Systems*, pages 555–558, 2014.

[57] A. Palchaudhuri and A. S. Dhar. Redundant Arithmetic Based High Speed Carry Free Hybrid Adders with Built-In Scan Chain on FPGAs. *IEEE International Conference on High Performance Computing*, pages 104–113, 2017.

[58] J. Rajendran, O. Sinanoglu, et al. Regaining trust in vlsi design: Design-for-trust techniques. *Proceedings of the IEEE*, 102(8):1266–1282, 2014.

[59] Rambus. Circuit Camouflage Technology. https://www.rambus.com/security/cryptofirewall-cores/circuit-camouflage-technology/, 2022. Last accessed on 10/04/2022.

[60] M. Renovell, P. Faure, et al. IS-FPGA: A New Symmetric FPGA Architecture with Implicit SCAN. *IEEE International Test Conference 2001*, pages 924–931, 2001.

[61] A. Rezaei, R. Afsharmazayejani, et al. Evaluating the Security of eFPGA-Based Redaction Algorithms. *IEEE/ACM International Conference on Computer-Aided Design*, pages 1–7, 2022.

[62] J. A. Roy, F. Koushanfar, et al. EPIC: Ending Piracy of Integrated Circuits. *IEEE/ACM Design, Automation and Test in Europe*, pages 1069–1074, 2008.

[63] I. Ruczinski, C. Kooperberg, et al. Logic Regression. *Journal of Computational and Graphical Statistics*, pages 475–511, 2003.

[64] G. Sengar, D. Mukhopadhyay, et al. Secured Flipped Scan-Chain Model for Crypto-Architecture. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 26(11):2080–2084, 2007.

[65] D. Sirone and P. Subramanyan. Functional Analysis Attacks on Logic Locking. *IEEE Design, Automation Test in Europe Conference Exhibition*, pages 936–939, 2019.

[66] W. Snyder. Verilator. https://verilator.org, 2021. Last accessed on 10/04/2022.

[67] P. Subramanyan, S. Ray, et al. Evaluating the Security of Logic Encryption Algorithms. *IEEE International Symposium on Hardware Oriented Security and Trust*, pages 137–143, 2015.

[68] Synopsys. Design Compiler NXT. https://www.synopsys.com/implementation-and-signoff/rtl-synthesis-test/design-compiler-nxt.html, 2022. Last accessed on 10/04/2022.

[69] Synopsys. Formality Equivalence Checking. https://www.synopsys.com/implementation-and-signoff/signoff/formality-equivalence-checking.html, 2022. Last accessed on 10/04/2022.

[70] J. Szefer. Survey of Microarchitectural Side and Covert Channels, Attacks, and Defenses. *Journal of Hardware and Systems Security*, pages 219–234, 2019.

[71] X. Tang, E. Giacomin, et al. OpenFPGA: An Opensource Framework Enabling Rapid Prototyping of Customizable FPGAs. *International Conference on Field Programmable Logic and Applications*, pages 367–374, 2019.

[72] The Commission on the Theft of American Intellectual Property. The IP Commission Report. https://www.nbr.org/wp-content/uploads/pdfs/publications/IP_Commission_Report.pdf, 2013. Last accessed on 10/04/2022.

[73] K. Tiri and I. Verbauwhede. A Logic Level Design Methodology for a Secure DPA Resistant ASIC or FPGA Implementation. *IEEE Design, Automation and Test in Europe Conference and Exhibition*, pages 246–251 Vol.1, 2004.

[74] A. Tiwari and K. A. Tomko. Scan-chain Based Watch-Points for Efficient Run-Time Debugging and Verification of FPGA Designs. *IEEE Asia and South Pacific Design Automation Conference.*, pages 705–711, 2003.

[75] R. Torrance and D. James. The State-of-the-Art in IC Reverse Engineering. *International Conference on Cryptographic Hardware and Embedded Systems*, pages 363–381, 2009.

[76] I. Tuomi. The Future of Semiconductor Intellectual Property Architectural Blocks in Europe. *European Communities*, 2009.

