A practical application of game theory to optimize selection of hardware Trojan detection strategies

Jonathan Graf 1,2 · Whitney Batchelor 1 · Scott Harper 1 · Ryan Marlow 1 · Edward Carlisle IV 1 · Peter Athanas 2

Received: 16 April 2019 / Accepted: 4 December 2019 © The Author(s) 2019

Abstract
A wide variety of Hardware Trojan countermeasures have been developed, but less work has been done to determine which are optimal for any given design. To address this, we consider not only metrics related to the performance of the countermeasure, but also the likely action of an adversary given their goals. Trojans are inserted by an adversary to accomplish an end, so these goals must be considered and quantified in order to predict these actions. The model presented here builds upon a security economic approach that models the adversary and defender motives and goals in the context of empirically derived countermeasure efficacy metrics. The approach supports formation of a two-player strategic game to determine optimal strategy selection for both adversary and defender. A game may be played in a variety of contexts, including consideration of the entire design lifecycle or only a step in product development. As a demonstration of the practicality of this approach, we present an experiment that derives efficacy metrics from a set of countermeasures (defender strategies) when tested against a taxonomy of Trojans (adversary strategies). We further present a software framework, GameRunner, that automates not only the solution to the game but also mathematical and graphical exploration of “what if” scenarios in the context of the game. GameRunner can also issue “prescriptions,” a set of commands that allows the defender to automate the application of the optimal defender strategy to their circuit of concern. Finally, we include a discussion of ongoing work to include additional software tools, a more advanced experimental framework, and the application of irrationality models to account for players who make subrational decisions.

Keywords Hardware Trojan horse · Trust · Risk · Game theory · Optimization

1 Introduction

Reasoning about hardware Trojan horse (HTH) detection strategies requires considering a more complex set of influences than those treated in traditional circuit verification practice. In addition to traditional concerns about the coverage, a verification method might accomplish with respect to some defect, the relationship between the creator of the Trojan (the adversary) and the developer of Trojan detection methods (the defender) is governed by strategies, incentives, and creativity. While an undetected defect (e.g., a manufacturing flaw or an implementation bug) being searched for by a traditional verification method may cause outcomes as grave as those caused by an HTH, there is no guiding intelligence making rational choices about how to optimally produce those outcomes. Rather, we often assume they occur randomly, and this assumption is then built into the requirements of verification techniques to discover the subject defect. For example, when pursuing optimal coverage of stuck-at-0/1 or bridging faults [1], we may treat their occurrence in different circuit regions as more or less equivalently likely. Questions of detection approach optimality collapse into an overall circuit coverage metric. With all other costs constant, a method that accomplishes the best overall circuit coverage is the most preferred.

Conversely, with HTH detection, the adversary does have the opportunity to make optimal choices about whether, which, and where to use an HTH to accomplish their goals. When considering how to optimally defend against such adversaries, we must consider more than rudimentary metrics about defensive solutions such as likelihood of detecting a Trojan. We must also consider the adversary’s desired outcome, their strategies for accomplishing it, and the resources they have at their disposal to expend in pursuit of it. As we
will see, in many cases, the intuitions derived from verification best practices can fail in the face of an informed adversarial attack. It will be shown that the HTH detection methods with both the highest probability of detection and the lowest false alarm rates are not always the optimal play for defenders that face a rational adversary who is simultaneously optimizing their HTH strategy. In fact, we will see that high-performing detection methods can fool the defender into creating a Maginot Line of defenses in an arrangement that guarantees only that the adversary will attack in a manner for which they have not prepared [2].

Given this challenge, a framework for optimal decision making in this adversary-defender interaction is necessary. Our previous work in [3–7] suggested the use of game theory to produce the requisite formal, reliable, neutral, and automated framework for quickly optimizing defender strategies. This paper updates and extends that work by refining the game variables and probabilities; demonstrating empirical means of populating the probability values involved; performing an experiment on an example dataset; and creating software to automate this experimentation, analyze and visualize its results, and automate the user response.

1.1 Contribution summary

The transformative quantifiable assurance approach introduced in this paper provides the following contributions:

1. The introduction of security economic models that incorporate the efficacy of FPGA Trojan detection methods and the incentives of both the adversary and defender in hardware Trojan encounters.
2. The introduction of a simple two-person strategic game theoretic model that leverages the aforementioned security economic models and the Nash equilibrium solution concept.
3. An automated method, embodied by software entitled GameRunner, of determining and applying optimal detection strategies for hardware Trojan detection, based on the models of Contributions 1–2.
4. Novel visualization techniques which allow for the exploration of the investment-security state space within the context of complex adversary-defender engagements.

1.2 Paper organization

This paper is organized as follows. In Sect. 2, we briefly treat the background topic of hardware Trojans and detection methods and more completely treat the background of game theory in the context of HTH scenarios. In Sect. 3, we discuss our rationale for the construction of games and the selection of game solution concepts, including updated insights on correct game construction that have emerged from our recent empirical applications of past game constructions. In Sect. 4, we present the experiment conducted to demonstrate the practical value of the game, illustrating the construction of player strategies, the use of game variables, and the automated production of results. In Sect. 5, we discuss the results of this experiment as well as the software we created for exploring game solutions and directing defensive responses. Section 6 discusses applications and new directions that are presently emerging from our ongoing work. We conclude in Sect. 7.

2 Background

HTH design and detection are the subjects of a large and maturing literature. We briefly introduced the area in Sect. 2.1. Less work has been done on the use of game theory in hardware security contexts, but there have been some significant new advancements since our publication of [3], which we believe to be the first examination of hardware security using game theory tools. Section 2.2 summarizes the emerging literature of game theory in hardware security.

2.1 Hardware Trojan design and detection

In the last 15 years, hardware Trojan design and detection have been well studied and characterized [8]. Several taxonomies [9, 10] have been published that sort HTH’s into categories based on the step in a product lifecycle at which they are inserted, their activation mechanism, their effects, their location, and their physical characteristics. Similarly, a maturing literature reveals a wide variety of countermeasures for design-time Trojan detection via behavioral functional validation, formal verification, side-channel analysis, or varieties of structural analysis. Countermeasures may also be employed that seek to make it difficult to insert a Trojan in the first place or make it easier to detect Trojans in running devices [8].

For the purposes of our work in this paper, we need only recognize that it is possible to arrange Trojans into categories that have similar characteristics that allow their analysis in groups. The production of an optimal taxonomy is a worthy and ongoing research topic. For this paper, it suffices to say that we may group Trojans and countermeasures into categories and that the Trojan and countermeasure effects can be measured according to the metrics used in our game. What we illustrate with our theory and experimentation is that given a Trojan taxonomy, a set of adversary Trojans that fit within that taxonomy, and a set of defender countermeasure strategies, our game theoretic formulation provides value in guiding defender decision making.
2.2 Game theory and hardware Trojan detection

Game theoretic methods have been used to develop solutions to various cyber-security problems [11]. Among these approaches, there are now several that have modeled the adversary and defender as opponents in a two-player strategic game for the purpose of microelectronic device security. To our knowledge, the first publication to apply game theory to microelectronics security was our work in [3], which modeled the opponents similarly to how we suggest here but does so not for hardware Trojans but for an adversary wishing to tamper with deployed devices and a defender wishing to prevent such tamper. The first paper that applied 2-person strategic games to model hardware Trojan adversary and defender relationships was [12]. As compared to the work presented here, Kamhoua et al. [12] does so with a simplified model of the security economics of the adversary and defender. Specifically, the authors base their game outcomes on the specification of a fee that is paid by the adversary to the defender in the instance that a hardware Trojan is detected. This arbitrary fee avoids treating the constituent elements of adversary and defender security economics and limits the conclusions to the few situations where such a direct fee payment could be envisioned. In our work, as first presented in [4], we explain how to construct utility functions that model the concerns expressed by opponents in more realistic adversary/defender interaction. These utility functions may be used in conjunction with game theory to guide system designers in selecting optimal sets of existing detection methods. In [5], we extended the models from [4] to demonstrate that our framework is useful not only in selecting from among existing Trojan detection methods but also in guiding developers of new ones. In [7], we theorized about a software framework (an implementation of which is presented in this work) to automate the production of the variables in the game. In [6], we formulated system-level games using similar construction, allowing HTH threats to be considered simultaneously with threats to other components (bitstream, software) of a multi-processor system on chip device.

Since the publication of [4, 5], several related contributions have been made. Kamhoua et al. [13] extends the formulation of [12], leveraging a taxonomy similar to ours in [4, 5]. However, it continues to model the economic interaction between the adversary and defender in an oversimplified manner and lacks a means of connecting empirical test results to the games. It also uses a zero-sum game. A zero-sum game assumes the gain or loss of one player is exactly balanced by the loss or gain, respectively, of the other. While this simplifying assumption leads to conveniences in analyzing the game, this assumption does not hold as a correct description of the HTH adversary and defender. Rather, this artificial balancing of the economics of each player sidesteps the treatment of the independent constituent variables in each player’s desired outcome. Collectively, these decisions in setting up the game remove it from the reality of defender/adversary interactions and disqualify its conclusions from guiding industry-level decisions. Smith et al. [14] develops a game theoretic formulation for Trojan detection, but it is a zero-sum game that concerns itself not with multiple independent Trojan strategies but with one that optimally selects netlist regions most vulnerable to attack and sets the adversary’s goal to be maximal defender loss. As we will see, our work creates a scenario where adversary gain and defender loss are independent, which is often the case in asymmetric environments, and this allows us to consider multiple independent adversary and defender strategies and countermeasures, as opposed to just one. Galiardi et al. [15] expands the example in [14] by placing it in the context of software to secure the design workflow. Related to both our work and [12, 13], the work presented in [16] attempts to model less-than-perfectly rational players through the use of prospect theory, where player preferences are shaped by a subrational behavior model prior to the application of the Nash equilibrium solution concept. As such, it represents an initial construction of the security economics of HTH interactions with the security psychology of the players. However, the game played has a model of adversary/defender interaction that is limited, similar to [12]. In it, defender preferences related to HTH attacks are indexed by an undefined degree to which they are afraid of the Trojan and the defender is artificially constrained to choose two from among an available six detection strategies. Our utility functions model the adversary and defender motives with greater fidelity. Our approach is to address rational adversary/defender models first, then, later, we will revisit subrational behavior.

There are a few projects outside the realm of game theory whose outcomes can be related to portions of our work. Some use statistical models to consider optimization concepts, such as [17], which attempts to generally identify cyber vulnerabilities and quantify their impact, using the resulting values to prioritize the order in which they are addressed. While it is focused on software vulnerabilities, an extension to hardware may be possible. Other work, such as [18], attempts to quantify the costs related to error payloads and implementation. While this is not used in a game theoretic formulation, it could have use in an alternative formulation of utility functions.

3 Security economics and game theory

Game theory can tell us not only what the adversary is capable of but also what action they are reasonably going to take. Before we can approach what the adversary and defender might do in a given interaction with each other, however, we must first define their individual interests. Here, we turn to security economics, which merges its two eponymous fields
into a study of how incentives govern the behavior of actors in security-related interactions [19]. Thus, our first security economic task is to explore the value of an HTH and the ability to insert and to detect it from the perspectives of both our opponents. We do so by defining utility functions that algebraically represent the beliefs by which our players make their decisions. The resolution of a player’s utility function given values for each constituent variable gives us that player’s payoff under those conditions. Our adversary and defender utility functions draw from and extend [20], a model that considers how one can improve their security situation through additional investment. Specifically, in the selection of an HTH or a method to defend against it, we create utility functions that consider the improvement in the payoff of both of our players.

