

Optimizing Programmable Logic Design Security Strategies

Jonathan P. Graf

Dissertation submitted to the faculty of the Virginia Polytechnic Institute and State University in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

In

Computer Engineering

Peter M. Athanas, Chair

Ezra A. Brown

Mark T. Jones

Cameron D. Patterson

Joseph G. Tront

April 29, 2019

Blacksburg, Virginia

Keywords: FPGA, trust, design security, design integrity, design confidentiality, trustworthy computing, game theory

Copyright 2019 Jonathan P. Graf

Optimizing Programmable Logic Design Security Strategies

Jonathan P. Graf

(ABSTRACT)

Abstract – A wide variety of design security strategies have been developed for programmable logic devices, but less work has been done to determine which are optimal for any given design and any given security goal. To address this, we consider not only metrics related to the performance of the design security practice, but also the likely action of an adversary given their goals. We concern ourselves principally with adversaries attempting to make use of hardware Trojans, although we also show that our work can be generalized to adversaries and defenders using any of a variety of microelectronics exploitation and defense strategies. Trojans are inserted by an adversary in order to accomplish an end. This goal must be considered and quantified in order to predict the adversary's likely action. Our work 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 enables mathematical and graphical exploration of “what if” scenarios in the context of the game. GameRunner can also issue “prescriptions,” sets of commands that allow the defender to automate the application of the optimal defender strategy to their circuit of concern. We also present how this work can be extended to adjacent security domains. Finally, we include a discussion of future work to include additional software, a more advanced experimental framework, and the application of irrationality models to account for players who make subrational decisions.

Optimizing Programmable Logic Design Security Strategies

Jonathan P. Graf

(GENERAL AUDIENCE ABSTRACT)

We present a security economic model that informs the optimal selection of programmable logic design security strategies. Our model accurately represents the economics and effectiveness of available design security strategies and accounts for the varieties of available exploits. Paired with game theoretic analysis, this model informs microelectronics designers and associated policy makers of optimal defensive strategies. Treating the adversary and defender as opponents in a two-player game, our security economic model tells us how either player will play if it is known in advance how their opponent plays. The additional use of game theory allows us to determine the optimal play of both players simultaneously without prior knowledge other than models of players beliefs.

Dedication

This work is dedicated to my wife, Leanna, and my son, Jack. Thank you for supporting me during the early mornings, late nights, and weekends that were required for Daddy to get his “doctor degree” and fulfill a life goal.

Acknowledgements

In addition to Leanna and Jack, heartfelt thanks are due to many others:

To Peter Athanas, who, for the 15 years after advising my master's degree studies, kept cheerfully prodding me to finish my doctoral dissertation. Your optimism continued when a sponsor declined to let me publish my original topic in 2008 as well as when I chased by professional goals instead of my academic ones. Your assessment in 2016 that my professional research concepts could be crafted to fulfill my dissertation requirements pushed me over the edge to re-commit to finishing. Without your encouragement, I never would have attempted something as challenging (foolhardy?) as trying to finish a doctorate while also starting a business. Thank you for being unrelentingly optimistic! There are so many other things to thank you for – the training you provided in your classes and while I worked in your lab, the time you always provide when asked, and the example you set for how to help people grow by giving them the tools to explore challenging ideas in a fun and creative environment. The list could go on. I am sincerely grateful.

To my committee, Ezra Brown, Mark Jones, Cameron Patterson, and Joseph Tront. Thank you for the many hours spent discussing ideas in your offices, refining concepts in document drafts, and philosophizing over coffee.

To Scott Harper, my most frequent collaborator in various capacities at Virginia Tech, Luna Innovations, MacAulay-Brown, and Graf Research. We have worked together on more projects than can be named – including this one – so suffice it to say that I am better both professionally and personally for your collaboration and friendship. I am deeply grateful for both.

To my Graf Research colleagues who collaborated directly with me on this work, Whitney Batchelor, Ed Carlisle, and Ryan Marlow. Your contributions to this work – ideas, refinements, corrections, conversations, travel, and software development – made it far better. It was also far more fun to complete due to the presence of each of you on the research team. Among the many contributions you each made in our collaboration, let me highlight a few for which I am specifically grateful. Whitney: thank you for exploring the background and related work with me, for contributions to the subrationality discussion, and for helping me properly express many of the mathematical ideas written here. Ed: thank you for work implementing the GameRunner software, for the weeks spent back and forth refining the game visualization techniques to their present form, and for performing empirical Trojan testing. Ryan: thank you for your work developing Trojan detection techniques and automating Trojan detection testing. Again, I am deeply grateful to each of you.

To my Graf Research colleagues who collaborated with me on closely related research programs, Ali Asgar Sohangpurwala, Timothy Dunham, Michael Capone, and Alan Cook. Thank you for the privilege of working with you, exchanging ideas, refining concepts, and creating new technologies to solve interesting microelectronics security challenges. Solving problems with creative teams of bright people like you is the joy of my career; thank you for your contributions to such a fulfilling professional experience. Thank you also for the friendship and fun that have accompanied the R&D.

To my former colleagues at Luna and MacAulay-Brown, with whom I collaborated on both large DARPA and Air Force programs and innumerable small projects related to microelectronics security. In addition to my named co-authors – Scott Harper, Lee Lerner, Stephen Craven, Barry Polakowski, Stephen Baka, Kevin Urish, John Hallman, Brian Knight, and others – there are too many brilliant innovators and software developers who made major contributions both to projects and my career to name. Thank you all for the personally and intellectually fulfilling journey together while collaborating on those projects. Thank you also for all the enormously fun conversations over beer.

To Barry Polakowski, my boss and collaborator at Luna Innovations and MacAulay-Brown. Thank you for your support of my early doctoral coursework and my career. I am grateful for the professional growth experiences of our collaboration and for the fond

memories from our many travels working on microelectronics security projects, crisscrossing the country to share what our R&D teams accomplished.

To Luna Innovations, thank you for sponsoring the coursework for my doctorate way back in the mid-aughts.

Finally, to my sponsors across multiple branches of the United States Department of Defense who supported my research. Thank you for the privilege of working for and with the very best in order to make our small contribution to make our country and the world just a little bit better. While there are too many to name, two deserve special thanks in the context of this work. To Dean Collins, thank you for your support of my team's early ideas in microelectronics trust. It was a privilege to work for you in the crucible of the DARPA Trust program, which not only refined early trust ideas but also set my career direction. To Matthew Casto, sponsor of portions of this work, it is a rare pairing to be sponsored by an ambitious researcher working on a mid-career PhD while attempting the same myself. Thank you for your support and congratulations on finishing!

This work was performed partially under internal research funding from the Graf Research Corporation and partially under funding from the United States Air Force. The portions of this work derived from publications [9] and [10] are based upon work supported by the United States Air Force under Contract No. FA8650-17-C-1148 and released under case numbers and 88ABW-2019-1182 and 88ABW-2018-4079, respectively. 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 or the Graf Research Corporation.

Contents

| | |
|--|--------------|
| List of Figures | xi |
| List of Tables | xv |
| List of Equations | xvii |
| List of Acronyms | xviii |
| Chapter 1. Introduction | 1 |
| 1.1 Summary of Contributions..... | 4 |
| 1.2 Document Organization | 7 |
| 1.3 Related Contributions | 9 |
| 1.4 Summary of Sponsorship..... | 10 |
| Chapter 2. Background | 12 |
| 2.1 Attack and Defense | 15 |
| 2.1.1 Trust Background..... | 15 |
| 2.1.2 FPGA Anti-Tamper Background..... | 25 |
| 2.2 Metrics, Strategies, and Games..... | 29 |
| 2.2.1 Models and Metrics for IC Trojans..... | 29 |
| 2.2.2 Game Theory | 33 |

| | |
|---|-----------|
| Chapter 3. Security Economics and Game Theory..... | 39 |
| 3.1 Players..... | 40 |
| 3.2 Collecting Variables of Concern..... | 41 |
| 3.2.1 Strategies..... | 41 |
| 3.2.2 Probabilities | 42 |
| 3.2.3 Economic Variables | 43 |
| 3.3 Assembling Utility Functions | 46 |
| 3.4 Step Games | 47 |
| 3.5 Games and Solution Concepts | 49 |
| 3.5.1 Player Rationality..... | 50 |
| 3.5.2 Computing Game Solutions | 52 |
| 3.6 Example Trust Game | 52 |
| 3.6.1 Playing the Trust Game | 53 |
| 3.6.2 Solving Related Games..... | 57 |
| Chapter 4. Experimental Game..... | 59 |
| 4.1 Game Scenario: Defender and Adversary..... | 59 |
| 4.1.1 Defender and Adversary Economics | 60 |
| 4.1.2 Step Games: HDL and 3PIP | 62 |
| 4.2 Strategies and Probabilities..... | 63 |
| 4.2.1 Adversary Strategies | 64 |
| 4.2.2 Defender Strategies | 68 |
| 4.2.3 Empirically-Derived Probabilities | 72 |
| 4.3 Automation and GameRunner | 74 |
| Chapter 5. Results..... | 79 |
| 5.1 Results and Nash Equilibria..... | 79 |

| | | |
|--|---|------------|
| 5.2 | Analysis..... | 85 |
| 5.2.1 | Expected Utility Theory and the Nash Equilibrium | 85 |
| 5.2.2 | Analysis of Sensitivity to Error..... | 88 |
| 5.2.3 | Further Exploration of Results with GameRunner | 96 |
| 5.3 | Expanding the Defender Strategies: Sets of Countermeasures..... | 107 |
| 5.4 | Discussion: How do we know this works? | 116 |
| Chapter 6. Ongoing Work and Applications..... | | 124 |
| 6.1 | Advancing the Framework..... | 125 |
| 6.1.1 | Hardware Trojan Test Article Database | 125 |
| 6.1.2 | Automated Detection Method Application..... | 126 |
| 6.1.3 | A User Software Application..... | 127 |
| 6.1.4 | OpTrust Framework..... | 127 |
| 6.2 | Advancing the Theory..... | 134 |
| 6.2.1 | Sequential Games and Repeated Play..... | 135 |
| 6.2.2 | Alternative Solution Concepts and Subrational Game Play | 136 |
| 6.2.3 | How Should We Then Play?..... | 141 |
| 6.3 | Applications | 142 |
| 6.3.1 | Direct Application of this Work | 142 |
| 6.3.2 | Applications of the General Detection Game | 144 |
| 6.3.3 | Applications of the General Prevention Game | 147 |
| Chapter 7. Conclusions..... | | 150 |
| 7.1 | Final Contribution Summary | 150 |
| Appendix A: Detection Method Results..... | | 153 |
| Bibliography | | 156 |

List of Figures

| | |
|---|----|
| Figure 1. Maginot Line between France and Germany [3]..... | 3 |
| Figure 2. Attack Surface Available to the APT-Enabled Adversary | 22 |
| Figure 3. Desired Utility of Security Economics/Game Theory for Hardware Trojan Detection Strategy Selection | 40 |
| Figure 4. The ASIC and FPGA Device Games Decomposed into Step Games ... | 48 |
| Figure 5. Glitch State Trojan in RS232 (RS232-TjGlitchState) [10] | 67 |
| Figure 6. SCOA clustering result for circuit RS232-T1200 [10]..... | 70 |
| Figure 7. GameRunner Block Diagram [9]..... | 75 |
| Figure 8. GameRunner Graphical User Interface | 78 |
| Figure 9. Defender Strategies as $Z_D(\text{SCOA})$ sweeps from 50% to 150% of its estimated value; all other variables fixed; HDL Step Game in the Consumer Economy. | 91 |
| Figure 10. Defender Strategies as $Z_D(\text{STRC})$ sweeps from 50% to 150% of its estimated value; all other variables fixed; HDL Step Game in the Consumer Economy. | 92 |
| Figure 11. Defender Strategies as $Z_A(\text{STSQ})$ sweeps from 50% to 150% of its estimated value; all other variables fixed; HDL Step Game in the Consumer Economy. | 94 |

| | |
|--|-----|
| Figure 12. Defender Strategies as $Z_A(\text{GLST})$ sweeps from 50% to 150% of its estimated value; all other variables fixed; HDL Step Game in the Consumer Economy | 95 |
| Figure 13. GameRunner Solving the HDL Step Game in the Kickstarter Economy | 97 |
| Figure 14. Adversary Strategies as L sweeps from \$0 to \$375,000 with $G = L/2$ and $Z_{find} = 2G$; all other variables fixed; HDL Step Game | 98 |
| Figure 15. Defender Strategies as L sweeps from \$0 to \$375,000 with $G = L/2$ and $Z_{find} = 2G$; all other variables fixed; HDL Step Game | 100 |
| Figure 16. Adversary Strategies as $P_D(\text{GLST}, \text{BOOL})$ sweeps from 50% to 100%; all other variables fixed; HDL Step Game in the Consumer Economy | 103 |
| Figure 17. Defender Mixed Strategies as $P_D(\text{GLST}, \text{BOOL})$ sweeps from 50% to 100%; all other variables fixed; HDL Step Game in the Consumer Economy | 104 |
| Figure 18. Adversary Mixed Strategies as $Z_A(\text{GLST})$ sweeps from \$0 to \$750,000; all other variables fixed; HDL Step Game in the Consumer Economy | 106 |
| Figure 19. GameRunner Solving the HDL Step Game, Consumer Economy, Defender Strategies Employed in Countermeasure Sets | 110 |
| Figure 20. Adversary Strategies as Z_{find} sweeps from \$0 to \$20,000,000; all other variables fixed; HDL Step Game with Defender Strategies Employed in Countermeasure Sets | 111 |
| Figure 21. Defender Strategies as Z_{find} sweeps from \$0 to \$20,000,000; all other variables fixed; HDL Step Game with Defender Strategies Employed in Countermeasure Sets | 113 |

Figure 22. Adversary Strategies as $Z_A(\text{GLST})$ sweeps from \$0 to \$750,000; all other variables fixed; HDL Step Game under the Consumer Economy with Defender Strategies Employed in Countermeasure Sets 114

Figure 23. Defender Strategies as $Z_A(\text{GLST})$ sweeps from \$0 to \$750,000; all other variables fixed; HDL Step Game with Defender Strategies Employed in Countermeasure Sets..... 115

Figure 24. GameRunner-Created 3D Graph of Adversary (Red) and Defender (Blue) Utility Function Payoffs under All Possible Mixed Strategies; HDL Game in the Consumer Economy..... 119

Figure 25. GameRunner-Created 2D Graph of Defender Mixed Strategy Sets (Top) and the Resulting Defender Payoffs (Below) When the Adversary Plays the Nash Equilibrium-Selected Optimal Strategy; HDL Game in the Consumer Economy 122

Figure 26. GameRunner-Created 2D Graph of Defender Mixed Strategy Sets (Top) and the Resulting Defender Payoffs (Below) When the Adversary Plays the Nash Equilibrium-Selected Optimal Strategy; HDL Game in the Consumer Economy 123

Figure 27. Red Team Table Generation..... 128

Figure 28. Threat Team Standard Game Generation 131

Figure 29. Separation of Pre-Computation Environment and Developer Environment..... 133

Figure Acknowledgements

Figure 1 was created by Wikimedia user Goran tek-en and was shared at https://commons.wikimedia.org/wiki/File:Maginot_Line_ln-en_svg.svg under Creative Commons Attribution-ShareAlike 3.0 Unported. Please see <https://creativecommons.org/licenses/by-sa/3.0/>). No changes were made to the image.

Figure 5 and Figure 6 are derived from [10], a collaborative work between the author of this dissertation and cited co-authors of [10].

Figure 7 is from [9], a collaborative work between the author of this dissertation and the cited co-authors of [9].

All other figures are the original work of the author.

List of Tables

| | |
|--|----|
| Table 1. The Trust Game in Normal Form | 53 |
| Table 2. Example Adversary Strategies and Costs | 54 |
| Table 3. Example Defender Strategies and Costs | 55 |
| Table 4. Table of $P_D(\sigma_A, \sigma_D)$ Values | 55 |
| Table 5. Table of $P_{FA}(\sigma_D)$ Values..... | 55 |
| Table 6. The Trust Game in Normal Form Showing Tuples of (Adversary Utility, Defender Utility) in \$K's | 56 |
| Table 7. Hypothetical Trust Game Solution via IEDS | 56 |
| Table 8. Game Scenarios | 61 |
| Table 9. Adversary Strategies and Costs | 65 |
| Table 10. Defender Strategies and Costs | 71 |
| Table 11. Three Economies (All Values \$) | 72 |
| Table 12. P_D by Adversary and Defender Strategy | 80 |
| Table 13. P_{FA} by Defender Strategy | 80 |
| Table 14. Two-Player Strategic Game: HDL Step, Kickstarter Economy | 81 |

| | |
|---|-----|
| Table 15. Two-Player Strategic Game: HDL Step, Consumer Economy..... | 82 |
| Table 16. Two-Player Strategic Game: HDL Step, Network Device Economy... | 82 |
| Table 17. Two-Player Strategic Game: 3PIP Step, Kickstarter Economy..... | 83 |
| Table 18. Two-Player Strategic Game: 3PIP Step, Consumer Economy | 84 |
| Table 19. Two-Player Strategic Game: 3PIP Step, Network Device Economy ... | 84 |
| Table 20. P_D by Adversary and Defender Strategy (SCOA, STRC) | 89 |
| Table 21. P_{FA} by Defender Strategy (SCOA, STRC) | 89 |
| Table 22. P_D by Adversary and Defender Strategy with Defender Strategies Employed in Countermeasure Sets | 107 |
| Table 23. P_{FA} by Defender Strategy with Defender Strategies Employed in Countermeasure Sets..... | 107 |
| Table 24. Two-Player Strategic Game: HDL Step, Consumer Economy, Defender Strategies Employed in Countermeasure Sets | 109 |
| Table 25. | 139 |
| Table 26. Raw Detection Method Results by Adversary Strategy Taxonomy Category | 153 |

List of Equations

| | |
|--|-----|
| Eq. 1. Adversary Utility Function..... | 46 |
| Eq. 2. Defender Utility Function..... | 47 |
| Eq. 3. Simplified Adversary Utility Function..... | 53 |
| Eq. 4. Adversary Expected Utility | 86 |
| Eq. 5. Defender Expected Utility | 86 |
| Eq. 6. Possible Rationality-Weighted Adversary Utility Function..... | 140 |
| Eq. 7. Possible Rationality-Weighted Defender Utility Function..... | 141 |
| Eq. 8. Defender Utility Function in the Prevention Game..... | 149 |
| Eq. 9. Adversary Utility Function in the Prevention Game..... | 149 |

List of Acronyms

| | |
|-------|--|
| 3PIP | Third-Party Intellectual Property |
| AES | Advanced Encryption Standard |
| AI | Artificial Intelligence |
| APT | Advanced Persistent Threat |
| ASIC | Application Specific Integrated Circuit |
| ATPG | Automated Test Pattern Generation |
| AWS | Amazon Web Services |
| BLE | Boolean Logic Equivalence (generally) |
| BNCH | Testbench (Trojan defender strategy in this work) |
| BOOL | Boolean Logic Equivalence (Trojan defender strategy in this work) |
| CCMP | Combinational Comparator (Trojan adversary strategy in this work) |
| CEP | DARPA Common Evaluation Platform |
| CPU | Central Processing Unit |
| CSIM | Constrained Random Simulation (Trojan detection method in this work) |
| DARPA | Defense Advanced Research Projects Agency |
| DMEA | Defense Microelectronics Activity |
| DoD | United States Department of Defense |
| DONT | Do Nothing (Trojan adversary strategy in this work) |
| DPA | Differential Power Analysis |
| DSP | Digital Signal Processor |
| DUT | Design/Device Under Test |

| | |
|-------------|--|
| ECTR | Event Counter (Trojan adversary strategy in this work) |
| EDA | Electronic Design Automation |
| EEE | Enumeration of Extreme Equilibria algorithm |
| eFPGA | Embedded Field Programmable Gate Array |
| EUT | Expected Utility Theory |
| FPGA | Field Programmable Gate Array |
| GATE | Change Gates (Trojan adversary strategy in this work) |
| GLST | Glitch State Trojan (Trojan adversary strategy in this work) |
| GNU | GNU's Not Unix! |
| GPU | Graphics Processing Unit |
| GUI | Graphical User Interface |
| HDL | Hardware Description Language |
| HLL | High Level Language |
| HLS | High Level Synthesis |
| HRNG | Hardware Random Number Generator |
| HTH | Hardware Trojan Horse |
| IC | Integrated Circuit |
| IEDS | Iterated Elimination of Dominated Strategies |
| IO | Input/Output |
| IP | Intellectual Property |
| JSON | JavaScript Object Notation |
| LRS | Revised Reverse Search vertex enumeration algorithm |
| MPI | Message Passing Interface |
| MPSoC | Multi-Processor System-on-Chip |
| NP-Complete | Non-Polynomial Complete |
| PUF | Physically Unclonable Function |
| RAM | Random Access Memory |
| REWR | Rewire (Trojan adversary strategy in this work) |
| SEM | Scanning Electron Microscope |
| SCOA | Sandia Controllability Observability Algorithm (Trojan defender strategy in this work) |

| | |
|--------|---|
| SHARP | Stochastic Human behavior model with Attractiveness and Probability weighting |
| SoC | System-on-Chip |
| SRAM | Static Random-Access Memory |
| STRC | Structural detection (Trojan defender strategy in this work) |
| STSQ | State Sequence (Trojan adversary strategy in this work) |
| SU-QRE | Subjective Utility Quantal Response Equilibrium |
| TCL | Tool Command Language |
| TSS | Trusted Silicon Stratus |
| VLSI | Very Large Scale Integration |
| VM | Virtual Machine |

Chapter 1. Introduction

Microelectronics security commonly requires a designer to play the role of defender. This defender must protect components of their design from an adversary who wishes to gain from some aspect of the designer's system at the expense of the designer. These encounters between the adversary and the defender may be modeled as two-player strategic games. From these games, means of optimizing strategies for both players to accomplish their goals may be discovered. This dissertation concerns itself with these encounters, including how to accurately construct game representations of the subject encounters and how to interpret the conclusions of the games to the advantage of each player. These encounters are explored principally in the context of integrated circuit design security, with a focus on Field Programmable Gate Arrays (FPGAs) and the binary files that configure them: bitstreams. We delve in depth into the considerations of a designer who wishes to ensure that they produce an FPGA bitstream design that is free from any unwanted content.

This unwanted content is produced by an adversary who wishes to accomplish an exploitation goal by inserting a Hardware Trojan Horse (HTH). FPGAs present two major HTH domains of concern. The first is the traditional domain, wherein the HTH is a malevolent Hardware circuit constructed during the manufacture of a device. The second domain comprises HTH circuits configured by programming otherwise benign portions of the FPGA hardware through malicious alteration of the bitstream. We also treat, more briefly, the general area of design security for systems that contain bitstreams, other firmware, software, and hardware considerations. Additionally, we posit that similar game models may be used to determine design confidentiality strategies, where the designer

Chapter 1. Introduction

wishes to keep secret certain properties of their design – most often, its implementation details – from an adversary who wishes to extract those properties.

Within our primary focus, we must establish a mechanism for reasoning about Hardware Trojan Horse (HTH) detection strategies. This 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 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].

As in Figure 1, if a defender invests enormous resources in a baseline defensive strategy – but does not consider how their strategy will cause their adversary to react – they risk being vulnerable to an adversary who optimizes their approach to a defender’s known weaknesses. Verification best practices provide our Maginot Line. They are static defenses that consider only those HTH attacks that take the expected form of errors and bugs that can be detected by verification methods. As we will see, if the adversary knows which verification methods are available, they will go to the trouble of creating useful HTH attack vectors that do not conform to the defender’s expectations. They will “invade through the Low Countries” and bypass the defenses.

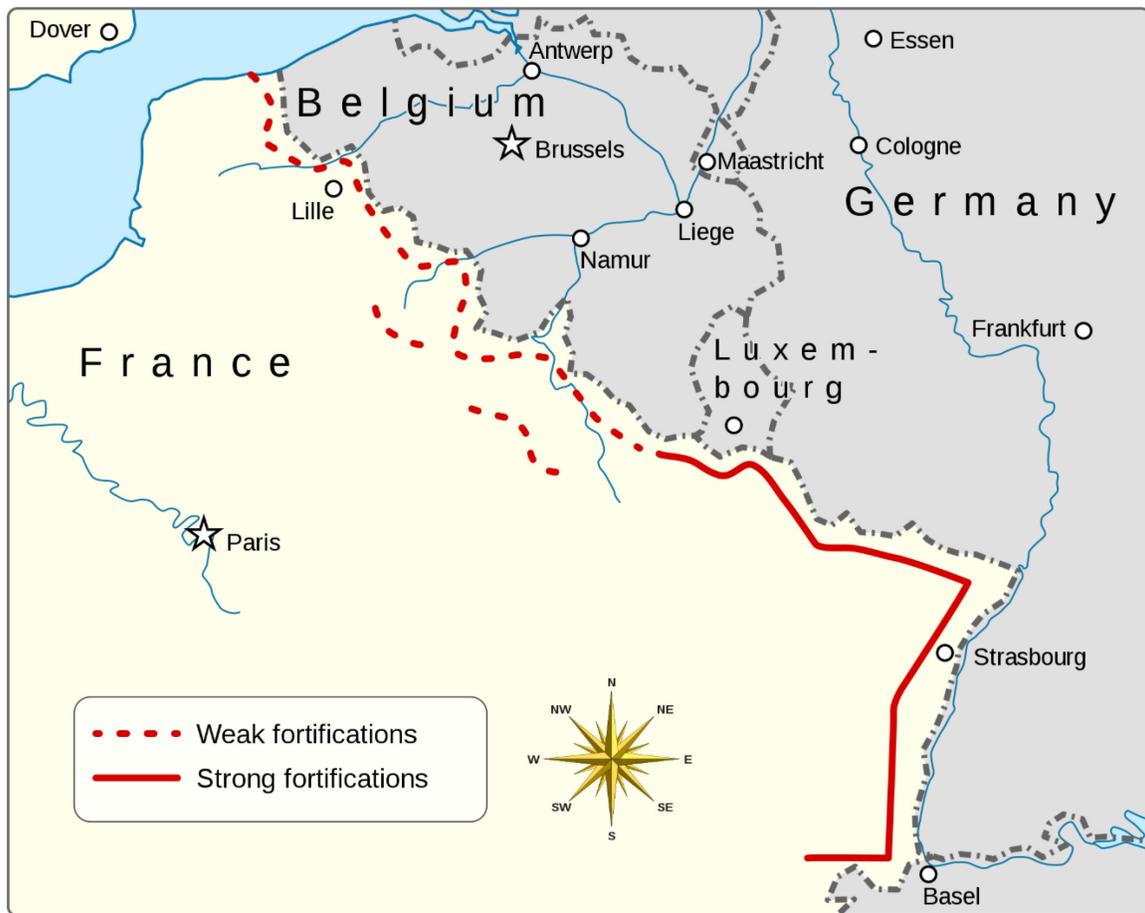


Figure 1. Maginot Line between France and Germany [3]

Given this challenge, a framework for optimal decision making in this adversary-defender interaction is necessary. This work introduces a novel use of game theory to produce the requisite formal, reliable, neutral, and automated framework for quickly optimizing defender strategies [3,5,6,7,8,9]. The games are constructed to consider first the baseline strategies of each player and optimize the selection of adversarial and defensive strategies in that context. We present and refine 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. We further demonstrate its applicability both to systems and other domains of microelectronics security.

1.1 Summary of Contributions

Contribution 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. Prior models applied to this problem ignored the components of adversary and defender beliefs.

Impact: Allows an adversary or a defender to optimize their strategy if they know the strategy of the other party. Because the components of the security economic models are built to allow data from empirical testing of both adversary and defender strategies to directly affect model outcomes, data from both red team testing and real-world encounters may now directly improve our knowledge of adversary and defender strategy elections. This claim is demonstrated through the implementation of a software tool that makes use of these models and a series of experiments in Chapters 3-5.

Contribution 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. The body of work documented in this dissertation holds claim to the first publication that applied two-person strategic games to microelectronics security. The refinement of that model into the FPGA-Trojan-focused game model featured in this work

Chapter 1. Introduction

combines the benefits of the Contribution 1 player models with the ability to solve games to predict those players' actions.

Impact: Allows an adversary or defender to simultaneously predict their opponent's optimal play as well as their own best response. Red team and real-world encounter data can be used to continuously improve these predictions, due to the fidelity of the Contribution 1 models. This claim is demonstrated through the implementation of a software tool that makes use of these models and a series of experiments in Chapters 3-5.

Contribution 3: Improvement to the FPGA Trojan game model to account for the entire attack surface available to an adversary, margin of error in derived and measured game variables, and mixed strategy solutions for adversary and defender. Contemporary works in games for Hardware Trojans consider only 3rd-Party IP and device vendors as potential adversaries; this work considers and models the design and deployment lifecycle of the device as the attack surface available to the adversary. This work also provides software mechanisms to explore game results when model variables are altered.

Impact: The first fully comprehensive model that simultaneously accounts for every adversarial entry point into a bitstream, predicts the optimal adversary behavior, and suggests optimal defense. This claim is demonstrated through the implementation of a software tool that makes use of these models and a series of experiments in Chapters 3-5.

Contribution 4: Universal game models for microelectronics security for activities that involve the detection and prevention of malice. The models allow any microelectronics security encounter that involves an adversary and a defender to be structured as a 2-person strategic game, including in the context of design integrity, design security, anti-counterfeiting, and other scenarios. To our knowledge, this dissertation marks the first attempt in literature to express general microelectronics security game models that can be applied to this range of solutions.

Impact: Demonstrates the broad application of the game theoretic approach to other adversary/defender interactions in microelectronics security. The models that

Chapter 1. Introduction

satisfy this claim are documented as the general system game, general detection game, and general prevention game in Chapter 6.

Contribution 5: An automated method of determining and applying optimal detection strategies for hardware Trojan detection, based on the models of Contributions 1-3. This contribution is evidenced by a software guidance tool, GameRunner, that maintains security economic metrics and game models, provides guidance based on input adversary and defender strategy metrics, and provides means of quantitatively demonstrating the claimed improvement on outcomes. To our knowledge, this is the first such guidance method that automates the application of optimal strategies for hardware Trojan detection.

Impact: Provides automated guidance of optimal defensive strategies without requiring the user to have special knowledge of security economics or game theory, enabling broad application. This automation obviates the need for users of the tool to understand the game theory behind the decisions made by the tool. They must simply apply the guidance by running the script. This claim is demonstrated through the implementation of GameRunner and a series of experiments in Chapters 3-5.

Contribution 6: Novel visualization techniques which allow for the exploration of the investment-security state space within the context of complex adversary-defender engagements. This contribution is evidenced by GameRunner, a software program that can be used to depict such strategies and their benefits in exploring domain-specific complex games. Prior software tools for game solving are general game solvers.

Impact: Provides a simplified way of reasoning about where investment should be made to improve outcomes for either opponent while abstracting the advanced mathematics underlying the tool behind a graphical interface. The visualizations that result from this analysis allow for the ability to explore the impact of individual variables on the mixed strategy results leads to useful observations, including visual cues of where the influence of individual variables begin to dominate the decision making of the players. This claim is demonstrated through a series of analyses in Chapter 5.

1.2 Document Organization

This document is organized as follows.

Chapter 2. Background. In this chapter, we cover the background of general microelectronics design security practices, FPGA-specific design security practices, and game theory as applied to microelectronics security.

Chapter 3. Security Economics and Game Theory. In this chapter we establish the security economic and game theoretic underpinnings of our model. We discuss our rationale for the construction of games and the selection of game solution concepts, including insights on correct game construction. We explore the variables of concern, the utility functions, and the concept of “step games” used in our model. We briefly treat player rationality and mechanisms for computing game solutions. We include a set of example game solutions, demonstrating the use of the iterated elimination of dominated strategies (IEDS) and Nash equilibrium solution concepts. This “by hand” solution concept illustrates the value of the game in a simplified scenario prior to the more in-depth assessments of the following chapters. Portions of this chapter were published as the following works:

- [5] **J. Graf**, “Toward Optimal Hardware Trojan Detection through Security Economics and Game Theory,” in *GOMACTech 2016 Proceedings*, March 2016, paper 14.4.
- [6] **J. Graf**, “Trust games: How Game Theory Can Guide the Development of Hardware Trojan Detection Methods,” in *Hardware Oriented Security and Trust (HOST), 2016 IEEE International Symposium on*, May 2016.

The publication of these works contributed to invitations to present the following invited lectures derived in part from those works:

- [20] **J. Graf**, “Optimizing Forward Design Trust for FPGAs,” invited lecture delivered to *Single Event Effects Symposium / Military and Aerospace Programmable Logic Devices Workshop*, San Diego, CA, May 25, 2017.
- [23] **J. Graf**, “Measuring Trust,” invited lecture delivered to *Single Event Effects Symposium / Military and Aerospace Programmable Logic Devices Workshop*, San Diego, CA, May 24, 2018.

Chapter 1. Introduction

- [24] **J. Graf**, “Optimal Trust Strategies,” invited lecture delivered to the National Academy of Sciences for the *National Academies Study on Secure and Reliable Microelectronics for AF Systems*, Washington, DC, June 19, 2018.

Chapter 4. Experiment. In this chapter, we present an experiment conducted to demonstrate the practical value of the game, illustrating the construction of player strategies, the use of game variables, the design of our game solving tool, and its automated solving functions.

Chapter 5. Results. In this chapter, we discuss the results of this experiment as well as the GameRunner software we created for exploring game solutions and directing defensive responses. We use GameRunner to explore several “what-if” scenarios that emerge from the game in the experiment. We include an analysis of whether we can be confident that this approach provides value. The discussion in Chapters 4 and 5 were published as:

- [9] J. Graf, W. Batchelor, S. Harper, R. Marlow, E. Carlisle, and P. Athanas, “A Practical Application of Game Theory to Optimize Selection of Hardware Trojan Detection Strategies,” manuscript submitted to the *Journal of Hardware and Systems Security*, 2019.
- [10] R. Marlow, S. Harper, W. Batchelor, and J. Graf, “Hardware Trojan Detection using Xilinx Vivado,” in *2018 IEEE National Aerospace and Electronics Conference*, 2018.

Chapter 6. Ongoing Work and Applications. In this chapter we discuss the continuing work that has resulted from the above-documented research and results. This includes exploration of subrational game play and applications to a variety of new domains of microelectronics security. While many publications are planned related to this work as it continues, the following papers have been published thus far based on the work described in this chapter:

- [3] J. Graf and P. Athanas, “How Threats Drive the Development of Secure Reconfigurable Devices,” in *2008 IEEE National Aerospace and Electronics Conference*, 2008, pp. 239–245.

- [7] J. Graf, “Towards system-level adversary attack surface modeling for microelectronics trust,” in *2016 IEEE National Aerospace and Electronics Conference (NAECON) and Ohio Innovation Summit (OIS)*, 2016, pp. 474–477.
- [8] J. Graf, “OpTrust: Software for Determining Optimal Test Coverage and Strategies for Trust,” in *GOMACTech 2017 Proceedings*, March 2017.

Chapter 7. Conclusions. In this chapter, we present our concluding thoughts and speculation about future directions.

1.3 Related Contributions

Prior to this work, the author has made other contributions to the field of secure microelectronics. A subset of this work is cited in this publication, and is collected here for the committee’s consideration:

- [10] **J. Graf** and P. Athanas, “A key management architecture for securing off-chip data transfers,” in *Field Programmable Logic and Application, ser. Lecture Notes in Computer Science*, J. Becker, M. Platzner, and S. Vernalde, Eds. Springer Berlin Heidelberg, 2004, vol. 3203, pp. 33–42.
- [12] A. J. Mahar, P. M. Athanas, S. D. Craven, J. N. Edmison, and **J. Graf**, Design and Characterization of a Hardware Encryption Management Unit for Secure Computing Platforms,” in *39th Hawaii International International Conference on Systems Science (HICSS-39 2006), CD-ROM / Abstracts Proceedings, 4-7 January 2006, Kauai, HI, USA*, 2006.
- [13] P. Athanas, J. Bowen, T. Dunham, C. Patterson, J. Rice, M. Shelburne, J. Suris, M. Bucciero, and **J. Graf**, “Wires on demand: Run-time communication synthesis for reconfigurable computing,” in *2007 International Conference on Field Programmable Logic and Applications*, Aug 2007, pp. 513–516.
- [14] **J. Graf**, J. Hallman, and S. Harper, “Trust in the FPGA Supply Chain using Physically Unclonable Functions,” in *GOMACTech 2010 Proceedings*, March 2010, paper 22.4, pp. 317–319.

- [15] **J. Graf**, S. Harper, and L. Lerner, “Ensuring Design Integrity through Analysis of FPGA Bitstreams and IP cores,” in *Proceedings of the International Conference on Engineering of Reconfigurable Systems and Algorithms (ERSA)*, 2012.
- [16] **J. Graf**, S. Harper, and L. Lerner, “The Integrity of FPGA Designs: Capabilities Enabled by Unlocking Bitstreams and 3rd-Party IP,” in *GOMACTech 2012 Proceedings*, March 2012, paper 18.3, pp. 201–204.
- [17] **J. Graf**, S. Harper, J. Hallman, and B. Knight, “Managing Risk to Field Programmable Gate Array Trust: A Deployment Framework for DoD Instruction 5200.44,” in *GOMACTech 2014 Proceedings*, March 2014, paper 6.1, pp. 77–80.
- [18] S. Baka, J. Hallman, S. Harper, and **J. Graf**, “Trust and Reuse of 3rd-Party IP,” in *GOMACTech 2014 Proceedings*, March 2014, paper 2.4, pp. 25–28.
- [19] K. Urish and **J. Graf**, “Mitigation of Space-Reliability Reduction Trojans in FPGA Designs,” in *GOMACTech 2015 Proceedings*, March 2015, paper 7.1, pp. 91–94.
- [20] **J. Graf** and A. A. Sohanghpurwala, “Private Verification for FPGA Bitstreams,” in *GOMACTech 2017 Proceedings*, March 2017.
- [21] S. Harper, **J. Graf**, W. Batchelor, T. Dunham, and P. Athanas, “Introducing a Trust Metric Foundation and Deriving Trust-for-Buck,” in *GOMACTech 2019 Proceedings*, March 2019.

1.4 Summary of Sponsorship

This work has been sponsored by the United States Air Force under a series of Small Business Innovative Research contracts awarded to the Graf Research Corporation:

- AF161-150, Optimal Strategies for Cloud-Based Trust Assessment, Phase 1, contract FA8650-16-M-1808, June 14, 2016 – March 13, 2017; \$149,998; Principal Investigator: Jonathan Graf.
- AF161-150, Optimal Strategies for Cloud-Based Trust Assessment, Phase 2, contract FA8650-17-C-1148, September 28, 2017 – December 30, 2019, \$749,740; Principal Investigator: Jonathan Graf.

Chapter 1. Introduction

- AF161-150, Optimal Strategies for Cloud-Based Trust Assessment, Phase 3, contract in negotiations #TBD, 2019 – 2026 (estimated), \$49.9M (estimated); Principal Investigator: Jonathan Graf.