[77] Z. UlAbideen, T. D. Perez, et al. A Security-aware and LUT-based CAD Flow for the Physical Synthesis of eASICs. *arXiv preprint arXiv:2207.05413*, 2022.

[78] A. Varas, R. Varadarajan, et al. Strengthening the Global Semiconductor Supply Chain in an Uncertain Era. https://www.semiconductors.org/wp-content/uploads/2021/05/BCG-x-SIA-Strengthening-the-Global-Semiconductor-Value-Chain-April-2021_1.pdf, 2021. Last accessed on 10/04/2022.

[79] H. Wang, D. Forte, et al. Probing Attacks on Integrated Circuits: Challenges and Research Opportunities. *IEEE Design & Test*, 34(5):63–71, 2017.

[80] Y. Wang, P. Chen, et al. The Cat and Mouse in Split Manufacturing. *ACM/IEEE Design Automation Conference*, pages 1–6, 2016.

[81] L. Woods, Z. István, et al. Ibex: An Intelligent Storage Engine with Support for Advanced SQL Off-loading. *Proceedings of the VLDB Endowment*, pages 963–974, 2014.

[82] Y. Xie and A. Srivastava. Anti-SAT: Mitigating SAT Attack on Logic Locking. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, pages 199–207, 2019.

[83] Xilinx. Vivado Design Suite User Guide: Programming and Debugging. https://docs.xilinx.com/r/en-US/ug908-vivado-programming-debugging/Generating-Encrypted-and-Authenticated-Files-for-UltraScale-and-UltraScale, 2021. Last accessed on 10/04/2022.

[84] B. Yang, K. Wu, et al. Scan Based Side Channel Attack on Dedicated Hardware Implementations of Data Encryption Standard. *IEEE International Conference on Test*, pages 339–344, 2004.

[85] B. Yang, K. Wu, et al. Secure Scan: A Design-for-Test Architecture for Crypto Chips. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, pages 2287–2293, 2006.

[86] M. Yasin and O. Sinanoglu. Transforming Between Logic Locking and IC Camouflaging. *IEEE International Design & Test Symposium*, pages 1–4, 2015.

[87] J. Yoon, Y. Seo, et al. A Bitstream Reverse Engineering Tool for FPGA Hardware Trojan Detection. *ACM SIGSAC Conference on Computer and Communications Security*, page 2318–2320, 2018.

Appendix

A Scan-Chain Access for FPGA Designs

Similar to scan chains in application-specific integrated circuits (ASICs), scan chains can be utilized in field-programmable gate arrays (FPGAs), as shown in Figure 10. Scan chains allow arbitrary test patterns to be loaded into flip-flops on an FPGA [57, 60, 74]. Note that testability is important for embedded FPGAs (eFPGAs) as they are integrated with an ASIC design.

Usually, scan chains are not open to end-users or attackers. Techniques, such as flipped scan [64], XOR scan [1], double feedback XOR scan [7], and sub-chains based scan [5, 46, 47, 56], protect scan chains and restrict their access; however, they are broken by attacks, including Mukesh *et al.* [1], Subhadeep *et al.* [7], Banik *et al.* [6], and Cui *et al.* [22]. Some techniques disable access to scan chains using AES algorithm [85]—however, such AES-based techniques are vulnerable to attacks, including DaRolt *et al.* [23] and Ali *et al.* [2]. Some other scan-chain protections were developed using the concept of logic locking, such as Encrypt Flip-Flop [41] and dynamic scan obfuscation (DynScan) [42]. Yet, these scan-chain protections are vulnerable to attacks, such as ScanSAT [3] and DynUnlock [51]. In this case, the attacker is able to obtain unauthorized scan-chain access to perform *FuncTeller* after the testing stage. Currently, there are secure scan-chain techniques robust to all existing attacks, such as *dishonest oracle* (*DisORC*) [50]. However, DisORC may not sufficiently protect a circuit from recovery by *FuncTeller* because DisORC's effectiveness is limited. For example, the Hamming distance (HD) between the correct output and the output on applying a random incorrect key is 11.30% for b20 in DisORC [50], but the ideal HD should be 50% on average [58].