### 3.1 Players

The players in our game are an adversary, who is seeking to accomplish a gain by placing an HTH in a circuit, and a defender, who is seeking to avoid the loss that would take place if the HTH is placed and successful. The gain of the adversary and the loss of the defender are not symmetrical.\(^1\) One can easily imagine a scenario where an adversary HTH might cost a large business or government enormous resources in reaction to the HTH’s effect but the adversary only captures some small portion of the lost value. A core assumption in traditional game theory is that the players are rational and that they select strategies that optimally realize their self-interest. Their self-interests are represented in utility functions, which may be said to capture their beliefs. As stated in the previous section, we use make of this assumption in order to devise purely rational solutions. That is, we model player beliefs as comporting exactly with rationality; players with irrational beliefs are presently under consideration in related work.

### 3.2 Collecting variables of concern

To define the opponents’ utility functions, we must account for human elements (strategies and incentives) as well as empirically derived efficacy measures for both Trojans and Trojan-detection methods. This section explores both.

#### 3.2.1 Strategies

Each player has at their disposal a set of strategies. The defender has \(n\) available strategies in the set \(S_D^n = \{\sigma_{D0}, \sigma_{D1}, \cdots, \sigma_{Dn}\}\). Within this set, \(\sigma_{D0}\) carries special significance: it represents a baseline strategy of electing to do nothing beyond their planned verification practice to specifically target hardware Trojans. We have designed the game in this way to recognize that most parties developing devices and systems already have verification practices, which will have a varying degree of impact upon hardware Trojan detection. For our model to be accurate, we must first characterize the impact \(\sigma_{D0}\) has on hardware Trojan detection. The choice the defender is optimizing in this game is the optimal additional investment to address the Trojan threat. Defender strategies \(\sigma_{D1}, \cdots, \sigma_{Dn}\) are HTH countermeasures. In this paper, the countermeasures we consider are detection methods that discover the Trojan during the design process; however, alternative countermeasures can be considered with the same or similar game formulations. Each strategy may itself be one detection method, a set of detection methods, or even a specific sequence of detection methods.

For the adversary, we represent their \(m\) available strategies as the set \(S_A = \{\sigma_{A0}, \sigma_{A1}, \cdots, \sigma_{Am}\}\). Their baseline strategy, \(\sigma_{A0}\) represents electing to do nothing further. We say further because the adversary will have made a prior investment simply to get to the point of being able to make a strategic decision about an attack method. This baseline concept will be treated more when we discuss our game scenarios. The other strategies \(\sigma_{A1}, \cdots, \sigma_{Am}\) represent an attack of one kind or another. In upcoming sections, we will see that for our current experiment, selecting an adversary strategy means selecting the appropriate category of hardware Trojan from a taxonomy of available Trojans.

Our game is adaptable to accommodate any defensive strategy that can be measured with the efficacy probabilities and economic concerns of the next sections. The probabilities and costs that compose our game are based on the selection of strategies from sets \(S_A\) and \(S_D\).

#### 3.2.2 Probabilities

The probabilities in our game are meant to be empirically derived. In this way, our game can be updated quickly, quantitatively, and without human bias when innovations in either adversary or defender strategies are discovered. An inspiration was the set of probabilities described by [21], which formed the basis for the empirical metrics used in the DARPA TRUST program [22, 23]. We have adapted and extended those probabilities for specific use in our utility functions. We make use of the following set.

\[
P_D(\sigma_A, \sigma_D): \text{The probability of detection. This represents the probability that the selected defender strategy, } \sigma_D, \text{ will detect the selected attacker strategy, } \sigma_A. \text{ We treat this as a continuous probability in } [0, 1].
\]

\[
P_{FA}(\sigma_D): \text{The probability of false alarm. This is the probability that the selected defender strategy, } \sigma_D, \text{ will incorrectly indicate an attack has taken place.}^2 \text{ This is also a continuous probability in } [0, 1].
\]

\(^1\) Thus, our emphasis that this is not a zero-sum game.

\(^2\) For the purposes of our paper, we have described the false alarm rate as dependent only on the selected defensive strategy. In related work, we are considering whether false alarm rates have a dependency on the adversary strategy as well as whether producing false alarms in the defensive method may itself be an adversary strategy.
The probability of success. This represents the likelihood that the adversary HTH, \(\sigma_A\), will work given that it is inserted. This is a continuous probability. As we will see, for the purposes of the experiments in the next section, we use only a discrete probability of 0 or 1 for this value. As will be seen in our utility functions, we use it in this manner to be able to eliminate the possibility of the adversary accomplishing a gain from a hardware Trojan attack in the instance they elect not to attack. That is, for our use in these experiments, \(P_A(\sigma_A) = 0\) and \(P_A(\sigma_A) = 1\) for all other strategies \{\(\sigma_{A1}, \ldots, \sigma_{An}\)\}. Alternatively, we could have elected to use \(P_A\) as a continuous function, but all games evaluated to date have been dominated by other factors. The above probabilities are the same as those we used in our past work.

\[P_{ATT}(\sigma_A, \sigma_D):\] The probability of attribution. This represents the likelihood that the given adversary strategy, \(\sigma_A\), can be attributed to the adversary in the instance the circuit is detected by defender strategy, \(\sigma_D\). \(P_{ATT}\) is new in this paper, as our past work ignored this value. We use this value to account for the fact that the adversary experiences no punitive costs unless their hardware Trojan is not only discovered but also attributed to them. Indeed, if the hardware Trojan circuit is discovered but not attributed, the adversary does not obtain the full value they sought to gain, but they lose much less if they are detected but not truly “caught” through attribution. Attribution is itself a challenge in cybersecurity—both technically and legally (see for example [24]). Thus, our utility functions must account for it.

For our game to be useful—moving beyond purely theoretical formulations to something that can provide meaningful guidance to industry—it is critical that reliable sets of these probabilities can be derived empirically. The experiments in this paper attempt to demonstrate how \(P_D(\sigma_A, \sigma_D)\) and \(P_{FA}(\sigma_D)\) can be produced for an example encounter and used to determine optimal decision making.

### 3.2.3 Economic variables

The economic variables of our game equation represent the monetized values of the motives of the adversary and defender (that is, what they stand to gain or lose by playing the game) as well as the costs incurred by making various strategic choices. In a few ways, the adversary and defender choices mirror each other, since each are setting a strategy with associated costs and rewards. However, the game is asymmetrical: the economics of each player differ leading to a game that is not zero sum. We first consider the costs of the defender, then those of the adversary.

\(L\): The defender’s financial loss if the adversary is successful. Our game is set up around the concept that the defender is attempting to protect themselves from loss. That is, the product they are producing (the circuit or device they are developing) already has some market value which would be diminished in the case that an adversary was able to weaken that value. In commercial markets, this value might be straightforwardly associated with the economics of the product’s target market. In national-defense situations, the value may be a strategic advantage, which would have to be monetized. For a safety-related product, there may be some value of human life that should be considered.\(^3\)

\(Z_D(\sigma_D):\) The direct cost of implementing a defensive strategy, \(\sigma_D\). This cost is measured in materials and time. That is, \(Z_D\) does not only include the cost of purchasing a specialized tool. To properly account for a selected defensive method, we must consider the defender-incurred labor costs of creating, implementing, and executing the selected method as well as interpreting and verifying the results. If there is a purchase price for the tool, that should be included in the material cost along with the cost of computing power and the cost of tradeoffs (if any) in the area/power/timing attributes of the design under test. Advanced analysis may consider the time-to-market advantage missed as a cost of slowing product development for the purpose of HTH detection.

\(Z_{FA}(\sigma_D):\) The cost of resolving false alarms for a given method, \(\sigma_D\). If a method were to claim an HTH is present, an effort would be undertaken to attempt to determine if it is a false alarm or an actual HTH. If it is determined to be a false alarm, the situation for the defender is not as dire as it would be if an HTH were detected; however, costs are still incurred to diagnostically that the HTH detection method was incorrect in its determination. In some cases, this cost might be low, such as if a Hardware Description Language (HDL) bug is detected as an HTH and the bug is simply removed. Higher costs might be incurred in cases where the selected strategy produces copious amounts of false alarm data that must be sorted through, even if the cost of resolving any one alarm is not high. In other cases, the cost of individual false alarms can be quite high, since it may require ammple reverse engineering and significant time to determine the root cause of an HTH indicator. In still worse cases, some of the false alarms will be unresolvable. That is, despite the defender’s best efforts, they may not be able to determine that a false alarm is indeed false. In this case, they will incur significant costs, since they will act as if a Trojan has been detected, when in fact one is not present. Thus, the \(Z_{FA}\) variable acts to properly account for the costs of methods with differing false alarm rates instead of naively pursuing high detection rates at all costs.

\(G\): The adversary’s financial gain if successful. This variable represents the adversary’s motive for inserting the HTH.\(^4\)

---

3. The monetization of strategies and human life may be unfamiliar to readers of a hardware security journal, but they are regularly performed practices of risk assessment. See [25, 26].

4. Note that the adversary’s goal is not to insert the HTH. Rather, it is optimally seek gain \(G\). This observation is both simple and extremely important, especially when considering exploits at the system level. The rational adversary will pursue their gain \(G\) in the optimal manner, which might involve an exploit of hardware, software, or firmware. See our parallel work in [6].
That is, they have an advantage to gain. In commercial markets, this advantage could be their own product’s time to market. In defense, they might gain from reducing their opponent’s advantage. In safety, they could have some political advantage to diminishing the safety of a product. Note that unlike other approaches, we do not simply say that the defender’s loss is the adversary’s gain; thus, our game is not necessarily symmetrical (unless variable relationships are set up to make it so). Thus, we are able to model complex game scenarios among asymmetrical opponents. Just like in the case of the defender’s loss, the non-monetary values that contribute to G must be monetized as a prerequisite for playing the game.

\[ Z_A(\sigma_A): \text{The direct cost of implementing the adversary’s strategy, } \sigma_A. \] Similar to the cost of the defender’s strategy, this is measured in time and materials. To properly account for those costs, one may need to consider costs long before the adversary and defender meet each other. The adversary will have to identify potential targets and do an initial qualification to see which are worth pursuing. They will have to perform an assessment to see which attacks make sense given the dimensions of the design they are seeking to exploit. For their strategies, they will have to similarly consider the creation, execution, and verification costs for their HTH. A unique adversary cost is also access. That is, the method of gaining access to the defender’s design such that they can exploit it with an HTH is a significant cost to the adversary.\(^5\) For each game scenario, it is essential to separate which of these are sunk costs and which should be attributed to each strategy uniquely. For example, if the interaction between the adversary and defender takes place after the adversary has already qualified the defender’s design as meeting their criteria for exploitation and gained access to the defender’s design network, the costs associated with target qualification and access do not need to be separately applied to each strategy in \(S_A\). The game model in this paper takes this approach. However, if the adversary is earlier in their qualification process (as may be the case if the adversary is deciding between attacking the system via a software exploit or a hardware exploit as in [6]), a different accounting will need to take place for the cost of each strategy.

\[ Z_{\text{false}}(\sigma_A): \text{The penalty to the adversary if their Trojan, } \sigma_A, \text{ is not only detected but also attributed to them. The } Z_{\text{false}} \text{ penalty could include lost direct financial value (e.g., the defender sues the adversary), lost market value due to reputation, jail time, or other penalties. In defense scenarios, lost reputation could be lost standing in the world, sanctions, or other international penalties.} \]

\(^5\) Note that the adversary’s goal is not to insert the HTH. Rather, it is optimally seek gain G. This observation is both simple and extremely important, especially when considering exploits at the system level. The rational adversary will pursue their gain G in the optimal manner, which might involve an exploit of hardware, software, or firmware. See our parallel work in [6].