Chapter 2. Background

Security in the context of FPGAs is a term whose meaning is as broad as the applications served by the devices. FPGAs are growing in resources and complexity, driven by demand from a burgeoning array of new applications. From the simple low-power traditional FPGAs whose silicon is dedicated almost exclusively to programmable logic to the emerging FPGA Multi-Processor System on Chip (MPSoC) device class, FPGAs continue to realize their promise for novel processing architectures in a variety of domains. Recently, the same-die close coupling of heterogeneous processing structures with hardened security resources has created the potential for FPGAs to serve a wide variety of new security applications. Modern FPGAs provide cryptographic accelerators, physically unclonable functions (PUFs), and hardware random number generators (HRNGs). On the MPSoC high end, these features are combined on the same die with programmable FPGA fabric, central-processing units (CPUs), graphics processor units (GPUs), digital signal processors (DSPs), and programmable input-output (IO) resources. This pairing of processing and security resources raises the possibility of using them to create tightly integrated, custom secure processors for future high-security networked applications. FPGAs with these features hold the potential of revolutionizing the security posture for high-security Internet of Things (IoT) applications such as autonomous vehicles, intelligent energy grid devices, home automation, various industrial and commercial control systems, and the datacenter. Within these applications, FPGAs can provide their embedded security features to a variety of security application domains, including information assurance, mission assurance, and cyber-physical systems security.

Chapter 2. Background

Each of those security applications may be undermined, however, if the design of the FPGA is not secure. The fulcrum on which FPGA design security pivots is the programming *bitstream*. The bitstream is a binary design container whose bit values mold the silicon resources of the FPGA into the user-specified application by populating logic look-up tables (LUTs) and block RAMs (BRAMs) while also parameterizing complex CPUs, GPUs, IOs, DSPs, and routing switch matrices. The majority of modern FPGAs store this bitstream at runtime in a static RAM (SRAM) memory array that is integrated with the silicon device resources.¹ The FPGA designer produces the bitstream using electronic design automation (EDA) tools.

The early part of the design cycle consists of steps similar to those used to create any digital electronic integrated circuit (IC). The implementation of the identified requirements for a new FPGA is usually accomplished using a hardware description language (HDL) such as Verilog or VHDL.² The designer's HDL description goes through a series of transformations to become a bitstream. These HDL statements are *synthesized* to become a netlist of connected logic primitives called a netlist. The netlist is *mapped* to resources of the types present on the target FPGA device. If sufficient resources are present, the EDA tools identify the specific individual resources on the FPGA where each function will be realized (*placement*) and connects them with specific routing resources (*routing*), resulting in a placed-and-routed netlist (*placelist*). At any of the previous design representations (HDL, netlist, or placelist), the designer may augment their design with circuits produced by and purchased from third parties, commonly referred to as 3rd-Party IP (3PIP). With the final placelist in hand, a bitstream compiler translates it into the bit settings required to

¹ There are other classes of FPGA that store the bitstream value in flash or anti-fuse memories, but this document focuses on SRAM FPGAs due to their ubiquity.

² New high-level synthesis (HLS) programming models are becoming popular that allow the use of high level languages (HLLs) to program FPGAs. Mature HLL-to-gates ecosystems include the C-to-gates functions of Xilinx Vivado HLS design environment and the OpenCL integration into Altera Quartus Prime. HLS typically does not bypass the steps described here; rather, it precedes and automatically produces the HDL that is then used in subsequent steps.

Chapter 2. Background

realize the design on the FPGA. Those bit settings are arranged in an addressable order that will be understood by the on-device FPGA configuration engine, allowing them to properly distribute the bits to the appropriate resource upon device startup. The bits in this order are referred to as the bitstream.

This bitstream sits at the midpoint between the two major fields of FPGA design security. The first field looks backwards from the point the bitstream is made and examines the above-described design flow. It asks whether the bitstream contains *only* what the designer intended *and no other function*. Specifically, this field considers whether an adversary may have introduced a malicious function into the bitstream in the form of a hardware Trojan. This field of study is referred to as *FPGA design integrity*, or more often, simply as *FPGA Trust*. The second field looks forward from the point a trusted bitstream is in hand and asks how the confidentiality and integrity of that bitstream might be preserved throughout the deployed lifecycle of the design. The collective name for using measures to prevent, detect, and respond to adversarial exploitation in order to preserve bitstream confidentiality and integrity is *FPGA anti-tamper*. As we will see in the section that follows, both of these fields emerged from early research concerns raised by the US Department of Defense (DoD) and have blossomed into robust areas of academic [25][36] and commercial [26][27] interest.

The primary field of our work in this document is FPGA Trust; thus, this is the primary focus of this background section. However, we include a treatment of FPGA anti-tamper for two reasons. First, as we will see in subsequent chapters, the fields are directly related. For example, we demonstrate that an HTH can be inserted for the sole purpose of making subsequent tamper easier in Chapter 4. Second, we will also show that the models used to assess FPGA trust have direct analogues in the fields of trust assessment in different devices (e.g., Application Specific Integrated Circuits (ASICs)), different embodiments (e.g., software), and adjacent fields of security (FPGA anti-tamper). We revisit alternative uses of our model in Chapter 6.

The remainder of this chapter is organized as follows. In Section 2.1 we cover the background for our major areas of concern for FPGA security, FPGA Trust, as well as a brief treatment of the adjacent field of FPGA anti-tamper. In 2.2, we cover the background

literature for work that attempts to measure and strategize about hardware Trojans as well as the general use of game theory to optimize security concerns.

2.1 Attack and Defense

Attacks against microelectronic devices and their related defenses can take many forms. In this document, we are primarily concerned with the fields of trust and anti-tamper.

2.1.1 Trust Background

In the last fifteen years, hardware Trojan design and detection have been studied and characterized [28]. Several taxonomies [29,30] have been published that sort HTH's into categories based on the stage 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 [28]. FPGA Trust is best understood in the context of the general challenge of microelectronics trust, which is concerned with keeping the physical implementation of a manufactured IC free of hardware Trojans.

2.1.1.1 Microelectronics Trust

The earliest public call to address microelectronics trust in general came in 2005 when the Defense Science Board highlighted it as a critical challenge for the DoD [31]. The responses to this call took many forms, but two are worth brief mention as historical references. First, on the basis of key learnings from a series of small exploratory projects (e.g., [32], which started in 2004), the Defense Advanced Research Projects Agency (DARPA) launched the DoD-focused Trusted Integrated Circuits (TRUST) program [33][34][35]. Second, soon after DARPA TRUST brought interest to the area, the first IEEE International Symposium on Hardware-Oriented Security and Trust (HOST) emerged from the test and verification community at the 2008 Design Automation Conference [36]. After these two seminal events initiated the organized response to the

hardware Trojan threat, research has grown into many areas, which are summarized in the following sections.

2.1.1.2 IC Hardware Trojans

The portion of integrated circuit trust that relates directly to FPGA bitstream trust is the concern of logical hardware Trojan insertion at design time. Hardware Trojans are generally defined as having both a *trigger* circuit and a *payload* circuit. The trigger “starts” the payload when an event is detected. The event detected by the trigger could be a signal transition, the present state of a state machine, an external influence, or a series of sequential states across multiple timeframes. Notably, the trigger could simply be the power-on of the device, in which case there would be no additional circuitry for the trigger and the payload circuit will always be active.

The traditional HTH taxonomies of [29] and [30] are constructed based on six hardware Trojan attributes:

1. Insertion Phase
2. Abstraction Level
3. Activation Mechanism
4. Effects
5. Location
6. Physical Characteristics

As we will see, in this work, we use taxonomies such as these as organizing principles that subdivide the attack surface upon which adversaries and defenders play their games. Both we and others have created HTH example circuits that fit into these taxonomies, providing ample circuits for experimentation. For example, the *trust-HUB.org* website, founded on the work of Salmani et al [37] and extended by others [38], contains benchmark circuits organized by the above attributes. This repository of circuits is used for benchmarking hardware Trojan detection methods through the use of common circuits. It allows any user to simply “dial-up” a circuit of interest by supplying its desired attributes. It should be noted that this feature may be used just as easily by an adversary as by a researcher. The circuits in the repository are, in some cases, quite sophisticated and specifically designed to evade detection.

2.1.1.3 IC Trojan Countermeasures

The most straightforward strategy to prevent hardware Trojans is to control every aspect of the design and manufacturing process. However, the costs involved make this strategy available only to wealthy companies and governments. For example, this is the primary strategy of the Defense Microelectronics Activity (DMEA) Trusted Foundry Program [39]. Since owning custom fabrication facilities and controlling all the personnel, software, and equipment in the microelectronics supply chain is beyond the means of most designers, alternative means of preventing IC Trojans is desired. For those who want to have access to the latest manufacturing processes but cannot afford to own their own leading edge fabs, potential exists in an emerging area of research called *split-manufacturing*, where a portion of IC manufacturing takes place in an untrusted fab with access to the latest silicon process technologies, leaving final metallization to be accomplished in a trusted facility. See for example the work of Vaidyanathan et al [40] and the other performers funded by the ongoing Intelligence Advanced Research Projects Agency “Trusted Integrated Chips” program [41]. However, since this requires two fabs (one contracted, one controlled) and coordination between the two, such an approach is still beyond the reach of most IC designers. Further, since it has no analog to FPGA Trojans in the bitstream, this document focuses on other prevention strategies.

If complete control over the process is not available, one proposed strategy has been to make it difficult to insert a hardware Trojan in the first place by either making the circuit hard to understand or leaving little room in the circuit for a Trojan to be inserted. Recent circuit obfuscation techniques include those proposed by Roy et al. [42], whose EPIC method inserts random gates to obfuscate circuit function; Chakraborty et al., whose methods obfuscate the data flow in circuits [43] [44]; Li et al. who apply structural transformations to the circuit that are unlocked by a secret key [45]; and Zhang et al [46], who use muxes whose paths are determined by the resolution of on-chip Physically Unclonable Functions (PUFs).

One challenge to using these methods to prevent hardware Trojan insertion is that they significantly increase the complexity of circuit test and often cost circuit area, making adoption impractical. Another challenge is that the introduction of re-organized circuits, randomized gates, re-mapped dataflow, and novel physical features introduce risks to yield.

This limits adoption due to the risk designers take in using an obfuscated netlist that puts other design goals (e.g., timing, area, performance) at risk. While some studies, such as those by Rajendran et al. [47], have shown that these obfuscation methods show some promise to preventing the reverse engineering of netlists protected by these methods, more recent literature disagrees. In general, the use of obfuscation is a departure from traditional security doctrine, wherein obfuscation and scrambling techniques are eschewed in favor of more mathematically provable techniques [48]. In the context of software obfuscation, Barak et al demonstrated that mathematically perfect obfuscation is theoretically impossible under general circumstances [49]. By a similar argument, Shamsi et al [50] argue that circuit locking – a popular obfuscation technique – can never be perfectly implemented. Machine-learning techniques have proven effective against a variety of circuit obfuscation strategies [56]. Nonetheless, research is ongoing and may yield future advances. A less costly strategy for preventing hardware Trojans in VLSI devices is simply to fill the circuit with “filler cells” and associated digital-signature-based self-authentication methods to prevent an adversary’s ability to add any new circuitry, per the work of Xiao et al [48][52].

In this document, the HTH countermeasures of primary interest are HTH *detection* strategies.³ Contrary to the efficacy controversies of prevention countermeasures, detection strategies are well attested. Furthermore, they can be made widely available and do not change the design. These may be divided into those that can be applied during design and those that are applied after the device is manufactured. This document focuses on those that may be applied by the designer themselves at design time, since those methods translate directly to the challenge of FPGA Trojan detection.

Within the category of detection methods, some have focused on reverse engineering circuits to expose their contents for examination. This has utility, for example, in the case of 3rd-Party IP evaluation or post-manufacturing ASIC evaluation. For example, REFSM was developed by Meade et al. [53]. It helps to extract control logic from a flattened netlist and permit the partitioning of the circuit. Another such top-down functional analysis tool,

³ While we focus on detection strategies, the models we develop are equally applicable to any HTH countermeasure strategy.

developed by Li et al. [54], mines data gathered from functional simulations to extract knowledge of functions. Other reverse engineering-oriented work attempts to develop new solutions (or circumventions) of the subgraph isomorphism problem – the common NP-complete graph theoretic challenge of searching for small circuit patterns within a larger circuit. For example, see Bouchaour et al. [55]. These methods can raise the abstraction level of the composition of the circuit, exposing implementation details along the way. Charabarty et al [56] demonstrated how machine learning can be brought to bear to assist in determining the function of circuits. Quijada et al [57] demonstrate automated extraction techniques from Scanning Electron Microscope (SEM) based circuit delayering for the purpose of post-manufacturing Trojan assessment. Circuit reverse engineering methods are revisited in the FPGA trust section.

Other Trojan detection methods that apply to design-stage evaluation rely instead on verification techniques to find hardware Trojans. For example, the FANCI technique of Waksman et al. [58] seeks to find Trojans based on models of Trojan triggers. It uses a Boolean functional analysis to determine which circuitry appears *stealthy* in its behavior when subjected to traditional logic simulation. They define stealthy to mean “nearly unused.” Similarly, the VeriTrust method from Zhang et al. [59] seeks to find only the trigger portion of the hardware Trojan using the assumption that it will not be activated. A limitation of both FANCI and VeriTrust is that they only look at one sequential stage of a circuit at any given time. That is, they only evaluate the combinational logic between two register stages. Taking advantage of this feature, Zhang et al [60] showed that stealthy implicitly-triggered circuits can be developed that escape FANCI, VeriTrust, and their more simplistic predecessors. Their DeTrust technique spreads the trigger circuitry across both combinational logic blocks and multiple sequential levels to allow its function to blend in with the rest of the good circuitry in the design. The same paper suggests improvements to FANCI and VeriTrust that may account for these even-stealthier triggers, but the techniques must be applied over at least as many sequential stages of the circuit as the Trojan trigger has been spread over. This can be a computationally expensive prospect, making it impossible to guarantee that it can be completed for any given circuit. Salmani [140] proposed another method for circuit detection based on static controllability and

observability calculations, processed by a machine learning analysis, which has the advantage of being computationally inexpensive.

Notably, this snapshot of the state-of-the-art is typical of the past decade of hardware Trojan and Trojan detection method development. Just as in any domain of cybersecurity, the adversary creates a method to which the defender reacts. The adversary reacts again with an improvement. The cycle continues.

2.1.1.4 FPGA Trust

Trojans that are purely logical – along with their related detection methods – are the same for FPGAs as they are for any general IC. Just as with ICs in general, Trojans implemented out of the FPGA’s logic resources may be inserted via modified 3PIP, altered HDL, altered netlists, or via malicious EDA tools. Thus, the above discussion of logical HTHs for ICs applies to FPGAs as well. However, there are notable differences between Trojans that must be physically realized in a traditional IC and those that will be realized as components of an FPGA bitstream. Those differences also demand a different perspective on detection methods in FPGAs, with most of the difference expressed in the unique manner in which FPGA circuits are created: by applying a programming bitstream to a set of programmable logic resources.

To illustrate this, let us consider an adversary who makes use of the now-common advanced persistent threat (APT) style of network attack to gain access to the defender’s network.⁴ On this network, the defender is producing a design for realization in either an ASIC or an FPGA. If that design is to be realized in an ASIC, the economies that drive ASIC markets indicate it will be implemented in many systems. The hardware Trojan they insert might risk accidental discovery simply because of how many systems might use that processor. This may deter the adversary from their goal for risk of discovery. Furthermore, the adversary faces the challenge that they are inserting their Trojan at design time. Unless the fab responsible for manufacturing the ASIC is also complicit in the act, they run the risk that the fab will discover the Trojan. Fabs use many testing techniques that are undisclosed to their users to ensure yield, which adds further discovery risk for the

⁴ A version of this discussion was published in [7].

Chapter 2. Background

adversary. Finally, even if the Trojan in the ASIC makes it all the way through the fab, the adversary has no means of testing the implemented Trojan to ensure it works prior to its deployment on a large scale.⁵ An adversary may be able to overcome these hurdles, though it is outside the scope of this work to speculate about how. It is, however, worthwhile to note that fewer of the above hurdles exist for an adversary wishing to insert a Trojan in an FPGA.

FPGAs are used for designs focused on specific applications whose smaller deployment base does not economically justify the manufacture of an ASIC. These applications can be large Internet routers, industrial control systems, defense systems, or cyber-physical systems. The adversary may target these systems with greater specificity if their entry point is a specific FPGA design used in those systems. Returning to the adversary with an APT-enabled network entry point, for an FPGA design, they are able to see every aspect of that design. The attack surface available to this type of adversary is illustrated in Figure 2. If they have access to the design systems, they are able to access not only the 3PIP, HDL, netlists, and placelists (just as in the ASIC case) but also the final deployment format: the bitstream. Quite often for FPGAs, all of these files reside on the same workstation. From these available choices – which represent the attack surface – the adversary may select the most appropriate point of Trojan insertion and the associated style of change based on what their goal might be. Thus, the adversary may not only insert the Trojan in the system, but – given an APT with the common feature of data exfiltration – they may also retrieve the bitstream with the Trojan in it for testing. Since FPGAs are commodity devices, the adversary can purchase the specific commodity FPGA for which the bitstream is destined to program and test their Trojan in the actual design and ensure it is working. The unique ability for adversaries to select a highly-specific target, the fact that there is no fab involved in the deployment process, and the fact that the Trojan can be tested “remotely” prior to deployment make FPGAs a unique target for Trojans. In these ways, FPGA bitstreams during the design step are similar to software in that they are vulnerable to comprehensive exploitation over-the-wire by a sufficiently advanced adversary.

⁵ Unless, again, they have cooperation with an insider at the fab.

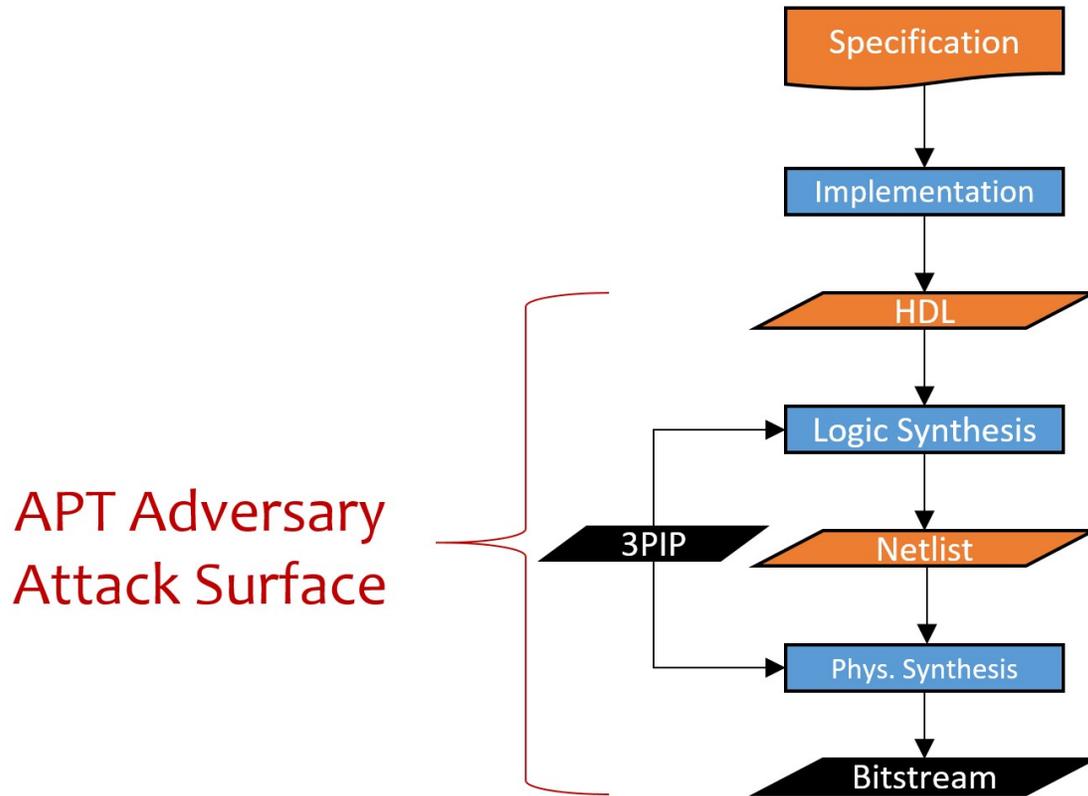


Figure 2. Attack Surface Available to the APT-Enabled Adversary

The bitstream itself presents an additional component of the attack surface to the adversary. Several papers have demonstrated that the bitstream is subject to reverse engineering. Note and Renaud [61] produced the first bitstream reverse engineering tool, though their approach was not comprehensive in understanding all features of the bitstream.⁶ Significant improvements were subsequently shown by Bergeron et al. [65],

⁶ Prior to Note and Renaud, several authors published *design tools* that performed bitstream-level manipulations and leverage detailed knowledge of the bitstream format. Examples include the Xilinx JBits design framework demonstrated by Guccione and Patterson [62][63], the alternate wire database of Steiner and Athanas [64], and the wires-on-demand runtime communications synthesis technique of Athanas et al. [13] (to which the author was a contributor). While these design tools contained knowledge of bitstream function, this section is concerned not with design tools but with tools explicitly developed to reverse engineer the bitstream into a general netlist format.

Benz et al. [66], and Ding et al. [67]. Notably, Ding et al. demonstrate a repeatable distributed computing approach to bitstream reverse engineering to address the computational complexity of the endeavor. Thus, even after the design is completed, a hardware Trojan may be inserted. That is, given the above described APT-enabled adversary, they may simply wait for the designer to finish their work on a Trojan-free bitstream, then modify that bitstream directly with their changes.

Swierczynski and Fyrbiak [68] demonstrated that this scenario is not mere speculation. They demonstrated an attack – on a bitstream protected by vendor-provided encryption, no less – that not only decrypted it but also searched for an insertion point and inserted a Trojan directly in the bitstream. What is notable about this attack is that their knowledge of the bitstream was sufficient that they did not need to reverse engineer it to a netlist to accomplish the attack. Furthermore, they demonstrated re-encrypting the bitstream with the discovered key to ensure it could re-deploy and be properly decrypted and authenticated to the target system. Thus, the APT-based over-the-wire attack on FPGA systems is realistic.

Methods that protect FPGA systems from hardware Trojans may similarly take advantage of the fact that FPGAs are implemented by programming bitstreams. Trimberger [69] was the first to publish the idea that verification techniques combined with software that understands the FPGA bitstream format could potentially enhance the trustworthiness of FPGA designs. In a related approach that was fully realized in practice, the technology that resulted from the FPGA tasks on the aforementioned DARPA TRUST program and its successor the DARPA IRIS program were published by Graf et al in [14][15][16][17].⁷ The technologies described include the Change Detection Platform (CDP) and Functional Derivation Platform (FDP). Collectively, this software could evaluate FPGA designs in any of the above-mentioned formats – HDL, netlist, placelist, and even the final bitstream itself. After assigning a “golden reference” to represent the expected Trojan-free designs,

⁷ The author served as principal investigator for the FPGA tasks of the DARPA TRUST program and as co-PI (with Dr. Scott Harper) for the FPGA tasks of the DARPA IRIS program. The summary here is drawn entirely from the referenced publications, which were approved for public release by DARPA as cited in each publication.

Chapter 2. Background

these platforms compared the design under test (DUT) to that reference and expose any differences. The theory of operation was that any differences exposed are assessed as potential hardware Trojans. Hardware Trojans were modeled loosely as changes from expectation, whether bit-level changes (in the bitstream), structural changes (in the placelist), or logical changes (in the netlist). In this approach, conversion from the bitstream format was automated, resulting in a placelist and associated implementation details. Those resulting files were evaluated for differences from expectation using methods that evaluate structure, simulate behavior, and use formal Boolean logic equivalence testing, assertion-based verification, and model checking. In the event that the golden reference was the original HDL source, this process was relatively straightforward. However, the system was also designed to be able to use HDL simulation models as golden references.

In these cases, the platform made use of advanced circuit partitioning and mapping techniques to break both the DUT and the golden reference into equivalent subcircuits to increase the granularity of behavioral evaluation and make assessment runtime tractable. In the instance that there was no HDL golden model, the system could accomplish a trust evaluation by relying on a comprehensive derivation of the circuits function (i.e., reverse engineering), comparing the exposed function to a datasheet, which is treated as the golden reference. In this case, the platform follows a combined approach of top-down and bottom-up automated reverse engineering. The theory of operation was that the bottom-up techniques successively improve the understanding of *circuit composition* by defining the interaction and structure of logic primitives, while the top-down approach successively improved the understanding of device function. At the point of convergence between top-down and bottom-up methods, the hierarchically-derived circuit composition was mapped to the known function of the device, leading to a complete knowledge of the system. Then the known system function is mapped to the set of expectations expressed by the datasheet, with any differences investigated as potential Trojans. As above in the description of logical Trojans, a few other researchers have also begun following the route of using reverse-engineering combined with logical evaluation to expose hardware Trojans [53][54].

One challenge for any reverse-engineering-based trust method is that despite the automated nature of some of the underlying methods, a large amount of human intervention

is needed to guide the effort. This increases the resource expenditure required of the designer when electing to use a reverse engineering based trust method. Furthermore, the fact that the design is reverse engineered at all can be problematic. For example, due to risks to the confidentiality of 3rd-Party IP implementation details and proprietary bitstream formats, there may be legal limits to using reverse engineering methods for trust.

A new method of bitstream assessment, PV-Bit, was proposed by Graf and Sohanchpurwala [70] wherein the bitstream of an FPGA is assessed against a trusted placelist without reverse engineering the bitstream back to source. This would allow designers to perform their own trust assessment by first producing a trusted placelist using traditional logical and 3PIP assessment methods, then producing a bitstream, then using PV-Bit to assess the trustworthiness of the produced bitstream. Such a method only assesses the final step of the FPGA design process. The optimization of the assessment methods for logical Trojans is not solved by PV-Bit and thus remains an open question addressed by this work.

2.1.2 FPGA Anti-Tamper Background

As with the general field of trust, the field of anti-tamper was a concern first considered with rigor by the US Department of Defense. By the 1990s anti-tamper doctrines and basic methods were well defined, as Huber and Scott detail in [71]. DoD was concerned that the IP in their electronic systems remain known only to them, not to their customers or competitors. Soon thereafter, a few providers of high-value microelectronic systems held similar concerns, which were addressed by either commodity tamper resistant products (such as the IBM 4758 [72] or early smart cards) or custom implementations. One famous example of a custom implementation was from Microsoft, which used hardware anti-tamper techniques to protect secrets in the original Xbox gaming console. While Anderson and Kuhn [73][74] demonstrated attacks against smartcards in the late 1990s, it was Huang's work in the early 2000s that circumvented the Xbox protections that brought a larger public consciousness to anti-tamper [75]. Design security and anti-tamper concerns for FPGAs has been an academic concern as well, as chronicled by Drimer [25]. Nowadays, hardware anti-tamper is a common commercial concern, including among FPGA vendors Xilinx [76] and Intel [77].

Chapter 2. Background

FPGA anti-tamper is directly related to FPGA trust. If a trusted bitstream can be produced by a designer, their next task is to secure it while it is fielded. The field of FPGA anti-tamper is focused on resisting, detecting, responding to, and recording evidence of adversary efforts to violate the bitstream's confidentiality (e.g., understand the design) and integrity (e.g., alter the design). Since the path to meaningful runtime alteration of the design is first to understand it, this section focuses on the study of maintaining bitstream confidentiality. The maintenance of bitstream confidentiality is based on methods to encrypt the bitstream and the challenge an adversary faces with non-public bitstream formats, and the difficulty of understanding potentially obfuscated integrated circuit designs.

2.1.2.1 Integrated Circuit Confidentiality

As with the relationship between IC and FPGA Trust, an illustrative starting point for FPGA confidentiality is to first consider IC confidentiality. For any embodiment – whether in an FPGA or an ASIC – those wishing to protect their IP from reverse engineering methods, a variety of circuit obfuscation methods are available. For example, at the physical level, SypherMedia International [78] offers an ASIC standard cell library that camouflages the function of its gates. The intention is to frustrate those who might attempt to reverse engineer a design through delayering and imaging. It is difficult from imaging alone to discern the difference between, for example, a camouflaged AND and OR gate. If the camouflaged gates are selectively distributed throughout the standard gates present in the design, reproducing a circuit for analysis becomes difficult. Since this technique relies on having the ability to change the physical design library of an ASIC, it does not have a direct analogy in FPGA design.

IC obfuscation techniques that do work equally well for FPGA designs as for IC designs include the logical obfuscators that were mentioned in the trust background above. As such, they offer the same benefits – and significant drawbacks. While the above IC obfuscation techniques may be used to improve the confidentiality of FPGA bitstream designs, they typically are not due to the complexity they introduce to the designer and due to the presence of vendor-provided bitstream confidentiality techniques.

2.1.2.2 FPGA Bitstream Confidentiality

The assumptions behind FPGA bitstream confidentiality methods have been called into question due to recent adversarial breakthroughs. The goal of FPGA bitstream confidentiality is to ensure that the intellectual property contained in the FPGA bitstream cannot be analyzed by an adversary, either for functional understanding or intellectual property (IP) theft. Recently, however, techniques have proliferated that are designed to circumvent vendor-provided encryption functions for SRAM FPGAs. Starting with the introduction of the Xilinx Virtex-II in 2001, a design was widely considered to be sufficiently protected if it made use of a symmetric block cipher (such as 3DES or AES) with a secret key that is only made accessible to the device configurator [79]. That is, the bitstream was encrypted at all times when it was stored in a non-volatile memory next to the FPGA, and it was only decrypted by the device itself using the secret key during configuration. Devices from many vendors followed this formula for more than a decade. It was thought that an adversary could not steal the configuration bitstream and decipher its meaning since the adversary only had access to this encrypted version of the bitstream. The belief was that this method would stop an adversary from even starting the arduous process of trying to understand the vendor-proprietary bitstream because they had to overcome a cryptographically-strong mathematical challenge prior to ever gaining access to the unencrypted bitstream.

What is mathematically perfect on paper often fails in implementation, however. Such was the case with bitstream encryption. While the symmetric block cipher algorithms were without fault mathematically, their implementation in silicon betrayed information in side channels that led to the demise of this IP protection approach. Moradi et al. [80] demonstrated the use of differential power analysis (DPA) attacks to recover the encryption key from Virtex-II. The same team of researchers later demonstrated related side-channel attacks against more relevant ciphers (e.g., AES) and more advanced FPGAs from both Xilinx and Altera [8180] [82]. Academic and industrial collaborations such as the DPA Contest [83] ensure that side channel analysis continues to improve. At present, commercial products, such as the Rambus DPA Workstation Platform [84], may be purchased to recover the encryption keys from FPGAs. In for many devices, this workstation is capable of determining the encryption by monitoring the power of a single

Chapter 2. Background

FPGA power-on cycle. It is thus easily possible to recover unencrypted bitstreams for many FPGA devices whose owners had attempted to protect their design IP via encryption.

Even with an unencrypted bitstream, it was traditionally thought to be difficult to decipher the design contained inside it due to the proprietary nature of the bitstream format. However, as mentioned in the trust section, academic research over the past decade has demonstrated significant progress in automating the translation from a bitstream into a logical netlist format. At present, the difficulty of deciphering a bitstream should not be considered an adequate deterrent to the theft of FPGA design IP. Furthermore, as noted in the cited works, the regular structure of FPGA bitstreams lends itself well to the production of automated tools that can reliably repeat the process of translating from bitstreams into logical netlists.

Because FPGA bitstreams can be exploited in this way, secure systems that rely on bitstream confidentiality as a measure of their security are put at risk. For example, many systems for securing data transfers (e.g., Graf et al [11]) or securing software executing on the FPGA (e.g., Mahar et al [12]) assume that their configuration is kept secret to avoid having their internal key management infrastructure compromised. Thus, a lack of bitstream confidentiality undermines the core root of trust for many FPGA-based secure computing platforms.

Notably, FPGA vendors have responded to these concerns. Leading-edge FPGA devices, such as the Xilinx UltraScale+ [85], Altera Stratix 10 [86], and Microsemi PolarFire [87], have updated their configuration engines and security infrastructure to include sophisticated encryption and key generation strategies that resist analysis from all published DPA attacks as well as related side channel and invasive analysis techniques. Nonetheless, in most cases, these design security strategies still unravel if the adversary is able to recover a single key from the device. Thus, the specter still remains that the advancement of key theft techniques may once again put IP at risk of automated key recovery and bitstream analysis techniques.

Thus, just as with hardware Trojans during the design process, the designer – assisted by the vendor – and the adversary continuously trade the advantage back and forth. One problem within FPGA design confidentiality is that the security for all designs ultimately rests on the security resources provided by the vendor, meaning that all designs deployed

on a given family of FPGAs will be put at risk if that FPGA has its configuration security resources broken and its bitstream deciphered. Despite advances in security and the ubiquity of bitstream reconfiguration potential in FPGAs, there has yet to emerge a systematic means of providing security solutions that are unique to each deployed design. The entire industry rests upon the FPGA vendor security solutions, thus designers remain vulnerable to the above-described break-once, break-all risk to their bitstream confidentiality. Furthermore, since a deployed FPGA must store the decryption key – or at least the means of reconstructing the decryption key – for the bitstream in the system to facilitate automated device boot-up, side channels and invasive means of recovering that key remain the adversary’s preferred means to sidestep the mathematical difficulty of brute-force attempts against any utilized encryption algorithms.

2.1.2.3 Concerns Beyond Bitstream Confidentiality

While bitstream confidentiality is the major anti-tamper concern unique to FPGAs, it should be noted that all standard IC anti-tamper concerns also apply to FPGAs. The FPGA realizes the designer’s circuit using its configuration bitstream, and the secrets of that circuit may themselves be stolen at runtime. For example, the aforementioned data transfer and secure processor [11][12] solutions rely on encryption cores that are individually realized in the configurable fabric of the device. The designers of those encryption cores must concern themselves with the above DPA attacks just as much as the FPGA vendors do when designing the FPGA configuration encryption cores. Furthermore, the design inside the FPGA may also wish to consider the fields of glitching and probing attacks that are becoming ubiquitous and inexpensive. We revisit the idea of glitching to exploit the design inside the bitstream in Chapter 4.

2.2 Metrics, Strategies, and Games

2.2.1 Models and Metrics for IC Trojans

Modeling a hardware Trojan is an ongoing challenge. The relationship between the creator of the Trojan (the *adversary*) and developer of Trojan detection methods (the *defender*) is governed not only by novel circuit design and test methods but also by human incentives and creativity. The adversary may be motivated to accomplish a variety of

Chapter 2. Background

objectives, including but not limited to, controlling the target device, reducing its reliability, causing it to exhibit aberrant behavior under certain conditions, or causing it to divulge secrets during operation [29]. The adversary has an array of means to accomplish said goals in that the Trojan may be inserted at any of a variety of points in the design cycle of the device and may manifest itself in any of a variety of physical embodiments that accomplish the desired effects in the circuit. Because of the spectrum of potential Trojans available to a creative adversary, it is difficult to describe a useful formal abstraction of a Trojan that can be used by defenders to measure the efficacy of detection methods. Thus, Trojan detection methods lack the mathematical tools employed in related fields that seek to detect unwanted effects in circuits. For example, in circuit defect testing, a fault model might be employed to model stuck-at-0/1 faults or bridging faults [1]. These models allow developers of automated test pattern generation (ATPG) software to formally quantify the benefits of novel methods with respect to their efficacy (e.g., fault coverage) and performance [89]. Thus, in fault testing, novel methods can be quickly and formally measured and compared to determine whether they do or do not make a contribution to the state of the art. This is not the case for IC Trojans.

A further difference between methods to detect Trojans and methods to test for faults is the actor that places the effect in the circuit. Faults are typically emergent effects of imperfect manufacturing methods. Thus, fault detection methods do not concern themselves with a guiding hand that employs one or more strategies. For hardware Trojans, attention must be paid to an intelligent adversary who can strategically develop and insert any of a variety of Trojans in order to accomplish their desired end. A human intelligence driven by human incentives is the adversary; the Trojan is the means to their end. Furthermore, the adversary must be thought of as having knowledge of the Trojan detection methods a reasonable defender might employ along with a comprehensive understanding of the attack surface (i.e., where the Trojan is inserted in the design) available to them.

This said, some hardware Trojan models have been created, each with a narrowly-limited scope in terms of taxonomic classification of the Trojan type and the attack surface available to the adversary. All formal models made to date focus on logic circuits with a trigger circuit, a limited representation of possible Trojans. The model used by FANCI and VeriTrust makes the assumption that the primary outputs of the circuit will be affected by

the circuit and that the trigger will be stealthy, and these attributes are used to enable their respective detection solutions. DeTrust showed that the stealthy trigger need not be as simplistic as assumed. Haider et al. [90] demonstrated that if the circuit does not affect the primary outputs of the design – instead leaking information in other ways – even the sequential updates to FANCI and VeriTrust cannot detect the circuit.

The model and detection method evolution over the above papers illustrates the challenge of formal hardware Trojan modeling. Each formal hardware Trojan model proposed to date has built-on simplifying assumptions (e.g., must be a logical change and must have a trigger circuit). Given those assumptions, methods are developed that fully cover circuits that match those assumptions (or prove lack of coverage). However, new Trojans are then developed that do not meet those assumptions that still exercise the purposes of the adversary. One such common limiting assumption is that most models are based on fixed notions of the “utility” of the circuit (e.g., affects primary outputs, leaks information out of side channels). Because of the difficulty in formalizing a measure of utility, utility is fixed as an assumption of the model. There is not a universal formal Trojan model that includes utility as a modeled value. This is because the payload of a circuit is the portion of the Trojan that defines its utility.⁸ The payload is simply a circuit with properties not easily differentiated from the surrounding, trustworthy circuitry. Thus, much of the recent literature (e.g., [90]) ignores Trojan payload utility entirely and concentrates on hiding and detecting triggers.

Without a universal hardware Trojan model, a universal formal notion of hardware Trojan “coverage” also does not exist. In fact, even with the above models in development, the efficacy of the detection methods derived from said models is most often expressed as empirical results with respect to certain taxonomically-classified benchmark circuits from the trust-HUB.org repository. As we will see, this does serve to provide a basis for the metrics we use in our models.

⁸ Kimura and Bibyk [91] attempted to quantify the utility of a hardware Trojan to the adversary by assigning a “Hardware Error Payload” score to various circuits. Their quantification was a heuristic-based, human assigned score, however, and not a formal circuit property.

Chapter 2. Background

A further complication to the challenge of formal coverage in practice, is that formal notions of detectability can fail based on implementation assumptions of Trojan detection methods. For example, when a trusted reference model is available to a designer, Boolean logic equivalence (BLE) testing is a common method of determining the absence of a logic Trojan. The Trojan model utilized in this approach is to investigate any non-equivalent logic as a potential Trojan. It is commonly thought that no Trojan can exist in a circuit that is logically equivalent to its trusted HDL reference. Urish and Graf [9] demonstrated a set of Space-Reliability Reduction (SRR) Trojans that escape BLE-based Trojan detection methods. As one example, for designs that use Triple-Modular Redundancy and voter circuits to improve an ICs resilience in the presence of radiation effects, hardware Trojans may be introduced to change the voter circuits from majority to minority vote circuits. Such a change goes undetected by BLE testing, since the circuits are logically identical except when radiation effects change the function of the design. Resilience is not a concept built into logic equivalence testing. The reliability of the modified design was reduced, but the circuit was still logically equivalent to the Trojan-free trusted reference design. Even with a hardware Trojan model based on formal logic equivalence, a Trojan was successfully inserted into the system, escaping detection. Another example is provided in future chapters.

Thus, to date, there is no universal formal model that captures all the potential purposes and forms of hardware Trojans. Because of this, there is also no universal formal model for how to define hardware Trojan detection coverage. As above, models may be used to compare method efficacy within the narrow scope of the formal attributes of each model. However, no formal model yet takes the entire attack surface available to the adversary – every hardware Trojan style and every entry point – and provides any guidance of where, whether, and what type of Trojan they might attack with. Since these models do not exist, comparison of methods with each other across detection strategies is only accomplished empirically. For example, to compare a detection method that leverages reverse engineering to one that uses functional analysis, one is left to compare the results of each method as determined empirically by their performance on benchmarks.