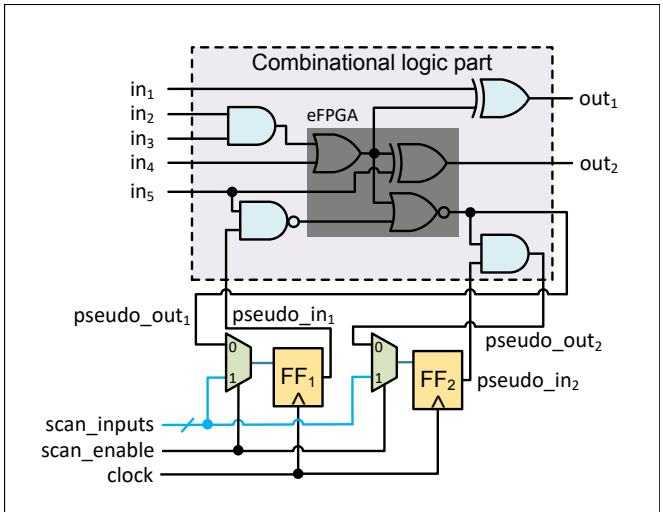

Figure 10: The eFPGA redacted design with the scan chain.

Algorithm 4: Scalable PI expansion

Input: O, w, m_0, PIT_{pred}, p, p_0
Output: PI_{pred}

Function `get_hard_dc`(m_0, PIT_{pred}):
  $hard\_dc := \varnothing$
  for $index \in \{1, \cdots, len(m_0)\}$ **do**
    $hard\_dc\_flag := is\_dc\_always(index, PIT_{pred})$
    if $hard\_dc\_flag == True$ **then**
      $hard\_dc := hard\_dc \bigcup \{index\}$
return $hard\_dc$
Function `get_iter_limit`($cube$, $index$, $hard\_dc$, p, p_0):
  $num\_dc := cube.count(\text{“-”})$
  if $index \in hard\_dc$ **then**
    $iter\_limit := min\{2^{num\_dc}, p_0\}$
  else
    $iter\_limit := min\{2^{num\_dc}, p \times num\_dc\}$
return $iter\_limit$
Function `expand_scalably`(O, w, m_0, p, p_0):
  $hard\_dc :=$ `get_hard_dc`(m_0, PIT_{pred})
  $cube := m_0$
  for $index \in \{1, 2, \cdots, len(m_0)\}$ **do**
    $iter\_limit :=$ `get_iter_limit`($cube$, $index$, $hard\_dc$, p, p_0)
    $cube :=$ `update_cube`(O, w, $cube$, $index$, $iter\_limit$) ▷ `update_cube()` in Algorithm 1
  $PI_{pred} := cube$
return PI_{pred}

B Scalably Expanding from Minterm to PI

Algorithm 4 provides a solution for scalable expansion from an ON-set minterm to a predicted prime implicant (PI) compared to Algorithm 1. In Algorithm 4, The time taken in determining a PI is proportional to the number of don't cares. The non-effective inputs are always don't cares in all the PIs. We refer to these as hard don't care (*hard_dc*) bits. To reduce the time taken in determining a potential don't care bit, the number of queries on the hard_dc bit is limited to a constant value p_0, where $p_0 < p \times num\_dc$ in most cases. We consider p_0 as the constant limitation parameter in Algorithm 4. Suppose, out of n inputs, there are n_r effective inputs ($n_r < n$). In the worst case, the complexity is $O(n_r^2)$, which is less than $O(n^2)$. Thus, Algorithm 4 has higher efficiency than Algorithm 1 since $O(n_r^2) \ll O(n^2)$ when $n_r \ll n$.