\(^6\) Differences in the cost of accessing software, firmware, and silicon for exploitation was explored in [6].

### 3.3 Assembling utility functions

We now have the requisite variables to populate our adversary and defender utility functions. These equations represent each party’s beliefs. That is, they model the reasoning by which they will make their decision. As stated, we first model our opponents as perfectly rational, considering irrationality later. Thus, for the adversary, we define the utility as:

\[
U_A(\sigma_A, \sigma_D) = (1 - P_D(\sigma_A, \sigma_D))P_S(\sigma_A)G - Z_A(\sigma_A)
\]

\[
- P_D(\sigma_A, \sigma_D)P_{\text{ATT}}(\sigma_A, \sigma_D)Z_{\text{false}}(\sigma_A)
\]

That is, the utility for the adversary of selecting a given strategy relates to the value of their desired gain (subject to the probability that their Trojan will both go undetected and work), less the cost of inserting that strategy, less the potential loss as the result of the HTH being found, subject to the probability their Trojan is both found and attributed to them.

For the defender, utility is modeled as:

\[
U_D(\sigma_A, \sigma_D) = [P_D(\sigma_A, \sigma_D) - P_D(\sigma_A, \sigma_D))]L - Z_D(\sigma_D)
\]

\[-Z_{\text{FA}}(\sigma_D)[P_{\text{FA}}(\sigma_A) - P_{\text{FA}}(\sigma_A)]\]

That is, the utility for the defender of electing a given strategy considers the value of their loss subject to the selected strategy’s reduction in the probability of incurring that loss with respect to the “do nothing additional” strategy, less the cost of deploying the strategy, less the cost of a false alarm subject to the selected strategy’s increase in the false alarm probability with respect to the “do nothing additional” strategy.

### 3.4 Step games

An additional consideration for both players in our game is where in the Application-Specific Integrated Circuit (ASIC) or Field-Programmable Gate Array (FPGA) design cycle their interaction takes place. One way of treating this selection is to consider all of the available HTH attack and countermeasure strategies—from writing the specification to deploying the finished devices—in one large game for every design. The complexity of such a large game leads to difficulties in drawing conclusions related to what a designer should do at each step of the process. An alternative is to zoom in on each step of the design cycle and treat them as separate decision points at which the adversary and defender must consider whether and how to attack or defend. This paper concentrates on this latter approach, adopting the term step games to describe the adversary/defender interaction at each point in the design cycle.
3.5 Games and solution concepts

In a game, if either player knows what their opponent is going to do, they can simply use their own utility function to select their optimal play. However, we cannot assume that each party will know in advance which strategy their opponent will select. It is reasonable that each party may have good knowledge of the strategies available to their opponent, however. That is, they will know what their opponents are capable of, but they will not know exactly what they will do. This is where game theory demonstrates its value. In rough terms, game theory can tell each player the optimal strategy to select while simultaneously optimizing their opponent’s selection. More specifically, the Nash equilibrium solution concept discovers the strategy tuple \((\sigma_A, \sigma_D)\) or tuples at which no player can do better by unilaterally changing their strategy.

When solving a strategic two-player game, the Nash equilibrium may resolve to a single pure-strategy in which there is only one strategy tuple that represents an equilibrium. With a single pure-strategy equilibrium, the optimal choice for each party is quite clear: select the strategies at the equilibrium point. Alternatively, some games present a mixed-strategy equilibrium, where the strategy selections are represented by a probability distribution across several strategies. Other games may resolve to multiple pure strategy equilibria, and still others may have both mixed and pure strategy equilibria. When dealing with games that have more than one or mixed equilibria, one or both players may have a set of rational choices to select from. Given a mixed or multi-equilibrium outcome, each player will know the probability by which their opponent may select from among their equilibrium strategies, and each also knows their own equilibrium strategies and the frequency with which they should be played.\(^6\) At first, it may seem that a mixed strategy answer would only confuse the matter of selecting an optimal HTH countermeasure. As we will see from our experiment below, an exploration of mixed strategies can provide robust insight into adversary/defender dynamics.

3.5.1 Player rationality

An underlying assumption of the Nash equilibrium solution concept is that each party is rational. A rational player seeks their own optimal best interest within the knowledge of what their opponent might do. While our work as well as [12, 14, 15] are built on this rationality assumption, others have rightly questioned whether human actors will behave in this way. For example, [16] uses Prospect Theory to model players who make irrational (or “subrational”) choices in ways that mimic human decision making. Prospect theory can account for adversaries who might take risks to gain big payoffs or defenders who are irrationally afraid of big losses—despite that in both cases those choices are less-than-optimal. Another subrational alternative to prospect theory that we have considered is quantal response equilibrium (QRE), which uses a statistical model instead of a deterministic method to solve games [28]. That statistical model includes a value, \(\lambda\), that bounds the rationality of each player. In this way, it does not introduce prospect theory’s specific models of how the player will behave irrationally, it simply describes a fixed limit to their rationality. Alternatively, Trembling hand perfect equilibrium considers whether a player might simply make a mistake, either by accident or by imperfectly modeling the strategies available to their opponent [29]. Revealed preference theory [30] may illuminate less-than-rational player choices in multi-play games by taking into account a history of strategy selections. More complex subrational player models can account for players who evaluate strategies differently and come to non-uniform conclusions from repeated play, such as in subjective utility quantal response equilibrium (SU-QRE) and the Stochastic Human behavior model with Attractiveness and Probability weighting (SHARP) model, in which concepts of attractiveness and vulnerability are used to model the manner in which players construct subrational choices [27, 31]. In all of the above non-Nash solutions, accidental/imperfect play is bounded to allow strategic reasoning to take place even in the case where subrational behaviors are exhibited by the players.

While the above work argues convincingly that players may be irrational in fixed or human-like ways—or might be perfectly rational against a mistaken model—we perform most of our analysis with rational players whose beliefs comport to reality, and we draw conclusions from the Nash equilibrium solution concept in this work. We do so for three reasons. First, the goal of our work is to demonstrate a direct connection that maps empirical testing practices to the most optimal hardware Trojan defense. In doing so, we want to allow empirically demonstrated HTH attack/countermeasure analysis to guide our game solutions to the maximum degree possible while minimizing the opportunity for expert assumptions to dominate. We explicitly want to avoid decision making by fear, uncertainty, and doubt and instead emphasize where the rational choices are. Second, we consider the rationality of the players to be a question that is independent of those pertaining to creating proper utility models for the HTH adversary and defenders. In fact, the same utility models can provide the foundation for solution concepts that introduce irrationality as an additional variable. Thus, to focus our work, we adopt the simplifying assumption that the players are rational, for now.

Third, in our ongoing work, we are exploring the above-listed irrationality models, using this work to define the baseline optimal selections based on rational players. That is, as we
advances our decision engine (described below) to include irrationality models, we are synthesizing strategy guidance from game solutions that include both rational and irrational player models and produce a recommendation that weights or otherwise reasons while taking both into account. Thus, the below work would still contribute to such a decision engine even after irrationality models and solution concepts are complete. Furthermore, when we do introduce irrationality models, we wish to do so with baseline utility functions that properly model the HTH problem at hand. We seek first to properly discover solutions for rational players with correct utility functions; after this, we will consider irrationality models that build on this foundation.

3.5.2 Computing game solutions

In our past work, we emphasized illustrating game solutions that could be solved on paper [5] in order to reduce the complexity of our discussion. Alternatively, others have reduced the strategies available to players to two (a binary game) to allow a simpler formal mathematical discussion of the game variables and their relationships; see for example [32]. As our goal in this work is to demonstrate a practical application of game theory to a diverse set of real-world strategies available to the HTH adversary and defender, the resulting games are quite large. These large games do not lend themselves easily to discussion using either by-hand solving or formal mathematics without over-constraining the assumptions and reducing the fidelity of the utility function model. Thus, we have adopted an approach that allows us to solve and analyze large games using a powerful multi-threaded symbolic solver engine and a novel approach to visualizing the solutions to the games.

4 Experiment

We constructed an experiment to demonstrate the value of game-assisted reasoning about hardware Trojan countermeasures. Our purpose was to establish a realistic game scenario—including reasonable players, economic variables, and available strategies—then allow the empirically derived metrics to complete the utility functions for each player. This section introduces the game scenario and a variant, the experimentation performed using actual Trojan/countermeasure interactions, and the software we developed to automate various processes, including an automated game solver and results visualization tool called GameRunner.

4.1 Game scenario: defender and adversary

We consider the game in which a defender is attempting to produce a trusted FPGA design that consists partially of defender-written hardware description language (HDL) code and partially of 3rd-Party IP cores (3PIP) purchased by the defender. For the sake of brevity, in this paper, we analyze only the HDL step game of this scenario. We model the adversary as a rival who seeks to undermine the defender’s product and, as a result, gain some of the market the defender loses as a result of the HTH. We use the pragmatic advanced persistent threat (pAPT) adversary model that we first introduced in [6]. The pAPT adversary is an APT adversary who behaves rationally. We assume that the pAPT adversary already has access to the network and computer systems upon which the defender is designing their product. The pAPT adversary has the ability to infiltrate and exfiltrate data from the designer, including modifying or replacing the design’s source files or 3PIP cores that become part of the defender’s system. The strategies available to the pAPT adversary include a taxonomy of HTH exploits, which can be placed into these files to become part of the defender’s FPGA bitstream. The purpose of the HTH is considered generally: they wish to cause malfunctions in the defender’s design to cause it to fail in the market, allowing them to gain market share for their rival product. Notably, we treat the cost incurred by the adversary to gain persistent access to the defender’s network to be a sunk cost. That is, our game takes place after the adversary has decided to explore the defender’s network to determine whether to attack further with an HTH. The adversary’s utility function considers and quantifies the merits of that HTH attack alone.

4.1.1 Defender and adversary economics

The defender seeks to use this FPGA as the core processing element for a product they are bringing to market. We consider three market scenarios, illustrated in Table 1 below. The first scenario, entitled “Kickstarter,” is that of a developer who wishes to sell a novel Web-connected Internet of Things (IoT) device on Kickstarter. Their goal is to sell 500 such devices. They are using an Arty board from Digilent that contains a Xilinx Zynq 7000 SoC FPGA [33]; we use the cost of that board to represent the total material cost of the device.