The first such framework was created by Wilt et al. [102] for use in the DARPA Trust program. Current metrics follow the same essential form as those suggested in that paper.

A detection performance metric and a rate of false positives is established. However, what has been lost in literature since that first paper on trust metrics is the emphasis on a measure of resources required to accomplish the detection. In that paper, the only resource measured was “Time to Evaluate.” As will be seen in future sections, the notion of the resources required for a specific detection method to detect a specific type of hardware Trojan is essential to the strategies adopted by the adversary. It therefore matters significantly to the developer of a Trojan detection method and to the designer who wishes to have a comprehensive Trojan detection strategy. Thus, to accomplish any sense of *optimality* in hardware Trojan detection, a different kind of model is needed.

2.2.2 Game Theory

Game theory has broad application to computer science and engineering. A large body of literature exists, as indexed in surveys by Shoham [102], Dulauf and Blume [103], Halpern [104], and many others. While endless subcategories exist, we may divide the body of work into two main areas:

1. How to represent and solve games efficiently using computers
2. How to apply game theory to various problems within computer science and engineering to accomplish optimization, guide behavior, and provide insight

This dissertation does not contribute to the former. While we make use of the game solving engines produced by Avis [147,152,153], McKelvey et al [149], and Rosenberg [151] in future chapters to accomplish our results, we are not improving upon their solvers. Rather, our work fits squarely in the second category. Among the most active areas within computer science and engineering to which people are applying game theory are artificial intelligence (see the work of Tennenholz [105] and Elkind and Leyton-Brown [106]), cloud computing (see the work of Jalaparti that optimizes for cost [107], the work of Yang that optimizes for energy [108], and the work of Nezerat and Dastghaibifard that generally approaches resource allocation [109]), and, of course, security.

2.2.2.1 *Game Theory and Computer Security*

Within the field of game theory for security, there are many contributions, as surveyed by Wang et al [110]. The field of game theory for network security is perhaps the most

active (see Liang and Xiao [111]), since it is natural to model entities in a network as agents with goals and beliefs about other agents in the network.⁹ However, related security fields, such as the work of Manshaei et al to optimize privacy [113], have also gained insights from the application of game theory.

The game theory for computer security work that comprises the contributions most closely related to the work of this dissertation are those that make similar game assumptions. Two devices we may use to narrow the large body of literature are (1) to sort it by the type of game we are playing or (2) to sort by the type of problem we are solving. To sort by the type of game, we may index our solution by the game theoretic assumptions that underly our game. Briefly stated, the assumptions of our primary game formulation are:

- The players only get *one play*. Thus, we model using a strategic game, not a dynamic/extensive form game with multiple stages and not a stochastic game with multiple states and probabilities associated with transitions between them.
- The knowledge of the game is *complete* and *symmetric*. That is, the players each have complete knowledge and the same knowledge of each other's strategies and beliefs. As we will explore in future chapters, this means the players know each other's utility functions.
- The knowledge of the game is not *perfect*. That is, players learn new information (e.g., whether the exploit works or is detected) at the end of the game that they did not know at the start. In effect, they are modeled as moving simultaneously due to this lack of advance knowledge of their opponent's choices.
- The knowledge of the game is *common*. That is, each player knows the other player has complete and imperfect knowledge.
- The game is not *cooperative*. That is, the players do not communicate with each other to find some mutual good.
- The players are *rational*. That is, the players seek their own best interests according to utility functions that are not weighted to account for irrationality.

⁹ Notably, Liang and Xiao call out the naivety of the economic models as a field that needs work, which we address in this work for microelectronics security.

Chapter 2. Background

These core assumptions are described, examined, and defended in more detail in subsequent chapters. Additionally, we explore the value of challenging several of these assumptions in Chapter 6. For the purposes of this background section, we simply want to know who else is playing this kind of game in computer security. In order of publication precedence, the earliest such work of which we are aware include:

- Jormakka and Molsa [112] who explore games related to information warfare
- Graf and Athanas¹⁰ [4] who explore games for microelectronics security
- He et al [114] who explore games for network security

The difference between these lies in the utility functions constructed for the players, which model their beliefs and motivations. In each case, different utility functions are required to describe the strategies at the disposal of the players and the consequences of playing each. This leads us to the second device we might use to sort through the works related to this dissertation: the type of problem we are trying to solve. This is the subject of the next section.

2.2.2.2 Game Theory and Microelectronics Security

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 the work of Graf and Athanas [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 two-person strategic games to model hardware Trojan adversary and defender relationships was by Kamhoua et al [93]. As compared to the work presented here, [93] 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

¹⁰ Several works referenced in this section serve both as background to the area and components of future chapters.

Chapter 2. Background

adversary and defender security economics and limits the conclusions to the few situations where such a direct fee payment could be envisioned. In [5], Graf elucidated the construction of 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 [6], Graf extended the models from [5] to demonstrate that a framework that is useful not only in selecting from among existing Trojan detection methods but also in guiding developers of new ones. In [8], Graf theorized about a software framework to automate the production of the variables in the game. In [7], Graf 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. In [9], Graf et al demonstrated an implementation of game solving software and its application to several experimental adversary/defender engagements.

Since the publication of [5,6], several related contributions have been made. In [98], Kamhoua et al extend the formulation of [93], leveraging a taxonomy similar to Graf's in [5,6]. However, [98] 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 players 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 [94] develop 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 [95] expand the example in [94] by placing it in the context of software to secure the design workflow. Related to both Graf's work and [93,98], the work presented by Saad et al in [96] 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. The game played has a model of adversary/defender interaction that is limited, similar to [93]. 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. In the work in this dissertation, our expected utility functions model the adversary and defender motives with greater fidelity. Subrational models are revisited in future sections; it will be seen that our adversary and defender models provide the unique ability to consider the manner in which players are behaving irrationally.

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 the work of Charles River Analytics in [97], 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 Kimura and Bibyk in [99], 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.

2.2.2.3 Game Solving Tools and Visualizations

As described in Section 2.2.2.1, our concentration in this dissertation is on applying game theory to microelectronics security, not in game solving. There are many game solvers, such as the aforementioned tools made by Avis [147,152,153], McKelvey [149], and Rosenberg [151]. Thus, the challenge of accomplishing correct solutions to games has largely been solved – at least insofar as we require for our purposes – by the mathematicians and computer scientists that produced those solver engines. What is lacking, however, are application specific means of exploring game results.

Chapter 2. Background

Past work is somewhat sparse on this topic. Traditional means of exploring mixed strategy results has been to depict them as bar charts showing the contribution each strategy makes in the mixed strategy result. Such bar charts can be seen, for example, in Saad's work in [96], where the probability of a player electing a given strategy is represented as different bars in a graph. However, the strategy is fixed to the value of a single game result. Another related visualization technique is from the Gambit software package by McKelvey et al [149]. In it, when solving games under the Quantal Response Equilibrium (QRE) solution concept, one may change the λ value, which represents a bound on the rationality of each player. In changing such a boundary, one can graph the mixed strategies within a boundary of rationality. In exploring the published future plans for this open-source project [150], the focus is primarily on new methods of solving games of novel types; there is no indication an improvement of game outcome visualization is planned.

We are unaware of any software toolset that lets a user explore the contribution of individual utility function variables on the outcome of games by producing multiple game solutions under different values for one or more utility function variables. This may be due to the fact that such a system is necessarily application specific. That is, the variables of interest to explore for microelectronic security games would be different from those required for games in political science or even elsewhere in computer security. The game theory software produced now is focused on solving games after their utility functions resolve into a payoff matrix, not in adjusting that payoff matrix in response to changes in the variables that comprise the utility functions. As will be demonstrated in Section 5.2, the ability to explore the impact of individual variables on the mixed strategy results leads to useful observations, including visual cues of where the influence of individual variables begin to dominate the decision making of the players. We focus on the variables of concern for microelectronic security games, but tools for exploring games of this type may be generalized for use in other games.

Chapter 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.¹²

The desired end result of our modeling work is illustrated in Figure 3. We seek to take metrics about the attack and defense strategies available to the adversary and defender, respectively, and create security economic models of each opponent. We will then use those models in a game theoretic formulation to determine the optimal set of actions each should take. For the defender, this will result in an optimal set of HTH detection methods. Note that this also predicts the optimal action of the adversary, and that this model could be useful to them as well. However, for our purposes in this work, we focus on the defender.

¹² The foundations of the field of security economics were laid by Anderson and Moore in [100].

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 of 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 the work of Gordon and Loeb in [101], 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.



Figure 3. Desired Utility of Security Economics/Game Theory for Hardware Trojan Detection Strategy Selection

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.¹³ 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. In EUT, their self-interests are represented in expected utility functions, which may be said to capture their *beliefs*. As

¹³ Thus, our emphasis that this is not a zero-sum game.

stated in the previous section, we make use 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 treated in our future 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 $S_D = \{\sigma_{D0}, \sigma_{D1}, \dots, \sigma_{Dn}\}$. Within this set, σ_{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 σ_{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 $\sigma_{D1}, \dots, \sigma_{Dn}$ are HTH countermeasures. In the main thread of this work, 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 $S_A = \{\sigma_{A0}, \sigma_{A1}, \dots, \sigma_{Am}\}$. Their baseline strategy, σ_{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 $\sigma_{A1}, \dots, \sigma_{Am}$ represent an attack of one kind or another. In future 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 [102], which formed the basis for the empirical metrics used in the DARPA TRUST program [116,117]. We have adapted and extended those probabilities for specific use in our utility functions. We make use of the following set.

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

$P_{FA}(\sigma_D)$: The probability of false alarm. This is the probability that the selected defender strategy, σ_D , will incorrectly indicate an attack has taken place.¹⁴ This is also a continuous probability in $[0,1]$.

$P_S(\sigma_A)$: The probability of success. This represents the likelihood that the adversary HTH, σ_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

¹⁴ For the purposes of the experiments in this document, we have described the false alarm rate as dependent only on the selected defensive strategy. Future work may consider 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.

experiments, $P_S(\sigma_{A0}) = 0$ and $P_S(\sigma_A) = 1$ for all other strategies $\{\sigma_{A1}, \dots, \sigma_{An}\}$. In the future, we can elect to use this as a continuous function, but all games evaluated to date have been dominated by other factors.

$P_{ATT}(\sigma_A, \sigma_D)$: The probability of attribution. This represents the likelihood that the given adversary strategy, σ_A , can be attributed to the adversary in the instance the circuit is detected by defender strategy, σ_D . 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 [118]). 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 the next section 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.¹⁵

$Z_D(\sigma_D)$: The direct cost of implementing a defensive strategy, σ_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, σ_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 diagnose 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 ample 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

¹⁵ The monetization of strategies and human life may be unfamiliar to readers, but they are regularly performed practices of risk assessment. See [119] and [120].

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.¹⁶ 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.

$Z_A(\sigma_A)$: The direct cost of implementing the adversary's strategy, σ_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.¹⁷ 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

¹⁶ 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 [7].

¹⁷ Differences in the cost of accessing software, firmware, and silicon for exploitation was explored in [7].

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 dissertation 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 [7]), a different accounting will need to take place for the cost of each strategy.

$Z_{find}(\sigma_A)$: The penalty to the adversary if their Trojan, σ_A , is not only detected but also attributed to them. The Z_{find} 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.

3.3 Assembling Utility Functions

We now have the requisite variables to populate our adversary and defender utility functions. Under EUT, 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

$$U_A(\sigma_A, \sigma_D) = \{[1 - P_D(\sigma_A, \sigma_D)]P_S(\sigma_A)\}G - Z_A(\sigma_A) - P_D(\sigma_A, \sigma_D)P_{ATT}(\sigma_A, \sigma_D)Z_{find}(\sigma_A). \quad \text{Eq. 1. Adversary Utility Function}$$

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

$$U_D(\sigma_A, \sigma_D) = [P_D(\sigma_A, \sigma_D) - P_D(\sigma_A, \sigma_{D0})]L - Z_D(\sigma_D) - Z_{FA}(\sigma_D)[P_{FA}(\sigma_D) - P_{FA}(\sigma_{D0})]. \quad \text{Eq. 2. Defender Utility Function}$$

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

As illustrated in Figure 4, an additional consideration for both players in our game is where in the ASIC or FPGA design cycle their interaction takes place. The design and deployment cycle of these devices may be divided into a sequence of subgames, each represented by a step game, $\Gamma_{\langle step \rangle}$, for each step in the design cycle. The illustration has separate step games for malicious changes to specification, logical synthesis, third-party IP (3PIP), physical synthesis, FPGA programming, ASIC mask generation, ASIC fabrication, and device deployment. The early steps of design cycle are similar for each device style. At the point of physical synthesis, the steps diverge to the unique design concerns related to the physical embodiment of each device type.

The step division in the figure is has utility – as we will illustrate. However, alternative divisions may be possible as well. In any division, however, games can be used sequentially to determine the set of protections to apply during that design and deployment stage to optimally protect that part of the process. 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 work concentrates on this latter approach, adopting the term *step games* to describe the adversary/defender interaction at each point in the design cycle.

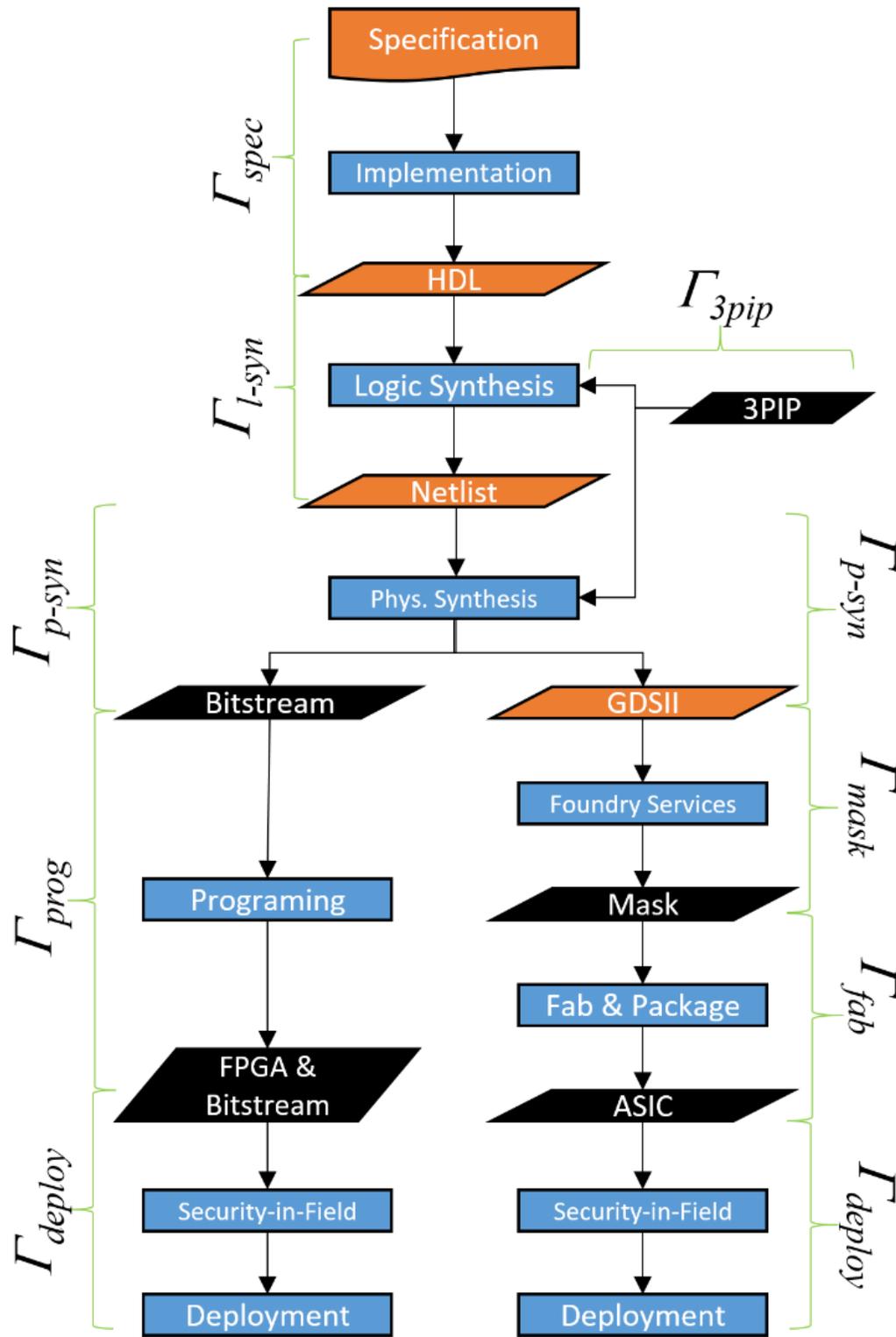


Figure 4. The ASIC and FPGA Device Games Decomposed into Step Games

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 (σ_A, σ_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.¹⁸ 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

¹⁸ The classic intuitive game example with a mixed strategy solution is Rock-Paper-Scissors. The optimal play for each player is to select from among three available strategies with equal probability for each. A discussion of mixed strategy Nash equilibrium and Rock-Paper-Scissors is in [124].

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 [93,94,95] are built on this rationality assumption, others have rightly questioned whether human actors will behave in this way. For example, [96] 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 [121]. That statistical model includes a value, λ , 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 [122]. Revealed preference theory [123] 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 nonuniform 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 [124][126]. 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 the future, we do plan to explore the above-listed irrationality models, using this work to define the baseline optimal selections based on rational players. That is, when we advance our decision engine (described below) to include irrationality models, we will likely synthesize 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.¹⁹ 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. In concurrent work that applies game theory to HTH detection challenges, the actual HTH detection component of the game is so oversimplified that no practical conclusion can be drawn from it. It is our hope that in this work, the math serves the problem - rather than oversimplifying the

¹⁹ 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.

problem to more easily serve the math. If we do the latter, we are simply setting up a math problem to conjecture about, not producing a model of the world about which we can draw conclusions. Thus, we seek first to properly describe the problem for rational players with correct utility functions; after this, we will consider irrationality.

3.5.2 Computing Game Solutions

When using game theory to reason about security, some researchers reduce 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 [127]. 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. While our future work will explore formal analysis of simplified variants of our games, for now, the software described in Section 4.3 below contributes to straightforward discussion of large complex games.

3.6 Example Trust Game

Before we explore more complex games with computed solutions in the next chapters, it is illustrative to consider a simple game which can be solved by hand. This allows the opportunity to develop an intuitive sense of the mechanics of game theoretic solutions in the context of HTH detection. For the purposes of this exercise, we simplify our adversary function by assuming that $P_S(\sigma_A) = 1$ for all adversary strategies and $P_{ATT}(\sigma_A, \sigma_D) = 1$ for all adversary/defender strategy tuples. This results in the simplified adversary utility function of Eq. 3. The utility function for the defender remains the same in this game construction.

$$U_A(\sigma_A, \sigma_D) = \{[1 - P_D(\sigma_A, \sigma_D)]P_S(\sigma_A)\}G - Z_A(\sigma_A) - P_D(\sigma_A, \sigma_D)Z_{find}(\sigma_A).$$

Eq. 3. Simplified Adversary Utility Function

To solve the game by hand, we are going to set up the game in a table. This arrangement is called *normal form*. To arrange in normal form, as in Table 1, we need to resolve the utility functions for each player under every possible strategy tuple available. As we will see, this arrangement allows a person to reason quickly about optimal play for simple games.

Table 1. The Trust Game in Normal Form

| | σ_{D0} | σ_{D1} | ... | σ_{Dn} |
|---------------|--|--|----------|--|
| σ_{A0} | $U_A(\sigma_{A0}, \sigma_{D0}), U_D(\sigma_{A0}, \sigma_{D0})$ | $U_A(\sigma_{A0}, \sigma_{D1}), U_D(\sigma_{A0}, \sigma_{D1})$ | ... | $U_A(\sigma_{A0}, \sigma_{Dn}), U_D(\sigma_{A0}, \sigma_{Dn})$ |
| σ_{A1} | $U_A(\sigma_{A1}, \sigma_{D0}), U_D(\sigma_{A1}, \sigma_{D0})$ | $U_A(\sigma_{A1}, \sigma_{D1}), U_D(\sigma_{A1}, \sigma_{D1})$ | ... | $U_A(\sigma_{A1}, \sigma_{Dn}), U_D(\sigma_{A1}, \sigma_{Dn})$ |
| \vdots | \vdots | \vdots | \ddots | \vdots |
| σ_{Am} | $U_A(\sigma_{Am}, \sigma_{D0}), U_D(\sigma_{Am}, \sigma_{D0})$ | $U_A(\sigma_{Am}, \sigma_{D1}), U_D(\sigma_{Am}, \sigma_{D1})$ | ... | $U_A(\sigma_{Am}, \sigma_{Dn}), U_D(\sigma_{Am}, \sigma_{Dn})$ |

In order to produce this arrangement, we need to play the “Trust game” with an example scenario by setting the values for the utility functions, then resolving them.

3.6.1 Playing the Trust Game

In our hypothetical scenario, the adversary’s goal is to cause the defender’s device to fail early by inserting a Trojan into a netlist representation of the design. Their intention is to create reliability concerns in the market for the defender’s device, allowing the adversary to capture a portion of that market with their rival device. The potential loss incurred by the defender is proportional to their lost market share, which we estimate as $L = \$10,000,000$. The desired gain for the attacker is proportional to the fraction of the defender’s lost market share that their rival device might gain, which we estimate to be G

= \$2,000,000. We estimate that the defender has a process for resolving false alarms that incurs cost $Z_{FA} = \$50,000$.²⁰

For our final fixed cost, we assume we only have a minimal ability to incur a penalty on the adversary. We set $Z_{find} = \$100,000$. We will assume that the adversary and defender each have four strategies. While taxonomies of Trojans and Trojan detection methods can likely yield games involving many more strategies for both players, such games often require commensurately more complex solution concepts, so we treat those in the following chapters when we have our automated solver. A simple game such as this suffices to illustrate the value of the approach.

The four strategies available to the adversary in this example are to do nothing, to use a simple failure circuit that is triggered by an expected binary value using a comparator, to use a more stealthy failure circuit, and to make a fully-customized 0-day circuit. The costs associated with each strategy are listed in Table 2. Given the expense of σ_{A3} , intuition might indicate that the attacker would be unlikely to spend half the cost of their expected gain, G , on a Trojan; however, it remains for our game to determine if that is a rational, optimal choice.

Table 2. Example Adversary Strategies and Costs

| Strategy | Description | $Z_A(\sigma_A)$ |
|---------------|------------------|-----------------|
| σ_{A0} | No Trojan | \$0 |
| σ_{A1} | Triggered Trojan | \$115,000 |
| σ_{A2} | Stealthy Trojan | \$200,000 |
| σ_{A3} | 0-Day Trojan | \$1,000,000 |

For the defender, we consider the case where they have just developed a new detection method called Magic Detector. As we will see, Magic Detector is an order of magnitude more expensive than the next best method, and it improves the detection probability against the stealthiest Trojans, though only marginally. The defender does not know whether this small detection probability improvement is worth the cost. The defender's four strategies are to do nothing in addition to standard test and verification, to additionally perform an

²⁰ In future chapters, we produce additional estimates that are justified more comprehensively. In this section, values are notional for the purpose of illustration.

advanced simulation-based detection technique, to perform an advanced Boolean logic equivalence (BLE) testing technique in addition to the additional advanced simulation, or to use the Magic Detector in conjunction with all of the above. The defender costs are shown in Table 3.

Table 3. Example Defender Strategies and Costs

| Strategy | Description | $Z_D(\sigma_D)$ |
|---------------|--------------------------------|-----------------|
| σ_{D0} | No Additional Trojan Detection | \$0 |
| σ_{D1} | Advanced Simulation | \$50,000 |
| σ_{D2} | Advanced BLE + Sim | \$100,000 |
| σ_{D3} | Magic Detector + BLE + SIM | \$1,000,000 |

We assume that empirical testing²¹ has been performed to pit the adversary and defender strategies against each other and determine the likely resulting probabilities of detection, as in Table 4. We also assume that the adversary and defender are using the same estimated P_D values, though our model can accommodate the condition when they are using different values.

Table 4. Table of $P_D(\sigma_A, \sigma_D)$ Values

| | σ_{D0} | σ_{D1} | σ_{D2} | σ_{D3} |
|---------------|---------------|---------------|---------------|---------------|
| σ_{A0} | 0 | 0 | 0 | 0 |
| σ_{A1} | 0.5 | 0.6 | 0.99 | 0.99 |
| σ_{A2} | 0.25 | 0.5 | 0.9 | 0.95 |
| σ_{A3} | 0.01 | 0.05 | 0.25 | 0.4 |

Further, we assume that the false alarm rates drop for the defender as more methods are employed in conjunction with one another, as shown in Table 5.

Table 5. Table of $P_{FA}(\sigma_D)$ Values

| σ_{D0} | σ_{D1} | σ_{D2} | σ_{D3} |
|---------------|---------------|---------------|---------------|
| 0.005 | 0.0001 | 0.00005 | 0.00001 |

²¹ The next chapters illustrate how this testing will take place.

Finally, as discussed above, we use $P_S(\sigma_A)$ in this treatment only to indicate whether the adversary has elected to insert a Trojan at all. It is 0 when σ_{A0} is selected; it is 1 for all other strategies. Using the normal form and resolving the utility functions, the game matrix is shown in Table 6. Units are in thousands of dollars, rounding as needed.

Table 6. The Trust Game in Normal Form Showing Tuples of (Adversary Utility, Defender Utility) in \$K's

| | σ_{D0} | σ_{D1} | σ_{D2} | σ_{D3} |
|---------------|---------------|---------------|---------------|---------------|
| σ_{A0} | 0, 0 | 0, -50 | 0, -100 | 0, -1000 |
| σ_{A1} | 835, 0 | 625, 950 | -194, 4800 | -194, 3900 |
| σ_{A2} | 1275, 0 | 750, 2450 | -90, 6400 | -195, 6000 |
| σ_{A3} | 979, 0 | 895, 350 | 475, 2300 | 160, 2900 |

To solve this game, we use the iterated elimination of dominated strategies (IEDS) solution concept. A player's strategy is said to dominate another strategy for that player in the instance that the former yields equal or better payoff no matter what their opponent chooses. We can see that neither player has a single strategy that dominates all of their other strategies regardless of the other player's selection. Thus, it is unclear from the initial formulation what each player's best strategy might be. To find this, we must iteratively remove individually dominated strategies, a process illustrated in Table 7.

Table 7. Hypothetical Trust Game Solution via IEDS

| | σ_{D0} | σ_{D1} | σ_{D2} | σ_{D3} |
|---------------|---------------|---------------|---------------|---------------|
| σ_{A0} | 0, 0 | 0, -50 | 0, -100 | 0, -1000 |
| σ_{A1} | 835, 0 | 625, 950 | -194, 4800 | -194, 3900 |
| σ_{A2} | 1275, 0 | 750, 2450 | -90, 6400 | -195, 6000 |
| σ_{A3} | 979, 0 | 895, 350 | 475, 2300 | 160, 2900 |

First, in the move highlighted in orange, we eliminate the adversary strategy σ_{A0} , since it is dominated by σ_{A3} , meaning the adversary would never play σ_{A0} . Therefore, we know the adversary will attack. With that strategy eliminated, we are left with a 3×4 game matrix less strategy σ_{A0} , and we continue solving. In this new game, defender strategy σ_{D0} is dominated by σ_{D3} , so it is eliminated (highlighted in blue). Thus, we know the defender must do more than standard test and verification, and the resulting 3×3 game is played. At this point, adversary strategy σ_{A3} dominates both σ_{A1} and σ_{A2} (green highlight), meaning both of the dominated strategies are eliminated. Thus, the adversary strategy is selected as

σ_{A3} . Given this, the defender will select their highest payoff, which is σ_{D3} , eliminating σ_{D1} and σ_{D2} (yellow highlight).

Notably, the discovered strategy tuple represents the game's Nash Equilibrium. That is, $(\sigma_{A3}, \sigma_{D3})$ is the set of strategies for which no player can do better by unilaterally changing their strategy. Thus, using two solution concepts, we arrive at the conclusion that the most rational choice for the adversary is to use their 0-Day Trojan despite the fact that it costs half what they hope to gain. Furthermore, the most rational choice for the defender is to use their Magic Detector. By a simple examination of the security metrics and costs of the Magic Detector, this was not intuitive prior to constructing the game. However, using the game, we can see this is the rational choice.

Note, as we explore below, these conclusions hold only for this game based on the estimates that went into it. Finally, we have also shown that this game provides as much utility to the adversary as it does to the defender. During the game analysis, the adversary learned that their 0-day Trojan was their best strategy. Thus, the adversary may make use of the Trust Game to construct optimal hardware Trojans as easily as the defender may do so to make optimal detection methods.

3.6.2 Solving Related Games

Using the same utility functions and different assumptions of costs and probability values yields different results. For example, if we use all the same values from the above game but drop the efficacy of the Magic Detector method against the 0-day Trojan from $P_D(\sigma_{A3}, \sigma_{D3}) = 0.4$ to $P_D(\sigma_{A3}, \sigma_{D3}) = 0.33$, the Nash Equilibrium becomes $(\sigma_{A3}, \sigma_{D2})$, meaning this less effective Magic Detector is not the best strategy for the defender despite the fact that the adversary's best strategy is still the 0-day Trojan. Alternatively, if our only change to the above game was to reduce the value of the device market by cutting L to \$1,000,000 and G to \$200,000, the Nash Equilibrium becomes $(\sigma_{A0}, \sigma_{D0})$, meaning the most rational choices for the adversary and defender become not to attack with a Trojan and to do nothing additional for Trojan detection, respectively. Thus, our game setup is sufficient to account for the emergence of new Trojans, changes in detection method efficacy, and changes in the economic valuations, all of which can be used to guide the development of new Trojan detection methods. In future chapters, this type of game analysis is explored in much

greater detail, and our software solver, GameRunner, is demonstrated as an tool for visualizing such exploration.

As a final note on the Trust Game, we highlight that for illustration purposes we constructed each of the scenarios to resolve to what is called a pure strategy Nash Equilibrium. That is, the strategies of each player are completely specified in the various Nash Equilibria we discovered. Using other probability and cost values, the Trust Game can also resolve to what is called a mixed strategy Nash Equilibrium. In this case, the strategies of each player are assigned a probability value rather than having a single strategy emerge as the best. As we will see in the next chapters, mixed strategy results are common in the large complex games we will likely encounter when using this methodology.

Chapter 4. Experimental Game

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 chapter 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. 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.²² We use the pragmatic Advanced Persistent Threat (pAPT) adversary model, a variant of the APT adversary introduced in Chapter 2. The pAPT adversary is an APT adversary who behaves rationally. We assume that the pAPT adversary already has access to the network and computer

²² Note that this game represents only one of the many scenarios that can be explored using our utility functions.

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.²³ 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 8 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 [128]; we use the cost of that board to represent the total material cost of the device.²⁴ 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, 5000 are sold. The third scenario is a different device: a high-end network processing system. They

²³ [7] explores additional potential desired outcomes and strategies for the pAPT adversary.

²⁴ The assumptions that build our economic values are simplified throughout this section 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.

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 [129]. The sale price for the third scenario is also estimated at twice the material cost.

Table 8. Game Scenarios

| | Kickstarter | Consumer | Network |
|-------------------------------|-------------|-------------|--------------|
| Quantity Sold | 500 | 5000 | 4000 |
| Material Cost Per Item | \$149 | \$149 | \$6,995 |
| Sale Price Per Item | \$298 | \$298 | \$13,990 |
| Market Value of Item | \$149,000 | \$1,490,000 | \$55,960,000 |

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, $Z_{find}(\sigma_A)$, is set to half of G for all strategies.²⁵

To determine the costs of HTHs 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 [130], assigned a labor wrap rate²⁶ of 3.1 to represent a contractor with expensive software tools, and

²⁵ Note that these assumptions may be questioned, modified, and analyzed in the GameRunner software we produced. Forthcoming discussion illustrates these features.

²⁶ A labor wrap rate is a unitless scalar multiplier value applied to a salary in order to calculate the cost of an hour of labor, including all overheads. We estimated using data from [131], which lists labor wrap rates for government contractors. We took the average wrap and added 1 to it to create a high rate, which accounts for the amortized cost of traditional electronic design automation (EDA) software, including synthesis tools and simulators. The use of government contractor data is driven by the public availability of data. It is difficult to acquire such information for purely commercial enterprises.

divided by a 2080 hour work year to get an hourly rate of \$167.22. We make the simplifying assumption that labor costs the same for the adversary and defender. To account for the cost of compute time, we use \$0.33/hr from [132] 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 dissertation, we do not make use of methods that require long periods of unattended computing, so we simply assume there is one hour 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 ($Z_A(\sigma_A)$ 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 and 3PIP

In our experiment, we model the HDL and 3PIP step games [5]. In each case, the defender and adversary are selecting one strategy to play in 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. As modeled here, the HDL and 3PIP steps are examined independently to allow easier discussion.

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. The fact that the defender has the source to reference enables different countermeasures to be used in the HDL case than those available in the 3PIP case. 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 the 3PIP step game the defender similarly employs countermeasures to test the 3PIP core for HTHs. Defender countermeasures that require reference to the source code are disallowed in our 3PIP experiment. For the adversary,

similar to the HDL case, their approach is to replace the original 3PIP core with one that contains their HTH. In our experiments, we use the same set of designs in the HDL case as in the 3PIP case, altering only the strategies available to the defender.

In the future, we may consider 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. For further discussion, see Chapter 6.

4.2 Strategies and Probabilities

In our experiment, we attempt to illustrate strategies that are realistically available to both an adversary and defender. In [8] we detailed the large task required to establish a reliable dataset upon which to base industry-level recommendations derived from this game-based methodology. The immense task of building this dataset will require interaction with industry on a scale beyond our scope in this work, but it is planned for future work. For our work in this dissertation, we attempt 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 [133,134] 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 judgement 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 [135]. In discovery, they review the circuit to determine the best place in the circuit to attack.²⁷ If that answer is “nowhere,” they employ attack σ_{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 hours. 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 hours to customize for the target circuit (the implementation stage) and an additional 120 hours 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 σ_{A0} .

For the representative HTHs we produced, our taxonomy is derived from [29,30] and simplified to divide the circuits into categories based on the trigger mechanism involved. The circuits we used were primarily the set from [133,134] as refined by [136]. However, to provide more variety of input design and demonstrate progress towards the desired future

²⁷ This may be where the adversary employs a methodology similar to [94], where they use game theory to optimize HTH placement in a circuit.

large dataset required by [8], we also included designs from the DARPA Common Evaluation Platform (CEP) [137] and placed a variety of HTH’s in them. 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 9.

Table 9. Adversary Strategies and Costs²⁸

| Strategy | Taxonomy Description | Abbrev. | Qty | HTH Circuits | Labor Cost (Hrs) | Machine Time Cost (Hrs) | Material Cost (\$) | $Z_A(\sigma_A)$ (\$) |
|---------------|-----------------------------------|---------|-----|---|------------------|-------------------------|--------------------|----------------------|
| σ_{A0} | Do Nothing | DONT | 6 | AES-notj-top, BasicRSA-notj, CEP-gps-notj, PIC16F84-NoTj, RS232-NoTjGate, RS232-NoTj | 120 | 120 | 0 | 20,106 |
| σ_{A1} | Always on: Rewire | REWR | 8 | AES-reversebit1, AES-reversebyte1, BasicRSA-TReverseByte1, CEP-gps-Tj-disable-aes, CEP-gps-Tj-reversebit1, PIC16F84-TReverseBit1, RS232-TjReverseBit1, RS232-TjReverseBit2 | 480 | 480 | 0 | 80,426 |
| σ_{A2} | Always on: Change Gates | GATE | 4 | AES-T100, AES-T200, AES-T300, RS232-T1800 | 480 | 480 | 0 | 80,426 |
| σ_{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 |
| σ_{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 |
| σ_{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 |
| σ_{A6} | Trigger: Glitch State | GLST | 2 | PIC16F84-TjGlitchState, RS232-TjGlitchState | 480 | 480 | 0 | 80,426 |

²⁸ Circuits marked “CEP” apply TrustHub Trojans to DARPA Common Evaluation Platform designs; all “rewire” circuits are our own custom designs, circuits marked “TjGlitchState” apply custom-designed Glitch State Trojans to TrustHub designs, all others are named according to the nomenclature of the TrustHub benchmark suite. A few circuits in the latter category required minor modifications and bug fixes to be properly included in our Trojan set.

Our first two HTH categories, REWR and GATE, are not triggered at all – they are always on. σ_{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. σ_{A2} (GATE) represents HTH circuits which are always on and include modified gates and logic in addition to wires; four are included.

Strategies σ_{A3} , σ_{A4} , σ_{A5} , and σ_{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 σ_{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 σ_{A4} (CCMP) use a combinational comparator as a trigger. That is, they activate the HTH payload when a set of 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 σ_{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 σ_{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” 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 Figure 5 below. 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.

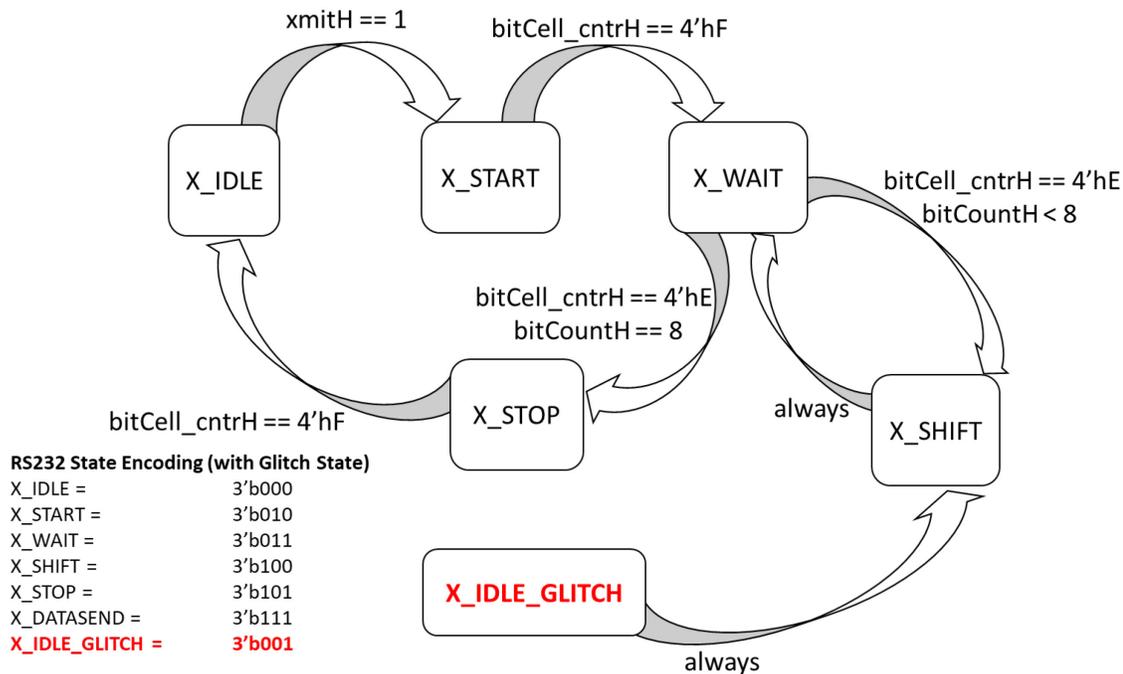


Figure 5. Glitch State Trojan in RS232 (RS232-TjGlitchState) [10]

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 [138,139]. Neither of the HTH circuits comprising σ_{A6} 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 – will be a focus of our future 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.²⁹ 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. In future σ_{A0} datasets, the varieties of circuit test mechanisms (such as scan chains, etc.) that may be present and cause methods to false alarm should be included.