C Analysis of Parameters in *FuncTeller*

Time Limit T. With greater T, *FuncTeller* can achieve a higher prediction accuracy because the total number of PIs increases. **Linear Limitation Parameter p.** In the worst case, the number of queries in each PI expansion is $\sum_{i=1}^{n} p \times i = p \times \frac{n^2+n}{2}$, where n is the number of inputs. Thereby, in the worst case, the time complexity of each PI expansion is $O(n^2)$. **Constant Limitation Parameter $\mathbf{p_0}$.** Suppose the PI expansion uses Algorithm 4. Assume a predicted PI has n_r effective inputs. In the worst case, the number of queries in the PI expansion is $\sum_{i=1}^{n_r} p \times i + \sum_{i=1}^{n-n_r} p_0 = p \times \frac{n_r^2+n_r}{2} + p_0 \times (n - n_r)$. Thus, the time complexity of the scalable PI expansion is $O(n_r^2)$. When $n_r \ll n$ (most inputs are non-effective inputs), the acceleration of the PI expansion process (comparing Algorithm 4 to Algorithm 1) is huge since $O(n_r^2) \ll O(n^2)$. **Distance Parameter $\mathbf{d_0}$.** In our simulations, we choose $d_0 = 2$ based on a case study of distance distribution on IBEX [81], as described in Section 5.4. **Convergence Parameter $\mathbf{p_{conv}}$.** Let c be the confidential probability of the following event: *FuncTeller consecutively visits p_{conv} OFF-set minterms.* Suppose r^{ON} is the ratio of the ON-set minterms in the search space for the next ON-set minterm. Thus, $c = (1 - r^{ON})^{p_{conv}}$. Further, if we consider p_{conv} as a fixed parameter, $r^{ON} = 1 - c^{\frac{1}{p_{conv}}}$. In our simulations, we choose $p_{conv} = 50$. If this event happens, then $10\% < c \leq 100\%$ implies $0 \leq r^{ON} < 4.5\%$. Thus, after consecutively visiting OFF-set minterms for $p_{conv} = 50$ times during the search for the next ON-set minterm, we conclude that ON-minterms are rare in the current search space since r^{ON} is negligibly small.

D FPGA Hardware Implementation

We construct an FPGA-based prototype to mimic an eFPGA by solving two challenges: (i) **High communication slack** between *FuncTeller* and oracle affects the attack time significantly. To reduce this communication slack, we implement *FuncTeller* and oracle on the same SoC platform. (ii) **Lack of parallelization:** To match the attack time of the software-based results of *FuncTeller*, we implement multiple oracles on the same FPGA. The number of oracles on the FPGA is limited by resource constraints.

Implementation. We attack the circuits in Table 2 using the FPGA prototype shown in Figure 11, using the PYNQ-Z2 board. The PYNQ-Z2 board contains a Zynq-7000 series SoC platform containing both an FPGA (performing oracle) and a processor (running *FuncTeller*). We use the Xilinx AXI4 interface [83] to reduce communication slack. Further, for larger designs, we repeat this process for multiple boards. Finally, all PITs are sent to a remote computer.

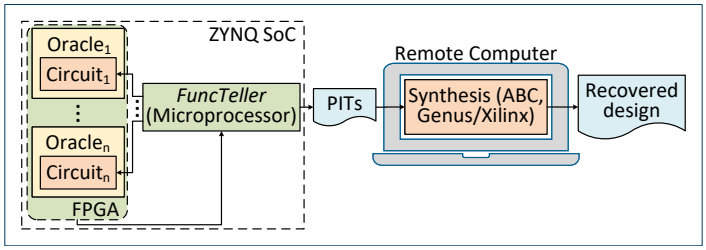

Figure 11: FPGA prototype setting.

Results and Analysis. We empirically observe that the attack needs approximately twice as long to recover the design with the same accuracy compared to the simulation-based version. These results on selected circuits are shown in Table 4. While the software simulation runs on an Intel Xeon processor running at 2.6GHz, our hardware implementation is on a 650MHz dual-core Cortex-A9 processor. This leads to a degradation in performance. Thus, the time taken per query is approximately 2× the time taken per query in the simulation-based setting, as there is a software overhead while generating new queries. In this case, we are limited due to SoCs available. Note that the three circuits (b20, IBEX, and GPS) in Table 2 are not included in Table 4. Due to the FPGA resource constraints, the extraction of these designs was not complete. To further explain this, we present a scalability analysis for these circuits.