---

7 It may be in this analysis that strategy selection is dominated by factors other than the rationality of the players, leading the recommendations of models that rely on different assumptions of player rationality to converge. In this case, it is valuable to know the baseline solution with rational players as a point of comparison to irrational player models.

8 For discussion of the 3PIP step game and further analysis of this experiment, see [33].

9 Note that this game represents only one of the many scenarios that can be explored using our utility functions.

10 [6] explores additional potential desired outcomes and strategies for the pAPT adversary.

11 The assumptions that build our economic values are simplified throughout this paper to focus on the hardware Trojans and countermeasures. If the players were involved in a different economy (e.g., defense), the contributors to the economic values would be dramatically different.
high-end connectivity and memory resources [34]. The sale board, which has a large Xilinx Virtex-7 690T FPGA and which we estimate based on a high-end Digilent NetFPGA processing system. They are pursuing a smaller market by seeking a consumer device; in this case, 5000 are sold. The successful enough to be sold in high enough quantities to be estimated using data from [36], which lists labor wrap rates for government order to calculate the cost of an hour of labor, including all overheads. We these features.

We set the sale price of the item to be twice the material cost, and the market value of the item to be the quantity sold multiplied by that sale price. The second scenario is the “Consumer” scenario, which is when the Kickstarter item is successful enough to be sold in high enough quantities to be considered a consumer device; in this case, 500 are sold. The third scenario is a different device: a high-end network processing system. They are pursuing a smaller market by seeking to only sell 4000 devices. Their material cost is higher, which we estimate based on a high-end Digilent NetFPGA board, which has a large Xilinx Virtex-7 690T FPGA and high-end connectivity and memory resources [34]. The sale price for the third scenario is also estimated at twice the material cost. We set the loss value, L, which the defender seeks to protect to be the market value of the item. In this game, we model the adversary as a rival who seeks to undermine the defender’s product and, as a result, gain some of the market the defender loses as a result of the HTH. We assume the adversary’s gain, G, will be half of the market value of the item. We assume if the adversary is discovered, the penalty, \( Z_{\text{find}}(\sigma_A) \), is set to half of G for all strategies.

To determine the costs of HTH and countermeasures, we need to place value on person-time and resource-time. To develop an hourly labor cost, we considered that the skills necessary to design or counter HTHs are similar to the skills of a design verification engineer. We took the national average for a design verification engineer, assigned a labor wrap rate of 3.1 to represent a contractor with expensive software tools [36], and divided by a 2080 h work year to get an hourly rate of $167.22.

We make the simplifying assumption that labor costs are the same for the adversary and defender. To account for the cost of compute time, we use $0.33/h from [37] based on the cost of an m5.2xlarge Amazon Web Services (AWS) virtual machine instance with 8 vCPUs and 32 GiB of memory, a computer sufficient to run the EDA tools. Comparing the cost of labor to that of compute time, one can easily conclude that labor costs will dominate. In this paper, we do not make use of methods that require long periods of unattended computing, so we simply assume there is 1 h of compute time for every hour of labor. That is, the defender and adversary are utilizing computers like m5.2xlarge every time a human is performing labor. As we will see, to develop the costs of the adversary and defender strategies listed below \( (Z_A(\sigma_A) \text{ and } Z_D(\sigma_D)) \), we assign each with a time cost of labor, a time cost of computing, and a material cost (if any) to represent any licenses that need to be specially purchased to enact that strategy.

### 4.1.2 Step games: HDL, 3PIP, and others

In our experiment for this paper, we model the HDL step game. Similar games may be played for the 3PIP step, the bitstream generation step, and other steps (see [33]); however, the HDL game in this paper illustrates the analysis possible using our method. The defender and adversary are selecting one strategy to play at each step. In our game, the defender can select the “do nothing additional” or one or more countermeasures at each step. The adversary selects “do nothing” or one HTH. For the defender, in the HDL case, they are writing their own source code (which they trust), and their countermeasures determine if the logical netlist synthesized from their trusted source code contains an HTH. For the pAPT adversary in the HDL step game, they are choosing whether to replace the synthesized netlist from the synthesis tool with a modified netlist that contains their HTH. As an APT adversary, they are able to analyze the synthesized netlist from afar, using their own resources to perform the analysis after they have exfiltrated the netlist from the defender’s network.

In our ongoing work, we are considering the 3PIP and HDL games along with every other step in the design cycle either simultaneously or serially to determine how decisions in each step affect each other. In this analysis, the adversary would consider all available HTH insertion strategies and select not only the optimal HTH but also the optimal step in the design cycle to insert. Similarly, the defender would consider all steps to protect and spread resources among the steps. The experiment here illustrates a smaller attack surface available to the adversary for clarity, but larger models are presently under construction to consider the entire development process.

### 4.2 Strategies and probabilities

In our experiment, we attempt to illustrate strategies that are realistically available to both an adversary and defender. In [7], we detailed the large task required to establish a reliable and comprehensive dataset upon which to base industry-level recommendations derived from this game-based
methodology. For our work in this paper, we attempt simply to illustrate the value of game-based reasoning by having a representative set of strategies available to the adversary and defender. For the adversary, we borrow selections from the TrustHub benchmark suite [38, 39] and produce a few of our own to illustrate attack and circuit test principles not present in TrustHub. This allows us to demonstrate an HTH dataset that is statistically interesting and divisible into a taxonomy that is compatible with our methodology. However, since we do not claim that it is representative of all HTH’s in the wild, we caution against drawing industry-guidance conclusions from this experiment.

For the defender strategies, we similarly wish to emphasize that we have a representative set. In order to avoid the impression that we are making industrial recommendations, the only software we name is Xilinx Vivado Simulator, which is used as the baseline of industry common-practices for verification. Then, we implemented our own HTH detection methods, borrowing theory from interesting methods available in the literature. We do not represent these methods as perfect replicas of the literature that inspired them; they simply represent realistic HTH detection methods that fit into a taxonomy compatible with our utility functions. Finally, we made use of one additional industry tool, a Boolean logic equivalence (BLE) verification tool. Again, since the purpose of our work is to demonstrate the value of our game method, we do not name the BLE software used, to avoid having our conclusions misconstrued as a value judgment on that software.

4.2.1 Adversary strategies

We model a disciplined adversary whose approach to HTH insertion is based on known cyber-offensive practices. Their approach involves three stages: discovery, implementation, and exploitation [40]. In discovery, they review the circuit to determine the best place in the circuit to attack. 14 If that answer is “nowhere,” they employ attack $\sigma_{A0}$, which we abbreviate in our tables as DONT. However, because they had to review the circuit in the discovery stage to make this determination, there is a cost associated. We assume this decision takes 120 h, based on three persons working a 40 h week. Note that our estimates of hourly labor expenditure—both here and throughout this work—are reasonable but not exact. 15

We further assume the offensive EDA software set required of the adversary is a sunk cost that is not directly attributable to the current circuit. Thus, there are no material costs, meaning the cost of $Z_A(\sigma_{A0})$ is $20,106.

For all other circuits in the HTH taxonomy, we assume that the circuit required for the attack is already designed prior to the engagement. That is, they have at their disposal a variety of HTH exploits, and the cost of inserting the circuit in strategies where discovery has indicated that an HTH should be placed is dominated by labor. We assume that every HTH takes 240 h to customize for the target circuit (the implementation stage) and an additional 120 h to place the HTH in the circuit and put the modified circuit back in the defender’s network via pAPT. Thus, including the discovery stage, there are 480 total hours of labor associated with every attack in which placing an HTH of any kind is the optimal strategy, meaning that $Z_A(\sigma_{A}) = 80,466$ for all strategies other than $\sigma_{A0}$.

For the representative HTHs we produced, our taxonomy is derived from [9, 10] and simplified to divide the circuits into categories based on the trigger mechanism involved. The circuits we used were primarily the set from [38, 39] as refined by [41]. However, to provide more variety of input design and demonstrate progress towards the desired large dataset required by [7], we also included designs from the DARPA Common Evaluation Platform (CEP) [42] and placed a variety of HTH’s in them. The payloads enabled by these triggers include leaking information, changing function, denying service, degrading performance, and reducing reliability. Additionally, as we will discuss below, we included an additional category of HTH and placed it in several designs. We list the circuits below in Table 2. Our first two HTH categories, REWR and GATE, are not triggered at all—they are always on. $\sigma_{A1}$ (REWR) represents HTHs that are always on and only rewire the circuit—they do not actually add gates or alter logic. We included eight such circuits in the dataset. $\sigma_{A2}$ (GATE) represents HTH circuits which are always on and include modified gates and logic in addition to wires; four are included.

Strategies $\sigma_{A3}$, $\sigma_{A4}$, $\sigma_{A5}$, and $\sigma_{A6}$ are all triggered. That is, they are not active when the circuit first turns on; they have to be activated by a triggering mechanism. The portion of the circuit that turns on when the trigger is activated is called the payload. The fifteen HTH circuits in strategy $\sigma_{A3}$ (ECTR) are activated by an event counter. This counter can be as simple as a clock counter (e.g., a time-delayed activation) or it can count other events in the circuit until an activation threshold is reached. The thirteen circuits representing strategy $\sigma_{A4}$ (CCMP) use a combinational comparator as a trigger. That is, they activate the HTH payload when a set of

---

14 This may be where the adversary employs a methodology similar to [14], where they use game theory to optimize HTH placement in a circuit.

15 Our purpose in this work is to illustrate our analytic method in a representative game. The structure of assumptions we expose here provides the opportunity for ongoing refinements within the same framework. Hour estimates are meant to be realistic but not exact.
signals in the circuit are equal to a preset value. This could be a secret coded value on an input, or any set of internal signals that accomplish the expected trigger value. A more complex type of trigger is represented by the fifteen circuits in $\sigma_{A5}$ (STSQ). These are triggered by a sequence of events in the circuit. These can be a transition sequence in a state machine or a series of events that are monitored from disparate circuit regions.