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 $Z_D(\sigma_{D0})$ 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_{D0} (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, the costs associated with them are unique among labor, machine time, and material licenses. For example, strategy σ_{D1} (CSIM) also uses simulation and a testbench, but we used a technique called constrained/directed random simulation to improve the test coverage.³⁰ 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

²⁹ Of course, a defender can also false alarm on a circuit that contains a Trojan, but ascribing a HTH to a circuit with a HTH for an incorrect reason has a significantly lower consequence than falsely ascribing a HTH to a circuit with no HTH present.

³⁰ This is related to the software black-box testing technique referred to as fuzzing.

primary input of the circuit.³¹ 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 $Z_D(\sigma_{D1}) = \$2,681$ based on the assumption that it takes 16 hours 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_{D2} (SCOA) is a static controllability/observability and analysis method based on the theory presented by Salmani in [140] using the implementation we published³² in [141]. The method makes use of static controllability and observability analysis [142] 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 [141,143].

The results of this tool are visually illustrated in Figure 6. In the figure, the SCOA results are graphed on the left for an RS232 core with no hardware Trojan and on the right for the same RS232 core infected by hardware Trojan *RS232-T1200* from our dataset. The graphs depict each gate in the circuit according to our calculated controllability (CC) values on the x-axis and the calculated observability (CO) values on the y-axis. Lower values mean those gates are easy to control or observe, with difficulty increasing as the values get higher. The CC and CO values have no units and are useful only in reference to each other.

³¹ More advanced future 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 a 1 to 0 or 0 to 1.

³² The implementation details of our work in [141] 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 [140] 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.

Note the extreme difference in the scale of each axis between the graphs on the left and right. The k -means clustering analysis is set to find three clusters. In the Trojan-free circuit, the clusters have little distinction: each cluster (represented by red, green, and blue dots) have very similar CO and CC values. In the infected circuit, the gates in the Trojan clearly evidence themselves. The cluster of red dots evidence the Trojan-free portion of the circuit while the blue dots are gates in the Trojan that are hard to observe and the green dots are gates in the Trojan that are hard to control.

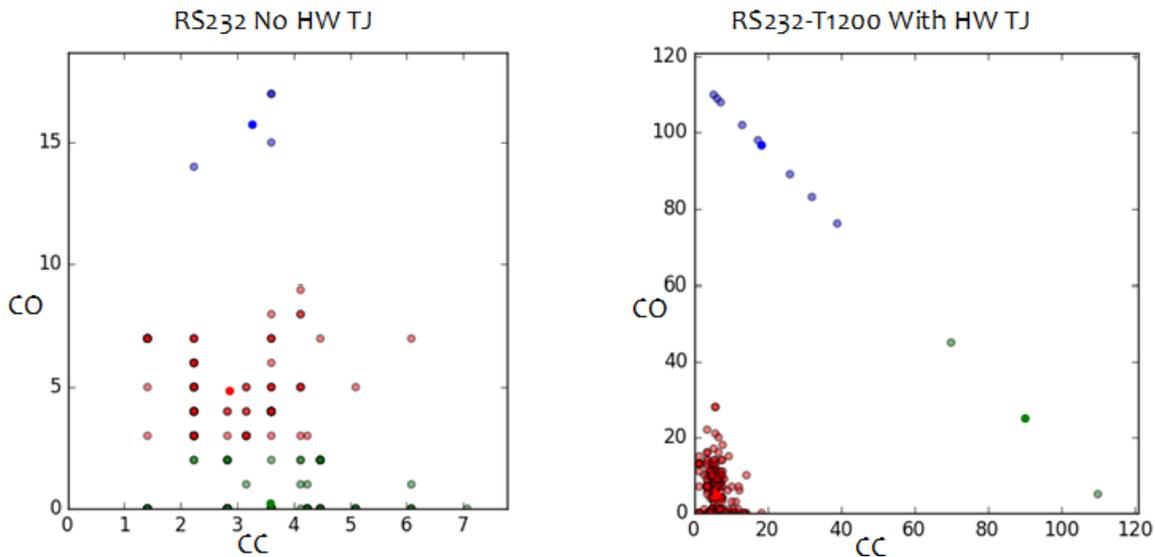


Figure 6. SCOA clustering result for circuit RS232-T1200 [10]

For our purposes, we assume such a tool would be moderately simple to use (16 hours of labor and machine time), since it is mostly automated, and modestly priced if sold on the market (\$15,000 license). Thus, $Z_D(\sigma_{D2}) = \$17,681$.

The strategy σ_{D3} (STRC) is a structural pattern matching method similar to those in [144,145]. 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. As with SCOA, we implemented this using the TCL interfaces exposed by Xilinx Vivado. For this method, we adopt the same cost model as σ_{D3} , which means $Z_D(\sigma_{D3}) = \$17,681$.

Chapter 4. Experimental Game

Finally, for strategy σ_{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. As such, this method applies only to the HDL step game, since source is assumed to be unavailable in the 3PIP step game. The labor and machine time to set up, run, and review the result from the tool are set to 24 hours each. The tool license is \$65,000. The total cost of the strategy is \$69,021.

Table 10. Defender Strategies and Costs

| Strategy | Taxonomy Description | Abbreviation | Labor Cost (Hrs) | Machine Time Cost (Hrs) | Material Cost (\$) | $Z_D(\sigma_D)$ (\$) |
|---------------|---|--------------|---------------------|-------------------------|--------------------|-------------------------|
| σ_{D0} | Nothing but testbench | BNCH | 0 | 0 | 0 | 0 |
| σ_{D1} | Constrained/directed random simulation | CSIM | 16 | 16 | 0 | 2,681 |
| σ_{D2} | Static controllability/observability method with k-means clustering | SCOA | 16 | 16 | 15,000 | 17,681 |
| σ_{D3} | Structural pattern matching | STRC | 16 | 16 | 15,000 | 17,681 |
| σ_{D4} | Boolean Logic Equivalence testing | BOOL | 24 | 24 | 65,000 | 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. For this situation, we assume that the defender will incur different costs based on whether the HTH is indicated in the HDL or in a 3PIP core. If in the HDL, 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 hours, or \$2,681 to do so. If the false alarm occurs in a 3PIP core, the forensic examination is far more costly, since we assume the source is not immediately available. In this case, we model the resolution as the designer buying the source to ensure that the design does not have HTH's in it for a cost of \$15,000. We further assume this

requires 100 hours of extra work at a cost of \$16,722. Thus, false alarms in 3PIP result in \$31,722 of cost for the defender.³³

Table 11, below, 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.

Table 11. Three Economies (All Values \$)

| Economy | G | L | Z_{find} | Z_{FA} (HDL) | Z_{FA} (3PIP) |
|----------------|------------|------------|------------|----------------|-----------------|
| Kickstarter | 74,500 | 149,000 | 149,000 | 2,681 | 31,755 |
| Consumer | 745,000 | 1,490,000 | 1,490,000 | 2,681 | 31,755 |
| Network Device | 27,980,000 | 55,960,000 | 55,960,000 | 2,681 | 31,755 |

4.2.3 Empirically-Derived Probabilities

Our experiment determined two of the core probabilities of our model: P_D and P_{FA} . For every (σ_A, σ_D) tuple, $P_D(\sigma_A, \sigma_D)$ was developed by testing every individual defender detection strategy (σ_D) against all circuits in the benchmark set that contained an HTH that was categorized as a member of adversary strategy set σ_A . $P_D(\sigma_A, \sigma_D)$ 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 σ_{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.

³³ One may argue a designer would not forensically examine to determine if the HTH indication is false or not in an IP core: they would simply select a different IP core with the same functionality. This alternative response would require the license of a second IP core. For our purposes, we assume this cost to be in the same order of magnitude as the one we modeled.

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 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 [168,169]. 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.³⁴ 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 launches each commercial or custom EDA tools 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. Future versions of this test harness will include more complex software applications, but this arrangement was sufficient for the testing performed here. Both the TSS cloud and future automation strategies required to develop much larger test sets are described in more detail in Section 6.3.

The complete set of raw results are listed in Appendix A on page 150. Note that the tables in this section include the expanded set of defender strategies that result when allowing the defender to make use of all possible combinations of CSIM, SCOA, STRC,

³⁴ 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.

and BOOL. Section 5.2 discusses the combinations in more detail. It also includes a summary of these results and the analysis of the games that result when using those P_D and P_{FA} values.

4.3 Automation and GameRunner

In order to automate the process of defining, solving, and exploring game solutions, we developed the concepts alluded to in [8] into a software tool entitled GameRunner, the architecture of which is illustrated in Figure 7 below. 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.³⁵ 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 [146]. 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 solvers for our set through a review of computer algebra systems [148], game-solving software tools [148], and software solver libraries [147] which led us to believe that the following three were promising:

- The set of solvers provided by Gambit [148], an open-source, cross-platform C/C++ library for game-theory computation.
- The EEE algorithm as implemented by [151] as a Java application, which we call, parsing its output by redirecting STDOUT.

³⁵ 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.

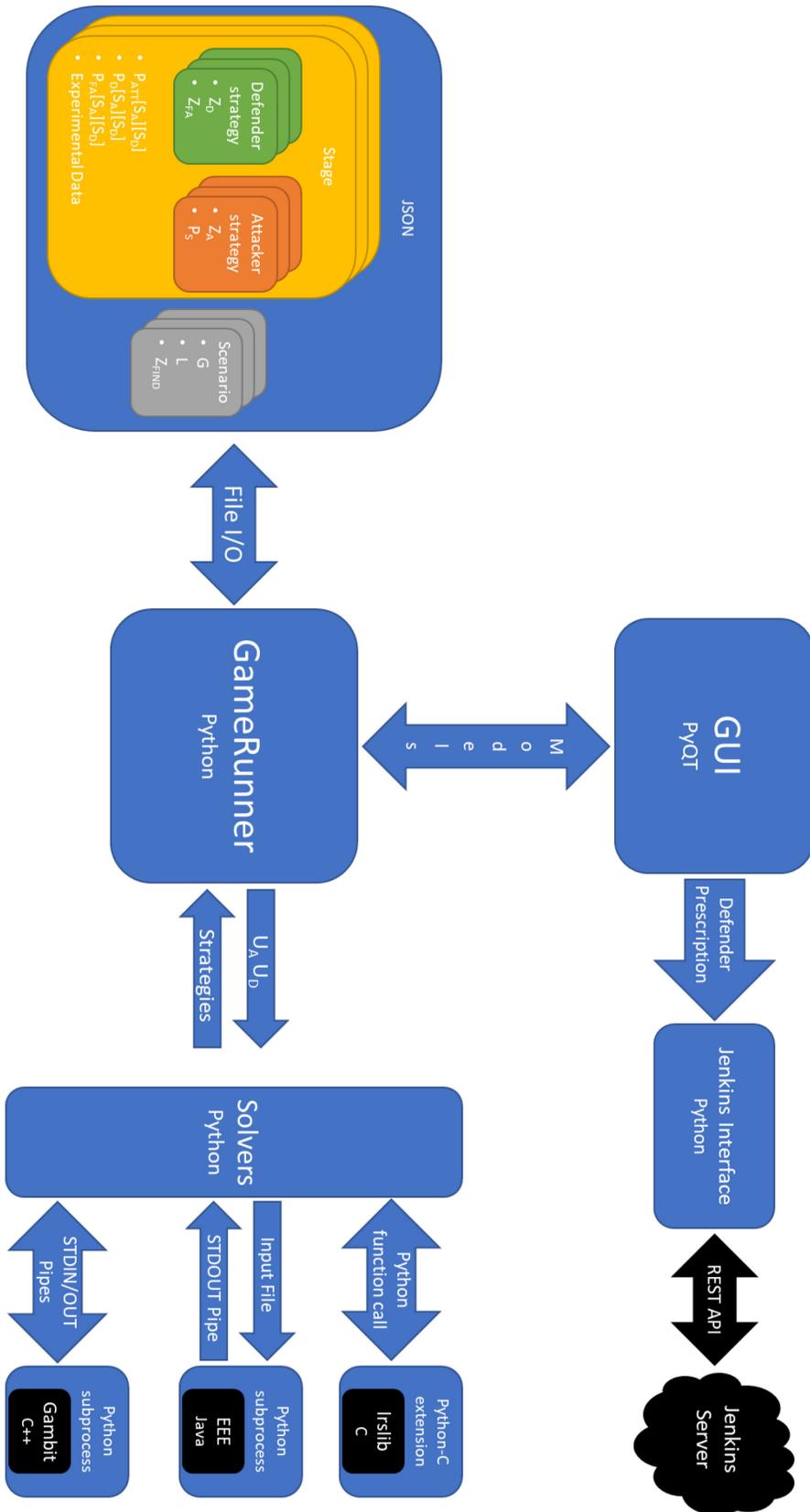


Figure 7. GameRunner Block Diagram [9]

- The LRS libraries [152]: C libraries which implement the lexicographic reverse search vertex enumeration algorithm in the form of [153]. 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. Notably, these libraries contain a single-threaded solver (*lrs*) as well as a multi-threaded solver that issues parallel threads on one machine (*plrs*) and a multi-threaded solver that issues parallel threads that can be computed across multiple machines using MPI libraries (*mplrs*).

We compared these three to each other using thousands of 25x25 games. Comparing for performance³⁶, implementation stability, and the correctness of the result, we selected *lrs* as the solver for the solutions presented in this dissertation. Its performance was adequate, and its solutions were consistently correct. As we will see in the next section, many of our game exploration methods require re-solving games repeatedly with slight variations on the utility functions, which made file IO a bottleneck. We addressed this bottleneck by writing a Python C extension that allows us to call the *lrs* solvers directly in Python, avoiding the use of STDOUT/STDIN. This resulted in an order-of-magnitude performance improvement when solving multiple small games repeatedly, as will be

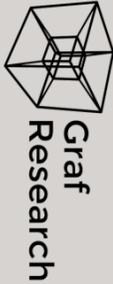
³⁶ The computational complexity of these solver engines depends on many factors, including not only (1) the size of the game in question but also (2) its game-specific properties (e.g., degeneracy) and (3) the properties of the restatement of the game as a problem mapped into a new solution domain (a common method of automated solvers). This makes simple formal comparisons of the engines difficult. Established practice is to empirically compare solvers across a reasonable variety of games with different sizes, properties, and implementation strategies, as was done by Avis and Jordan in [147]. Our goal in this work is to find a solver sufficient to provide correct answers in a time that is reasonable to allow our desired analytics; we do not seek to improve the solver. We relied on the work of Avis and Jordan and our own partial replication of its comparison methods to make this selection.

common in our use of GameRunner’s analysis and “what if” features. We will see these features illustrated in the next chapter.

There are promising features of each of the three solvers that may lead us to transition solvers in the future. For Gambit, one item we did not explore were some of its advanced game solving algorithms which, in some cases, have performance that greatly improved upon *lrs*. However, they require guiding the game in unique ways by performing some pre-conditioning based on assumptions about the game structure, which would require us to modify the structure of our game specifically for Gambit. For the EEE algorithm, we would like to reconsider it when implemented in C or C++ to perform a more controlled performance comparison. For the LRS libraries, the Nash solver was written to call the *lrs* solver, not *plrs* or *mplrs*. Rewriting to call one or the other parallel solver was scoped to be nontrivial. In the end, we did not pursue game solver performance improvements, since the *lrs* approach gave us performance sufficient for the analysis in the Results section below. However, if we run into performance limitations in the game solver in the future, we will revisit one or more of the above avenues.

Figure 8 illustrates the basic GameRunner interface. Reading data from the JSON file, it populates the strategy data and game inputs into the left two columns. The user may edit the strategy set, probability values, or economic values and select a solver algorithm. The utility functions are populated when the “Calculate” button is pressed, and the “Solve” button sends the utility matrices to the game solver. This populates the equilibrium windows. If a pure strategy equilibrium is discovered, it is indicated. If mixed strategies are discovered, they are displayed and highlighted, along with the probability of play. The GameRunner GUI also includes a number of features not pictured in the figure, including many ways of 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 chapter were generated directly by the GameRunner tool.

Chapter 4. Experimental Game



**Graf
Research**

GameRunner

File View Tools

Strategies

Stage HDL + -

Adversary + -

oA0 DONT: Do nothing
 oA1 REWR: Always on change - rewrite
 oA2 GATE: Triggered - change counting gates
 oA3 ECTR: Triggered - combinational comparator
 oA4 CCMF: Triggered - combinational comparator
 oA5 STSQ: Triggered - by state sequence
 oA6 CLST: Triggered - glitch state

Inputs

Scenario Consumer

Gain (G) L/2

Loss (L) \$1,490,000.00

ZFind C+2

Adversary Data

| Adversary | PS | ZD | ZFA |
|-----------|----------|---------|--------------------------|
| oA0 DONT | \$20,106 | 0.00% | \$2,681 |
| oA1 REWR | \$80,426 | 100.00% | oD1 CSIM \$2,681 ZFA[0] |
| oA2 GATE | ZA[1] | 100.00% | oD2 SCOA \$17,681 ZFA[0] |
| oA3 ECTR | ZA[1] | 100.00% | oD3 STRC \$17,681 ZFA[0] |
| oA4 CCMF | ZA[1] | 100.00% | oD4 BOOL \$69,021 ZFA[0] |
| oA5 STSQ | ZA[1] | 100.00% | |

Probability of Attribution (PAAT)

| Adversary | PS | ZD | ZFA |
|-----------|-------|-------|-------|
| oA0 DONT | 0.00% | 0.00% | 0.00% |
| oA1 REWR | 5.00% | 5.00% | 5.00% |
| oA2 GATE | 5.00% | 5.00% | 5.00% |
| oA3 ECTR | 5.00% | 5.00% | 5.00% |
| oA4 CCMF | 5.00% | 5.00% | 5.00% |
| oA5 STSQ | 5.00% | 5.00% | 5.00% |

Defender Data

| Defender | PS | ZD | ZFA |
|----------|----------|-------|----------|
| oD0 BNCH | \$0 | 0.00% | \$2,681 |
| oD1 CSIM | \$2,681 | 0.00% | oA1 REWR |
| oD2 SCOA | \$17,681 | 0.00% | oA2 GATE |
| oD3 STRC | \$17,681 | 0.00% | oA3 ECTR |
| oD4 BOOL | \$69,021 | 0.00% | oA4 CCMF |

Detection Probabilities (pD)

| Adversary | PS | ZD | ZFA |
|-----------|--------|---------|---------|
| oA0 DONT | 0.00% | 0.00% | 0.00% |
| oA1 REWR | 87.50% | 100.00% | 100.00% |
| oA2 GATE | 50.00% | 100.00% | 100.00% |
| oA3 ECTR | 53.33% | 73.33% | 100.00% |
| oA4 CCMF | 30.77% | 46.15% | 100.00% |
| oA5 STSQ | 13.33% | 60.00% | 100.00% |

False Alarm Probabilities (pFA)

| Defender | PS | ZD | ZFA |
|----------|-------|--------|--------|
| oD0 BNCH | 0.00% | 0.00% | 16.67% |
| oD1 CSIM | 0.00% | 50.00% | 16.67% |
| oD2 SCOA | 0.00% | 50.00% | 16.67% |
| oD3 STRC | 0.00% | 50.00% | 16.67% |
| oD4 BOOL | 0.00% | 50.00% | 16.67% |

Defender

| Adversary | PS | ZD | ZFA |
|-----------|----|-----|-----|
| oA0 DONT | 0 | -3 | -19 |
| oA1 REWR | 0 | 184 | -18 |
| oA2 GATE | 0 | -3 | -19 |
| oA3 ECTR | 0 | 227 | 379 |
| oA4 CCMF | 0 | 439 | 211 |
| oA5 STSQ | 0 | 676 | 181 |
| oA6 CLST | 0 | -3 | 727 |

Adversary

| Defender | PS | ZD | ZFA |
|----------|------|------|------|
| oD0 BNCH | -20 | -20 | -20 |
| oD1 CSIM | -155 | -155 | -155 |
| oD2 SCOA | 255 | 255 | -155 |
| oD3 STRC | 228 | 64 | 9 |
| oD4 BOOL | 412 | 286 | 286 |
| oD5 STSQ | 555 | 173 | 446 |
| oD6 CLST | 665 | 255 | 665 |

Mixed-Strategy Equilibrium

| | |
|----------|------|
| oA0 DONT | 0% |
| oA1 REWR | 0% |
| oA2 GATE | 0% |
| oA3 ECTR | 0% |
| oA4 CCMF | 0% |
| oA5 STSQ | 0% |
| oA6 CLST | 100% |

Pure-Strategy Equilibrium

| | |
|----------|-----|
| oD0 BNCH | 0% |
| oD1 CSIM | 0% |
| oD2 SCOA | 70% |
| oD3 STRC | 30% |
| oD4 BOOL | 0% |

Solutions

Isbnash-extension Solve

G ZA ZFind PD
 L ZD ZFA PFA

One error per calculation

Utility Matrices

Calculate Adversary Random

| Adversary | oD0 BNCH | oD1 CSIM | oD2 SCOA | oD3 STRC | oD4 BOOL |
|-----------|----------|----------|----------|----------|----------|
| oA0 DONT | -20 | -20 | -20 | -20 | -20 |
| oA1 REWR | -52 | -155 | -255 | -155 | -155 |
| oA2 GATE | 255 | 255 | 255 | -155 | -155 |
| oA3 ECTR | 228 | 64 | 9 | -155 | -155 |
| oA4 CCMF | 412 | 286 | 160 | 286 | -155 |
| oA5 STSQ | 555 | 173 | 446 | 446 | -155 |
| oA6 CLST | 665 | 255 | 255 | 665 | 665 |

Figure 8. GameRunner Graphical User Interface

Chapter 5. Results

In this chapter, we start by listing the raw output of the game results and discussing the Nash equilibria of each game. 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 12 summarizes our experimental P_D results; Table 13, the 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 12. 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% |

Table 13. P_{FA} by Defender Strategy

| BNCH | CSIM | SCOA | STRC | BOOL |
|------|------|------|------|------|
| 0% | 0% | 50% | 17% | 0% |

Table 14 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 (PD and PFA) are drawn from Table 12 and Table 13, and the remaining utility function variables are set as described in Chapter 4. This table – as with all in this dissertation where we depict utility function results – is listed rounded to the nearest thousand dollars.

Table 14. Two-Player Strategic Game: HDL Step, Kickstarter Economy

| | | $U_D(\sigma_A, \sigma_D)$ (\$ Thousands) | | | | |
|---|------|--|----------|-----------|-----------|-----------|
| | | BNCH | CSIM | SCOA | STRC | BOOL |
| $U_A(\sigma_A, \sigma_D)$ (\$ Thousands) | DONT | (-20,0) | (-20,-3) | (-20,-19) | (-20,-18) | (-20,-69) |
| | REWR | (-78,0) | (-88,16) | (-88,0) | (-78,-18) | (-88,-50) |
| | GATE | (-47,0) | (-47,-3) | (-47,-19) | (-88,56) | (-88,5) |
| | ECTR | (-50,0) | (-50,-3) | (-66,11) | (-71,22) | (-88,1) |
| | CCMP | (-31,0) | (-44,20) | (-56,27) | (-44,5) | (-88,34) |
| | STSQ | (-17,0) | (-17,-3) | (-55,51) | (-28,2) | (-88,60) |
| | GLST | (-6,0) | (-6,-3) | (-47,55) | (-47,56) | (-6,-69) |

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 don't attack 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.

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 we will see in Section 6.3.1, 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 PD given the adversary's likely action (STRC). Note that if the rationality of the players is in question, this should have been modeled prior to the point of accomplishing a game solution, which is also discussed in Chapter 6.

The tables for the Consumer economy and the Network economy are in Table 15 and Table 16, below.

Table 15. Two-Player Strategic Game: HDL Step, Consumer Economy

| $U_D(\sigma_A, \sigma_D)$ (\$ Thousands) | | | | | |
|--|--------------|------------|------------|------------|-------------|
| | BNCH | CSIM | SCOA | STRC | BOOL |
| $U_A(\sigma_A, \sigma_D)$ (\$ Thousands) | DONT (-20,0) | (-20,-3) | (-20,-19) | (-20,-18) | (-20,-69) |
| | REWR (-52,0) | (-155,184) | (-155,167) | (-52,-18) | (-155,117) |
| | GATE (255,0) | (255,-3) | (255,-19) | (-155,727) | (-155,676) |
| | ECTR (228,0) | (228,-3) | (64,279) | (9,379) | (-155,626) |
| | CCMP (412,0) | (286,227) | (160,439) | (286,211) | (-155,963) |
| | STSQ (555,0) | (555,-3) | (173,676) | (446,181) | (-155,1222) |
| | GLST (665,0) | (665,-3) | (255,726) | (255,727) | (665,-69) |

Table 16. Two-Player Strategic Game: HDL Step, Network Device Economy

| $U_D(\sigma_A, \sigma_D)$ (\$ Thousands) | | | | | |
|--|----------------|--------------|---------------|---------------|---------------|
| | BNCH | CSIM | SCOA | STRC | BOOL |
| $U_A(\sigma_A, \sigma_D)$ (\$ Thousands) | DONT (-20,0) | (-20,-3) | (-20,-19) | (-20,-18) | (-20,-69) |
| | REWR (969,0) | (-2878,6992) | (-2878,6976) | (969,-18) | (-2878,6926) |
| | GATE (12511,0) | (12511,-3) | (12511,-19) | (-2878,27962) | (-2878,27911) |
| | ECTR (11485,0) | (11485,-3) | (5329,11173) | (3277,14905) | (-2878,26046) |
| | CCMP (18429,0) | (13694,8607) | (8959,17199) | (13694,8591) | (-2878,38673) |
| | STSQ (23796,0) | (23796,-3) | (9433,26096) | (19692,7443) | (-2878,48430) |
| | GLST (27900,0) | (27900,-3) | (12511,27961) | (12511,27962) | (27900,-69) |

In both economies, the defender’s optimal play is to play SCOA 70% of the time and STRC 30% of the time. It may be surprising that it is never optimal for the defender to play BOOL, the one solution (per Table 12 and Table 13) 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. We will reconsider this question shortly.

The mixed strategy solutions to the 3PIP step game (where BOOL is not available to the defender) follow a similar pattern.

Table 17. Two-Player Strategic Game: 3PIP Step, Kickstarter Economy

| | | $U_D(\sigma_A, \sigma_D)$ (\$ Thousands) | | | | |
|--|------|--|----------|-----------|-----------|----------|
| | | BNCH | CSIM | SCOA | STRC | |
| $U_A(\sigma_A, \sigma_D)$ (\$ Thousands) | DONT | (-20,0) | (-20,-3) | (-20,-34) | (-20,-23) | |
| | REWR | (-78,0) | (-88,16) | (-88,-15) | (-78,-23) | |
| | GATE | (-47,0) | (-47,-3) | (-47,-34) | (-88,52) | |
| | ECTR | (-50,0) | (-50,-3) | (-66,-4) | (-71,17) | |
| | CCMP | (-31,0) | (-44,20) | (-56,12) | (-44,0) | |
| | STSQ | (-17,0) | (-17,-3) | (-55,36) | (-28,-3) | |
| | | GLST | (-6,0) | (-6,-3) | (-47,41) | (-47,52) |

In the 3PIP Kickstarter game, the adversary should play DONT 69% of the time and GLST 31% of the time; the defender should play BNCH 65% of the time and STRC 35% of the time.

Table 18. Two-Player Strategic Game: 3PIP Step, Consumer Economy

| | | $U_D(\sigma_A, \sigma_D)$ (\$ Thousands) | | | |
|--|------|--|------------|------------|------------|
| | | BNCH | CSIM | SCOA | STRC |
| $U_A(\sigma_A, \sigma_D)$ (\$ Thousands) | DONT | (-20,0) | (-20,-3) | (-20,-34) | (-20,-23) |
| | REWR | (-52,0) | (-155,184) | (-155,153) | (-52,-23) |
| | GATE | (255,0) | (255,-3) | (255,-34) | (-155,722) |
| | ECTR | (228,0) | (228,-3) | (64,264) | (9,374) |
| | CCMP | (412,0) | (286,227) | (160,425) | (286,206) |
| | STSQ | (555,0) | (555,-3) | (173,662) | (446,176) |
| | GLST | (665,0) | (665,-3) | (255,711) | (255,722) |

In the 3PIP Consumer game, the adversary has again reached the point where they should always attack: 2% of the time with STSQ and 98% of the time with GLST. The defender should defend with SCOA 70% of the time and STRC 30% of the time.

Table 19. Two-Player Strategic Game: 3PIP Step, Network Device Economy

| | | $U_D(\sigma_A, \sigma_D)$ (\$ Thousands) | | | |
|--|------|--|--------------|---------------|---------------|
| | | BNCH | CSIM | SCOA | STRC |
| $U_A(\sigma_A, \sigma_D)$ (\$ Thousands) | DONT | (-20,0) | (-20,-3) | (-20,-34) | (-20,-23) |
| | REWR | (969,0) | (-2878,6992) | (-2878,6961) | (969,-23) |
| | GATE | (12511,0) | (12511,-3) | (12511,-34) | (-2878,27957) |
| | ECTR | (11485,0) | (11485,-3) | (5329,11158) | (3277,14900) |
| | CCMP | (18429,0) | (13694,8607) | (8959,17185) | (13694,8586) |
| | STSQ | (23796,0) | (23796,-3) | (9433,26081) | (19692,7438) |
| | GLST | (27900,0) | (27900,-3) | (12511,27946) | (12511,27957) |

As with the HDL game, by the time the economy takes the shape of the 3PIP Network game, the adversary is attacking with the GLST HTH nearly 100% of the time, and the defender is defending with SCOA 70% of the time and STRC 30%.

At this point we should stress again that these results hold for the game we constructed, not necessarily for any specific defender who may be reading this dissertation. In our future results section, we revisit the question of what it would take to use this game to be prescriptive for industry.

5.2 Analysis

These results may be analyzed through many lenses. In this section, we try on a few. First, we consider the question of what actions the players may have taken – and what outcomes they could have expected – had they not been assisted by the Nash equilibrium solution concept. We do this in Section 5.2.1 by comparing the outcome anticipated by Expected Utility Theory (EUT) to that prescribed by the Nash equilibrium. Next, we question our assumptions for the economic variables and perform an analysis of the sensitivity of our game to errors in those assumptions in Section 5.2.2. In Section 5.2.3, we explore additional visual analytic techniques enabled by GameRunner.

5.2.1 Expected Utility Theory and the Nash Equilibrium

To perform comparison between the outcome prescribed by the Nash equilibrium and other potential outcomes of the modeled encounter, we may use EUT. In this analysis, we might phrase our question as, “What if we knew the security economic information but did not have access to the Nash equilibrium?” Expected utility theory allows us to quantify what outcome we might expect. For our analysis, we consider the resolution of the utility functions in Table 15 to be the *consequences* (per the nomenclature of EUT) of electing to play that strategy tuple. In this first application of EUT, we consider only pure strategy tuples for simplicity. If the adversary has m pure strategies, the defender has n pure strategies, and $x = m \times n$ is the total number of possible consequences of pure strategy pairings, EUT requires us to assign a probability p_1, p_2, \dots, p_x , to each consequence such that

$$\sum_{i=1}^x p_i = 1.$$

The adversary and defender may each set their own probability values for each outcome. The components of the adversary’s set are represented by p_a values and the defender’s by

p_d values.³⁷ To represent the consequences, we set up a consequence function U_a for the adversary and U_d for the defender. We develop our consequence function entries by stepping through the table of adversary and defender utility functions and setting $U_a(i) = U_A(\sigma_A, \sigma_D)$ for each entry, with i in the range of 1 to x , σ_A in the range of 1 to m , and σ_D in the range of 1 to n . Similarly, we step through the table and set $U_d(i) = U_D(\sigma_A, \sigma_D)$ for each entry in the table with i in the range of 1 to x , σ_A in the range of 1 to m , and σ_D in the range of 1 to n . Each player may establish an expected utility for an encounter by assigning a probability to the related consequence in the table and summing. For the adversary, we have

$$EU_a = \sum_{i=1}^x U_a(i)p_{a_i}. \quad \text{Eq. 4. Adversary Expected Utility}$$

And for the defender, we have

$$EU_d = \sum_{i=1}^x U_d(i)p_{d_i}. \quad \text{Eq. 5. Defender Expected Utility}$$

In the absence of Nash or another guiding principle, each player is unguided with respect to the play of their opponent. In this initial analysis, they may consider every consequence to be equally probable. That is, $p_{a_i} = p_{d_i} = \frac{1}{x}$. If we perform this EUT analysis on the HDL game in the Consumer economy and permit only pure strategy play, we get the following expected utility for the adversary:

$$EU_a = \sum_{i=1}^x U_a(i)p_{a_i} = \sum_{i=1}^x \frac{U_a(i)}{x} = \$191,270.99.$$

Similarly, we get the following expected utility for the defender:

$$EU_d = \sum_{i=1}^x U_d(i)p_{d_i} = \sum_{i=1}^x \frac{U_d(i)}{x} = \$167,830.94.$$

³⁷ The standard variables of EUT are similar to other variables elsewhere in this document. To be clear, P_D is the probability of detection in our empirical testing, whereas p_{d_i} is the probability of the i^{th} defender consequence in our EUT analysis.

For relative reference, note that the total market value of the Consumer economy is \$1,490,000. The Nash equilibrium outcome was \$254,824.00 for the adversary and \$726,246.60 for the defender. For the adversary, the Nash prescription modestly exceeds the expected utility derived by EUT; for the defender, the Nash prescription is considerably better.

We may also use EUT to compare the outcome of an unguided player who is playing against an opponent guided by the Nash equilibrium solution. While it may be difficult to imagine a pAPT adversary who goes to the trouble of infiltrating a network but does not perform some optimization on the best possible means of exploitation, it is quite easy to imagine a defender who simply does not want to go to the trouble of any defensive optimization when they would prefer simply to focus on design. To model this scenario, we again make use of the HDL game in the Consumer economy. The adversary plays the Nash-prescribed optimal strategy and accomplishes their aforementioned outcome of \$254,824.00. We may enumerate the available consequences under EUT for the defender as the set available when limited by the adversary's election to play the Nash equilibrium strategy.

In this analysis, the defender consequences under mixed strategies need to be considered, since the Nash equilibrium itself in this game is a mixed strategy.³⁸ To enumerate the consequences, we use an approximation method enabled by GameRunner, which is comprehensively explained in Section 5.4. For this section, briefly, the utility functions for the defender are calculated against an adversary playing the Nash equilibrium and the defender playing the mixed strategy sets allowed when strategy contributions are considered in increments of 10%. Using this strategy set as the x contributors to the consequence function, U_d , and again considering all $p_{a_i} = \frac{1}{x}$, we find that $EU_d = \$274,588.49$. That is, in a population of defenders who simply pick a mixed strategy at random in the face of an adversary who optimizes using the Nash equilibrium solution

³⁸ As a reminder, the Nash equilibrium called for the adversary to play GLST nearly 100% and STSQ nearly 0% of the time and for the defender to play SCOA about 70% and STRC about 30% of the time.

concept, the defenders should expect an average utility of \$274,588.49. This is again far from the Nash-optimal value.

What perhaps might be more likely, however, is that in a population of defenders, there are many who would be biased towards complete inaction. This scenario can be modeled by saying that half of the defenders in a population will elect to do nothing, and the other half will pick a defensive strategy at random from the remaining mixed strategies. We model this by setting the p_d value for the consequence of the defender's pure DONT strategy to 0.5 and set $p_{d_i} = \frac{0.5}{x-1}$ for the consequence of all other strategies in the same set. In this case, $EU_d = \$137,454.07$, which is worse than the expected utility of picking completely at random.³⁹

This section establishes that the Nash equilibrium strategy is *better* than what either opponent could reasonably expect from unguided play. For further illustration of the question of whether this means the play is *optimal*, please see Section 5.4.

5.2.2 Analysis of Sensitivity to Error

A likely source of error in our game model are the economic variables. While the probability values have an empirical basis, the economic variables are derived from estimated market values. We would like to determine how sensitive the outcome of our games might be to an error in estimation. To demonstrate how this analysis, we again make use of the HDL Game in the Consumer economy. We analyze the results of varying the economic values – G , L , Z_{find} , and all Z_A , Z_D , and Z_{FA} values– each in turn by +/- 50% of their estimated value. We then re-solve the game at those points and graph the outcomes using a GameRunner feature that will be more fully explained in Section 5.2.3.

This analysis revealed that there is no change in the Nash equilibrium across this range of error when considering G , L , Z_{find} , and all Z_{FA} values. Furthermore, there was no change

³⁹ While we noted it may be less reasonable to expect, the converse analysis is also possible. If a defender plays the Nash equilibrium strategy and the adversary chooses any random mixed strategy, the adversary's expected utility $EU_a = \$112,691.85$, using the same approximation of mixed strategies method.

in the Nash equilibrium across this error range for $Z_A(\text{DONT})$, $Z_A(\text{REWR})$, $Z_A(\text{GATE})$, $Z_A(\text{ECTR})$, and $Z_A(\text{CCMP})$ as well as $Z_D(\text{BNCH})$, $Z_D(\text{CSIM})$, and $Z_D(\text{BOOL})$. The only variables for which the mixed strategy Nash equilibrium betrayed any sensitivity in this range of error were for costs associated with the strategies within the Nash equilibrium: $Z_D(\text{SCOA})$, $Z_D(\text{STRC})$, $Z_A(\text{STSQ})$, and $Z_A(\text{GLST})$. Prior to performing this analysis, it is helpful to recall the experimental performance of said strategies. Reducing the data from Table 12 and Table 13 to only show SCOA and STRC performance, we depict P_D and P_{FA} data in Table 20 and Table 21, respectively.

Table 20. P_D by Adversary and Defender Strategy (SCOA, STRC)

| | SCOA | STRC |
|--------------|------------|------------|
| DONT | 0% | 0% |
| REWR | 100% | 88% |
| GATE | 50% | 100% |
| ECTR | 73% | 80% |
| CCMP | 62% | 46% |
| STSQ | 60% | 27% |
| GLST | 50% | 50% |
| Total | 68% | 60% |

Table 21. P_{FA} by Defender Strategy (SCOA, STRC)

| SCOA | STRC |
|------|------|
| 50% | 17% |

Note that SCOA and STRC are equally effective against the adversary’s favored GLST Trojan. SCOA has a much higher overall false alarm rate but better detection performance against the adversary’s significantly less favored STSQ strategy. Figure 9 shows the effect of changing $Z_D(\text{SCOA})$ from 50% to 100% of its estimated value. When SCOA is very inexpensive (closer to 50% of its estimated value), the cost of its high false alarm rate is overwhelmed by its superior performance against STSQ. At these low values SCOA is the favored defensive strategy nearly 100% of the time. As the cost of SCOA approaches 95% of the estimated value, the base game’s Nash equilibrium mixed strategy becomes the

Chapter 5. Results

optimal play: SCOA 70% and STRC 30% of the time. This strategy remains the Nash optimal strategy all the way through 150% of the estimated value. The adversary strategy of playing STSQ nearly 0% of the time and GLST nearly 100% of the time does not change.⁴⁰

Similarly, Figure 10 depicts the sweep of $Z_D(\text{STRC})$ from 50% to 150% of its estimated value. From 50% to about 105% of the estimated STRC cost, the Nash equilibrium remains the same as the base game. But when STRC becomes too expensive (above 105% of the estimated costs), the SCOA strategy becomes the overwhelming favorite. It is clear that the relative cost between two closely matched defensive strategies (SCOA and STRC) and their performance differences between false alarms and STSQ detection are the variables driving the defensive strategy selection. As with the SCOA cost sensitivity analysis, the adversary strategy does not change.