Scalability Analysis. On the one hand, the circuits (IBEX, GPS, and b20) excluded from Table 4 have a large number of input/output ports; on the other hand, the interfaces available on the SoC platform are limited. As a result, the number of oracles that can be implemented on the SoC is limited, and consequently, *FuncTeller* cannot be completed. For instance, we can only implement 5 IBEX oracles on a single SoC. This can be overcome by parallelizing the attack on multiple SoCs to recover the entire design. Due to the lack of such resources, we are unable to parallelize to the extent possible in software (50 outputs in parallel). Next, we estimate *FuncTeller*'s performance if additional SoCs were available.

Table 5 shows the estimated time needed to match the simulation-based accuracy if the number of oracles is 50. Recall that the number of oracle implementations on a single SoC is limited by the number of AXI4 interfaces. Thus, more SoCs are needed to parallelize the attack to at least 50 oracles. We derive the time estimate in Table 5 based on empirical observations and validate these observations based on tests for individual outputs. As stated previously, *FuncTeller* needs approximately twice as long to recover the design if the number of oracles is the same. This is due to the communication slack being 2× higher than the software-based implementation of *FuncTeller*. Notably, the communication slack is only 2× higher while the Cortex-A9 processor is approximately 3× slower compared to the Intel Xeon processor used for simulation. Based on this analysis, we can conclude that powerful SoCs which have a large number of AXI interfaces and powerful onboard CPUs can effectively recover large-scale redacted designs as *FuncTeller* can scale to larger designs depending on the hardware platforms used.

E Results on CSAW Benchmarks

Table 6 shows *FuncTeller*'s performance on the FPGA-based circuits from Cybersecurity Awareness Worldwide (CSAW) 2021 logic locking event. For all circuits across the main benchmark sets, *FuncTeller* has an accuracy of $> 90\%$.

Table 4: The results of *FuncTeller* on the FPGA prototype.

Circuit	# oracles	Attack time (hours)	Accuracy (%)
c432	7	0.16	99.26
c880	26	1.96	95.23
c1355	32	2.34	52.33
c1908	25	2.21	79.43
c7552	46	2.53	88.02
b14	107	1.97	90.34
MIPS	75	2.63	93.24

Table 5: Estimated time and resources for *FuncTeller* to match the simulation-based accuracy on FPGAs.

Circuit	Max. # oracles on each FPGA	# FPGAs required	Estimated time (hours)
b20	15	4	20
IBEX	5	10	144
GPS	1	50	88

Table 6: Comparison of performance between *FuncTeller* and Chen *et al.* [18] on CSAW 2021 competition circuits [43].

Circuit		# inputs	# outputs	Accuracy (%)	
				Chen *et al.* [18]	*FuncTeller*
MAIN	set1_1	94	90	83.11	100.00
	set1_2	98	94	83.33	100.00
	set1_3	102	98	78.40	100.00
	set1_4	106	102	78.29	100.00
	set1_5	110	106	76.73	99.68
	set1_6	114	110	75.78	99.68
	set1_7	118	114	78.19	99.71
	set2_1	94	90	84.26	100.00
	set2_4	106	102	76.34	99.81
	set2_5	110	106	75.82	98.63
	set2_6	114	110	76.71	99.41
	set2_7	118	114	78.47	98.51
	set3_1	94	90	83.34	99.82
	set3_2	98	94	82.92	99.70
	set3_3	102	98	77.57	99.79
	set3_4	106	102	76.39	98.90
	set3_5	110	106	75.27	99.17
	set3_6	114	110	76.70	99.08
	set3_7	118	114	78.84	98.80
	set4_1	94	90	82.24	100.00
	set4_2	98	94	81.97	100.00
	set4_3	50	46	92.36	100.00
	set4_4	186	182	65.30	99.42
	set4_5	110	106	76.75	99.44
	set4_6	114	110	75.82	99.47
	set4_7	118	114	77.99	99.20
BONUS	set1	198	194	63.15	95.84
	set2	214	210	62.71	96.13
	set4	246	242	66.42	96.23
	set5	262	258	67.79	96.74
	set6	278	274	67.21	96.65
	set7	294	290	67.83	96.62