The activation mechanism for the strategy set $\sigma_{A6}$ (GLST) is a glitch. That is, the HTH consists of a state added to an incompletely specified state machine which will make it easier for a fault-injecting adversary to perform a future clock or power glitching attack. The state encoding for this added state is selected to be a hamming distance of 1 from one or more initialization states of the circuit. This design makes the “glitch state”

Table 2: Adversary strategies and estimated costs

| Strategy | Taxonomy description | Name | Qty | HTH circuits | Labor cost (h) | Machine time cost (h) | Material cost ($) |
|----------|---------------------|------|-----|--------------|-----------------|---------------------|------------------|
| $\sigma_{A0}$ | Do Nothing | DONT | 6   | AES-notj-top, BasicRSA-notj, CEP-gps-notj, PIC16F84-NoTj, RS232-NoTjGate, RS232-NoTj | 120 | 120 | 0 | 20,106 |
| $\sigma_{A1}$ | Always on: Rewire | REWR | 8   | AES-reversebit1, AES-reversebyte1, BasicRSA-TReverseByte1, CEP-gps-Tj-disable-ae, CEP-gps-Tj-reversebit1, PIC16F84-TReverseBit1, RS232-TjReverseBit1, RS232-TjReverseBit2 | 480 | 480 | 0 | 80,426 |
| $\sigma_{A2}$ | Always on: Change Gates | GATE | 4   | AES-T100, AES-T200, AES-T300, RS232-T1800 | 480 | 480 | 0 | 80,426 |
| $\sigma_{A3}$ | Trigger: Event Counter | ECTR | 15  | AES-T1200, AES-T1500, AES-T1700, AES-T1900, AES-T2100, AES-T900, BasicRSA-T300, BasicRSA-T400, CEP-gps-Tj-reset-counter, PIC16F84-T100, PIC16F84-T200, PIC16F84-T300, PIC16F84-T400, RS232-T300, RS232-T500 | 480 | 480 | 0 | 80,426 |
| $\sigma_{A4}$ | Trigger: Combinational Comparator | CCMP | 13  | AES-T1000, AES-T1300, AES-T1800, AES-T400, AES-T600, AES-T700, BasicRSA-T100, BasicRSA-T200, RS232-T100, RS232-T400, RS232-T800, RS232-T1300, RS232-T1700 | 480 | 480 | 0 | 80,426 |
| $\sigma_{A5}$ | Trigger: State Sequence | STSQ | 15  | AES-T1100, AES-T1400, AES-T1600, AES-T2000, AES-T500, AES-T800, CEP-gps-AES-T500, RS232-T600, RS232-T700, RS232-T900, RS232-T901, RS232-T1200, RS232-T1600, RS232-T1900, RS232-T2000 | 480 | 480 | 0 | 80,426 |
| $\sigma_{A6}$ | Trigger: Glitch State | GLST | 2   | PIC16F84-TjGlitchState, RS232-TjGlitchState | 480 | 480 | 0 | 80,426 |

Fig. 1 Glitch state Trojan in RS232 (RS232-TjGlitchState)
easily reachable by a glitch condition upon device power up. The only function of this state is to transition immediately into a state which should have required a complex sequence of states and inputs to reach. For example, the state machine alteration comprising one of the GLST HTH’s is illustrated in Fig. 1. A new state, X_IDLE.GLITCH is encoded as 3'b001 to have a Hamming distance of 1 from both the X_IDLE and X_WAIT states. Those states are targeted since the circuit is temporally likely be in them at startup or when the device containing the circuit is idle. This would give a later, physically present adversary the opportunity to glitch the circuit, getting into the X_SHIFT without going through a proper state sequence.

This is meant to represent how an HTH could interact with an emerging class of glitching attacks which allow adversaries to bypass security mechanisms such as ARM TrustZone with glitches provided either locally or remotely [43, 44]. Neither of the HTH circuits comprising σ_A0 have TrustZone or other security states, but the glitch states do bypass multiple states in the proper sequence of the state machine to allow the circuit to start in a later state. This circuit category is a simple representative of the emerging set of hardware exploits that go beyond the simple triggers found in most published HTHs. Emerging Trojans of this type—those that make later exploitation easier rather than being an exploit by themselves—are a continuing focus of our ongoing work. As will be seen, traditional verification mechanisms do not consider such attacks, so they do not tend to detect them.

With the “do something” strategies described, it bears revisiting σ_A0, since it is when the adversary selects σ_A0 that the defender must take the most care to avoid a false alarm.16 The six circuits in our σ_A0 dataset that represent the adversary doing nothing are simply the HDL source of the base designs with no changes.

### 4.2.2 Defender strategies

As with the adversary, we model a disciplined defender. This defender’s baseline strategy σ_D0 involves using a testbench and simulator to exercise the design. Because this would likely be done under any circumstance regardless of whether an HTH concern was present, costs of that simulator and testbench are considered sunk and are not attributed to any strategy. Thus, we set Z_D(σ_D0) to $0. Furthermore, for every circuit in the test set, we created a basic testbench that exercises all the features of the circuit to some limited extent if one did not already exist. For strategy σ_D1 (BNCH) if this testbench reveals circuit behavior that varies from the expected output, we mark that as an HTH detection. For all testbench simulations, we used Xilinx Vivado Simulator.

For our other defensive strategies, illustrated in Table 3, the costs associated with them are unique among labor, machine time, and material licenses. For example, strategy σ_D2 (CSIM) also uses simulation and a testbench, but we used a technique called constrained/directed random simulation to improve the test coverage.17 Knowing how long to perform directed random simulation is an art form in the test and verification industry. For our illustrative purposes, we used a random number generator to produce 1 million random input vectors for each primary input of the circuit.18 The cost of employing this method is minimal, since the major cost—that of a simulator that supports it—is free by virtue of our assumption that Vivado Simulator is a sunk cost. Thus, we claim Z_D(σ_D2) = $2681 based on the assumption that it takes 16 h of person time and machine time to adequately write and review the results from each directed random testbench including the time required to set up, run, and analyze a sizable circuit.

The strategy σ_D3 (STRC) is a static controllability/observability and analysis method based on the theory presented in [45] using the implementation we published19 in [46]. The method makes use of static controllability and observability analysis [47] information as the basis of an unsupervised clustering analysis, which allows the separation of signals that are components of hardware Trojans and those that are not. The implementation makes use of the TCL interfaces exposed by Xilinx Vivado and performs the k-means clustering using the scikit-learn Python library [46, 48]. We assume this automated tool would be simple to use (16 h of labor and machine time) and modestly priced if sold on the market ($15,000 license). Thus, Z_D(σ_D3) = $17,681.

The strategy σ_D4 (STRC) is a structural pattern matching method similar to those in [49, 50]. It claims a Trojan is present whenever it detects disconnected or undriven wires or asynchronous feedback loops, features common in published HTHs and extremely uncommon in traditional circuit design.

16 Of course, a defender can also false alarm on a circuit that contains a Trojan, but ascribing an HTH to a circuit with a HTH for an incorrect reason has a significantly lower consequence than falsely ascribing an HTH to a circuit with no HTH present.

17 This is related to the software black-box testing technique referred to as fuzzing.

18 More advanced uses of a constrained random may set a toggle coverage threshold. That is, instead of simulating 1 million random input vectors and stopping, random inputs would be generated and simulated until some pre-set high percentage of the wires in the circuit have transitioned from 1-to-0 or 0-to-1.

19 The implementation details of our work in [46] are interesting but they diverge from this discussion here. The original contribution of the paper was the implementation, not the algorithm. We demonstrated the advanced Trojan detection method of [45] while avoiding the use of any advanced EDA software. Rather, we implemented it using only Xilinx Vivado—which all Xilinx FPGA developers already have—and a few Python scripts.
As with SCOA, we implemented this using the TCL interfaces exposed by Xilinx Vivado. For this method, we adopt the same cost model as \( \sigma_{D3} \), which means \( Z_D(\sigma_{D3}) = $17,681 \).

Finally, for strategy \( \sigma_{D4} \) we make use of a commercial Boolean logic equivalence checker (BOOL). For this method, we used a commercial EDA tool to determine if the synthesized netlist is logically equivalent to the trusted source code. The labor and machine time to set up, run, and review the result from the tool are set to 24 h each. The tool license is $65,000. The total cost of the strategy is $69,021.

The cost of a false alarm, \( Z_{FA}(\sigma_D) \), deserves special consideration. This cost is incurred to the defender whenever they elect a countermeasure strategy that results in false indication of an HTH where none is actually present. In the HDL step game, we assume they can peruse their own code to determine that the HTH is not present, using standard EDA tools at their disposal to do so. We assume this will cost them 16 h, or $2681 to do so. It is worth noting that in the 3PIP step game, forensic examination to determine if an HTH indication is false is far more costly, since source code for the 3PIP core may not be available—and if it is, it was produced by a party other than the designer.

Table 4 summarizes the three economies we are using to demonstrate our game. In addition to the strategies being different between step games, the \( Z_{FA}(\sigma_D) \) values are as well. All other economic variables are set to be the same between the step game to allow us to examine the results derived by changing only strategies and false alarm penalties.

### 4.2.3 Empirically derived probabilities

Our experiment determined two of the core probabilities of our model: \( P_D \) and \( P_{FA} \). For every \((\sigma_A, \sigma_D)\) tuple, \( P_D(\sigma_A, \sigma_D) \) was developed by testing every individual defender detection strategy against all circuits in the benchmark set that contained an HTH that was categorized as a member of an adversary strategy set \( \sigma_A \). \( P_{FA}(\sigma_A, \sigma_D) \) was determined by the proportion of circuits in the subject HTH set that were detected by the subject detection method. One of the adversary strategies tested in this way was \( \sigma_A0 \), when the adversary elects not to insert an HTH and performs no alteration to the original defender circuit. \( P_{FA}(\sigma_D) \) was determined by the proportion of these unaltered circuits that a defender strategy marked as having an HTH in it.

We emphasize that our testing was performed empirically. That is, our results are not determined by projecting which adversary Trojan should be theoretically detectible by each defender countermeasure. Rather, we tested each Trojan in our adversary set against each detection method in our defender set. We are seeking results of real-world practical value. This requires us to consider implementation defects in the detection methods that might cause them to report erroneous information. That is, if a detection method reports the absence of a Trojan due to an implementation error, we believe that should count against it.

The testing took place in two environments. For defender strategies BNCH, CSIM, SCOA, and STRC, our tests were performed on an Ubuntu Linux 16.04 LTS virtual machine within the Graf Research virtual machine pool. It was assigned 16 threads from a 2.9 GHz Intel Xeon processor along with 64 GB of memory. For defender strategy BOOL, we made use of a software license and virtual machine provided by the Air Force Trusted Silicon Stratus (TSS) cloud [51, 52]. As TSS is built in a special enclave of the AWS GovCloud, the TSS VM was an AWS c4.8xlarge instance with 36 CPUs and 60 GB of RAM.\(^{20}\) The testing was automated by a “test harness” which can quickly accept new adversary or defender strategies. Once a strategy for either party is defined, the testing process is entirely automated. At present, the test harness is a GNU makefile, which

**Table 3** Defender strategies and estimated costs

| Strategy | Taxonomy description | Name  | Labor cost (h) | Machine cost (h) | Material cost ($) | \( Z_D(\sigma_D) ($) \) |
|----------|----------------------|-------|----------------|------------------|------------------|------------------------|
| \( \sigma_{D0} \) | Nothing but testbench | BNCH  | 0              | 0                | 0                | 0                      |
| \( \sigma_{D1} \) | Constrained/directed random simulation | CSIM  | 16             | 16               | 0                | 2681                   |
| \( \sigma_{D2} \) | Static controllability/observability method with k-means clustering | SCOA  | 16             | 16               | 15,000           | 17,681                 |
| \( \sigma_{D3} \) | Structural pattern matching | STRC  | 16             | 16               | 15,000           | 17,681                 |
| \( \sigma_{D4} \) | Boolean logic equivalence testing | BOOL  | 24             | 24               | 65,000           | 69,021                 |

**Table 4** Three economies (all values estimated $)

| Economy     | \( G \) | \( L \) | \( Z_{find} \) | \( Z_{FA}(\text{HDL}) \) |
|-------------|--------|--------|----------------|-------------------------|
| Kickstarter | 74,500 | 149,000| 149,000        | 2681                    |
| Consumer    | 745,000| 1,490,000| 1,490,000      | 2681                    |
| Network device | 27,980,000| 55,960,000| 55,960,000     | 2681                    |

\(^{20}\) Neither virtual machine utilized exactly matches the AWS VM we used for the cost model in our example game. However, none of our benchmarks were multithreaded nor did any of them require more than the 32 GB of memory used in the cost model. Thus, the difference is not relevant to our example game.
launches each commercial or custom EDA tool with the appropriate options and some Python scripts that parse the output logs of the tools involved to determine whether or not a detection has taken place.