⁴⁰ We selected two significant digits to represent the mixed strategy results for ease in depiction and discussion. Because of the degree to which assumptions guide the economic values, we did not believe that more than two digits could be justified. It is likely that the values are changing during this and other sweeps, just below the resolution of our analysis.

Chapter 5. Results

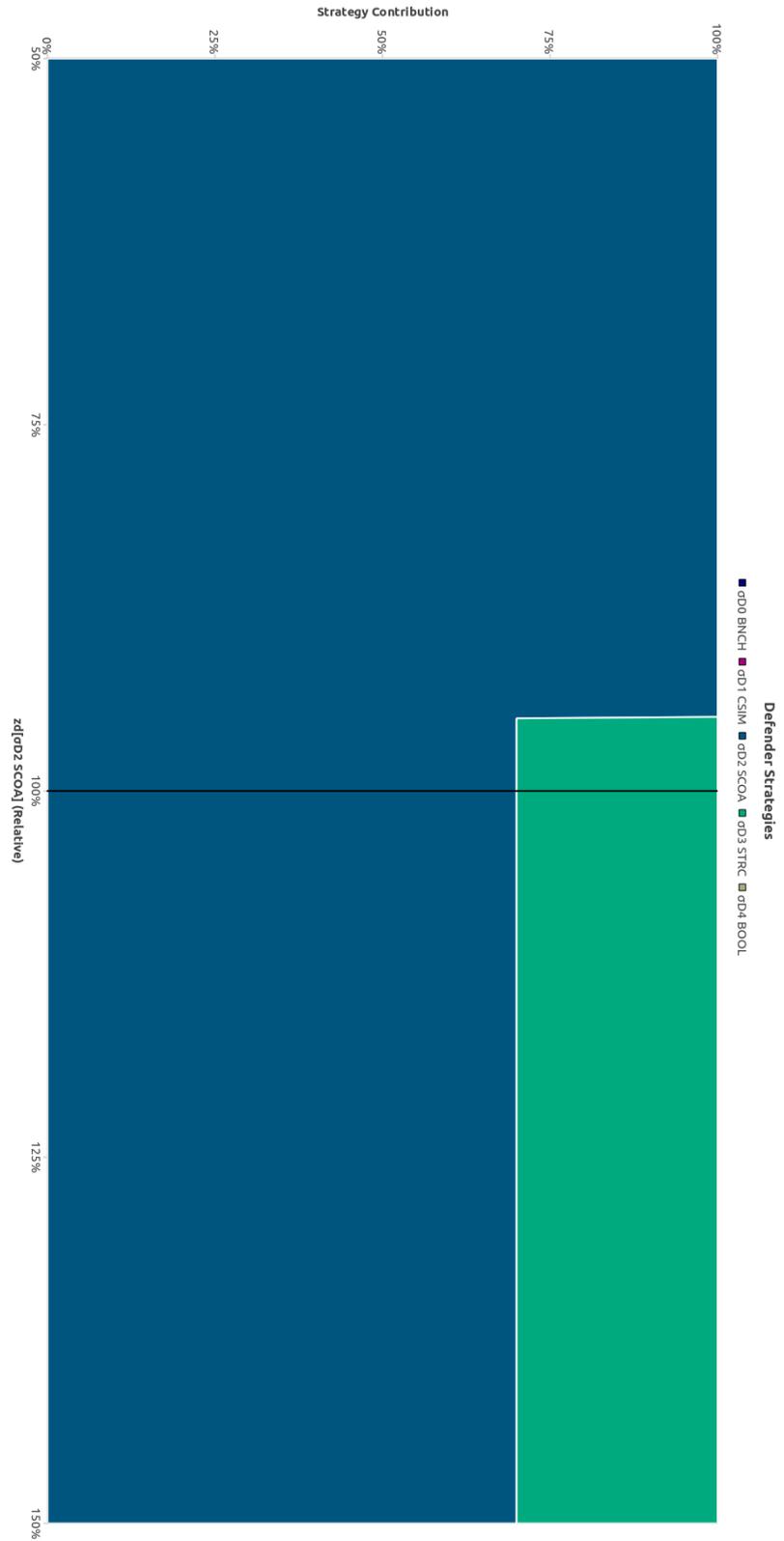


Figure 9. Defender Strategies as $Z_D(SCOA)$ sweeps from 50% to 150% of its estimated value; all other variables fixed; HDL Step Game in the Consumer Economy

Chapter 5. Results

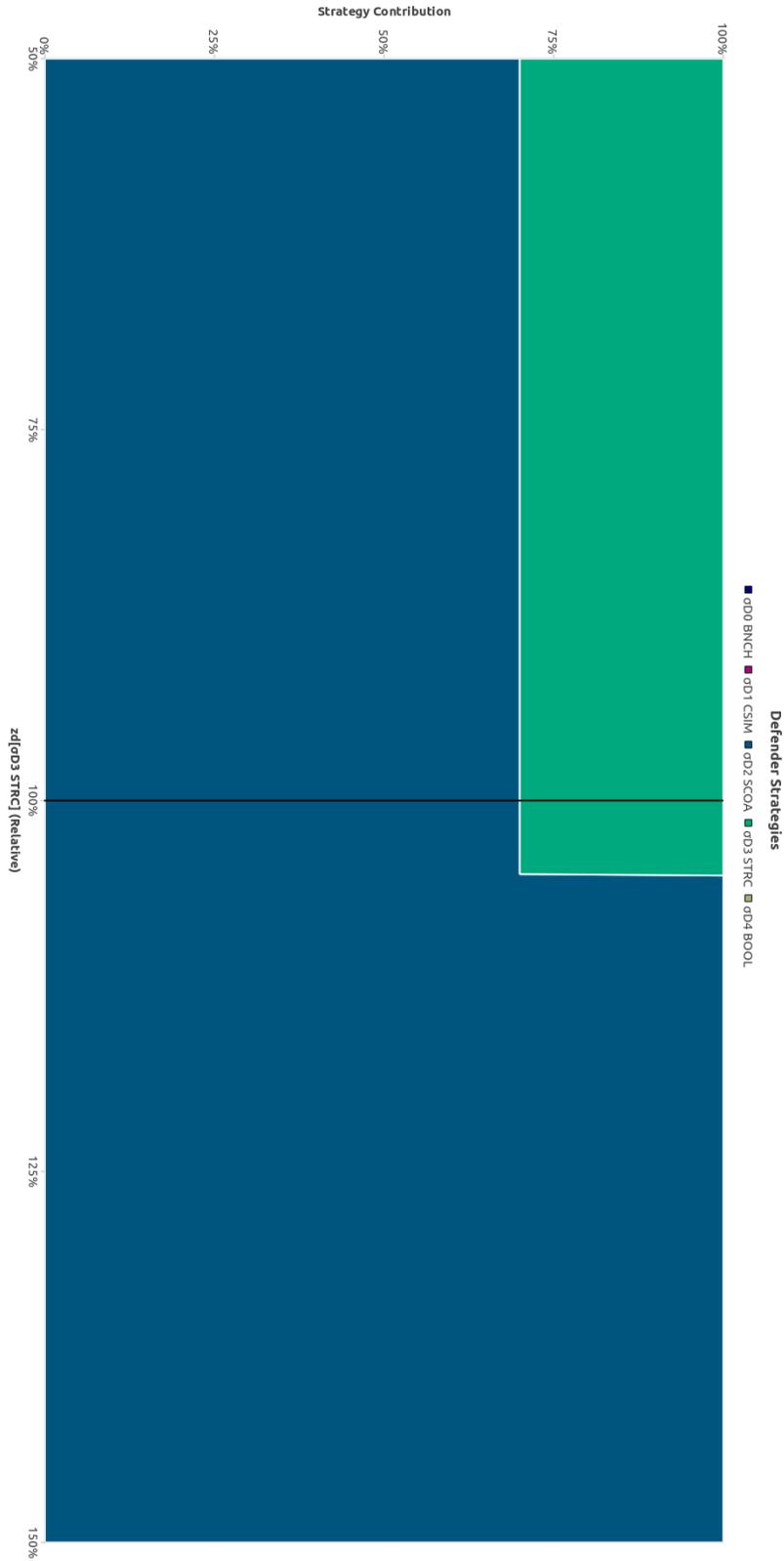


Figure 10. Defender Strategies as $Z_D(\text{STRC})$ sweeps from 50% to 150% of its estimated value; all other variables fixed; HDL Step Game in the Consumer Economy

Chapter 5. Results

The sensitivity analysis of the adversary costs $Z_A(\text{STSQ})$ and $Z_A(\text{GLST})$ shows a related outcome – for the defender. Never does the adversary’s strategy change from its nearly 100% GLST and nearly 0% SCOA. In Figure 11, we see the effect on the defender’s strategy of sweeping the value of $Z_A(\text{STSQ})$ from 50% to 150% of its estimated value. At its lowest extreme (50%), the defender plays SCOA 85% of the time and STRC 15% of the time. At the highest extreme (150%), the defender plays SCOA 55% of the time and STRC 45% of the time. If the cost of STSQ is lower, the chance of the adversary playing STSQ is very slightly greater (though still beneath the two significant digits we capture in our analysis), so the defender shifts towards SCOA, reasoning it is slightly more acceptable to suffer its higher false alarm rate if it means a dramatic improvement in the detectability of STSQ. If the cost of STSQ is greater, the chance of the adversary playing STSQ is again very slightly higher, so the defender shifts towards STRC.

For the same reasons, the cost of GLST has a similar effect, as depicted in Figure 12. When it is very inexpensive, the adversary is very slightly more likely to play it, leading to an increased defender preference for STRC (55% SCOA, 45% STRC). When it is very expensive, the adversary becomes very slightly less likely to play it as compared to STSQ, leading to a 85%/15% defender preference for SCOA/STRC.

Chapter 5. Results

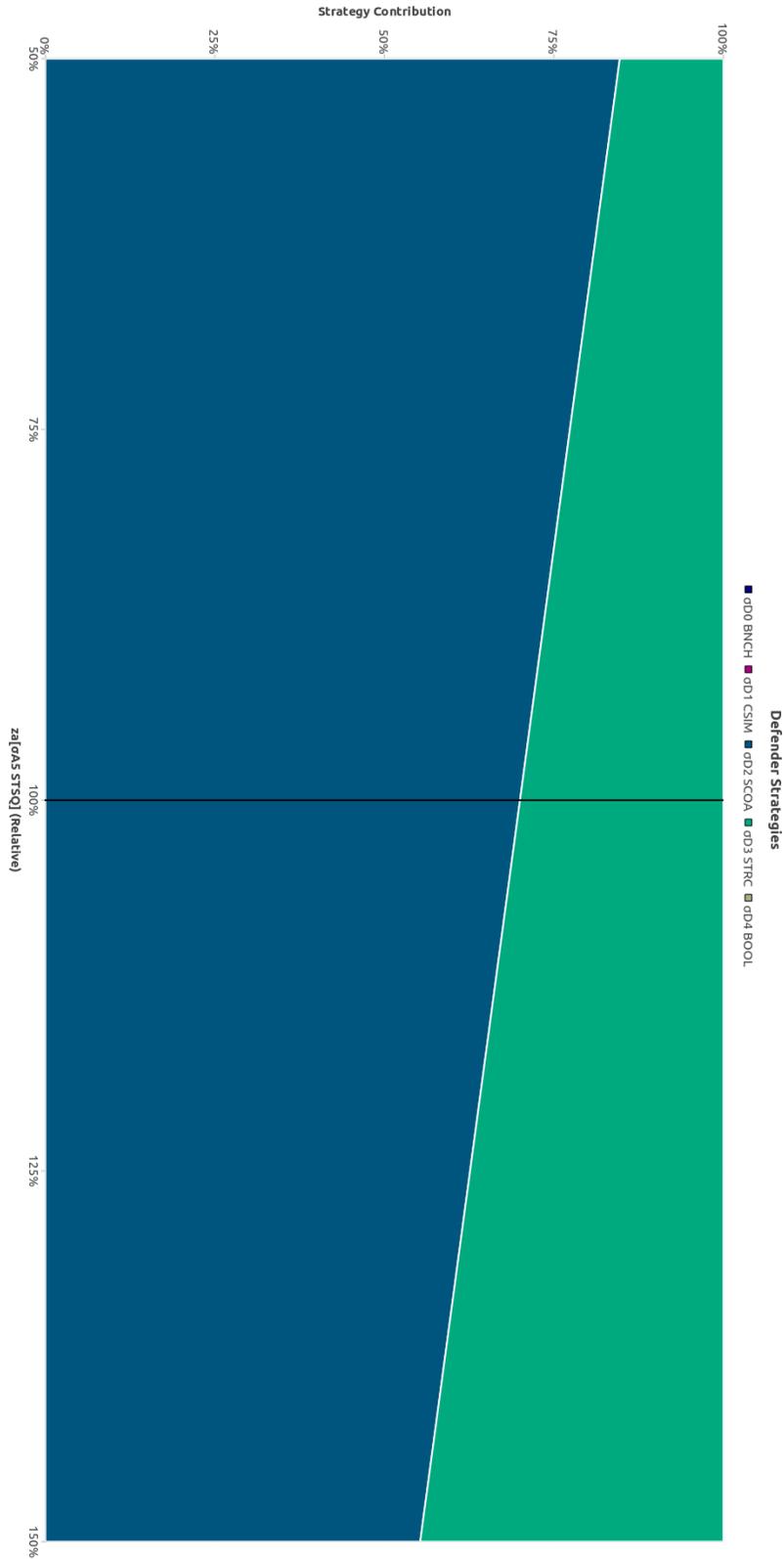


Figure 11. Defender Strategies as $Z_A(\text{STSQ})$ sweeps from 50% to 150% of its estimated value; all other variables fixed; HDL Step Game in the Consumer Economy

Chapter 5. Results

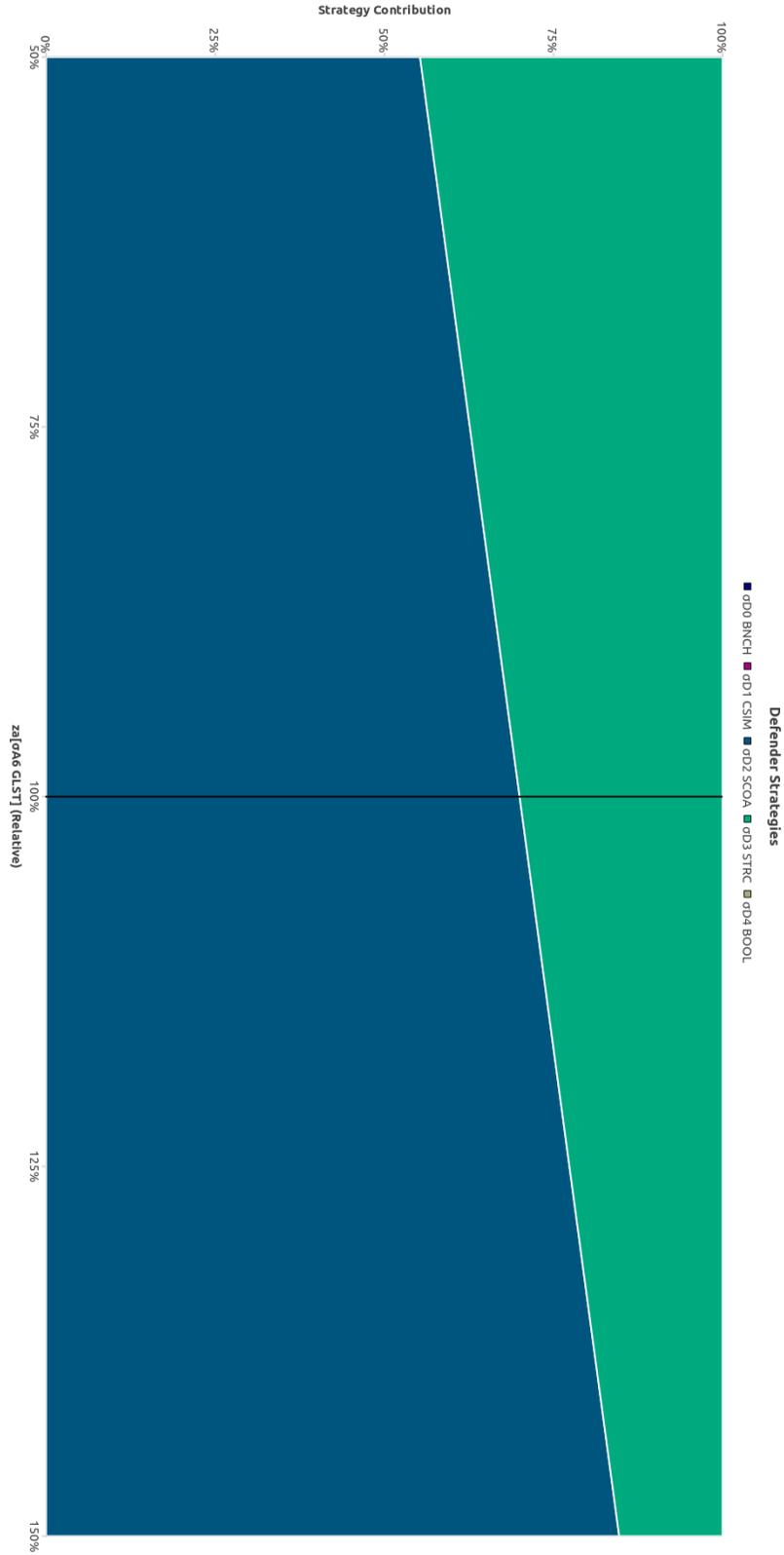


Figure 12. Defender Strategies as $Z_A(GLST)$ sweeps from 50% to 150% of its estimated value; all other variables fixed; HDL Step Game in the Consumer Economy

5.2.3 Further Exploration of Results with GameRunner

Figure 13 depicts the solution to the HDL game in the Kickstarter economy. 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 Figure 14. 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 variables constant, as determined in Chapter 4. That is the adversary gain G is always half of L and the consequence of getting caught, Z_{find} , is always twice the gain. In so doing, we maintain the market dynamic as we sweep L .⁴¹

The graph in Figure 14 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 line shows where $L = \$149,000$, since that was the value of the Kickstarter game that launched this analysis process in GameRunner.

⁴¹ Of course, GameRunner is also capable of making all these variables independent.

Chapter 5. Results



Graf Research

Strategies

Stage HDL

Adversary

- oA0 DONT: Do nothing
- oA1 REWR: Always on change - rewrite
- oA2 GATE: Always on change - changed gates
- oA3 ECTR: Triggered - event counter
- oA4 CCMP: Triggered - combinational comparator
- oA5 STSQ: Triggered - by state sequence
- oA6 GLST: Triggered - glitch state

Defender

- oD0 BNCH: Costimulation with testbench
- oD1 CSIM: Constrained Random Costimulation with fuzzer
- oD2 SCSA: Static controllability/observability analysis
- oD3 STRC: Structural pattern based static detection
- oD4 BOOL: Boolean logic equivalence testing

Inputs

Scenario Kickstarter

Gain (c) L2

Loss (l) \$149,000.00

Zfind Gr2

Adversary Data

| | | |
|----------|----------|---------|
| oA0 DONT | \$20,106 | 0.00% |
| oA1 REWR | \$80,426 | 100.00% |
| oA2 GATE | ZA[1] | 100.00% |
| oA3 ECTR | ZA[1] | 100.00% |
| oA4 CCMP | ZA[1] | 100.00% |
| oA5 STSQ | ZFA[1] | 100.00% |

Defender Data

| | | |
|----------|----------|---------|
| oD0 BNCH | \$0 | \$2,681 |
| oD1 CSIM | \$2,681 | ZFA[0] |
| oD2 SCSA | \$17,681 | ZFA[0] |
| oD3 STRC | \$17,681 | ZFA[0] |
| oD4 BOOL | \$69,021 | ZFA[0] |

Probability of Attribution (PA)

| | | | | |
|----------|-------|-------|-------|-------|
| oA0 DONT | 5.00% | 5.00% | 5.00% | 5.00% |
| oA1 REWR | 5.00% | 5.00% | 5.00% | 5.00% |
| oA2 GATE | 5.00% | 5.00% | 5.00% | 5.00% |
| oA3 ECTR | 5.00% | 5.00% | 5.00% | 5.00% |
| oA4 CCMP | 5.00% | 5.00% | 5.00% | 5.00% |
| oA5 STSQ | 5.00% | 5.00% | 5.00% | 5.00% |

Detection Probabilities (PD)

| | | | | |
|----------|---------|---------|---------|---------|
| oD0 BNCH | 0.00% | 0.00% | 0.00% | 0.00% |
| oD1 CSIM | 100.00% | 100.00% | 100.00% | 100.00% |
| oD2 SCSA | 87.50% | 100.00% | 100.00% | 100.00% |
| oD3 STRC | 50.00% | 100.00% | 100.00% | 100.00% |
| oD4 BOOL | 53.33% | 73.33% | 80.00% | 100.00% |
| oD5 STSQ | 30.77% | 46.15% | 46.15% | 100.00% |
| oD6 GLST | 13.33% | 76.67% | 100.00% | 100.00% |

False Alarm Probabilities (PPA)

| | | | | |
|----------|-------|--------|--------|-------|
| oD0 DONT | 0.00% | 50.00% | 16.67% | 0.00% |
| oD1 REWR | 0.00% | 50.00% | 16.67% | 0.00% |
| oD2 GATE | 0.00% | 50.00% | 16.67% | 0.00% |
| oD3 ECTR | 0.00% | 50.00% | 16.67% | 0.00% |
| oD4 CCMP | 0.00% | 50.00% | 16.67% | 0.00% |
| oD5 STSQ | 0.00% | 50.00% | 16.67% | 0.00% |
| oD6 GLST | 0.00% | 50.00% | 16.67% | 0.00% |

Utility Matrices

Adversary

| | | | | |
|----------|-----|-----|-----|-----|
| oA0 DONT | -20 | -20 | -20 | -20 |
| oA1 REWR | -78 | -88 | -88 | -78 |
| oA2 GATE | -47 | -47 | -47 | -88 |
| oA3 ECTR | -50 | -50 | -66 | -71 |
| oA4 CCMP | -31 | -44 | -56 | -44 |
| oA5 STSQ | -17 | -17 | -55 | -28 |
| oA6 GLST | -6 | -6 | -47 | -47 |

Defender

| | | | | | |
|----------|---|----|-----|-----|-----|
| oD0 BNCH | 0 | 16 | 0 | -19 | -18 |
| oD1 REWR | 0 | -3 | -19 | 56 | |
| oD2 GATE | 0 | -3 | 11 | 22 | |
| oD3 ECTR | 0 | 20 | 27 | 5 | |
| oD4 CCMP | 0 | -3 | 51 | 2 | |
| oD5 STSQ | 0 | -3 | 55 | 56 | |
| oD6 GLST | 0 | -3 | 55 | 56 | |

Solutions

Iternash-extension Error 0.00%

Solver Algorithm

G ZA Zfind PD

L ZD ZFA PPA

One error per calculation

Pure-Strategy Equilibrium

| | |
|----------|-----|
| oA0 DONT | 76% |
| oA1 REWR | 0% |
| oA2 GATE | 0% |
| oA3 ECTR | 0% |
| oA4 CCMP | 0% |
| oA5 STSQ | 0% |
| oA6 GLST | 24% |

Mixed-Strategy Equilibrium

| | |
|----------|-----|
| oD0 BNCH | 65% |
| oD1 CSIM | 0% |
| oD2 SCSA | 0% |
| oD3 STRC | 35% |
| oD4 BOOL | 0% |

Combined Equilibrium

| | | | | | |
|----------|-----|----|----|----|-----|
| oA0 DONT | 49% | 0% | 0% | 0% | 26% |
| oA1 REWR | 0% | 0% | 0% | 0% | 0% |
| oA2 GATE | 0% | 0% | 0% | 0% | 0% |
| oA3 ECTR | 0% | 0% | 0% | 0% | 0% |
| oA4 CCMP | 0% | 0% | 0% | 0% | 0% |
| oA5 STSQ | 0% | 0% | 0% | 0% | 0% |
| oA6 GLST | 16% | 0% | 0% | 0% | 8% |

Figure 13. GameRunner Solving the HDL Step Game in the Kickstarter Economy

Chapter 5. Results

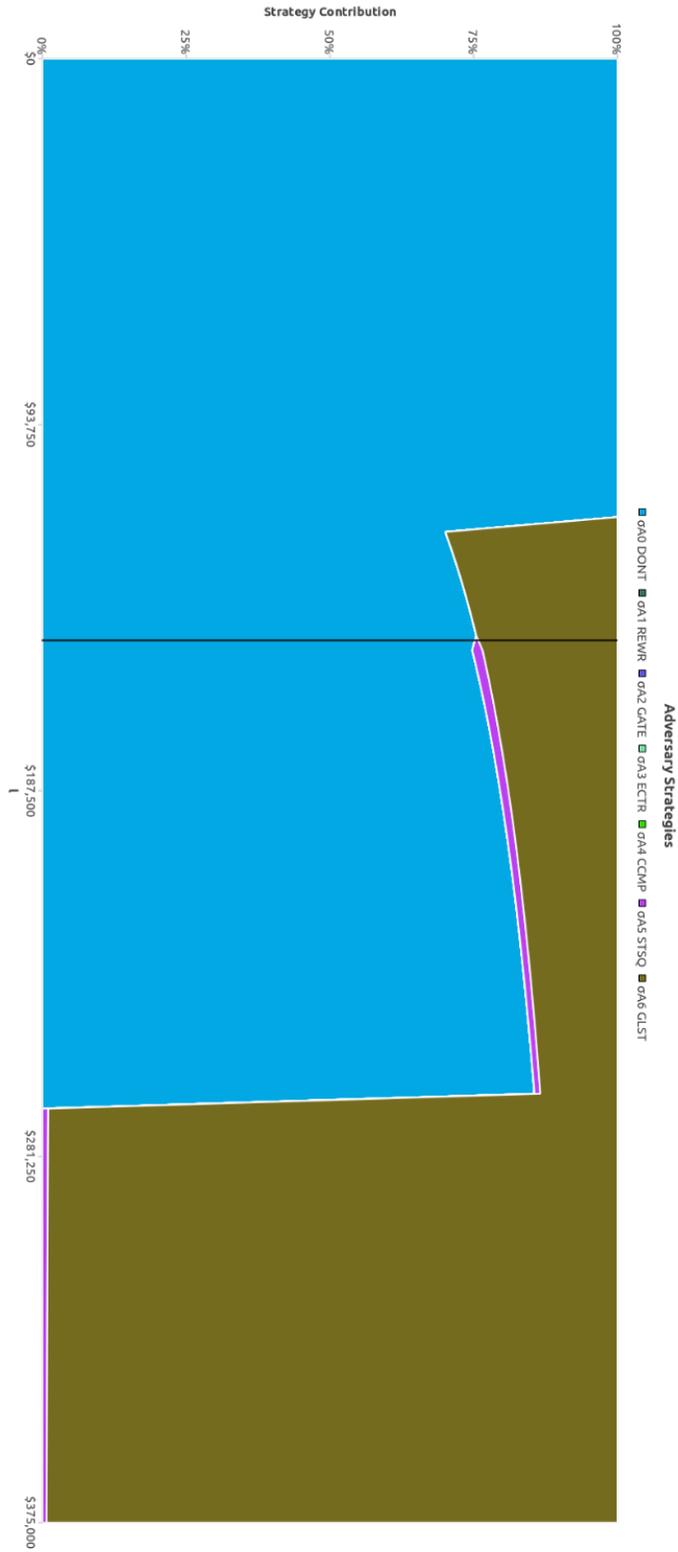


Figure 14. Adversary Strategies as L sweeps from \$0 to \$375,000 with $G = L/2$ and $Z_{find} = 2G$; all other variables fixed; HDL Step Game

Chapter 5. Results

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 can also be graphed, which is depicted in Figure 15. 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.

Chapter 5. Results

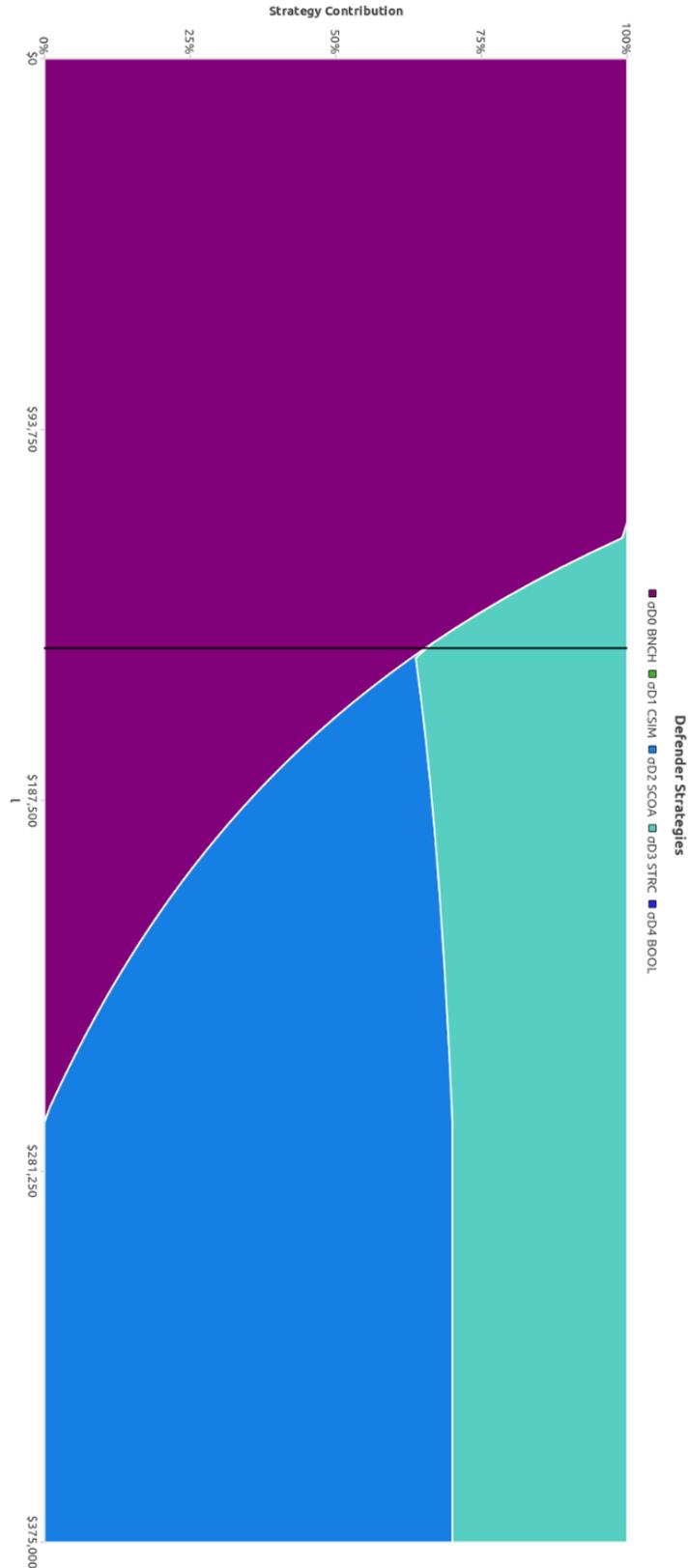


Figure 15. Defender Strategies as L sweeps from \$0 to \$375,000 with $G = L/2$ and $Z_{find} = 2G$; all other variables fixed; HDL Step Game

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 [154,155,156,157,158,159,160,161,162,163]. 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 Section 6.3.3. 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 overall goal of our analytic technique. As Figure 3 illustrated, 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 adding a GLST 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 $P_D(\text{GLST}, \text{BOOL})$ from 0% to 100%. Figure 16 depicts this sweep, zoomed in on the range of 50% to 100% to highlight where the game outcome changes. 1000 games were solved with $P_D(\text{GLST}, \text{BOOL})$ sweeping from 50% to 100% to produce the figure. At $P_D(\text{GLST}, \text{BOOL}) \approx 83\%$, 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. Figure 17 depicts the change in defender strategies over the same range of $P_D(\text{GLST}, \text{BOOL})$. Notably, at the same range where $P_D(\text{GLST}, \text{BOOL}) \approx 83\%$, the BOOL method is enough of a deterrent that it does not even have to be played by the defender every time. The less costly STRC and SCOA

Chapter 5. Results

methods can be gradually introduced to the mixed strategy solution set for the defender, although playing BOOL is always the primary solution of the mixed strategy set.

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 the BOOL. Based on the assumptions of our analysis, this would truly be the “game changing” HTH detection method.

Chapter 5. Results

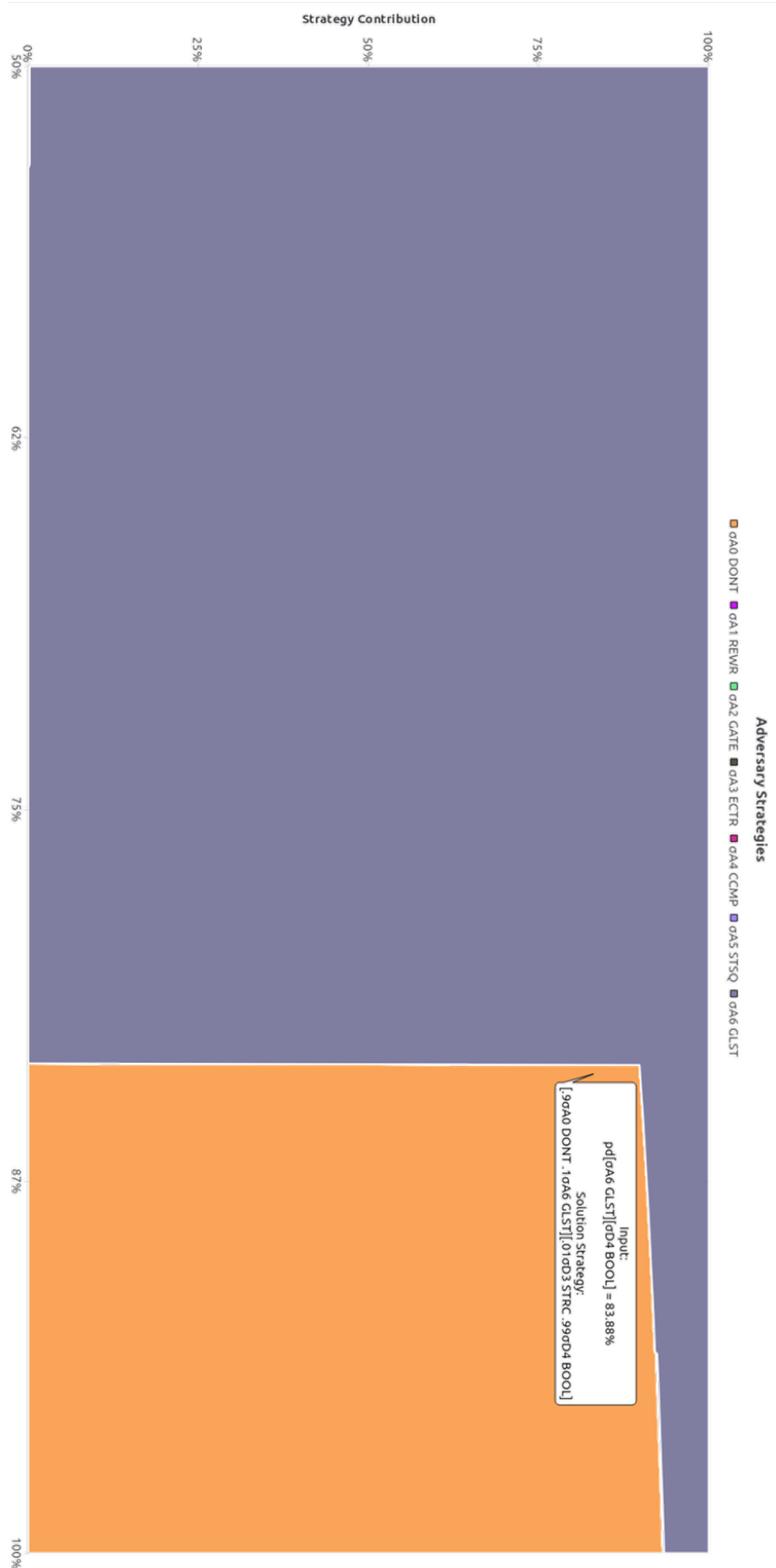


Figure 16. Adversary Strategies as $P_D(\text{GLST}, \text{BOOL})$ sweeps from 50% to 100%; all other variables fixed; HDL Step Game in the Consumer Economy

Chapter 5. Results

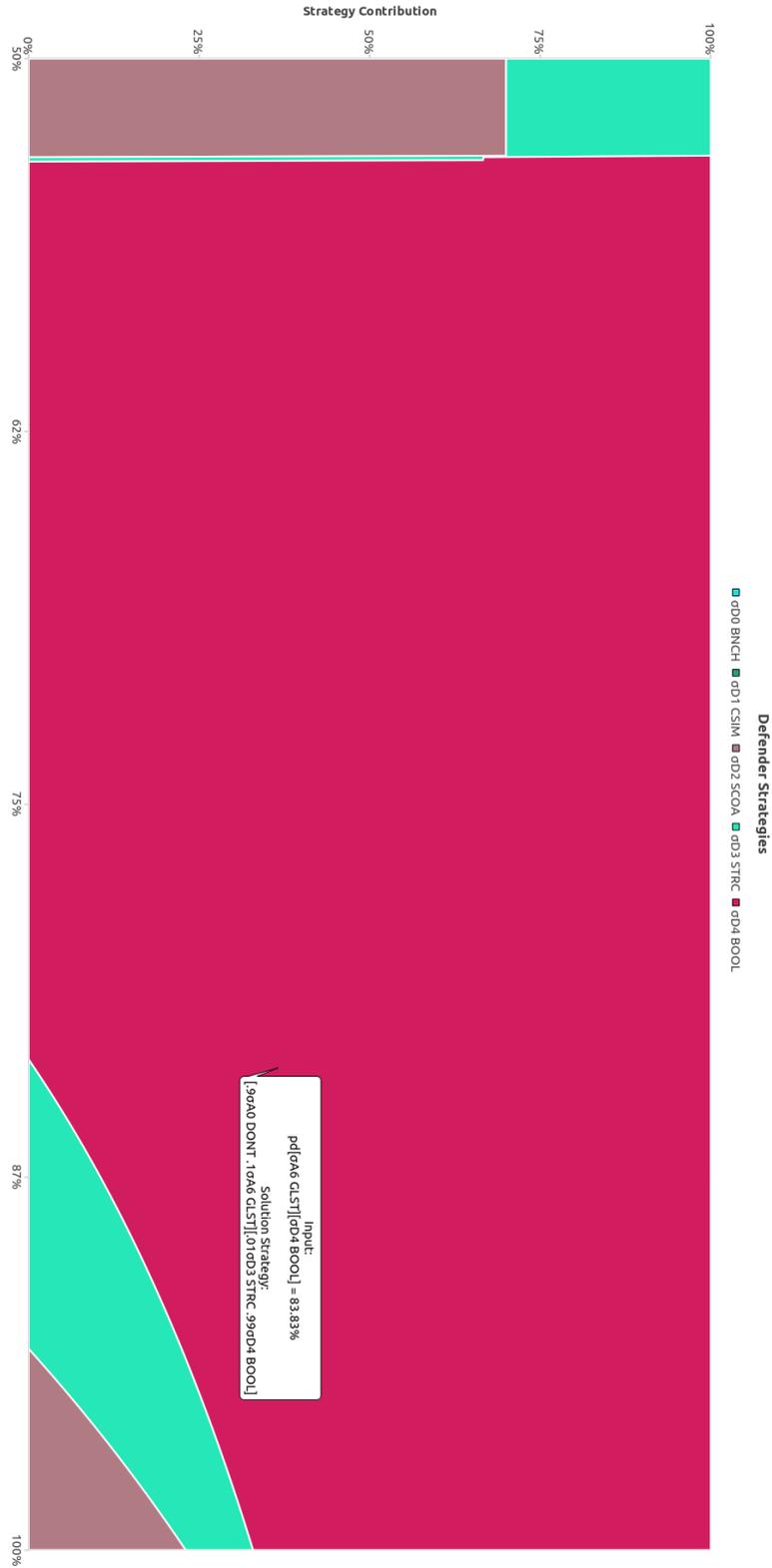


Figure 17. Defender Mixed Strategies as $P_D(\text{GLST}, \text{BOOL})$ sweeps from 50% to 100%; all other variables fixed; HDL Step Game in the Consumer Economy

Chapter 5. Results

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 imagine (but do not describe) 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 Figure 18. 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.”

One note on this analysis is that it is played without additional cost to the defender, since our sweep only considered the adversary cost variable. In this analysis, the cost of GLST gets higher while the defender continues to play detection-based HTH countermeasures set to their original costs. A more accurate way of modeling this situation would be to allow the defender to play prevention-oriented strategies in a game that considers their costs in comparison to the outcomes borne by both players. This new model is explored in Section 6.3.3.

Chapter 5. Results

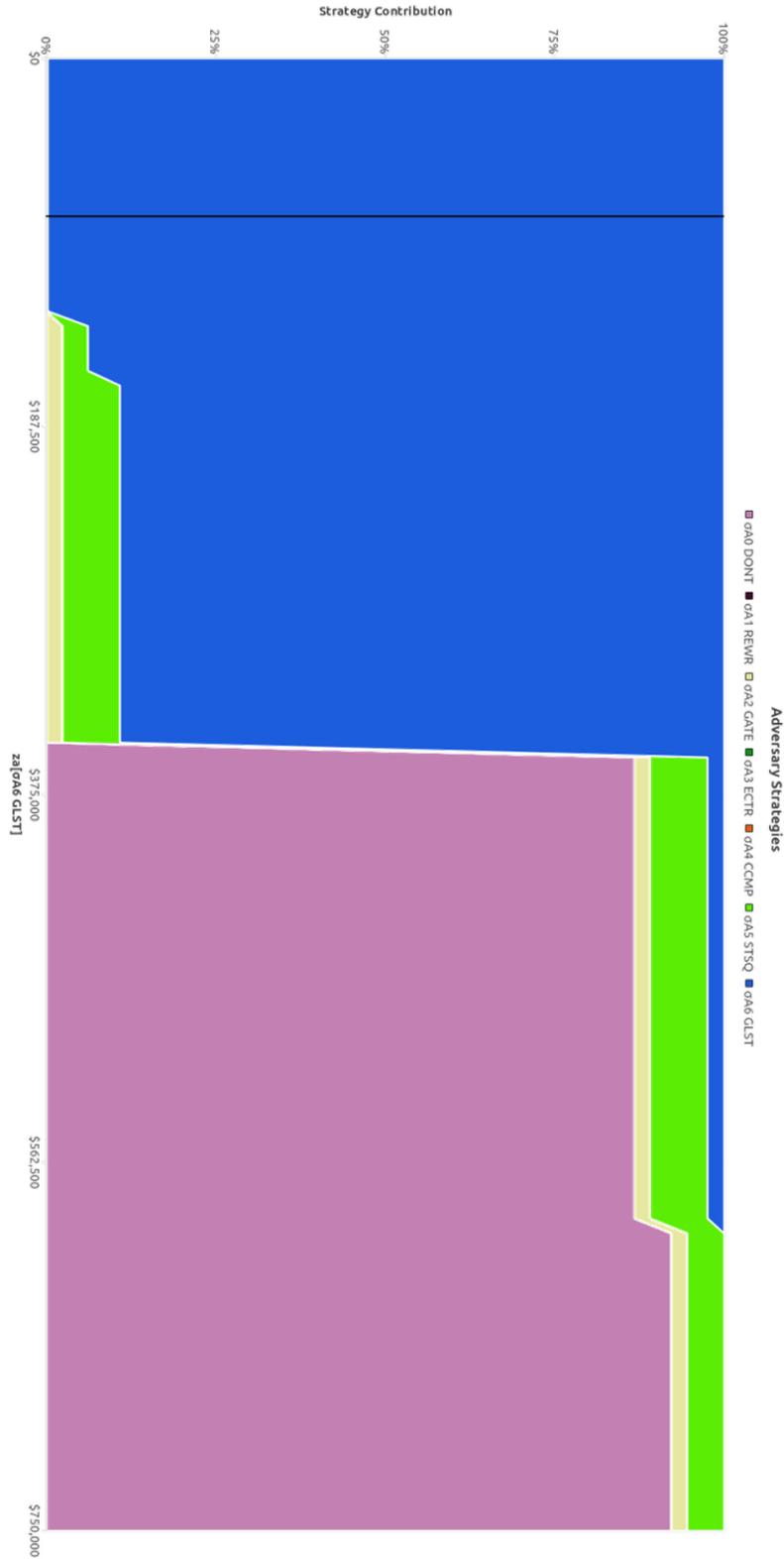


Figure 18. Adversary Mixed Strategies as $Z_A(\text{GLST})$ sweeps from \$0 to \$750,000; all other variables fixed; HDL Step Game in the Consumer Economy

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 this end, we also assembled the results of every combination of strategies beyond the defender’s “do nothing” strategy of BNCH. For now, the application of these strategies are not dependent on the order they are applied; in the future, we will consider how the order of operations may lead to refinement and reduction in false alarm rates. Table 26 in Appendix A on Page 153 has the raw results, which are summarized in Table 22 for P_D and Table 23 for P_{FA} , below. In this case, each strategy is a set of countermeasures, and the set is referred to and analyzed as a single defender strategy.

Table 22. P_D by Adversary and Defender Strategy with Defender Strategies Employed in Countermeasure Sets

| | BNCH | CSIM | SCOA | STRC | BOOL | CSIM SCOA | CSIM STRC | SCOA STRC | CSIM SCOA STRC | CSIM BOOL | SCOA BOOL | CSIM STRC SCOA | STRC BOOL | CSIM STRC SCOA STRC | SCOA STRC SCOA | CSIM STRC SCOA STRC SCOA |
|--------------|------------|------------|------------|------------|------------|--------------|--------------|--------------|----------------------|--------------|--------------|----------------------|--------------|------------------------------|----------------------|--------------------------------------|
| DONT | 0% | 0% | 0% | 0% | 0% | 0% | 0% | 0% | 0% | 0% | 0% | 0% | 0% | 0% | 0% | 0% |
| REWR | 88% | 100% | 100% | 88% | 100% | 100% | 100% | 100% | 100% | 100% | 100% | 100% | 100% | 100% | 100% | 100% |
| GATE | 50% | 50% | 50% | 100% | 100% | 50% | 100% | 100% | 100% | 100% | 100% | 100% | 100% | 100% | 100% | 100% |
| ECTR | 53% | 53% | 73% | 80% | 100% | 73% | 80% | 93% | 93% | 100% | 100% | 100% | 100% | 100% | 100% | 100% |
| CCMP | 31% | 46% | 62% | 46% | 100% | 69% | 54% | 62% | 69% | 100% | 100% | 100% | 100% | 100% | 100% | 100% |
| STSQ | 13% | 13% | 60% | 27% | 100% | 60% | 27% | 60% | 60% | 100% | 100% | 100% | 100% | 100% | 100% | 100% |
| GLST | 0% | 0% | 50% | 50% | 0% | 50% | 50% | 50% | 50% | 0% | 50% | 50% | 50% | 50% | 50% | 50% |
| Total | 40% | 46% | 68% | 60% | 96% | 70% | 63% | 77% | 79% | 96% | 98% | 98% | 98% | 98% | 98% | 98% |

Table 23. P_{FA} by Defender Strategy with Defender Strategies Employed in Countermeasure Sets

| | BNCH | CSIM | SCOA | STRC | BOOL | CSIM SCOA | CSIM STRC | SCOA STRC | CSIM SCOA STRC | CSIM BOOL | SCOA BOOL | CSIM STRC SCOA | STRC BOOL | CSIM STRC SCOA STRC | SCOA STRC SCOA | CSIM STRC SCOA STRC SCOA |
|--|------|------|------|------|------|--------------|--------------|--------------|----------------------|--------------|--------------|----------------------|--------------|------------------------------|----------------------|--------------------------------------|
| | 0% | 0% | 50% | 17% | 0% | 50% | 17% | 50% | 50% | 0% | 50% | 50% | 17% | 17% | 50% | 50% |

Chapter 5. Results

The result is that we now have a large set of strategies (any of them that contain both BOOL and either STRC or SCOA) that improve total detection to 98%. False alarm rates of the countermeasure sets depict the same variation between 0% and 50%. For space, we only analyze one step game in one economy. We selected the HDL step game in the Consumer economy; the two-player strategic game in normal form that results from the utility functions for this expanded game is depicted in Table 24, below. Note that while this is a much larger game, it is still only representative. In future analysis, we anticipate analyzing every available defender strategy and every combination thereof at every step game. These games will be extremely large. As we get larger, the use of GameRunner as an automated tool becomes ever more necessary, since humans cannot hand-solve games of such large dimension.

The solution to the game for the adversary 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 may seem surprising that despite adding countermeasure sets that proved to have an increase in P_D . Some of them include 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.⁴²

⁴² 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 12 and Table 13. 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.

Table 24. Two-Player Strategic Game: HDL Step, Consumer Economy, Defender Strategies Employed in Countermeasure Sets

| | | $U_D(\sigma_A, \sigma_D)$ (\$ Thousands) | | | | | | | | | | | | | | | |
|--|----------|--|-------------|-------------|------------|--------------|-------------|-------------|-------------|-------------|--------------|--------------|--------------|--------------|--------------|--------------|--------------|
| | | CSIM | | | | SCOA | | | | CSIM | | | | SCOA | | | |
| | | BNCH | CSIM | SCOA | STRC | BOOL | STRC | SCOA | STRC | BOOL | STRC | SCOA | STRC | BOOL | STRC | SCOA | STRC |
| | | | | | | | | | | | | | | | | | |
| $U_A(\sigma_A, \sigma_D)$ (\$ Thousands) | DONT | (-20, 0) | (-20, -3) | (-20, 19) | (-20, -18) | (-20, -69) | (-20, 22) | (-20, -21) | (-20, -37) | (-20, -39) | (-20, -72) | (-20, -88) | (-20, 91) | (-20, -87) | (-20, 90) | (-20, -106) | (-20, 108) |
| | REWR | (-52, 0) | (-155, 184) | (-155, 167) | (-52, -18) | (-155, 117) | (-155, 165) | (-155, 165) | (-155, 150) | (-155, 147) | (-155, 115) | (-155, 98) | (-155, 96) | (-155, 99) | (-155, 96) | (-155, 81) | (-155, 78) |
| | GATE | (255, 0) | (255, 3) | (-255, -19) | (255, 727) | (-155, 676) | (255, -22) | (-155, 724) | (-155, 708) | (-155, 706) | (-155, 673) | (-155, 657) | (-155, 654) | (-155, 658) | (-155, 655) | (-155, 639) | (-155, 637) |
| | ECTR | (228, 0) | (228, -3) | (64, 279) | (9, 379) | (-155, 626) | (64, 276) | (9, 377) | (-100, 559) | (-100, 557) | (-155, 624) | (-155, 607) | (-155, 605) | (-155, 608) | (-155, 606) | (-155, 590) | (-155, 587) |
| | CCMP | (412, 0) | (286, 227) | (160, 439) | (286, 211) | (-155, 963) | (97, 551) | (223, 323) | (160, 422) | (97, 534) | (-155, 960) | (-155, 943) | (-155, 941) | (-155, 944) | (-155, 942) | (-155, 926) | (-155, 923) |
| | STSQ | (555, 0) | (555, -3) | (173, 676) | (446, 181) | (-155, 1222) | (173, 674) | (446, 178) | (173, 659) | (173, 656) | (-155, 1220) | (-155, 1203) | (-155, 1201) | (-155, 1204) | (-155, 1202) | (-155, 1186) | (-155, 1183) |
| GLST | (665, 0) | (665, -3) | (255, 726) | (255, 727) | (665, -69) | (255, 723) | (255, 724) | (255, 708) | (255, 706) | (665, -72) | (255, 657) | (255, 654) | (255, 658) | (255, 655) | (255, 639) | (255, 637) | |

This new game with countermeasure sets offers us a new basis for analysis with some of GameRunner’s advanced features. Figure 19 shows the GameRunner GUI playing the expanded game in the HDL step in the consumer economy. Using this game as the basis, we again use GameRunner’s sweeping tool, but we ask a new question. Keeping all other market variables constant, we are interested in the question of *how much punishment* must be applied to an adversary in the event that they are caught inserting a Trojan. To answer this question, we swept the value of Z_{find} . Figure 20 shows how the adversary strategy changes across 1,000 game solutions that incrementally change Z_{find} from \$0 to \$20,000,000. At the extreme left end – with \$0 of Z_{find} – the adversary is sure to attack. Due to the extremely low P_{ATT} value (5%), the punishment of the adversary must exceed \$12,500,000 to make it approach a vanishingly small chance that it is rational to attack.

Chapter 5. Results

The screenshot displays the GameRunner interface with the following sections:

- Strategies:** A list of strategies for both Adversary and Defender, such as 'gA0 DONT: Do nothing' and 'gD0 BNCH: Cosimulation with testbench'.
- Inputs:** Scenario (Consumer), Random Fixed Costs, and Adversary Data (ZA, PS, ZD).
- Probability of Attribution (PAT):** A table showing probabilities for various strategies, with values ranging from 0.00% to 100.00%.
- False Alarm Probabilities (FPA):** A table showing false alarm rates for different strategies, with values ranging from 0.00% to 16.67%.
- Utility/Matrices:** A table showing utility values for Adversary and Defender across different strategy combinations.
- Solutions:** A section for solver algorithms (Flinnash-extension, Error 0.00%) and Pure-Strategy Equilibrium (gA0 DONT: 0%, gA1 REWR: 0%, gA2 GATE: 0%, gA3 ECTR: 0%, gA4 RCMP: 0%, gA5 STSQ: 0%, gA6 GLST: 0%).
- Mixed-Strategy Equilibrium:** A table showing mixed strategy results, with gD1 CSIM at 70% and gD2 SCOA at 30%.
- Combined Equilibrium:** A table showing combined equilibrium results, with gD1 CSIM at 70% and gD2 SCOA at 30%.

Figure 19. GameRunner Solving the HDL Step Game, Consumer Economy, Defender Strategies Employed in Countermeasure Sets

Chapter 5. Results

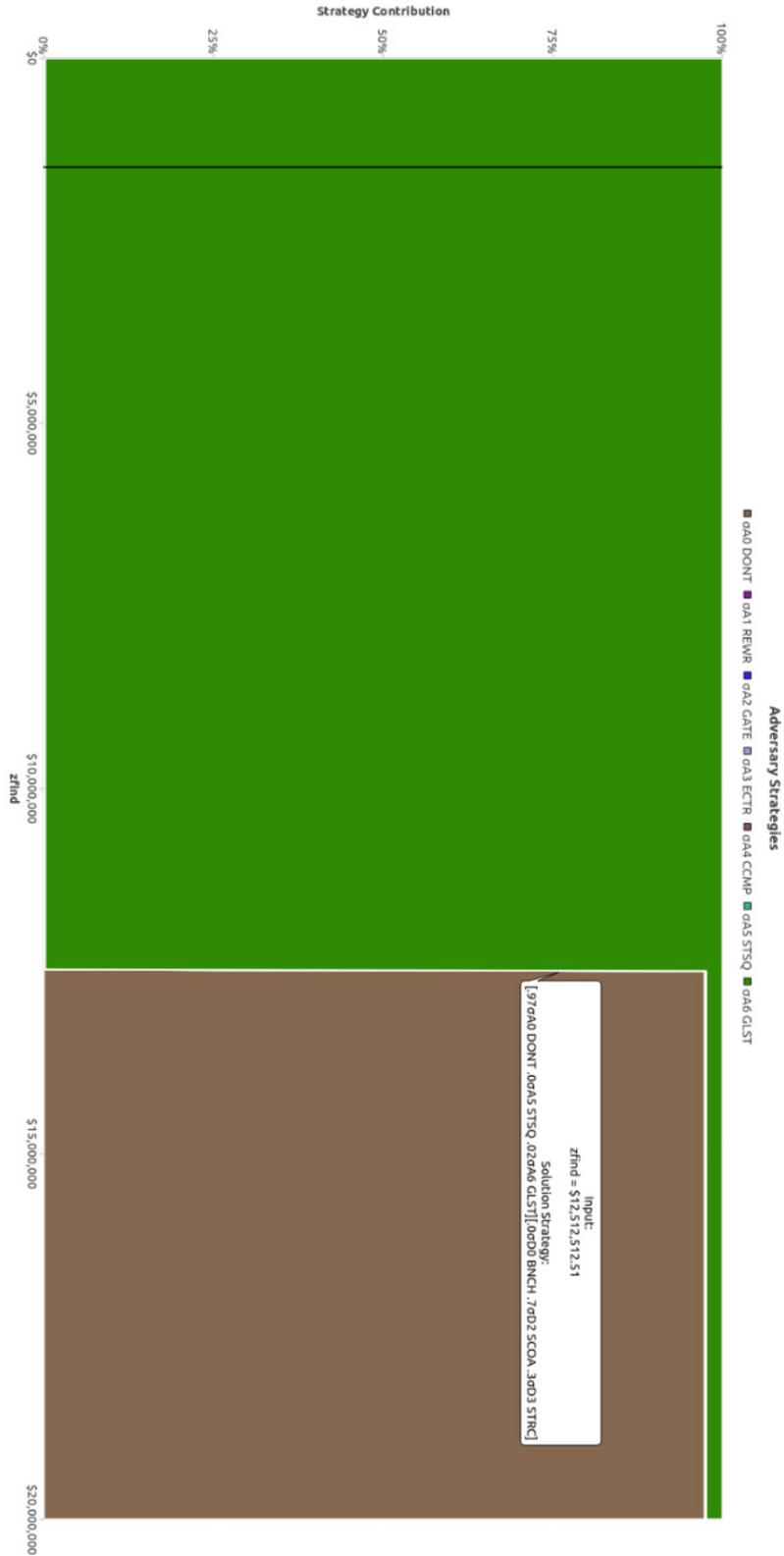


Figure 20. Adversary Strategies as Z_{find} sweeps from \$0 to \$20,000,000; all other variables fixed; HDL Step Game with Defender Strategies Employed in Countermeasure Sets

Chapter 5. Results

Note that this punishment is many multiples of the market value for the item they are attacking. This provides an excellent illustration of just how likely an adversary might be to attack when the consequences of the attack are unlikely to reach them. A conclusion that could be drawn here might be that law enforcement must set the penalties for cyber attacks using HTHs to be extremely severe in order to prevent parties from attempting them. Alternatively, we can interpret this a major motivation to improve attribution in HTH scenarios. Otherwise, the adversary impulse to try to insert an HTH is too often the most rational choice. The defender mixed strategy solutions across the same sweep are shown in Figure 21 below. As the punishment of the adversary continues to be advantageous for them, the 70% SCOA / 30% STRC mixed strategy solution holds. The default strategy, BNCH, gains a share of the mixed strategy solution as the size of the punishment increases, since the adversary becomes less likely to attack at all.

Finally, we can re-ask the question of how expensive the adversary's most-favored method – GLST – would have to be to make them play something else in this game that includes countermeasure sets. Figure 22 shows the adversary mixed strategy equilibria as we sweep of $Z_A(\text{GLST})$ from \$0 to \$750,000 in the Consumer economy in the HDL step game with the defender playing all available sets of countermeasures. Figure 23 shows the defender strategies over the same sweep. We can see that as we approach \$375,000, the increasing GLST cost causes the game to flip to where the adversary's mixed strategy equilibrium becomes dominated by the DONT strategy, with a small likelihood of playing various attack strategies. This is very similar to the analysis for the game that considered a defender that could only play one countermeasure at a time. As before, this analysis should improve in fidelity with the new game described in section 6.3.3.

Chapter 5. Results

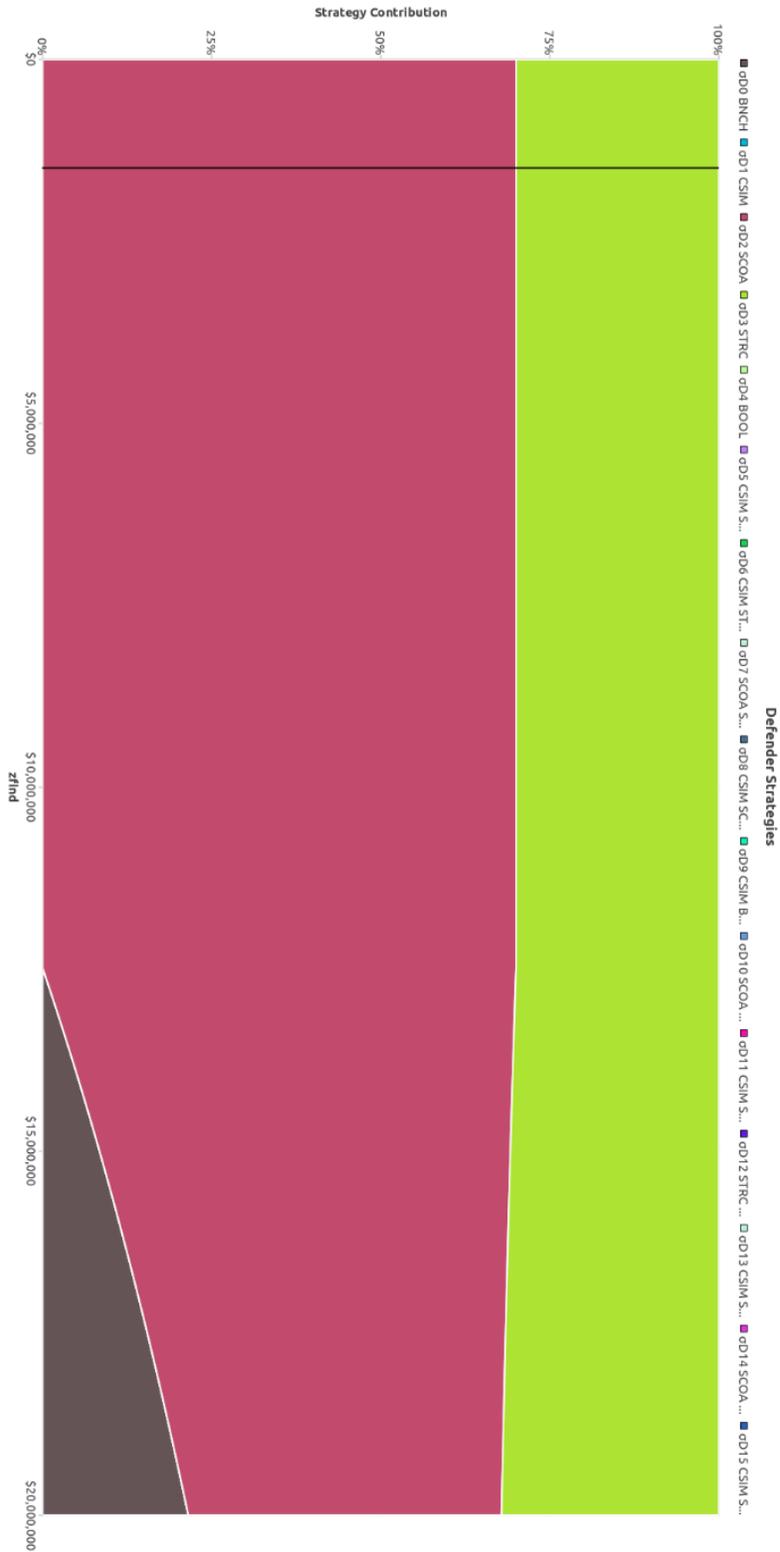


Figure 21. Defender Strategies as Z_{find} sweeps from \$0 to \$20,000,000; all other variables fixed; HDL Step Game with Defender Strategies Employed in Countermeasure Sets

Chapter 5. Results

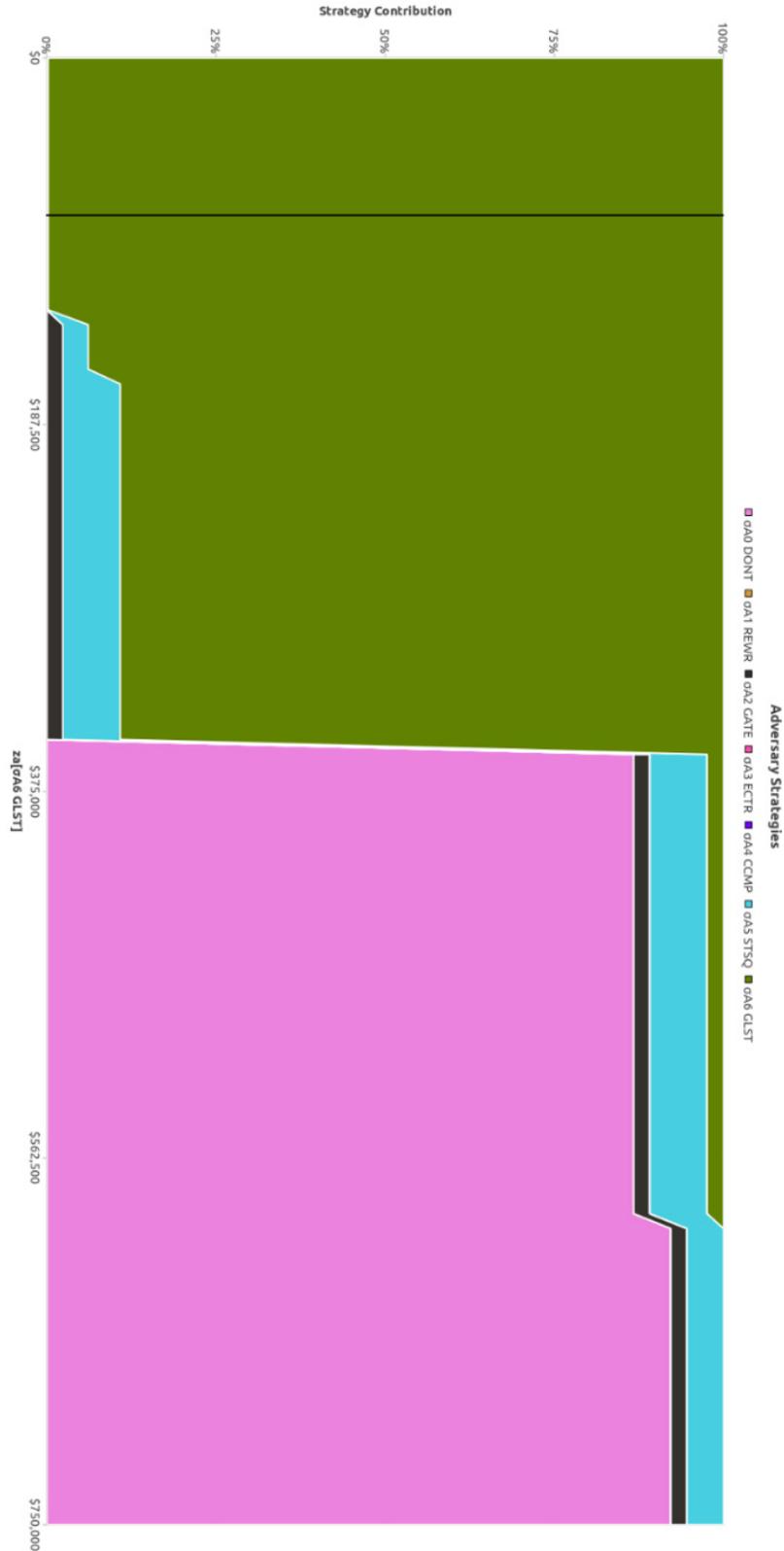


Figure 22. Adversary Strategies as $Z_A(GLST)$ sweeps from \$0 to \$750,000; all other variables fixed; HDL Step Game under the Consumer Economy with Defender Strategies Employed in Countermeasure Sets

Chapter 5. Results

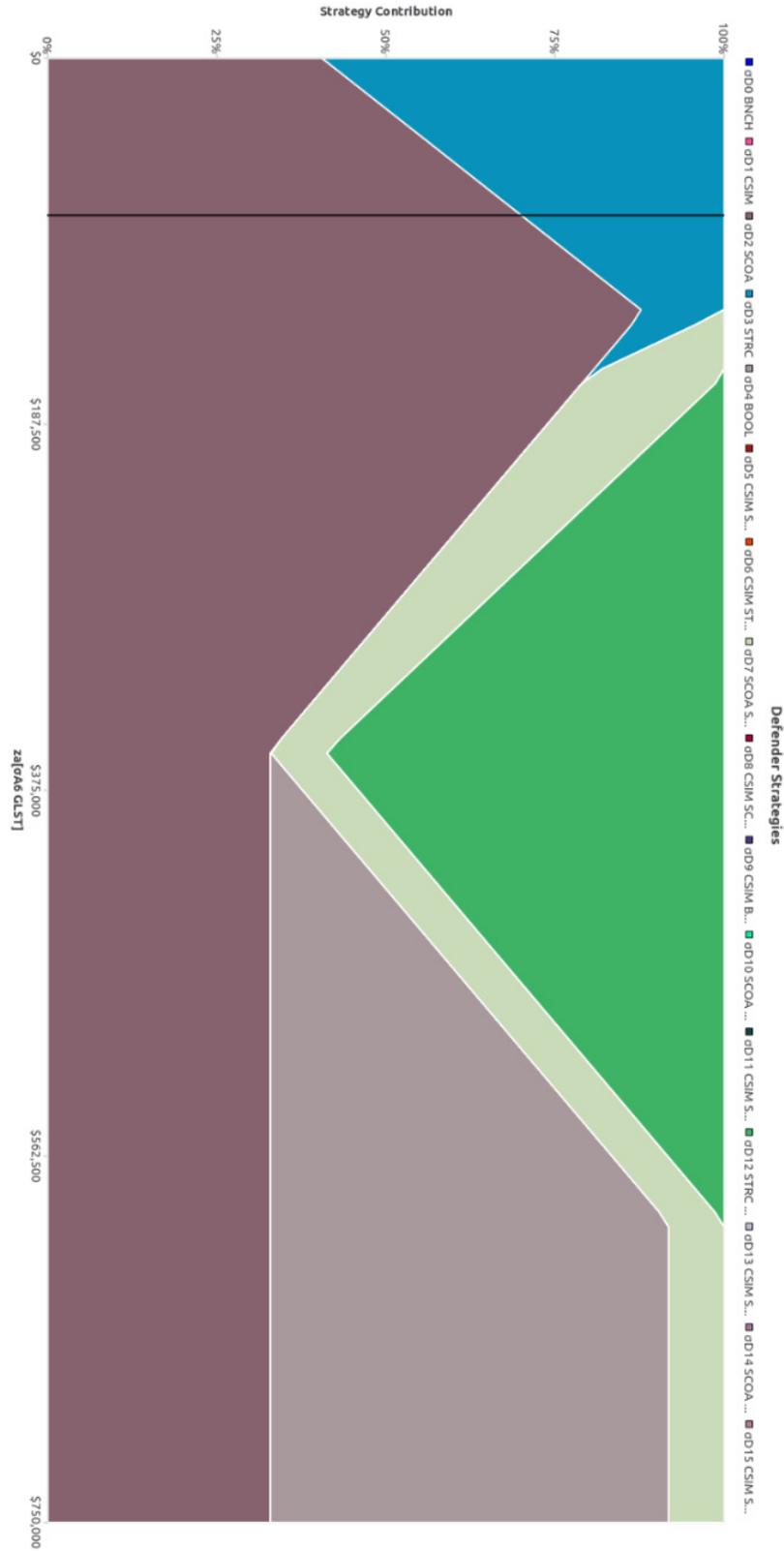


Figure 23. Defender Strategies as $Z_A(\text{GLST})$ sweeps from \$0 to \$750,000; all other variables fixed; HDL Step Game with Defender Strategies Employed in Countermeasure Sets

5.4 Discussion: How do we know this works?

The novel nature of our approach naturally leads to the question: does this work? The simple answer is that if (1) we get our utility functions right and (2) the Nash equilibrium solution concept correctly describes optimal decision making, then this approach provides value. However, more discussion is warranted, since it will expose the limitations, dependencies, and future variants of our approach. We break the question of “Does this work?” into three sub-questions:

1. Do the expected utility functions correctly represent the considerations (variables) within each player’s beliefs and the relationships between each player?
2. Are the values of the variables in the expected utility functions correctly determined?
3. Does the Nash equilibrium solution concept provide the optimal answer?

Question 1 is one of the correctness of the construction of our utility functions. That is, if we are able to algebraically define the beliefs of each player, we can use that model to predict their behaviors. We indicate by use of the term beliefs that we do not care as much about the actual scenario as we do about what each player thinks the actual scenario is. The players take action based on what they believe; the relationship between that belief and reality is a separate discussion. With this said, our primary aim in this work is to model players with *correct* beliefs about the engagement, while future work will explore players whose beliefs diverge from fully rational decision making.

Whether our utility functions represent rational beliefs for each player hinges on the narrative of sections 2 and 3. We believe we have represented the defender correctly: they are concerned with detection method efficacy and false alarm rates, the costs of deploying each, the costs of false alarms, and the cost of losing an engagement to a successful HTH adversary. If more considerations are involved, we believe this expected utility function would be modified, but not radically. We also believe that the adversary’s expected utility will involve concern for what they stand to gain, the effectiveness of their planned attack strategy, and the consequences and probabilities associated with getting caught. As with the defender, we believe this at minimum models the player’s major considerations when making decisions. As will be seen in the next chapter, establishing the equations in the

Chapter 5. Results

described arrangement provides a foundation for future exploration of subrational players. In addition, we inform these variables' relationships to each other in our utility functions by leveraging and adapting the standardized approach of security investment described by Gordon and Loeb [101]. Thus, our confidence in an affirmative answer to Question 1 is high.

The second question builds on the first. That is, if we have modeled the components of each player's decision-making processes correctly, have we also determined the values of those variables correctly? Our determination of the probability values related to detection and false alarms is well defended for the experiment we ran because of the demonstrated methodology for empirically determining the values. Again, we do not claim that the experimentation performed thus far is sufficient for industry-level conclusions, but we have demonstrated the methodology to generate detection and false-alarm values empirically. This methodology can produce the data necessary for industry-level conclusions. The probability of success if inserted and probability of attribution values are set in our model only notionally. Each require further study of actual HTH adversary/defender engagements and their consequences. Here we run into the "lack of data" problem pervasive in the study of HTH engagements, and we must turn to arguments from other fields such as leveraging success and attribution information from software exploits. Because of the direct analogy to the software domain, we believe our arguments are sound, but we do lack empirical data directly from the HTH domain.

The economic variables within our model are the least well defined empirically. For this work, we set many of them with a basis in reality by reasoning against labor and material costs and times required to perform tasks. We set others – such as the adversary gain and defender loss variables – by arguing with respect to market values gained and lost. While we have demonstrated the steps necessary to set these economic considerations empirically, they are large values that may be subject to errors. For example, if the adversary and defender were to mis-predict the future market values of the products in our game – an area fraught with error in any market – that could change their behavior. If they predicted those values differently, they each may be acting with a different set of beliefs, which, again, is the subject of our future work on subrational models. In the end, for this question we have made large strides towards building games within a framework of

empirical practice – especially when compared to other approaches that avoid the question entirely – but more work needs to be done here in the future. So, our answer to the second question is “probably” but we have at least defined the road ahead to a stronger answer in the affirmative.

Finally, the third question – *Does the Nash equilibrium solution concept provide the optimal answer?* – is based either in unfamiliarity with or skepticism of the Nash equilibrium solution concept. First we must answer what *optimal* means in this context. The utility functions model the values we are optimizing for each player: higher numbers are better outcomes. Optimal play is when each player maximizes the resolution of their utility function (their payoff) given the likely play of their opponent. In particular, in our case, we want to know the optimal defender play given the likely play of the adversary. Since the adversary is rational and, thus, also attempting to optimize their outcome, the Nash equilibrium solves for the set of adversary/defender strategies that permits neither party to accomplish a better outcome by unilaterally changing their strategy. With optimality defined, we can illustrate what it looks like to select the Nash equilibrium as the outcome selected by our methodology. The GameRunner tool again offers a unique set of visualizations.

To create a picture of the problem space from which the Nash equilibrium is selecting the optimal play, we can draw a three-dimensional grid on which all possible mixed strategies are represented. The figure that results from this exercise is shown below in Figure 24 using HDL step game in the Consumer economy. Along the X-axis are all possible defender mixed strategies. This includes all defender strategies played from 0% to 100% of the time in sets that must all add up to 100%. For example, one set is playing BNCH 100% of the time and all others 0%; one is playing BNCH 10% of the time, STRC 40% of the time, and CSIM 50% of the time; one is playing BNCH, CSIM, STRC, SCOA, and BOOL 20% of the time each. The possible set of mixed strategies includes all such combinations that add up to 100%. The Y-Axis of this graph is all such adversary mixed strategies: all mixed strategy sets that add up to 100% with every possible individual strategy contribution represented. Finally, the Z-axis represents the resolution (“payoff”) of the utility functions.