4.2.4 Automation and GameRunner

In order to automate the process of defining, solving, and exploring game solutions, we developed the concepts alluded to in [7] into a software tool entitled GameRunner. The core of GameRunner is a Python application that reads a custom JavaScript Object Notation (JSON) file that contains the data for one or more game scenarios. GameRunner assembles that data to present to the user in a Graphical User Interface (GUI) or to pass to one of a variety of game solvers. The user can also make use of the GUI to extract a prescription, which is the game-suggested mitigation strategy in a format that permits the automation of the tools required to perform that strategy. This prescription presently takes the form of a Jenkins file, which can be used in conjunction with the open-source Jenkins workflow automation software to run the software that composes the optimal detection strategy sets [53]. That is, GameRunner does not simply tell the user what the optimal strategy might be; it also issues a file that automates the implementation of that optimal strategy.

We selected a solver for Game Runner through a review of computer algebra systems [54], game-solving software tools [54], and software solver libraries [55]. We compared these three such systems to each other by solving thousands of 25 × 25 games. Comparing for performance, implementation stability, and the correctness of the result, we selected the lrs libraries [56] as the solver for the solutions presented in this paper. lrs is a set of C libraries which implement the lexicographic reverse search vertex enumeration algorithm in the form of [57]. lrs can be used to solve linear programming problems. The Nash equilibrium can be expressed as a linear programming problem; thus, lrs libraries can solve for Nash.

The GameRunner GUI includes a number of features related to visualizing game results. These visualizations, along with the data interactivity allowed by GameRunner, permit a wide variety of “what if” analyses. All the illustrations in the next section were generated directly by the GameRunner tool.

5 Results

Our results discussion starts with the raw output of the game results and a discussion of the Nash equilibria of each game.

21 Note that in deployment, GameRunner is not intended for use by general users. Rather, an expert would make use of GameRunner to produce prescriptions and more simplified software would be available to users to select the appropriate prescription without the requirement of understanding the game theory behind the software.

| Table 5 | $P_D$ by adversary and defender strategy |
|---------|------------------------------------------|
|         | BNCH | CSIM | SCOA | STRC | BOOL |
| DONT    | 0%   | 0%   | 0%   | 0%   | 0%   |
| REWR    | 88%  | 100% | 100% | 88%  | 100% |
| GATE    | 50%  | 50%  | 50%  | 100% | 100% |
| ECTR    | 53%  | 53%  | 73%  | 80%  | 100% |
| CCMP    | 31%  | 46%  | 62%  | 46%  | 100% |
| STSQ    | 13%  | 13%  | 60%  | 27%  | 100% |
| GLST    | 0%   | 0%   | 50%  | 50%  | 0%   |
| Total   | 40%  | 46%  | 68%  | 60%  | 96%  |

We follow this with an exploration of the games, demonstrating the contribution of the GameRunner tool to game analysis. Finally, we explore the outcome of games which have been modified to allow the defender to employ more than one countermeasure.

5.1 Results and Nash Equilibria

Table 5 summarizes our experimental $P_D$ results; Table 6, our $P_{FA}$ results. BNCH and CSIM were poor HTH detectors, with CSIM slightly better, but neither returned any false alarms. SCOA and STRC returned improved detection results, but SCOA has a disadvantageous false alarm rate. BOOL returned by far the best HTH detection rate—nearly perfect at 96% with no false alarms.

Table 7 depicts the two-player strategic game that emerges in the Kickstarter economy at the HDL step. In this table and those that follow, we depict the game in normal form; each entry in the table represents the tuple that results from calculating $(U_A(\sigma_A, \sigma_D), U_D(\sigma_A, \sigma_D))$ for the given defender and adversary strategies. The detection method efficacy metrics ($P_D$ and $P_{FA}$) are drawn from the data in Table 5 and Table 6, and the remaining utility function variables are set as described in Sect. 4. Table 7 entries are listed rounded to the nearest thousand dollars. The solution to this game is a mixed strategy for both players. The adversary should not attack (play the DONT strategy) 76% of the time and play the GLST HTH 24% of the time. Having DONT as the most common is due to the low value of the target. The GLST Trojan is quite effective, however, meaning that despite the low return, playing it occasionally has value. For the defender, since the adversary is so unlikely to attack, it is optimal to play BNCH 65% of the time and STRC, which has some effect against GLST with a low false alarm rate, 35% of the time.

| Table 6 | $P_{FA}$ by defender strategy |
|---------|-------------------------------|
|         | BNCH | CSIM | SCOA | STRC | BOOL |
| BNCH    | 0%   | 0%   | 50%  | 17%  | 0%   |
| CSIM    | 0%   | 0%   | 50%  | 17%  | 0%   |
| SCOA    | 0%   | 0%   | 50%  | 17%  | 0%   |
| STRC    | 0%   | 0%   | 50%  | 17%  | 0%   |
| BOOL    | 0%   | 0%   | 50%  | 17%  | 0%   |
The interpretation of the mixed strategy results bears some consideration. If one has a population of defenders who are facing a population of adversaries, the optimal defender guidance in this case is to tell 65% of them to run only BNCH and the other 35% to run STRC. As discussed further in Sect. 6, this is exactly the situation for which we have created this game: guiding a population of designers who are using the TSS cloud to secure their designs against intrusion. However, if one is providing guidance to only one defender, the guidance requires some conditioning. Some designers may wish to bias their selection towards the strategy guidance that is simply the most likely case (in this case, BNCH), towards the one that costs the least (again, BNCH), or biases towards providing the best $P_D$ given the adversary’s likely action (STRC).

The normal form game for the Consumer economy is depicted in Table 8. The normal form game for the Network economy is not depicted for brevity; however, the solution for the Consumer and Network economies were determined by GameRunner to be the same. In both economies, the defender’s optimal play is to play SCOA 70% of the time and STRC 30% of the time.\(^\text{22}\) It may be surprising that it is not optimal for the defender to play BOOL, the one solution with a nearly perfect detection rate and a perfect 0% false alarm rate. The reason is that the adversary in this case has a mixed strategy solution that is approaching a pure strategy solution: they should play GLST almost 100% of the time while playing STSQ nearly 0% of the time but not quite. That is, given the performance of BOOL against REWR, GATE, ECTR, CCMP, and STSQ HTH strategies, the adversary will always do better to play GLST. This means that given a rational adversary in this game, it is never optimal for the defender to play BOOL. Rather, they should alternate between the methods that tie for the best performance against the GLST HTH, STRC, and SCOA. While STRC has a much lower false alarm rate, the vanishing—but still existent—possibility of the adversary playing STSQ Trojans pushes SCOA to be weighted higher due to its better performance against SCOA. Notably, once we reach the Consumer economy model, we have reached the point where it is always rational for the adversary to attack.

Another way to interpret this result is to hearken back to the Maginot line analogy mentioned in the introduction. In these results, BOOL represents such a Maginot line: a defensive fortification that is so effective, the adversary is required to circumvent it. If we only looked at the $P_D$ and $P_{FA}$ data, we would be tempted to be confident in playing BOOL. However, the game theoretic solution teaches us that a rational adversary will take note of BOOL’s efficacy metrics as well and adjust their strategy accordingly. We should stress again that these results hold for the game we constructed, not necessarily for any specific defender who may be reading this paper seeking guidance. In Sect. 6, we revisit the question of what it would take to update our data set and test methodology to be prescriptive for industry.

### 5.2 Exploring the results with GameRunner

Once a game has its initial solution, GameRunner allows the exploration of a variety of interesting questions, including alternative values for variables. One such analysis style is illustrated in Fig. 2. In this case, we have started with the HDL step game in the Kickstarter economy. We asked GameRunner to analyze the consequence of sweeping the variable $L$ (the loss incurred by the defender if the adversary HTH is successful) from $0$ to $375,000$. In this case, we have instructed GameRunner to keep the relationships between the variables constant, as determined in Sect. 4. That is, the adversary gain $G$ is always half of $L$ and the consequence of getting caught, $Z_{final}$, is always twice the gain. In so doing, we maintain the market dynamic as we sweep $L$.\(^\text{23}\)

The graph in Fig. 2 is accomplished by telling GameRunner the number of interim “points” to solve between the extremes of the variable. GameRunner then incrementally changes the variable values linearly between those extremes in increments defined by that number of points. In the figure, we set the number of points to 100, which means GameRunner re-solved the HDL step game 100 times, incrementing the variable $L$ by 100 even increments between $0$ and $375,000$. All variables set to be dependent on the value of $L$ are also changed; all others are held to constant values indicated by the game that called this analysis. The vertical black line shows where $L = 149,000$, since that was the value of the Kickstarter game that launched this analysis process in GameRunner.

\(^{22}\) Because our methodology attempts to predict human behavior, the ultimate proof of correctness for this method is to implement it—along with alternatives—and see which best produces optimal outcomes in the real world. Alternatively, the Nash-prescribed results may be analyzed through many lenses. First, we may consider the question of what actions the players might have taken—and what outcomes they could have expected—had they not been assisted by the Nash equilibrium solution concept. One technique is to use expected utility theory (EUT) in the absence of guidance. Equally weighting all possible strategies in EUT can provide something akin to an average payoff of all possible outcomes. Separately, we performed this analysis on this game and demonstrated the superiority of the Nash-prescribed strategy. We can also use GameRunner to assess the game’s economic values for sensitivity to error. This analysis has the potential to expose the game as one where our assumptions—rather than our experimental testing—dominate the prescribed outcomes. These lengthy analyses—along with a visual exploration of the Nash equilibrium provided by GameRunner—may be found in [33].

\(^{23}\) Of course, GameRunner is also capable of making all these variables independent.
The colors in the graph show the mixed strategy contributions each adversary strategy makes for each value of $L$. On the extreme left end—where $L$ and the equivalent market values are very low—the rational adversary will never attack. At the extreme right end of the chart, we see that in more valuable markets, the adversary will almost certainly attack, with the use of the GLST Trojan approaching 100% likelihood. In between, we see adversary and defender dynamics defined by when it becomes optimal to attack with different attack methods. Again, since this represents the optimal play of the adversary in light of the likely optimal play of the defender, the shapes of the graph change as different strategies are applied by each under different market conditions defined by the variable or variables we are sweeping. In this case, we can see that eventually—with valuable enough markets—the adversary is going to attack with GLST. The defender responses to the same sweep are depicted beneath the adversary graph. On the far left, the defender always elects to only run BNCH. On the far right, the defender strategy has settled into the 30% STRC/70% SCOA mixed strategy that we saw in all games more valuable than the Kickstarter. Again, the line on the graph represents the $L$ value of the Kickstarter game.