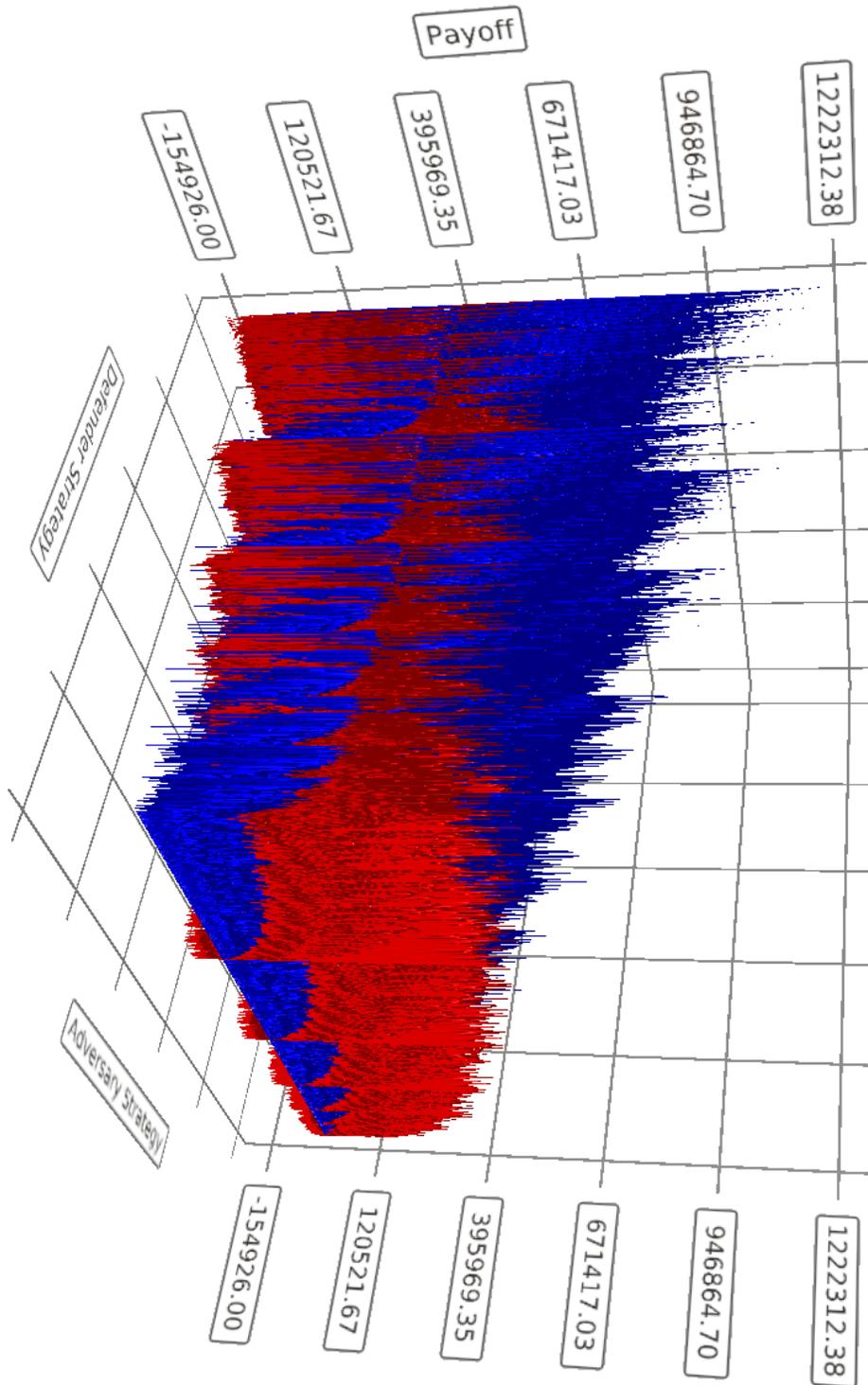


Figure 24. GameRunner-Created 3D Graph of Adversary (Red) and Defender (Blue) Utility Function Payoffs under All Possible Mixed Strategies; HDL Game in the Consumer Economy

Chapter 5. Results

Note that in GameRunner, we must select the resolution at which we want to consider the possible mixed strategies. In the figure, we picked a resolution that considered every possible strategy in contribution increments of 10%. That is, the X- and Y-axes represent the mixed strategies of the defender and adversary, respectively, with every possible combination of each strategy contributing 0%, 10%, 20%, ..., 100%.⁴³ The figure contains two intersecting planes. In blue, we see the plane of possible defender payoffs; in red, the plane of possible adversary payoffs. Neither are smooth planes, since the transitions between strategies can effect radical changes in the utility. This lets us visualize the “haystack of needles” from which we must select the needle that represents the mixed strategy set from which neither the adversary nor the defender can do better by unilaterally changing their strategy.

Recall that the Nash equilibrium solution concept prescribes that the defender’s optimal play is to play SCOA 70% of the time and STRC 30% of the time, while the adversary’s optimal strategy is to play GLST almost 100% of the time while playing STSQ nearly 0% of the time (but not quite). We can graphically explore the qualities of this equilibrium point by using another utility we built into GameRunner. Sticking with the same game, Figure 25 depicts an exploration of possible defender mixed strategies in the face of an adversary who selects the Nash-equilibrium mixed strategy. The top portion of the figure plots the defender mixed strategy sets at the same resolution as in Figure 24. Each strategy is assigned a unique color, and in the resulting plot, the 10% strategy stepping is readily visible. The bottom portion is aligned with the top and represents the resulting defender payoff for every new mixed strategy set for the defender when played against the fixed optimal mixed strategy set for the adversary. The blue line at the top of this graph represents the Nash-equilibrium mixed strategy payoff. Note that while several defender mixed strategy options come close, none does better than the Nash-equilibrium mixed strategy. Thus, if the adversary plays their Nash-equilibrium mixed strategy, the defender

⁴³ That this is an artifact only of the way we must draw the graphs; the Nash equilibrium considers the utility functions as continuous functions, not discrete points.

Chapter 5. Results

cannot do better by unilaterally changing their strategy from their Nash-equilibrium mixed strategy.

Next, we need to explore the adversary's options. Figure 26 represents the same game, but we have fixed the defender strategy at the Nash-equilibrium mixed strategy set, and we are exploring all possible mixed strategy adversary sets. The mixes of adversary strategies are plotted above, aligned with the graph below that shows the adversary payoffs. Again, the Nash-equilibrium adversary payoff is shown as the horizontal line, below which all other mixed strategy payoffs are graphed. No other mixed strategy matches or exceeds the payoff accomplished by the Nash-equilibrium payoff. Thus, the adversary joins the defender: we can clearly see visually that they cannot do better by unilaterally changing their strategy when their opponent selects the Nash-equilibrium payoff.

The above figures are only a visualization of the recommendation provided by our method to justify why we would declare it both a Nash equilibrium and an optimal strategy. Comprehensive proof that the Nash equilibrium "works" is beyond the scope of our work. However, the last of our three questions that compose the broader question of "Does this work?" has a justified affirmative answer. As a final note, there is also a practical answer to the question. In the end, this method works if the defender is able to use this method in the wild to accomplish lower overall resource expenditure in HTH prevention, detection, and recovery than they would following an expert-driven checklist that was not informed by this method. This idea is explored further in Chapter 6.

Chapter 5. Results

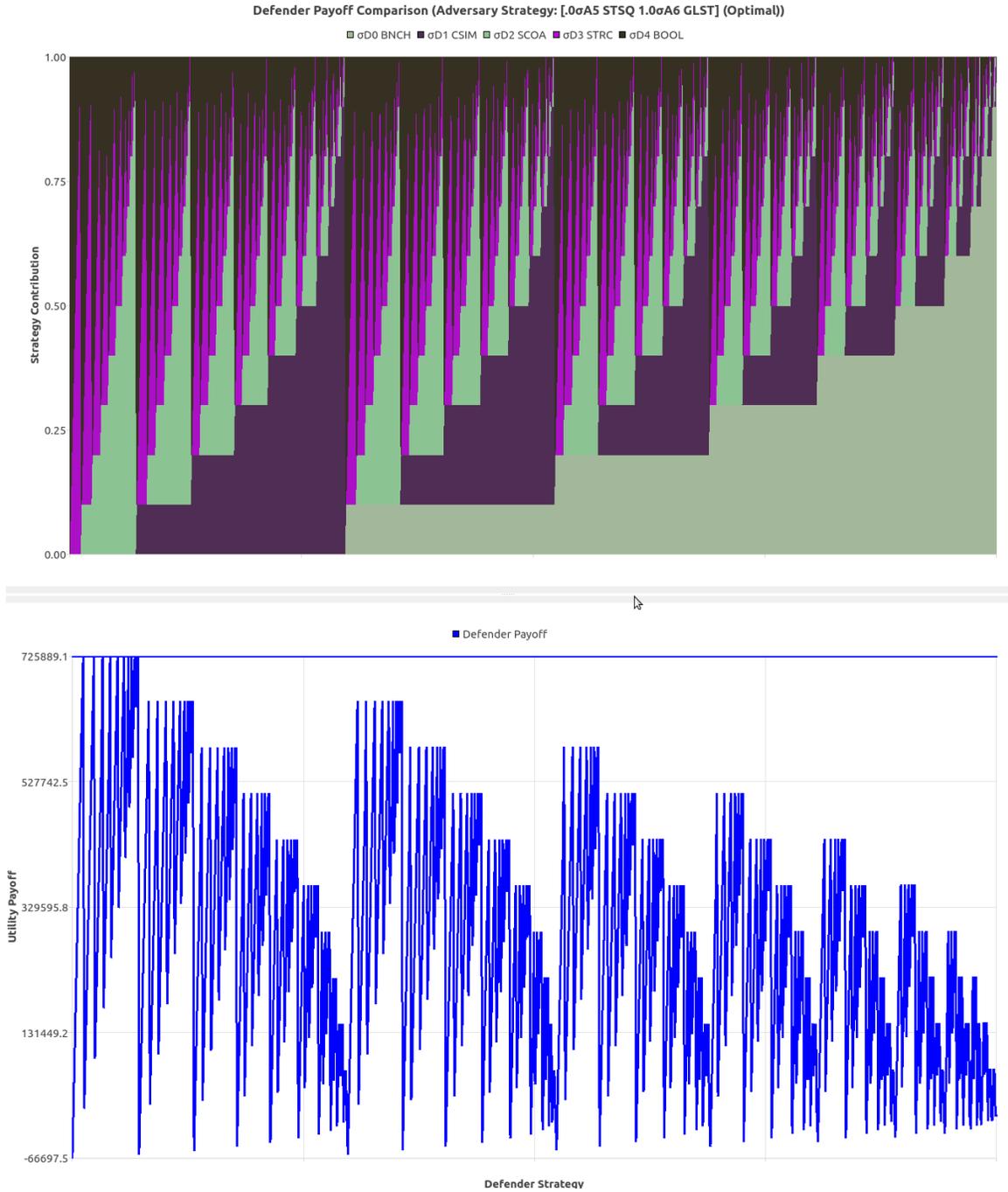


Figure 25. GameRunner-Created 2D Graph of Defender Mixed Strategy Sets (Top) and the Resulting Defender Payoffs (Below) When the Adversary Plays the Nash Equilibrium-Selected Optimal Strategy; HDL Game in the Consumer Economy

Chapter 5. Results

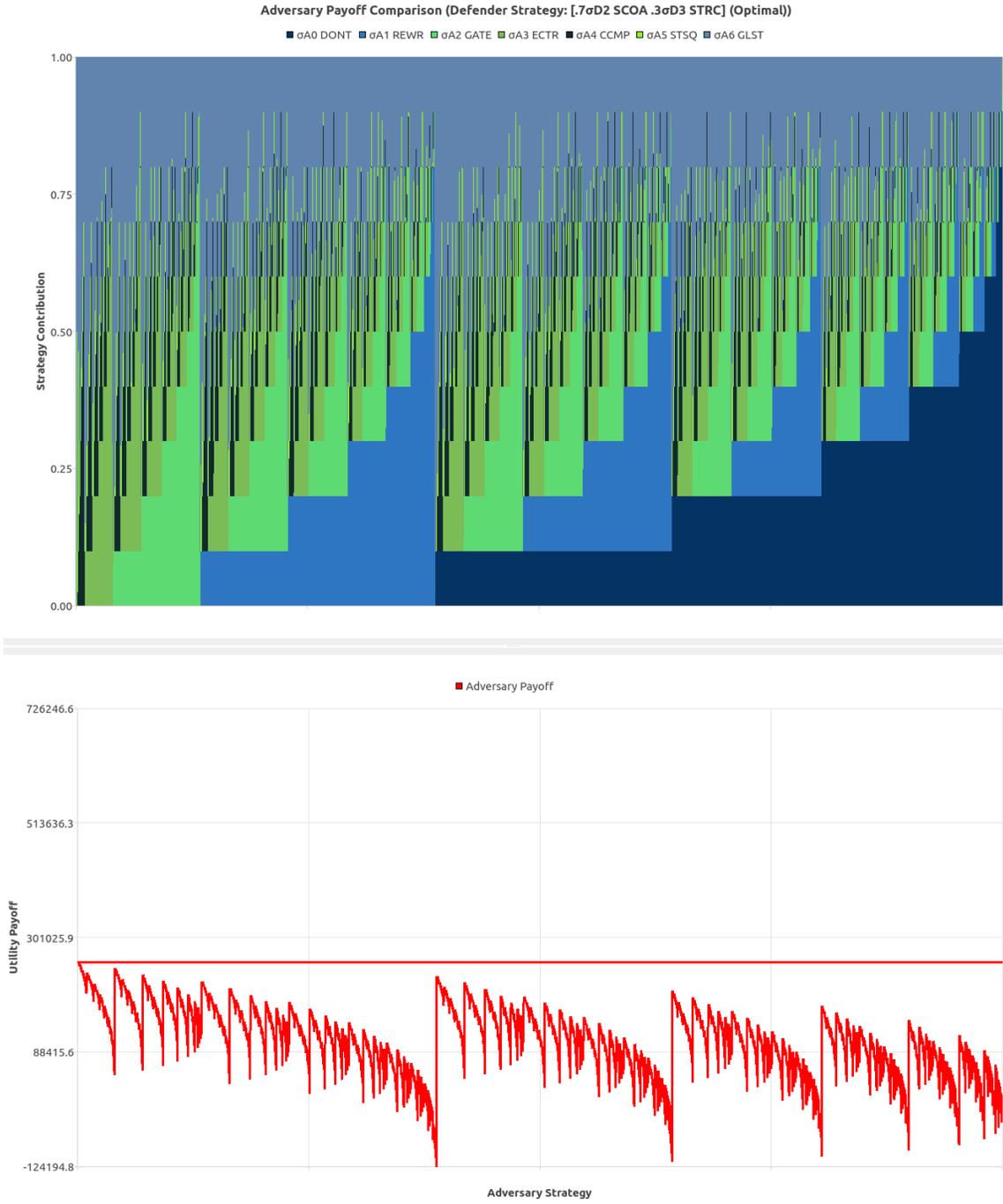


Figure 26. GameRunner-Created 2D Graph of Defender Mixed Strategy Sets (Top) and the Resulting Defender Payoffs (Below) When the Adversary Plays the Nash Equilibrium-Selected Optimal Strategy; HDL Game in the Consumer Economy

Chapter 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 dissertation 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 only two steps for the design lifecycle. To accomplish industry-level recommendations for HTH detection, we continue this research in a variety of new directions, which are summarized in this chapter. Furthermore, the security economic utility functions in this dissertation 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.

This chapter summarizes our emerging directions first by continuing the narrative directly from the previous two chapters. We first discuss how we are improving the framework presented in those two chapters in order 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. Next, we discuss how 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. Finally, we explore the applications beyond HTH detection in FPGAs that can be addressed by the methodologies presented in this work.

6.1 Advancing the Framework

The eventual end of this work is to guide industry in making decisions about optimal hardware Trojans countermeasures. To approach the ability to make such grand recommendations, we must be able to produce a basis of well-structured experiments that mirror industrial concerns and allow us to claim statistical relevance across very large sets of test articles. This task is not small. To give a brief summary of both planned and in-progress work to accomplish this end, we briefly describe our approach below.

6.1.1 Hardware Trojan Test Article Database

Continuing our work from [8], we are building a comprehensive set of HTH test circuits. They are to be held in a database of test articles defined by, at minimum, the following properties:

- HTH trigger
- HTH payload
- HTH design step insertion point
- Target circuit size
- Target circuit design style
- Target circuit insertion location

Our vision is to create a test article set that has statistically significant representation in each of these dimensions. This is an immense task. It will result in a database many orders of magnitude larger than the one tested in this work, and it will require continuous updating to be current to the latest threats.

The database will permit the study of taxonomies of various types. Taxonomies will be created by applying labels to the individual database entries, as automated by software currently in development. I will lead the construction of this tool to explore the development of an optimal taxonomy. We intend this to be a significantly larger database than those presently available publicly, such as TrustHub [133,134]. To the maximum

extent possible, we intend to both borrow from and share with ongoing related efforts, such as those in the Trusted and Assured MicroElectronics Forum [164]. Part of the database creation research will include exploration of automated Trojan insertion tools, such as [165] and [166]. These tools may be useful in quickly populating the database with relevant test articles.

6.1.2 Automated Detection Method Application

I am presently continuing work described in this document to automate the application of detection methods to the above database of HTH test articles. Our more advanced version, currently in-development, will automate the detection method application as a workflow, permitting not only fully automated detection methods but also those which require human-in-the-loop interventions and guidance. Furthermore, the ability to track the order in which detection methods are applied is in development. We anticipate this order will have an effect on outcomes. The combinations performed in this work did not consider order. For example, Method A might have a high detection rate but poor false alarm rates, and Method B might have a poor detection rate but provides concrete evidence that Method A's false alarms are, indeed, incorrect. Running these in sequence with knowledge of prior results may have effects that alter the outcomes of games. Notably, this will cause our strategy count to increase. Not only will combinations of detection methods matter, permutations will as well.

The end goal of this task is to automate the application of detection methods in a manner that realistically captures the ways they would be deployed in industry. We will also be able to take the key learnings from automating this detection method application to guide our prescription generation. That is, if we are able to automate test method application during the development of the test database, we can simply use the workflow for test method permutation determined to be optimal as a way of applying that test method permutation in an automated fashion. The workflow from the test automation simply becomes the prescription.

6.1.3 A User Software Application

The GameRunner software described in this document is designed to be used by threat analysts knowledgeable of the game theoretic underpinnings of the tool. Those threat analysts would use it to produce a database of prescriptions for a variety of contexts. GameRunner is not designed for the average FPGA and ASIC designers to use directly. We would rather have another tool that consumes the database of prescriptions and produces guidance for those designers after the input simple metrics related to the design they are trying to protect. We want this interface to abstract away the game theoretic decision engine to avoid requiring digital designers to learn game theory to make use of our tools. To this end, I have specified another application for users to input information about their design and threat concerns and simply receive the appropriate prescription. This application is one among many interfaces presently in development that offer different views and controls over the data models used in this system. We revisit this application in the context of its intended deployment framework in the next section.

6.1.4 OpTrust Framework

In addition to determining how to produce the database, automate test methods, and create a simpler software application for designer/users, I have specified a framework for how all these tools will interact with the various parties that will use it. The specifics of the interfaces are still in development, so in this section we refer more abstractly to the roles played by each party. We refer to the framework that comprises all the various applications and interfaces as OpTrust.

6.1.4.1 Roles within the OpTrust Framework

A core observation that initially led us to develop separate user roles for OpTrust is that the utility functions involve models and variables that will likely be determined by different parties. The first party is a red team capable of using the OpTrust tools for modeling and measuring the efficacy of the adversary and defender methods, including defining the opponents' strategy sets and using GameRunner to experimentally set the probability values required by the utility functions. The operation of the red team – producing the databases necessary for GameRunner – is illustrated in Figure 27. The red team is likely

composed of experts in digital test and verification and HTH detection. Thus, they are likely not the appropriate party to set the economic variables in the equations. This would be the role of a group of analysts we refer to as a threat team. We will revisit Figure 27 in more detail shortly.

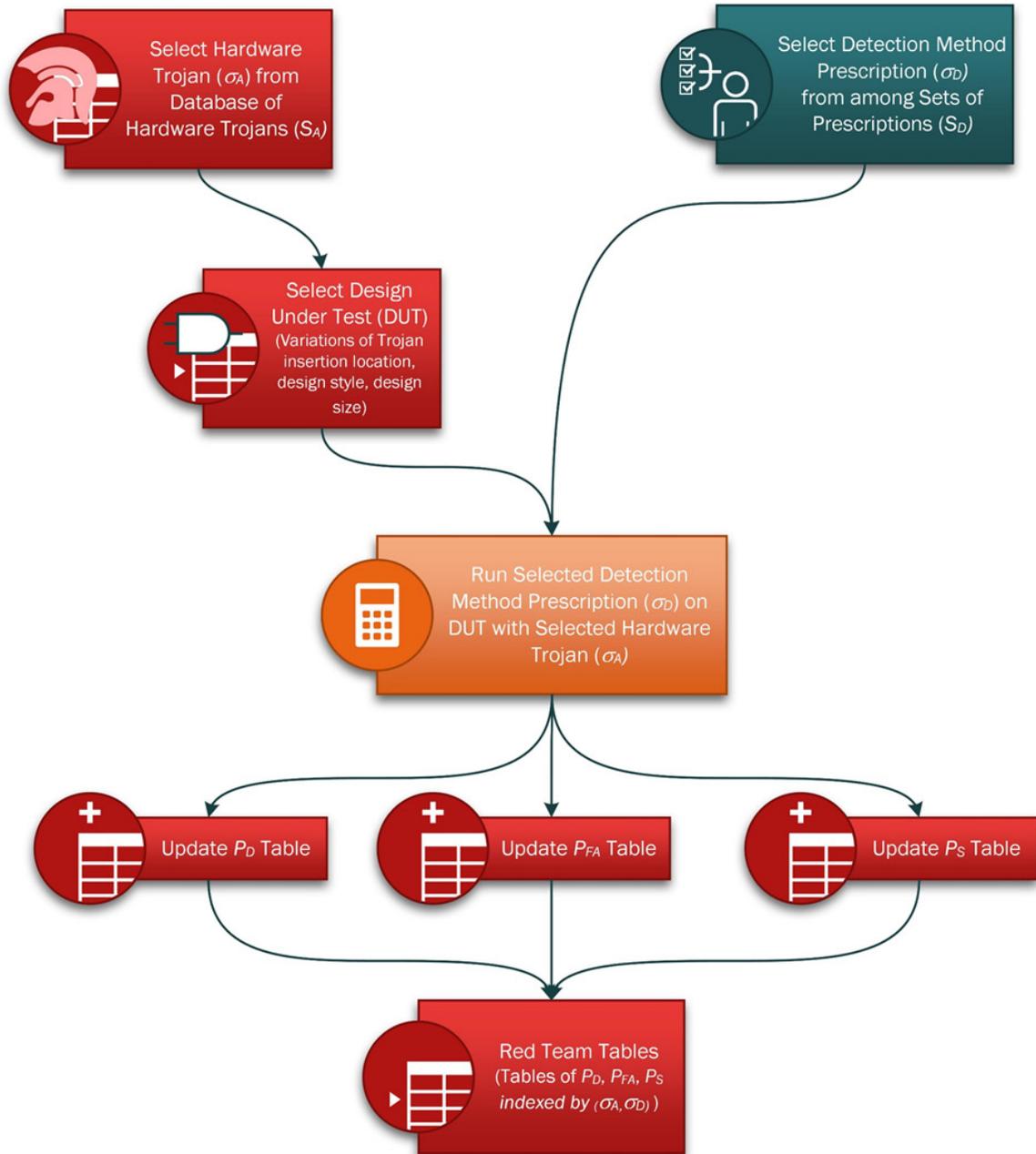


Figure 27. Red Team Table Generation

Chapter 6. Ongoing Work and Applications

Threat team experts are capable of quantifying the threat environment, including the costs of defensive strategies, costs of attacks, cost to the defender of resolving false alarms, cost to the adversary of detections, adversary financial gain if successful in attacking, and defender financial loss if adversary is successful. The threat team is also responsible for creating relative monetary estimates of non-financial strategic outcomes (e.g., how much the adversary or defender are willing to spend to avoid or gain a particular non-monetary result). Properly characterized, these threat environment variables will be selectable by referencing simplifying models I named “Standard Games” that are indexed on levels of criticality, with each selection involving appropriate changes to the underlying security economic variables.

In this section, we use examples describing five levels of criticality for simplicity. In reality, each criticality level will be a complete model of an attacker/defender scenario, and may be referred to in slightly more complex terms. For example, the user might wish to select an aggressive adversary, a risk-averse defender, a cost-averse defender, etc., and the threat team would have to have used the variables in the utility functions to construct those scenarios in advance. In Section 6.2 we define how our advancements in theory will allow them to do that. For the remainder of this section, however, we illustrate the framework principles with the aforementioned simplistic five security levels, envisioning them to sequentially increase both the criticality of the design being protected and the capabilities of the adversary as the levels increase from 1 to 5.

The final party – the microelectronics system developer – has the role of creating a trustworthy design. Their only interaction with OpTrust should be to select the criticality level of their design and determine the defensive strategies available to them at the point in the design flow for which they are responsible. This user action is what determines which prescription they should use. As we will see, the database of prescriptions will have been pre-computed prior to the user interacting with the OpTrust framework.

Separating the roles in the interface in this way has many benefits. One major benefit we have already mentioned is that the ASIC or FPGA designer can concentrate on developing and testing the design without being required to have special knowledge of test methods, threat modeling, and game theory. Another benefit is that both the red team and the threat definition team can concentrate on the aspects of the challenge that best fit their

competencies. A third benefit is that the interfaces themselves – and the information those interfaces provide access to – may be subjected to different security sensitivities. For example, a developer might be granted a level of privilege that allows them to receive guidance about their design without gaining access to all the red team, threat team, and game theoretic variable data involved in the guidance decision. It perhaps goes without saying, but a database of all known hardware Trojans organized into a searchable taxonomy that efficiently pairs to fully automated Trojan insertions tools is not something that should be released widely, lest it be used by malicious actors to nefarious ends. By separating the user roles and associated software, highly sensitive data might be used within the software used by the red and threat teams without exposing it for scrutiny by every designer.

6.1.4.2 Separating the Pre-Computation and Developer Environments

To secure sensitive data related to actual threats and hardware Trojans, it is preferable for the red and threat team data to be inaccessible to the developer. That is, it would be preferable if the recommendations could be precomputed by GameRunner for use in the in-development user application. The designer/user should simply receive a table of recommended detection methods (which we call “prescriptions”) that are calculated based on the “Standard Games,” which are themselves based on the criticality index from the threat team. Once each standard game is solved, the GameRunner produces a table of prescriptions. It is *only that table* that is provided to the developer in any format. This saves the developer from the computation time of OpTrust, and it saves the red and threat teams from having to share the raw information about hardware Trojans and threat environment assessments with the developer. The developer should just receive the prescriptions, not all the information in the databases and assessments that led to the creation of the prescriptions.

This desired structure brings up the fact that the red team and the threat team need to pre-compute the values for which they are responsible in the OpTrust security economic equations. We illustrated the red team precomputation process in Figure 27. The red team should test several HTH-exploited designs from each taxonomic entry described in Section 6.1.1 against each prescription of test methods. Note that every strategy in the defender’s set of methods is actually itself a permutation of test methods. For example, run

constrained random simulation to some toggle coverage, run Boolean logic equivalence, and run directed simulation to some toggle coverage (in that order) would be one strategy. Each other defender strategy is another permutation of test methods. Each time such a test is run, the red team tables (one for each probability variable) is updated.

Figure 28 depicts the pre-computation process required of the threat assessment team. This team is required to gather the information needed to establish the “Standard Games.” We – arbitrarily – set the number of games to five levels of criticality. That is, there are five sets of different values of the security economic variables determined by the threat team. These are the same criticality levels that the designer will select for their design, but this is a long time in the process prior to the designer’s selection. Instead, at this point, the threat team simply fills out a table that has all the variables in it for which they are responsible, as listed in the figure.

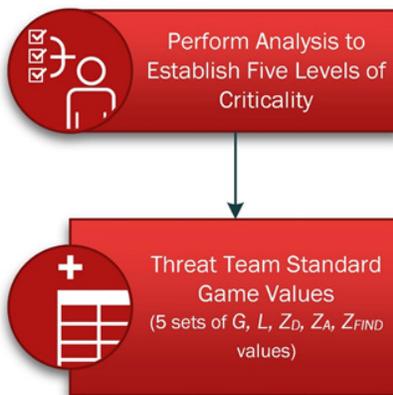


Figure 28. Threat Team Standard Game Generation

Figure 29 begins in its upper left-hand corner with the red team and threat team tables generated by operations in Figure 27 and Figure 28. These tables are computed in the “Pre-Computation Environment,” which for the OpTrust framework, we consider to be separate from the “Developer Environment.” After these two tables are created by the automated test method application referencing the hardware Trojan database in the Pre-Computation Environment, the remaining task in this environment is to use the variables described in those tables and use GameRunner to compute prescriptions for (in this example) each of the five “Standard Games” using the probabilities for every adversarial and defensive strategy described by the red team. Having finished this, we have a very simple outcome.

Chapter 6. Ongoing Work and Applications

We have a table of prescriptions (sets of tests required of the designer) that are indexed only by the criticality level of the design.

It is only this table that must be transferred to the Developer Environment. This table includes far less sensitive information than that which the red team and test team required in order to compute the table. Thus, the table of prescriptions should be able to travel into environments without the same security concerns as the Pre-Computation Environment, which must process raw information on existing hardware Trojans. The responsibilities in the Developer Environment include those for the developer (to select the prescription based on their design's criticality) and those for some sort of system for tracking the results of the tests required by the prescription to ensure compliance. Again, in wider usage, this table may be more complex than the five criticality levels listed here. However, even if the table is more complex, it will nonetheless abstract the user away from the full concerns of the pre-computation environment.

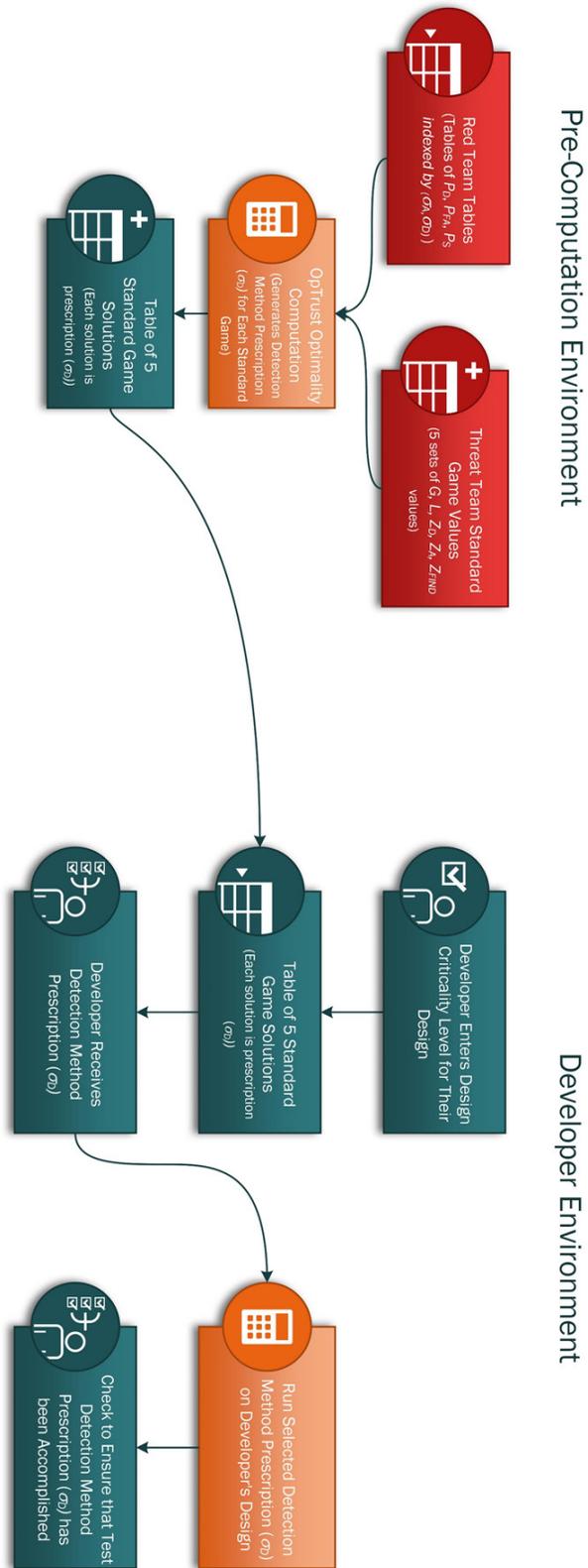


Figure 29. Separation of Pre-Computation Environment and Developer Environment

6.1.4.3 *Implementation*

The implementation of this framework and its constituent components is not only a large initial task, it is one that will require ongoing updates. The HTH database, threat team data, and the defender strategies are all time sensitive based on the activities of the adversaries and availability of defense methods. It must consistently be updated. The framework as described here as well as the methods for continuous updates have been specified and are planned for further refinement and implementation.

6.2 **Advancing the Theory**

In addition to advancing the framework, we are also exploring possible updates to the underlying theory of OpTrust. Some of this advancement is related to new applications of our tools and methods – created to solve the HTH problem – to new domains. That possibility is treated in Section 6.3. In this section, we consider more generally how the theoretical underpinnings of GameRunner might be advanced. Such advancement can take the form of exploring alternatives to the simplifying assumptions we made in constructing the games of this work:

- Assumption: The players only get *one play*. Alternative: We could instead consider multi-play games, such as the ubiquitous Stackelberg security game. We avoided adding the complexity of multiple plays thus far, since the first stage of any Stackelberg-modeled encounter would generally take the form of the strategic game played in this work. We focused in this work on getting that initial encounter right. With that encounter well modeled, Stackelberg games might be considered next.
- Assumption: The knowledge the game is *complete* and *symmetric*. Alternative: We could instead consider games where the parties are playing with incomplete information, or those in which one party holds an information advantage over the other. This latter case may be appropriate if one party can be certain they have a strategy the other is unaware of, such as a secret detection method.
- Assumption: The knowledge of the game is not *perfect*. Alternative: In games with turn-based, rather than simultaneous play, the potential exists for knowledge of outcomes to be perfect, which we might explore.

- Assumption: The knowledge of the game is *common*. Alternative: We might model opponents who are unaware of the amount of game knowledge their opponent has.
- Assumption: The game is not *cooperative*. Alternative: While this seems like an assumption we would never question, we might. A cooperative game only requires that the opponents have a means of communicating that enables them to pursue a solution that is not purely competitive. One could imagine a strategy where such a channel might accomplish a “do nothing” strategy if said communications allowed the opponent’s incentives to align in ways not foreseen by the base game in this work.
- Assumption: The players are *rational*. Alternative: Human players are rarely fully rational. “Subrational” play – that which is close to rational and has a definable mathematical relationship to rationality – is an area ripe for exploration. Much of Section 6.2.2 is dedicated to exploring alternatives to rationality.
- Assumption: The base metric of the game is *dollars*. Alternative: There could be hardware security games that are worth playing when we consider one player or another to have infinite resources. These games may be more driven by some other rank ordering of preferences other than dollars.

The next subsections expand further on which of the above game dimensions are most ripe for further exploration.

6.2.1 Sequential Games and Repeated Play

A consideration in step games that is not treated in this work is that they are played in sequence, as illustrated in Figure 4 on Page 48. That is, the adversary and defender have the opportunity to interact at the requirements game, then the HDL game, and so forth. Decisions made in a prior games should inform future steps through a feed-forward mechanism. For example, if the defender has adopted aggressive defensive countermeasures in the specification game, the 3rd-Party IP game, and the logical-synthesis game to ensure the absence of Trojans in the logical netlist, this knowledge should feed forward into the physical-synthesis game, potentially influencing the defender to apply fewer or different resources in that game. For simplicity, this feed-forward mechanism is ignored in this dissertation to focus the discussion on individual step games. However, I

am currently considering the feed-forward mechanism and how it leads to the construction of games representing the entire device design lifecycle. Similarly, playing a game that pits the same adversary and defender against each other many times will require a different feed-forward mechanism to track knowledge gained by each party from prior interactions.

6.2.2 Alternative Solution Concepts and Subrational Game Play

As discussed in Section 2.2, the Nash equilibrium may not be the only solution concept that provides value in determining optimal game play. Among the many potential concepts referred to in that section, we plan to explore those that let us consider player rationality. As has been stated previously in this work, the current model treats the players as perfectly rational, a scenario that is unlikely in reality.⁴⁴

6.2.2.1 General Subrational Beliefs

Some of alternative solution are strongly related to the Nash equilibrium and are accomplished simply by adequately applying weights to represent irrational beliefs in the utility functions prior to solving the Nash equilibrium using our existing solvers. We will see how this is done later in this section. Other solution concepts consider player rationality to be bounded, such as is the case for Quantal Response Equilibrium (QRE). QRE uses bounded rationality to describe the degree to which each player (the adversary and defender) makes decisions in accordance with the most rational play. The most rational play is still defined by and centered on the Nash equilibrium. By bounding the rationality of each player, we are able to avoid one pitfall of the Nash equilibrium concept: strict equilibrium offers no means of knowing whether a game was close to having a different outcome. All Nash tells you is what the most rational play is, not whether another play is

⁴⁴ Our work has primarily focused on the *implementation* of security (e.g., HTH's and detection methods) and the *economics* of security (e.g., modeling player beliefs as utility functions). As we step into player rationality – where and how player beliefs diverge from reality – we are stepping into the emerging field of the *psychology* of security. A catalyst that helped launch the modern resurgence of security psychology studies was Schneier's essay in [172], which remains an excellent primer on the subject.

very close to rational. With QRE, we can know both the most rational response and nearby responses that a less-than-perfectly rational player may instead select. Notably, since solving games with QRE is computationally expensive, the Pre-Computation Environment concept espoused above carries with it the added value that the developer does not have to wait for new games to be solved.

The challenge of applying subrational solution concepts is that they tend to be too general to reflect the specific ways in which the opponents in our game will be irrational. That is, those that apply weights to the utility function to represent an attraction to certain strategies, apply them to the entire utility function. This is the case with the Prelec function [170] as applied in [96]. Those like the general QRE function apply bounds to all possible ways the players might be irrational. What we want in our future work is a more fine-grained means of modeling player irrationality, allowing us to specifically and separately define the portions of the adversary/defender interaction about which they are exhibiting their subrational beliefs. The next section outlines our approach to this ongoing work.

6.2.2.2 Specific Subrational Beliefs

Because our work is motivated to describe how a human adversary and defender would actually behave when contending with each other in an HTH-related interaction, we must consider not only the most rational choice (as our work does, primarily) but also those human-directed strategy selections that might diverge from the most optimally rational. That is, humans do not make the best decisions, and this is reflected in their beliefs. Lest we be surprised by this, we need to be able to protect the defender against the potentially less-than-optimal choices of an irrational adversary. Because we have invested work in creating utility functions that comprise the components of rational decision making, the resulting variables allow us to describe the players as holding different beliefs about their constituent probabilities and costs. That is, prior to considering which general approach to subrationality we wish to make use of, we can first define the separate and specific ways in which the players might exhibit irrationality.

A few examples follow, with more complete illustrations of each to appear in forthcoming works:

1. The Irrationally Aggressive Adversary (IRRAAD). The IRRAAD may not believe that the defender's detection methods work as effectively as the defender believes them to. Thus, the adversary's P_D and P_{FA} could be qualified by the appropriate weighting function to reflect their belief that those values diverge from those we have empirically determined. The IRRAAD may also make choices without any thought to whether they might get caught. In this case, we could set P_{ATT} to be very low for their utility function. To provide guidance to the defender, we would then calculate the adversary's utility functions using these altered weights and the defender's utility functions without alteration. This calculates the defender's rationally optimal choice in the face of the IRRAAD. Re-solving the HDL game from Chapter 4 in the consumer economy with a rational defender playing the IRRAAD who believes that every P_D is half of the empirically determined value and every P_{ATT} is 0 yields an adversary willing to play GATE 23%, STSQ 34%, and GLST 43% of the time and a defender who plays SCOA 61%, STRC 8%, and BOOL 31% of the time.⁴⁵
2. The Counter-Your-Adversary (CYA) Defender. This defender holds a belief that they need to do something to counter any available adversary strategy, not just those the adversary is likely to play. This "checklist" style approach to security is a common requirement in some industries where the reward for imagining a threat and countering it outweighs the rewards for efficiencies due to avoiding unnecessary costs. We can model optimal decision making with the checklist requirement by enforcing a strategic restriction: eliminating defender strategies from the game that do not offer at least 50% probability of detection within each entry in the HTH taxonomy. For example, revisiting the HDL step

⁴⁵ This analysis does expose one of the assumptions in our model that is also worth future examination – that of perfect information. In this case, not only are the adversary and defender operating upon different beliefs, but they know that each other are operating on different beliefs as well as exactly what each other's differing beliefs are. Games with imperfect and unknown information is another topic of our ongoing work.

irrationality into a game related to HTH decision making. However, since the underlying utility functions in [96] suffered the previously-discussed artificial limitations, the application of Prelec to those utility functions simply weights an already oversimplified game. The synthesis of their observations related to Prelec and our utility functions does offer a unique outcome. We are able to apply Prelec or similar weighting functions to each constituent component of the utility functions, thus blending the empirical nature of the weighting functions with the empirically-constructed utility functions of our model.