This capability to visualize the effects of utility function variables on game outcomes offers some novel strategies for exploring game solutions. For example, here we explore the following question: “What can we do to stop the adversary’s beloved glitch state Trojan?” We explore two scenarios, using the HDL step game in the Consumer economy as the base game. The first scenario explores the question of whether we could add something to the BOOL strategy to improve the detection of GLST. The second explores the idea that we might perhaps make the GLST strategy more expensive to play. This introduces the possibility that a defender strategy could be simply to increase the cost required by the adversary to play their favorite strategy. In fact, this is exactly what one class of HTH countermeasure does, often via obfuscating the circuit, as in [58–67]. As with other refinements of our game, were we to include such strategies, we may want to reconsider utility functions; this is explored in detail in Sect. 6. For this analysis, we use the utility functions from our original formulation.

In both scenarios, we are seeking to discover quantified goals to provide the defender in order to reshape the outcome of the game in their favor. In a way, this is a corollary to the

---

### Table 7 Two-player strategic game: HDL step, Kickstarter economy

| $U_A(\sigma_A, \sigma_D)$ (S thousands) | DON'T | REWR | GATE | ECTR | CCMP | STSQ | GLST |
|--------------------------------------|-------|------|------|------|------|------|------|
| DON'T                               | (-20, 0) | (-80, 16) | (-50, 0) | (-66, 11) | (-31, 0) | (-17, 0) | (-6, 0) |
| REWR                                | (-78, 0) | (-88, 0) | (-47, 0) | (-66, 11) | (-31, 0) | (-44, 0) | (-6, 0) |
| GATE                                | (-47, 0) | (-88, 0) | (-47, -3) | (-71, 22) | (-44, 0) | (-56, 27) | (-6, -3) |
| ECTR                                | (-50, 0) | (-88, 0) | (-50, -3) | (-71, 22) | (-44, 0) | (-56, 27) | (-6, -3) |
| CCMP                                | (-31, 0) | (-88, 0) | (-44, 0) | (-71, 22) | (-44, 0) | (-56, 27) | (-6, -3) |
| STSQ                                | (-17, 0) | (-56, 27) | (-17, -3) | (-28, 2) | (-88, 34) | (-55, 51) | (-47, 55) |
| GLST                                | (-6, 0)  | (-88, 34) | (-6, -3)  | (-47, 55) | (-88, 34) | (-47, 55) | (-6, -3)  |

### Table 8 Two-player strategic game: HDL step, Consumer economy

| $U_D(\sigma_A, \sigma_D)$ (S thousands) | BNCH | CSIM | SCOA | STRC | BOOL |
|--------------------------------------|------|------|------|------|------|
| DON'T                                | (-20, 0) | (-20, -3) | (-20, -19) | (-20, -18) | (-20, -69) |
| REWR                                 | (-78, 0) | (-88, 16) | (-88, 0) | (-78, -18) | (-88, -50) |
| GATE                                 | (-47, 0) | (-88, 0) | (-47, -3) | (-88, 56) | (-88, 5) |
| ECTR                                 | (-50, 0) | (-88, 0) | (-50, -3) | (-71, 22) | (-88, 1) |
| CCMP                                 | (-31, 0) | (-88, 0) | (-44, 0) | (-71, 22) | (-88, 34) |
| STSQ                                 | (-17, 0) | (-56, 27) | (-17, -3) | (-28, 2) | (-88, 60) |
| GLST                                 | (-6, 0)  | (-55, 51) | (-6, -3)  | (-47, 55) | (-6, -69) |
overall goal of our analytic technique. We want empirical practice to drive our analytics. In this section, we describe how our analytics can, in turn, set threshold goals for metrics of empirical practice.

In our first scenario, we want to determine if we can stop the adversary from playing GLST by theoretically adding a GLST detection capability to the BOOL strategy. Recall that BOOL is the best $P_D$ and $P_{FA}$ performer against all other
Trojans, but it is completely blind to GLST. To find a threshold to set as a goal for BOOL, we sweep $P_{d}(\text{GLST, BOOL})$ from 0% to 100%. Figure 3 depicts this sweep, zoomed in on the range of 50 to 100% to highlight where the game outcome changes. One thousand games were solved with $P_{d}(\text{GLST, BOOL})$ sweeping from 50 to 100% to produce the figure. At $P_{d}(\text{GLST, BOOL}) \approx 83\%$, adversary mixed strategy abruptly begins to favor DONT. Below 83%, the adversary is almost sure to play GLST; above it, they will very likely not attack at all. This is an interesting result: an HTH detection method design challenge could be demanded to add a capability to a BOOL tool that has a GLST detection capability in excess of 83% without changing any other qualities of BOOL. Within the limitations of our game, this would truly be the “game changing” HTH detection method.

In the second scenario, we are seeking a similar game-changing threshold, but this threshold is based on forcing the GLST Trojan to become more costly to insert. In this analysis, we theorize about an obfuscation method that might make it specifically more costly to insert the GLST Trojan. We model this by sweeping $Z_{A}(\text{GLST})$ from $0$ to $750,000$, solving 100 games in even increments between those values, and plotting the mixed strategy results, as illustrated in Fig. 4. As with the previous scenario, this analysis discovers an abrupt shift in the game, where the primary strategy of the adversary switches from GLST to DONT. In this case, the strategy shifts at $Z_{A}(\text{GLST}) \approx 370,000$. This is, again, the threshold for a deterrence technique to be a “game changer.” If de-obfuscation could be shown to be more costly than this threshold, the defender “wins” by forcing a rational adversary to the DONT strategy.

### 5.3 Expanding the defender strategies: sets of countermeasures

The defender may be able to select more than one countermeasure in each interaction with the adversary. To evaluate this scenario, we assembled the results of every combination of defensive strategies (the defender’s “do nothing” strategy of BNCH was excluded from the combinations). In this new game, the application of these strategies was not dependent on the order they are applied. In our ongoing work, we further consider how the order of operations may lead to refinement and reduction in false alarm rates. When playing this game, each strategy is a set of countermeasures, and the set is referred to and analyzed as a single defender strategy.\(^{24}\) The result of assembling these sets was a large number of strategy sets that improve total detection to 98%; all such sets included BOOL. False alarm rates of the countermeasure sets depict the same variation between 0% and 50%.\(^{25}\)

\(^{24}\) The same technique could be applied to consider circuits wherein the adversary has employed multiple exploit strategies. Each combination of strategies would be treated in the game as a single strategy.

\(^{25}\) Detailed analysis of this and other game formulations will be released in [33].
The solution to the game for the adversary—when played in the Consumer and Network economies—is again to play the GLST Trojan nearly 100% of the time while playing STSQ a vanishingly small amount of time. The defender will continue to play SCOA 70% of the time and STRC 30%. This defender prescription may seem surprising that despite adding countermeasure sets that proved to have an increase in $P_D$. Again, the sets with the highest $P_D$ included BOOL—the best overall detector in total results—as well as STRC and/or SCOA—the best methods to play against the GLST Trojan. One of these combinations would seem intuitively to be the best play. Against a rational adversary, however, they do not improve the outcome for the defender. In fact, they make it worse by adding an expensive detection method—BOOL—that does not improve the outcome of the game against an adversary committed by their own rational best interest to play GLST. In this way, game theory provides an answer that may not be intuitively obvious to a defender who is not a strategist and/or an expert in hardware Trojan detection.\(^{26}\)

6 Ongoing work and applications

The games and methods illustrated in this work point the way towards how to develop industry-guiding recommendations of optimal HTH detection strategies. The experimental results in this work should not themselves be taken as industry recommendations, since we tested a limited set of HTH’s and detection methods arranged in a simple taxonomy in one step of the design lifecycle and used estimates for certain economic variables. To accomplish industry-level recommendations for HTH detection, we continue this research in a variety of new directions. Furthermore, the security economic utility functions in this work provide a foundation upon which to explore both more complex utility functions—including those that question the rationality of the players—and more detailed gameplay models. The work here also enables us to explore adversary/defender interactions both outside the context of FPGAs—and entirely outside the context of hardware Trojans. We are beginning to explore software vulnerabilities, system vulnerabilities, and even fields such as tamper and counterfeiting.

The first category of our ongoing research aims to create games out of a significant number of strategies across a significant number of test articles to claim statistical relevance sufficient to guide industry. This research area includes creating a large, well-structured hardware Trojan test article database, advancing our test harness that automates the application of detection

\(^{26}\) An expert may have already been able to come to this conclusion for this game for the same reasons as provided in the post-game analysis. It is the glaring omission in BOOL’s ability to see GLST Trojans that drives many of the game outcomes in our experiment, and that omission may have been detected by an astute reader upon first review of Table 5 and Table 6. However, for games that get larger than those in our examples or games whose detection countermeasures do not have such obvious blind spots, the methodology depicted here will provide value even to an expert.
methods to every HTH in the database, developing user software to simplify the use of GameRunner (hiding any game theoretic variables from standard users), and hosting the framework in a multi-user cloud. In a second ongoing work focus, we are advancing the theory that underlies the decision making, both by improving the utility functions and game models as well as questioning whether alternative decision-making processes are possible. We are looking at sequential games and repeated play and considering how information gained through game analysis performed at one step of the design process can be passed forward to influence subsequent steps. We are also considering alternative solution concepts, such as quantal response equilibrium (QRE) and subrationality models. Finally, we are exploring applications beyond HTH detection in FPGAs that can be addressed by the methodologies presented in this work. To this end, we are developing two general games: a prevention game and a detection game. The Trust game illustrated in this paper is a detection game. With these two general games, we aim to expand the application of GameRunner and our game-theory-based analysis to new security domains. Finally, how the solutions of these games should be combined with knowledge gained from other metrics systems (e.g., coverage metrics) to prescribe industry best practices is our ultimate goal.

7 Conclusions

We have presented a practical game theoretic approach to the selection of hardware Trojan detection strategies. We have demonstrated that utility functions may be constructed to represent the real-world beliefs of the players in the game, and that games may be constructed that represent the points in which realistic adversaries might face each other. We architected and implemented a tool, GameRunner, to solve and explore the solutions of the large, complex games that result from these interactions. An experiment was performed on a reasonable data set using well-considered adversaries to demonstrate both the models and the GameRunner tool’s ability to prescribe and automate optimal defender actions. Future work has been defined for how this tool can be applied at a much larger scale, paving the way for using methods such as this to determine industry-level optimal decision making in a variety of security domains. This work, along with its cited contemporaries, contributes to a promising trend towards developing quantifiable means of applying security assurances in microelectronics.

Acknowledgments This material is based upon work supported by the United States Air Force under Contract No. FA8650-17-C-1148 and was cleared for public release under case number 88ABW-2019-1182. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the United States Air Force.

Open Access This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article’s Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article’s Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit http://creativecommons.org/licenses/by/4.0/.

References

1. Mei KCY (1974) Bridging and stuck-at faults. IEEE Trans Comput C-23(7):720–727
2. Fazzari S (2018) “Hardware security: a history lesson,” in Trusted and assured MicroElectronics forum
3. Graf J and Athanas P (2008) “How threats drive the development of secure reconfigurable devices,” in 2008 IEEE National Aerospace and Electronics Conference, pp. 239–245
4. Graf J (2016) Toward optimal hardware Trojan detection through security economics and game theory. In: GOMACTech
5. Graf J (2016) “Trust games: How game theory can guide the development of hardware Trojan detection methods,” in 2016 IEEE International Symposium on Hardware Oriented Security and Trust (HOST), pp. 91–96
6. Graf J (2016) “Towards system-level adversary attack surface modeling for microelectronics trust,” in 2016 IEEE National Aerospace and Electronics Conference (NAECON) and Ohio Innovation Summit (OIS), pp. 474–477
7. Graf J (2017) OpTrust: software for determining optimal test coverage and strategies for trust. In: GOMACTech
8. Xiao K, Forte D, Jin Y, Karri R, Bhunia S, Tehranipoor M (2016) Hardware Trojans: lessons learned after one decade of research. ACM Trans Des Autom Electron Syst 22(1):6:1–6:23
9. Tehranipoor M, Koushanfar F (2010) A survey of hardware Trojan taxonomy and detection. IEEE Design Test of Computers 27(1):10–25
10. Rostami M, Koushanfar F, Karri R (2014) A primer on hardware security: models, methods, and metrics. Proc IEEE 102(8):1283–1295
11. Roy S, Ellis C, Shiva S, Dusgupta D, Shandilya V, and Wu Q (2010)”A survey of game theory as applied to network security,” in 2010 43rd Hawaii International Conference on System Sciences, pp. 1–10
12. Kamhoua CA, Rodriguez M, and Kwiat KA (2014) “Testing for hardware Trojans: a game-theoretic approach,” in Decision and Game Theory for Security, pp. 360–369
13. Kamhoua CA, Zhao H, Rodriguez M, Kwiat KA (2016) A game-theoretic approach for testing for hardware Trojans. IEEE Transactions on Multi-Stage Computing Systems 2(3):199–210
14. Smith AM, Mayo JR, Kammler V, Armstrong RC, and Vorobeychik Y (2017) “Using computational game theory to guide verification and security in hardware designs,” in 2017 IEEE International Symposium on Hardware Oriented Security and Trust (HOST), pp. 110–115
15. Galiardi M et al (2017) On modeling detection for quantitative trust analysis. In: GOMACTech
16. Saad W, Sanjab A, Wang Y, Kamhoua CA, Kwiat KA (Sep. 2017) Hardware Trojan detection game: a Prospect-theoretic approach. IEEE Trans Veh Technol 66(9):7697–7710
17. Charles River Analytics (2018) “How to avoid malice using linguistics-inspired exploit testing.” [Online]. Available: https://www.cra.com/work/case-studies/hamlet
18. Kimura AG and Bihy SB (2016) “Quantifying error payload and error implementation cost for hardware assurance,” in GOMACTech
19. Anderson R, Moore T (Oct. 2006) The economics of information security. Science 314(5799):610–613
20. Gordon LA, Loeb MP (2002) The economics of information security investment. ACM Trans Inf Syst Secur 5(4):438–457
21. Wilt DP, Metzler RC, DeVale J (2008) Metrics for trust in integrated circuits. In: GOMACTech
22. DARPA (2006) “Trusted Integrated Circuits (TRUST) (Archived),” [Online]. Available: https://www.darpa.mil/program/trusted-integrated-circuits
23. Collins DR (2007) “DARPA ‘Trust in IC’s’ Effort,” [Online]. Available: https://apps.dtic.mil/dtic/tr/fulltext/u2/a503809.pdf
24. Tsgourias N (2012) Cyber attacks, self-defence and the problem of attribution. J Confl Secur Law 17(2):229–244
25. USDOT (2016) “Revised departmental guidance on valuation of a statistical life in economic analysis,” [Online]. Available: https://www.transportation.gov/office-policy/transportation-policy/revised-departmental-guidance-on-valuation-of-a-statistical-life-in-economic-analysis
26. USEPA (2018) “Mortality risk valuation,.” [Online]. Available: https://www.epa.gov/environmental-economics/mortality-risk-valuation
27. Honner P (2018) “Why winning in rock-paper-scissors (and in life) isn’t everything,.” [Online]. Available: https://www.quantamagazine.org/the-game-theory-math-behind-rock-paper-scissors-20180402/
28. Mc Kelvey RD, Palfrey TR (1995) Quantal Response Equilibria for Normal Form Games. Games Econ Behav 10(1):6–38
29. Selten R (1975) Reexamination of the perfection concept for equilibria points in extensive games. Int J Game Theory 4(1):25–55
30. Varian HR, “Revealed preference.” [Online]. Available: http://people.ischool.berkeley.edu/~knbsluh/Papers/2005/revpref.pdf
31. Kar D, Fang F, Delle Fave F, Sintov N, and Tambe M (2015) “A game of thrones: when human behavior models compete in repeated stackelberg security games,” in Proceedings of the 2013 International Conference on Autonomous Agents and Multiagent Systems, Istanbul, Turkey, pp. 1381–1390
32. Johnson B, Laszka A, Grossklags J, Vasek M, and Moore T (2014) “Game-theoretic analysis of DDOS attacks against Bitcoin mining pools,” in Financial Cryptography and Data Security, Berlin, Heidelberg, pp. 72–86
33. Graf J, “Optimizing programmable logic design security strategies,” Doctoral Dissertation, available in May 2019. [Online]. Available: https://vtechworks.lib.vt.edu/handle/10919/89920
34. Digilent, “NetFPGA-SUME Virtex-7 FPGA Development Board,“ 2018. [Online]. Available: http://www.digilentinc.com/netfpga-sume-virtex-7-fpga-development-board/
35. Payscale.com, “Design Verification Engineer Cost of Living in Blacksburg, Virginia.” [Online]. Available: https://www.payscale.com/cost-of-living-calculator/Virginia-Blacksburg/—Design-Verification-Engineer
36. W. C. J. A. A. C. R. LaFleur, “2017 Government Contractor Survey,” 2017. [Online]. Available: https://www.granthornton.com/-/media/content-page-files/public-sector/pdi/surveys/2018/2017-government-contractor-survey
37. Amazon.com, “Amazon EC2 Pricing.” 2018. [Online]. Available: https://aws.amazon.com/ec2/pricing/on-demand/
38. Salmoni H, Tehraniipoor M, and Karri R (2013) “On design vulnerability analysis and trust benchmarks development,” in 2013 IEEE 31st International Conference on Computer Design (ICCD), pp. 471–474
39. Shakya B, He T, Salmoni H, Forte D, Bhunia S, Tehraniipoor M (2017) Benchmarking of hardware Trojans and maliciously affected circuits. J Hardware Syst Secur 1(1):85–102
40. GAO (2018) “Weapon system cybersecurity: DOD just beginning to grapple with scale of vulnerabilities,” [Online]. Available: https://www.gao.gov/products/GAO-19-128
41. Slayback SM (2015) “A computer Scientist’s evaluation of publically available hardware Trojan benchmarks,” Naval Postgraduate School
42. DARPA (2018) “CEP - Common Evaluation Platform (v1.2).” [Online]. Available: https://github.com/niit-ICEP
43. Tang A, Sethumadhavan S, and Stolfo S (2017) “CLKSCREW: exposing the perils of security-oblivious energy management,” in 26th USENIX security symposium
44. Tang A, Sethumadhavan S, Stolfo S (2018) Motivating security-aware energy management. IEEE Micro 38(3):98–106
45. Salmoni H (2017) COTD: reference-free hardware Trojan detection and recovery based on controllability and Observability in Gatelevel Netlist. Trans Info For Sec 12(2):338–350
46. Marlow R, Harper S, Batchelor W, and Graf J (2018) “Hardware Trojan Detection using Xilinx Vivado,” in NAECON 2018 - IEEE National Aerospace and Electronics Conference, pp. 86–91
47. Seth SC, Agraval VD, and Farhat H (1989) “A theory of testability with application to fault coverage analysis,” in [1989] Proceedings of the 1st European Test Conference, pp. 139–143
48. Pedregosa F et al (2011) Scikit-learn: machine learning in Python. J Mach Learn Res 12:2825–2830
49. Dabrowski A, Fejes P, Ullrich I, Krombholz K, Hobel H, Weippl E (2014) Poster: hardware Trojans – detect and react! In: Network and distributed system security (NDSS) symposium
50. A. Dabrowski, Hobel, J. Ullrich, I. Krombholz, and E. Weippl, “Towards a Hardware Trojan Detection Cycle,” in 2014 Ninth international conference on availability, Reliability and Security, 2014, pp. 287–294
51. Edwards M, “Trusted Silicon Stratus (TSS) Workshop,” 2010. [Online]. Available: https://apps.dtic.mil/dtic/tr/fulltext/u2/a540791.pdf
52. Nimbis, “Trusted Silicon Stratus: Joint Federated Assurance Center Distributed Transition Environment.” [Online]. Available: https://www.trustedstratus.org/
53. Jenkins, Build great things at any scale,” 2018. [Online]. Available: https://jenkins.io/
54. SageMath, [Online]. Available: http://www.sagemath.org/
55. Avis D and Jordan C (2015) “Comparative computational results for some vertex and facet enumeration codes,” CoRR, vol.abs/1510.02545
56. Avis D (2018) “Irs home page,”. [Online]. Available: http://cgem.cs.mcgill.ca/ibnspavis/C/irs.html
57. Avis D (1999) “Irs: a revised implementation of the reverse search vertex enumeration algorithm,”. [Online]. Available: http://cgem.cs.mcgill.ca/ibnspavis/doc/avis/Av99a.pdf
58. Chakraborty RS, Bhunia S (2009) HARPOON: an obfuscation-based SoC design methodology for hardware protection. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 28(10):1493–1502
59. Chakraborty RS and Bhunia S (2010) “RTL hardware IP protection using key-based control and data flow obfuscation,” in 2010 23rd International Conference on VLSI Design, pp. 405–410
60. Chakraborty RS, Bhunia SJ (2011) Security against hardware Trojan attacks using key-based design obfuscation. J Electron Test 27:767
61. Rajendran J, Pino Y, Sinanoglu O, and Karri R (2012) “Security analysis of logic obfuscation,” in DAC Design Automation Conference 2012 , pp. 83–89
62. Li L and Zhou H (2013) “A vertex enumeration algorithm,”. [Online]. Available: http://cgm.cs.mcgill.ca/ibnspavis/lrs/a540791.pdf
63. Chakraborty RS (2018) “A computer Scientist’s evaluation of publically available hardware Trojan benchmarks,” Naval Postgraduate School
63. Vaidyanathan K, Das BP, Sumbul E, Liu R, and Pileggi L (2014) “Building trusted ICs using split fabrication,” in 2014 IEEE International Symposium on Hardware-Oriented Security and Trust (HOST), pp. 1–6
64. Garg S, Gentry C, Halevi S, Raykova M, Sahai A, Waters B (2016) Candidate indistinguishability obfuscation and functional encryption for all circuits. SIAM J Comput 45(3):882–929
65. Zhang J (2016) A practical logic obfuscation technique for hardware security. IEEE Transactions on Very Large Scale Integration (VLSI) Systems 24(3):1193–1197
66. Tian J, Reddy GR, Wang J, Swartz W, Makris Y, and Sechen C (2017) “A field programmable transistor array featuring single-cycle partial/full dynamic reconfiguration,” in Design, Automation Test in Europe Conference Exhibition (DATE), 2017, pp. 1336–1341
67. Yu Q, Dofe J, Zhang Z, and Kramer S (2018) “Hardware obfuscation methods for hardware Trojan prevention and detection,” in The Hardware Trojan War, B. S. and T. M., Eds. Springer, pp. 291–325

Publisher’s note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.