For example, one possible such weighting scheme is illustrated in Eq. 6 and Eq. 7 below.⁴⁸ Here we introduce a scalar weighting representing each player’s rationality as related to each probability in the utility functions. Each weight is in $[0, \infty)$, with values less than 1 indicating that the player irrationality believes the subject probability is lower, a value of 1 meaning the player holds perfectly rational beliefs about this probability, and values greater than 1 indicating the player irrationally believes the probability is higher. In this formulation, each rationality weight is a scalar value, though in other formulations they could be functions of the strategies. For these example equations, the weights are:

- w_{AD} : Weight describing adversary rationality about probability of detection
- w_{AS} : Weight describing adversary rationality about probability of success
- w_{AA} : Weight describing adversary rationality about probability of attribution
- w_{DD} : Weight describing defender rationality about probability of detection
- w_{DF} : Weight describing defender rationality about probability of false alarm

$$U_A(\sigma_A, \sigma_D) = \{[1 - w_{AD}P_D(\sigma_A, \sigma_D)]w_{AS}P_S(\sigma_A)\}G - Z_A(\sigma_A) - w_{AD}P_D(\sigma_A, \sigma_D)w_{AA}P_{ATT}(\sigma_A, \sigma_D)Z_{find}(\sigma_A).$$

Eq. 6. Possible Rationality-Weighted Adversary Utility Function

attitudes when gambling with juice or tokens as currency [166]. With a hat tip to Dave Barry, “Juice Gambling Monkeys” would be an excellent name for a rock band.

⁴⁸ This formulation represents the general case of how to apply rationality weights to our equations. These are not the Prelec weighting functions, but the weights illustrated here could be appropriately replaced by Prelec functions if we wish.

$$U_D(\sigma_A, \sigma_D) = [w_{DD}P_D(\sigma_A, \sigma_D) - w_{DD}P_D(\sigma_A, \sigma_{D0})]L - Z_D(\sigma_D) - Z_{FA}(\sigma_D)[w_{DF}P_{FA}(\sigma_D) - w_{DF}P_{FA}(\sigma_{D0})].$$

**Eq. 7. Possible
Rationality-
Weighted
Defender Utility
Function**

We are continuing this work at the writing of this document. The most immediate work is how to derive weights that represent real-world decision making.

6.2.3 How Should We Then Play?

The irrationality discussion and the varieties of possible models leads naturally to the question of which model we should use to make defender recommendations. At present, GameRunner provides the guidance for the rational defender facing a rational adversary. It seems intuitively obvious, however, that in many high-value scenarios that we may want to consider how to counter adversaries who are irrationally aggressive, as in the IRRAAD model discussion above. We will certainly also want to consider the proper application of rationality weighting functions. While we do not yet know the outcome of this work, we are sketching an approach that will involve creating many models and solution concepts then comparing, blending, and following their recommendations. Continued empirical testing combined with feedback from following prescriptions in the real world should reveal outcomes that help us improve models and select from among the best of them.

How to blend all these models into a single recommendation is a topic of ongoing research. Similarly, we continue to consider how to instruct an individual user/designer player to play given a mixed strategy solution, as illustrated in this work. Furthermore, the games in this work never demonstrated multiple pure Nash equilibria, but this outcome is possible. In this case, if we are in a many-players scenario, we would suggest that the player population mixes their strategy selections across each pure Nash outcome with equal weighting. However, we also have to consider what to tell the individual player to do the first time and only time the game is played in that 1-member “population.” Without any means of the adversary and defender coordinating, there is no means of determining the absolute best play. Initially, we may just expose each as equal possibilities and let the

defender decide which prescription is most preferable. In selecting from among them, they would be choosing *an* optimal solution, but not the *only* optimal solution. Alternatively, we could suggest the equilibrium point that leads to the highest defender payoff. However, further consideration is warranted and is ongoing.

6.3 Applications

I envision many applications of this work, each of which are outlined briefly in this section. Initially, there is a direct application of the work with the only modification being completing the aforementioned in-progress applications and deploying them to a relevant environment. Next, we consider the idea that what we have referred to in this document as the Trust Game is an instance of a general case referred to as the Detection Game. While the inspiration of our game equations was to define optimal hardware Trojan detection strategies, the game can be generalized to treat any security scenario where an adversary is trying to hide an exploit and the defender is trying to detect it. This has applications not only to devices other than FPGAs but also to software. The observation that Detection Games operate upon the same metrics allows us to discuss the composability of games into large games that represent concerns of systems comprising multiple exploitable elements. Finally, we consider an alternative formulation we call the Prevention Game. This has applications to fields where the defender is attempting to prevent an exploit in the first place, such as countermeasures to prevent Trojans, tamper, or counterfeiting.

6.3.1 Direct Application of this Work

The primary application is the deployment of this method in the Air Force Trusted Silicon Stratus (TSS) Cloud [168][169]. TSS is a cloud of computing resources in a specialized enclave of the Amazon Web Services GovCloud with security modified to meet a variety of Air Force requirements. TSS is designed to be a centralized, secure research and design cloud that offers not only compute resources but also enterprise cloud licenses of commercial and government Electronic Design Automation (EDA) and validation and verification (V&V) software. Cloud resources will serve both as a means not only for (1) gaining access to the parallel compute capabilities required for processing the large datasets, solving the large games, and storing the large databases necessary for the back-

end of GameRunner and associated applications but also for (2) providing users with access to the resulting Jenkins workflows through the aforementioned user tool. The Jenkins workflows can be specifically constructed to reference the available cloud resources and EDA tools. Since the Air Force intends this cloud to be a means of securing the designs that many Air Force designers will be producing, placing our analytical technique and its guidance in this cloud would lead to deployment across a large population of developers.

The TSS application leads to some specific observations about the game model we have developed. One such notable observation is that the mixed strategy Nash equilibrium has a rare direct physical embodiment. There is a population of defenders (the Air Force designers) whose economic outcomes are borne by one entity (the Air Force). Thus, the mixed strategy outcomes of having some percentage of defenders play one strategy and some other percentages playing others maps directly to: the countermeasure recommendations will be different for many Air Force designers, but the overall outcome is optimal for the Air Force. Another observation is that the games can be constructed with knowable cost variables. For example, the Air Force can work with us to quantify the strategic value of markets and outcomes. Furthermore, in TSS, many of the mitigation strategy costs are either sunk (e.g., the Air Force has already purchased the enterprise tool license) or they are charged on a per-use basis in the TSS cloud. This gives us much of the defender economic data required for our models.

Finally, in the TSS cloud, we can perform a specific set of experiments related to “Does this work?” There are existing sets of recommendations for HTH mitigation via expert-developed checklists [171]. These checklist-based strategies can be enumerated and cost analyzed. Based on both empirical research and real-world results, the cost of performing the checklist-based mitigation and the cost of our optimized mitigation strategy may be compared longitudinally and for large user populations. If we are successful with this model, the overall cost of creating and deploying countermeasures and reacting to HTH attacks should be less than those of the expert-created checklist methods. The TSS cloud implementation is ongoing.

6.3.2 Applications of the General Detection Game

This work has focused on the configurable logic of field programmable gate arrays (FPGAs). I am developing games that address ASIC, mixed-signal ASIC, eFPGA, analog components and software/firmware for GPU/CPU and artificial intelligence (AI) processors. Each of these alternative games will have their own exploit and defense models for the adversary and defender. However, in each case, we observe that the underlying utility functions remain the same. The costs and probabilities associated with pitting an adversary's exploit strategies against a defender's detection strategies can be modeled using the same basic equations given in Eq. 1 and Eq. 2. This observation has led us to realize that our Trust Game is an instance of a general game we refer to as the Detection Game. The same security economic framework, the same utility functions, the same rationality weights, and the same solution concepts can apply to any scenario when an adversary is attempting to exploit the system and the defender is seeking to detect that exploit.

We can generalize further if we consider the varieties of what we mean by an "exploit." For example, a counterfeit device is a type of exploit where the adversary's attack is to produce an illegal copy of a legitimate product. In this case, the defender is not the designer of that product. Rather, they are the user of it. The adversary is trying to sell the defender an illegal copy, and the defender is trying to detect it to ensure they are only using genuine parts. The adversary might have many strategies – remarking/repackaging old parts, theft of overproduced parts from a fab, or manufacturing new parts from stolen masks. The defender may try to detect those counterfeits through visual inspection, side channel or power analytics, or thorough on-tester evaluations. Again, the same utility functions can apply, since this is another instance of the Detection Game.

Another result of determining that the Trust Game is a general instance of a Detection Game is that we can use the general Detection Game to reason about and compare dissimilar exploit strategies across a heterogenous attack surface. This is the type of attack surface that systems present: exploits of systems may emerge through their software, processors, firmware, printed circuit boards – or even through exploits of discrete electrical components such as capacitors or system fans. A Detection Game may be played related to the specific field of each type of exploit by modeling it as an adversary/defender

interaction related to that specific component. Additionally, a large Detection Game may be played across the entire attack surface simply by providing the adversary with all exploit strategies across the entire attack surface and providing the defender with all the defensive strategies. Note that, as with the games explored in detail in this dissertation, the defensive strategies would include permutations of detection strategies, but in this case, those permutations would select from detection methods that operate across all of the different components of the system.

As a proxy for a complete microelectronic system, we may consider an abstract model of the resources on the new class of FPGA Multi-Processor System on-Chip (MPSoC) to represent the subsystems of a complex microelectronic system. An FPGA MPSoC is quite complex and may include a variety different CPUs, GPUs, programmable logic resources, programmable security resources, and AI processing resources. For this simple example, we consider the system to have three subsystems worth exploiting: the software running on the CPUs, the bitstream running on the programmable logic, or the underlying ASIC silicon of the device itself. We consider the adversary to pragmatically consider where they would like to exploit this system. That is, they wish to optimally exploit. Similarly, the defender wishes to optimally select a set of exploit detection methods.

In the individual domains of concern, the adversary strategies are hardware Trojans implemented in the silicon of the device ($S_{AdversaryASIC}$), hardware Trojans implemented in the bitstream firmware that configures the programmable logic of the device ($S_{AdversaryFPGA}$), and viruses that exploit the software running on the CPU ($S_{AdversarySW}$). Those strategy sets are composed of exploits as below:

- $S_{AdversaryASIC} = \{\sigma_{HWTrojan0}, \sigma_{HWTrojan1}, \dots, \sigma_{HWTrojanW}\}$
- $S_{AdversaryFPGA} = \{\sigma_{BitstreamTrojan0}, \sigma_{BitstreamTrojan1}, \dots, \sigma_{BitstreamTrojanX}\}$
- $S_{AdversarySW} = \{\sigma_{Virus0}, \sigma_{Virus1}, \dots, \sigma_{VirusY}\}$

Notably, at the system level, the adversary may have attack strategies that split the attack across more than one domain of the system. For example, one portion of the attack might be a latent, hard-to-activate Trojan in ASIC silicon, and the other portion might be a software exploit that produces the activation signal. To capture this, we must consider

another set: $S_{AdversarySystem} = \{\sigma_{System0}, \sigma_{System1}, \dots, \sigma_{SystemZ}\}$. In each of the above sets the first entry ($\sigma_{HWTrojan0}$, $\sigma_{BitstreamTrojan0}$, σ_{Virus0} , and $\sigma_{System0}$) are the “do nothing” strategies.

In this simplified game, we assume the adversary is pursuing one goal (e.g., system control, denial of service, kill switch, etc.) that they want to accomplish through the selection of one exploit strategy, whether by attacking one subsystem or performing a system-level exploit. The full adversary strategy set for the system may be created simply by combining the sets:

$$S_{A_SYSTEM} = \left\{ \begin{array}{l} (\sigma_{HWTrojan0}, \sigma_{HWTrojan1}, \dots, \sigma_{HWTrojanW}), \\ (\sigma_{BitstreamTrojan0}, \sigma_{BitstreamTrojan1}, \dots, \sigma_{BitstreamTrojanX}), \\ (\sigma_{Virus0}, \sigma_{Virus1}, \dots, \sigma_{VirusY}), \\ (\sigma_{System0}, \sigma_{System1}, \dots, \sigma_{SystemZ}) \end{array} \right\}$$

The defender strategy set is similar. At minimum, they have detection methods in each domain to consider, as notated below:

- $S_{DefenderASIC} = \{\sigma_{HTHDetect0}, \sigma_{HTHDetect1}, \dots, \sigma_{HTHDetectM}\}$
- $S_{DefenderFPGA} = \{\sigma_{BHTHDetect0}, \sigma_{BHTHDetect1}, \dots, \sigma_{BHTHDetectN}\}$
- $S_{DefenderSW} = \{\sigma_{VirusScan0}, \sigma_{VirusScan1}, \dots, \sigma_{VirusScanO}\}$

If the defender is only allowed to play a single detection strategy, their set of strategies to protect the system would be:

$$S_{D_SYSTEM_SINGLE_STRATEGY} = \left\{ \begin{array}{l} (\sigma_{HTHDetect0}, \sigma_{HTHDetect1}, \dots, \sigma_{HTHDetectM}), \\ (\sigma_{BHTHDetect0}, \sigma_{BHTHDetect1}, \dots, \sigma_{BHTHDetectN}), \\ (\sigma_{VirusScan0}, \sigma_{VirusScan1}, \dots, \sigma_{VirusScanP}) \end{array} \right\}$$

However, limiting the defender to playing a single strategy in one domain to protect a system is unreasonable. They would want to at least have the opportunity to select a strategy set that protects them across multiple domains.⁴⁹ Within each domain, we would

⁴⁹ This system game illustrates a common strategic advantage enjoyed by the player on offense: they get to pick the battlefield. The adversary needs only select one exploit that

want to consider the permutations of each available individual strategy, since, as discussed earlier, the order of application matters in many cases. More specifically, we are interested in all the k -permutations of strategies within each domain. k -permutations are intuitively understood through their common description as “sequences without repetition.” We are interested in every possible non-repeated sequence of detection methods, including those sequences that do not use all available strategies. Between the domains, we do not anticipate order or detection method application having any bearing, unless an interrelationship between the domains can be demonstrated that would change this assumption. This is an untested intuition that needs to be more fully explored in the future. However, if it holds, the defender strategy set will include every combination of every permutation of available strategies within each domain strategy set.

This formulation will result in substantially larger games than those considered inside any individual domain. The ability to use a large game solver, such as those used in GameRunner, would be a requirement. Offline precomputation before a system designer interacts with a system-level OpTrust user tool would also prove valuable to ease of use for the end user. Furthermore, we expect that the strategy sets could be reduced through analysis. One way would be to define strategies within each domain for which order does not, in fact, matter, and eliminate k -permutations that represent effectively identical entries in the strategy set. This section represents a promising start to our ongoing work in system-level detection method optimization work.

6.3.3 Applications of the General Prevention Game

To this point, all of our game formulations have concerned themselves with varieties of the Detection Game, where an adversary seeks to exploit a system and the defender seeks to detect that exploit. However, detecting malice is not the only manner in which a defender might protect their microelectronic system. Another approach is to make it hard for the adversary to produce their malice in the first place. We previously discussed this

accomplished their end; the defender needs to consider all possible exploits. This conundrum leads many to argue that perfect security is provably impossible. Our work here hopes to at least make security strategies rationally optimal.

in the introduction in the context of hardware Trojan countermeasures – such as circuit obfuscation – which try to make it hard for the adversary to know how or where to attack. This prevention paradigm is also descriptive of the problem of tamper. The defender executes strategies in their design – obfuscating, encrypting, and otherwise protecting their design – after which they release their design into the wild where the adversary gets the chance to attempt tamper.⁵⁰ Another adversary/defender engagement that would benefit from a Prevention Game formulation is that of preventing counterfeiting. When we previously treated counterfeiting, it was in the context of detecting counterfeits after they have been built, which falls within the Detection Game. Many are working on strategies (such as logic locking) that make it hard to produce a usable a counterfeit in the first place. These scenarios would be better modeled by a Prevention Game.

We briefly sketch our ongoing approach to a general Prevention Game. The formulation is quite similar to the detection game, but with important differences. Both the adversary and defender have sets of discrete strategies, S_A and S_D , as in the Detection Game. Both also have their individual null strategies available to them, σ_{A0} and σ_{D0} , to represent, respectively, the adversary’s ability to elect not to attack and the defender’s ability nothing additional to traditional design practice to prevent the adversary’s goal. The core probability of interest is now the Probability of Prevention, $P_P(\sigma_A, \sigma_D)$. This probability is a function of both the strategy of the adversary and the strategy of the defender. The adversary is seeking gain G , which will be gained by their attack if they are successful, and the defender is seeking to minimize loss L , which would be lost if the adversary succeeds. The defender’s cost, $Z_D(\sigma_D)$, is dependent only on their own strategy selection whereas the adversary cost, $Z_A(\sigma_A, \sigma_D)$, considers both their own strategy and that of the defender. The defender is not detecting, so there is no need to model false alarms. Similarly, the adversary has no concern of being detected, so attribution probabilities and penalties upon attribution are not necessary. The resulting utility functions are given below in Eq. 8 and Eq. 9.

⁵⁰ Our earliest work using game theory in the context of microelectronics security was in the field of anti-tamper [3].

$$U_D(\sigma_A, \sigma_D) = [P_P(\sigma_A, \sigma_D) - P_P(\sigma_A, \sigma_{D0})]L - Z_D(\sigma_D)$$

**Eq. 8. Defender
Utility Function
in the Prevention
Game**

The defender's beliefs are concerned with the ability of their selected prevention strategy to increase the net likelihood (as compared to the null strategy) of preventing their loss L in the face of the adversary's selected attack strategy, less the cost of implementing that prevention strategy.

$$U_A(\sigma_A, \sigma_D) = \{[1 - P_P(\sigma_A, \sigma_D)]\}G - Z_A(\sigma_A, \sigma_D)$$

**Eq. 9. Adversary
Utility Function
in the Prevention
Game**

The adversary is concerned with the degree to which the defender's selected prevention strategy prohibits them from accomplishing their desired gain using their selected exploitation strategy, less the cost of applying their strategy to a system protected by that prevention strategy.

This Prevention Game formulation is just an initial treatment. When applied to specific domains with consideration for the economics and security concerns of those domains, I will modify the game to better match those scenarios. However, this is a promising start to applying the results of our HTH-focused strategy optimization work to other domains. GameRunner and the emerging comprehensive OpTrust framework provide tools that can easily be modified for significant utility in all the domains of interest in this chapter.

Chapter 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. Future work has been defined for how this tool can be applied at a much larger scale, paving the way for using this methodology to drive industry-level optimal decision making. This future work includes implementing a framework, OpTrust, for reasoning across large test article databases, automating detection method testing, and simplifying the user interface to the system to make it easier for designers to use. Further ongoing improvements include advancing the theory of the system to account for new styles of game play with different players and applications to a wide variety of new domains.

7.1 Final Contribution Summary

The central contributions of this work have been:

1. Security economic models, represented in this work primarily by the adversary and defender utility functions that underly the Trust Game, that consider the

Chapter 7. Conclusions

effectiveness of HTH detection methods when faced with Trojans from a taxonomy of threats, the probabilities associated with various activities of the involved parties, and the incentives of said parties, modeled as economic values. These models consider the individual beliefs of the opponents. If one opponent's strategy is known, the other can optimize their outcome by choosing the strategy that resolves their utility function to highest value.

2. A two-person strategic game, the Trust Game, constructed from the aforementioned utility functions, that allows us to resolve both opponents' optimal play at the same time through the use of the Nash equilibrium solution concept. This game accomplishes our main goal in allowing us to solve for the defender's optimal strategy in detection-based engagements.
3. An FPGA Trojan game model that includes not only the utility functions but also a description of the game's use in context of the design flow for FPGA bitstreams to consider the entire attack surface available to the adversary. The concept of step games was introduced to describe the interactions between adversary and defender at various places in the design cycle. A demonstration was performed using realistic variables – including representative Trojans and detection methods – to illustrate how to develop the variables of the utility functions realistically and apply the game solutions to predict optimal play for both players.
4. Universal microelectronics security game models and a discussion of how other considerations (such as the rationality of the players) can be applied to well-structured utility functions to consider specific questions. This illustrated the broad applicability of the core models used in FPGA Trust Game to other microelectronics security scenarios, including for varieties of malice (e.g., Trojans, tamper, and counterfeiting) and varieties of defensive scenarios (e.g., detection and prevention).
5. An automated method of determining and applying the guidance for the optimal play. This was illustrated by GameRunner, software that provides these features without requiring the user to understand the underlying game theory of the decision engine. The game solving engine provides utility for any game

Chapter 7. Conclusions

solution, demonstrating broad applications. In the context of the HTH Trust Game, the guidance provided may take the form of a prescription of automatable detection methods, demonstrating a realistic path to an easy application of our theory to real-world scenarios.

6. Novel visualization strategies for the exploration of game solutions. Again evidenced in GameRunner, the user may construct scenarios that explore the reaction of the players to modifications of utility function variables. This permits users to ask questions in the context of the modeled scenarios and receive graphical answers without facing the complexities of the game-based decision engine. Future work was defined to produce more software in the same vein: game-based decision making software to optimize microelectronics security strategies for industry.

Appendix A: Detection Method Results

Table 26 lists every benchmark circuit in our dataset, organized by our selected adversary taxonomy category, along with the results each detection method accomplished for the circuit. A 0 indicates that the detection method did not find an HTH. This is the correct answer for every row in which the adversary played the DONT strategy and the incorrect answer for all other adversary strategies. Conversely, a 1 indicates that the detection method found an HTH. This is the incorrect answer for every row in which the adversary played the DONT strategy and the correct answer for all other adversary strategies. The total number of circuits tested were 63.

In games where the defender’s strategies were combined into sets of multiple countermeasures, a logical OR function combined the results of the methods. That is, if any member of the set claimed there was an HTH present, it was marked a 1; else it was marked 0. As with the individual methods, these values were used in the P_D and P_{FA} calculations.

Table 26. Raw Detection Method Results by Adversary Strategy Taxonomy Category

| Adversary Taxonomy Category | Benchmark Circuit Name | BNCH | SCOA | | | CSIM | | SCOA | | CSIM SCOA STRC | SCOA SCOA STRC | CSIM SCOA STRC | SCOA SCOA STRC | CSIM SCOA STRC | SCOA SCOA STRC | CSIM SCOA STRC |
|-----------------------------|------------------------|------|------|------|------|------|------|------|------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|
| | | | CSIM | SCOA | STRC | BOOL | CSIM | STRC | SCOA | | | | | | | |
| DONT | AES-notj-top | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| DONT | BasicRSA-notj | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 0 | 0 | 1 |
| DONT | CEP-gps-notj | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 0 | 0 | 1 |
| DONT | PIC16F84-NoTj | 0 | 0 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 |

Appendix A: Detection Method Results

| Adversary Taxonomy Category | Benchmark Circuit Name | BNCH | | SCOA | | BOOL | | CSIM SCOA | | SCOA STRC | | CSIM BOOL | | SCOA STRC | | CSIM SCOA | |
|-----------------------------------|------------------------------|------|------|------|------|------|------|--------------|------|--------------|------|--------------|------|--------------|------|--------------|------|
| | | CSIM | STRC | CSIM | STRC | CSIM | STRC | CSIM | STRC | CSIM | STRC | CSIM | STRC | CSIM | STRC | CSIM | STRC |
| DONT | RS232-NoTj | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| DONT | RS232- NoTjGate | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| REWR | AES- reversebit1 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| REWR | AES- reversebyte1 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| REWR | BasicRSA- TReverseByte1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| REWR | CEP-gps-Tj- disable-aes | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| REWR | CEP-gps-Tj- reversebit1 | 0 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| REWR | PIC16F84- TReverseBit1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| REWR | RS232- TjReverseBit1 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| REWR | RS232- TjReverseBit2 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| GATE | AES-T100 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| GATE | AES-T200 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| GATE | AES-T300 | 0 | 0 | 0 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| GATE | RS232-T1800 | 0 | 0 | 0 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| ECTR | AES-T1200 | 1 | 0 | 0 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| ECTR | AES-T1500 | 0 | 0 | 0 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| ECTR | AES-T1700 | 0 | 0 | 0 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| ECTR | AES-T1900 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| ECTR | AES-T2100 | 0 | 0 | 0 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| ECTR | AES-T900 | 1 | 0 | 0 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| ECTR | BasicRSA-T300 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| ECTR | BasicRSA-T400 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| ECTR | CEP-gps-Tj- reset-counter | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| ECTR | PIC16F84-T100 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| ECTR | PIC16F84-T200 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| ECTR | PIC16F84-T300 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| ECTR | PIC16F84-T400 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| ECTR | RS232-T300 | 1 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| ECTR | RS232-T500 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| CCMP | AES-T1000 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| CCMP | AES-T1300 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| CCMP | AES-T1800 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| CCMP | AES-T400 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| CCMP | AES-T600 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| CCMP | AES-T700 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| CCMP | BasicRSA-T100 | 1 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

Appendix A: Detection Method Results

| Adversary Taxonomy Category | Benchmark Circuit Name | BNCH | CSIM | | | SCOA | | | CSIM | | | SCOA | | | CSIM | | |
|-----------------------------------|---------------------------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|
| | | | CSIM | SCOA | STRC | BOOL | CSIM | SCOA | STRC | BOOL | CSIM | SCOA | STRC | BOOL | CSIM | SCOA | STRC |
| CCMP | BasicRSA-T200 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| CCMP | RS232-T100 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| CCMP | RS232-T1300 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| CCMP | RS232-T1700 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| CCMP | RS232-T400 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| CCMP | RS232-T800 | 0 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| STSQ | AES-T1100 | 1 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| STSQ | AES-T1400 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| STSQ | AES-T1600 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| STSQ | AES-T2000 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| STSQ | AES-T500 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| STSQ | AES-T800 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| STSQ | CEP-gps-AES-T500 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| STSQ | RS232-T1200 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| STSQ | RS232-T1600 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| STSQ | RS232-T1900 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| STSQ | RS232-T2000 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| STSQ | RS232-T600 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| STSQ | RS232-T700 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| STSQ | RS232-T900 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| STSQ | RS232-T901 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| GLST | PIC16F84-TjGlitchState | 0 | 0 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 |
| GLST | RS232-TjGlitchState | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Bibliography

1. K. C. Y. Mei, “Bridging and Stuck-At Faults,” *IEEE Transactions on Computers*, vol. C-23, no. 7, pp. 720–727, Jul. 1974.
2. S. Fazzari, “Hardware Security: A History Lesson,” in *Trusted and Assured MicroElectronics Forum*, 2018.
3. Image from https://commons.wikimedia.org/wiki/File:Maginot_Line_in-en_svg.svg by Wikimedia user Goran tek-en. Shared under Creative Commons Attribution-ShareAlike 3.0 Unported (<https://creativecommons.org/licenses/by-sa/3.0/>). No changes made.
4. J. Graf and P. Athanas, “How Threats Drive the Development of Secure Reconfigurable Devices,” in *2008 IEEE National Aerospace and Electronics Conference*, 2008, pp. 239–245.
5. J. Graf, “Toward Optimal Hardware Trojan Detection through Security Economics and Game Theory,” in *GOMACTech*, 2016.
6. J. Graf, “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)*, 2016, pp. 91–96.
7. J. Graf, “Towards system-level adversary attack surface modeling for microelectronics trust,” in *2016 IEEE National Aerospace and Electronics Conference (NAECON) and Ohio Innovation Summit (OIS)*, 2016, pp. 474–477.
8. J. Graf, “OpTrust: Software for Determining Optimal Test Coverage and Strategies for Trust,” in *GOMACTech*, 2017.

Bibliography

9. J. Graf, W. Batchelor, S. Harper, R. Marlow, E. Carlisle, and P. Athanas, "A Practical Application of Game Theory to Optimize Selection of Hardware Trojan Detection Strategies," manuscript submitted to the *Journal of Hardware and Systems Security*, 2019.
10. R. Marlow, S. Harper, W. Batchelor, and J. Graf, "Hardware Trojan Detection using Xilinx Vivado," in *2018 IEEE National Aerospace and Electronics Conference*, 2018.
11. J. Graf and P. Athanas, "A key management architecture for securing off-chip data transfers," in *Field Programmable Logic and Application, ser. Lecture Notes in Computer Science*, J. Becker, M. Platzner, and S. Vernalde, Eds. Springer Berlin Heidelberg, 2004, vol. 3203, pp. 33–42.
12. A. J. Mahar, P. M. Athanas, S. D. Craven, J. N. Edmison, and J. Graf, Design and Characterization of a Hardware Encryption Management Unit for Secure Computing Platforms," in *39th Hawaii International International Conference on Systems Science (HICSS-39 2006), CD-ROM / Abstracts Proceedings, 4-7 January 2006, Kauai, HI, USA*, 2006.
13. P. Athanas, J. Bowen, T. Dunham, C. Patterson, J. Rice, M. Shelburne, J. Suris, M. Bucciero, and J. Graf, "Wires on demand: Run-time communication synthesis for reconfigurable computing," in *2007 International Conference on Field Programmable Logic and Applications*, Aug 2007, pp. 513–516.
14. J. Graf, J. Hallman, and S. Harper, "Trust in the FPGA Supply Chain using Physically Unclonable Functions," in *GOMACTech 2010 Proceedings*, March 2010, paper 22.4, pp. 317–319.
15. J. Graf, S. Harper, and L. Lerner, "Ensuring Design Integrity through Analysis of FPGA Bitstreams and IP cores," in *Proceedings of the International Conference on Engineering of Reconfigurable Systems and Algorithms (ERSA)*, 2012.
16. J. Graf, S. Harper, and L. Lerner, "The Integrity of FPGA Designs: Capabilities Enabled by Unlocking Bitstreams and 3rd-Party IP," in *GOMACTech 2012 Proceedings*, March 2012, paper 18.3, pp. 201–204.

Bibliography

17. J. Graf, S. Harper, J. Hallman, and B. Knight, "Managing Risk to Field Programmable Gate Array Trust: A Deployment Framework for DoD Instruction 5200.44," in *GOMACTech 2014 Proceedings*, March 2014, paper 6.1, pp. 77–80.
18. S. Baka, J. Hallman, S. Harper, and J. Graf, "Trust and Reuse of 3rd-Party IP," in *GOMACTech 2014 Proceedings*, March 2014, paper 2.4, pp. 25–28.
19. K. Urish and J. Graf, "Mitigation of Space-Reliability Reduction Trojans in FPGA Designs," in *GOMACTech 2015 Proceedings*, March 2015, paper 7.1, pp. 91–94.
20. J. Graf and A. A. Sohahngpurwala, "Private Verification for FPGA Bitstreams," in *GOMACTech 2017 Proceedings*, March 2017.
21. S. Harper, J. Graf, W. Batchelor, T. Dunham, and P. Athanas, "Introducing a Trust Metric Foundation and Deriving Trust-for-Buck," in *GOMACTech 2019 Proceedings*, March 2019.
22. J. Graf, "Optimizing Forward Design Trust for FPGAs," invited lecture delivered to *Single Event Effects Symposium / Military and Aerospace Programmable Logic Devices Workshop*, San Diego, CA, May 25, 2017.
23. J. Graf, "Measuring Trust," invited lecture delivered to *Single Event Effects Symposium / Military and Aerospace Programmable Logic Devices Workshop*, San Diego, CA, May 24, 2018.
24. J. Graf, "Optimal Trust Strategies," invited lecture delivered to the National Academy of Sciences for *National Academies Study on Secure and Reliable Microelectronics for AF Systems*, Washington, DC, June 19, 2018.
25. S. Drimer, "Security for Volatile FPGAs," University of Cambridge Computer Laboratory Technical Report UCAM-CL-TR-763 [Online]. Available: <https://www.cl.cam.ac.uk/techreports/UCAM-CL-TR-763.pdf>.
26. "Anti-tamper capabilities in fpga designs," Altera, Inc., White Paper, 2008. [Online]. Available: https://www.altera.com/content/dam/altera-www/global/en_US/pdfs/literature/wp/wp-01066-anti-tamper-capabilities-fpga.pdf
27. "Xilinx design security solutions," Xilinx, Inc., White Paper, 2016. Available: <http://www.xilinx.com/products/technology/design-security.html>

Bibliography

28. K. Xiao, D. Forte, Y. Jin, R. Karri, S. Bhunia, and M. Tehranipoor, "Hardware Trojans: Lessons Learned After One Decade of Research," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 22, no. 1, p. 6:1–6:23, 2016.
29. M. Tehranipoor and F. Koushanfar, "A Survey of Hardware Trojan Taxonomy and Detection," *IEEE Design Test of Computers*, vol. 27, no. 1, pp. 10–25, Jan. 2010.
30. M. Rostami, F. Koushanfar, and R. Karri, "A Primer on Hardware Security: Models, Methods, and Metrics," *Proceedings of the IEEE*, vol. 102, no. 8, pp. 1283–1295, Aug. 2014.
31. "Defense science board task force on high performance microchip supply," Office of the Under Secretary of Defense For Acquisition, Technology, and Logistics, Task Force Report, February 2005. [Online]. Available: <http://www.acq.osd.mil/dsb/reports/ADA435563.pdf>
32. "Technology for trusted circuits," Defense Advanced Research Projects Agency, SBIR Phase II, 2007. [Online]. Available: <https://www.sbir.gov/sbirsearch/detail/164119>
33. D. R. Collins, "Trust, a proposed plan for trusted integrated circuits," Defense Advanced Research Projects Agency, DTIC Report, 2006. [Online]. Available: <http://www.dtic.mil/cgi-bin/GetTRDoc?AD=ADA456459>
34. D. R. Collins, "Darpa "Trust in ICs" effort," Defense Advanced Research Projects Agency, DTIC Report, 2007. [Online]. Available: <http://www.dtic.mil/cgi-bin/GetTRDoc?AD=ADA503809>
35. "Trusted Integrated Circuits (TRUST)," Defense Advanced Research Projects Agency, DARPA BAA, 2007. [Online]. Available: <http://www.darpa.mil/program/trusted-integrated-circuits>
36. D. R. Collins, "Program for IEEE international workshop on hardware-oriented security and trust," 2008 Design Automation Conference, Conference Program, 2008. [Online]. Available: <http://www.engr.uconn.edu/HOST/HOST-2008.pdf>
37. H. Salmani, M. Tehranipoor, and R. Karri, "On Design vulnerability analysis and trust benchmark development", *IEEE Int. Conference on Computer Design (ICCD)*, 2013.

Bibliography

38. B. Shakya, T. He, H. Salmani, D. Forte, S. Bhunia, M. Tehranipoor, “Benchmarking of Hardware Trojans and Maliciously Affected Circuits”, *Journal of Hardware and Systems Security (HaSS)*, April 2017.
39. “Trusted foundry program,” Defense Microelectronics Activity, Tech. Rep., 2016. [Online]. Available: <http://www.dmea.osd.mil/trustedic.html>
40. K. Vaidyanathan, B. P. Das, E. Sumbul, R. Liu, and L. Pileggi, “Building trusted ICs using split fabrication,” in *Hardware-Oriented Security and Trust (HOST), 2014 IEEE International Symposium on*, May 2014, pp. 1–6.
41. C. McCants, “Trusted integrated chips (tic): Obtaining world-class performance without compromising security,” Intelligence Advanced Research Projects Agency, Tech. Rep., 2011. [Online]. Available: https://www.iarpa.gov/images/files/programs/tic/08-TIC_final.pdf
42. J. A. Roy, F. Koushanfar, and I. L. Markov, “Epic: Ending piracy of integrated circuits,” in *Design, Automation and Test in Europe, 2008. DATE '08*, March 2008, pp. 1069–1074.
43. R. S. Chakraborty and S. Bhunia, “Harpoon: An obfuscation-based SoC design methodology for hardware protection,” 43, vol. 28, no. 10, pp. 1493–1502, Oct 2009.
44. R. S. Chakraborty and S. Bhunia, “RTL hardware IP protection using key-based control and data flow obfuscation,” in *VLSI Design, 2010. VLSID '10. 23rd International Conference on*, Jan 2010, pp. 405–410.
45. L. Li and H. Zhou, “Structural transformation for best-possible obfuscation of sequential circuits,” in *Hardware-Oriented Security and Trust (HOST), 2013 IEEE International Symposium on*, June 2013, pp. 55–60.
46. J. Zhang, “A practical logic obfuscation technique for hardware security,” *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 24, no. 3, pp. 1193–1197, March 2016.
47. J. Rajendran, Y. Pino, O. Sinanoglu, and R. Karri, “Security analysis of logic obfuscation,” in *Design Automation Conference (DAC), 2012 49th ACM/EDAC/IEEE*, June 2012, pp. 83–89.

Bibliography

48. A. Kerckhoffs, "La cryptographie militaire," *Journal des Sciences Militaires*, vol. IX, pp. 5–38, Jan. 1883, pp. 161–191, Feb. 1883. https://www.petitcolas.net/kerckhoffs/crypto_militaire_2.pdf
49. B. Barak, O. Goldreich, R. Impagliazzo, S. Rudich, A. Sahai, S. Vadhan, and K. Yang, "On the (im)possibility of obfuscating programs," *Cryptology ePrint Archive, Report 2001/069*, 2001, <http://eprint.iacr.org/>.
50. K. Shamsi, D. Pan, and Y. Jin, "On the Impossibility of Approximation-Resilient Circuit Locking," in *Hardware-Oriented Security and Trust (HOST), 2013 IEEE International Symposium on*, May 2019
51. K. Xiao and M. Tehranipoor, "Bisa: Built-in self-authentication for preventing hardware trojan insertion," in *Hardware-Oriented Security and Trust (HOST), 2013 IEEE International Symposium on*, June 2013, pp. 45–50.
52. K. Xiao, D. Forte, and M. Tehranipoor, "A novel built-in self-authentication technique to prevent inserting hardware trojans," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 33, no. 12, pp. 1778–1791, Dec 2014.
53. T. Meade, S. Zhang, and Y. Jin, "Netlist reverse engineering for high-level functionality reconstruction," in *2016 21st Asia and South Pacific Design Automation Conference (ASP-DAC)*, Jan 2016, pp. 655–660.
54. W. Li, Z. Wasson, and S. A. Seshia, "Reverse engineering circuits using behavioral pattern mining," in *Hardware-Oriented Security and Trust (HOST), 2012 IEEE International Symposium on*, June 2012, pp. 83–88.
55. H. Bouchaour, M. Ouali, and Y. Lebbah, "Towards a method for VLSI circuit reverse engineering," in *Proceedings of the Third International Conference on Computer Science and its Applications (CIIA'11)*, December 2011.
56. P. Chakraborty, J. Cruz, and S. Bhunia, "SAIL: Machine Learning Guided Structural Analysis Attack on Hardware Obfuscation," in *Proc. 2018 Asian Hardware Oriented Security and Trust Symposium (AsianHOST)*, Dec. 2018, pp. 56–61.
57. R. Quijada, R. Dura, J. Pallares, et al., "Large-Area Automated Layout Extraction Methodology for Full-IC Reverse Engineering," *Journal of Hardware and Systems Security* (2018) 2: 322. <https://doi.org/10.1007/s41635-018-0051-4>

Bibliography

58. A. Waksman, M. Suozzo, and S. Sethumadhavan, “Fanci: Identification of stealthy malicious logic using boolean functional analysis,” in *Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security*, ser. CCS ’13. New York, NY, USA: ACM, 2013, pp. 697–708. [Online]. Available: <http://doi.acm.org/10.1145/2508859.2516654>
59. J. Zhang, F. Yuan, L. Wei, Y. Liu, and Q. Xu, “Veritrust: Verification for hardware trust,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 34, no. 7, pp. 1148–1161, July 2015.
60. J. Zhang, F. Yuan, and Q. Xu, “Detrust: Defeating hardware trust verification with stealthy implicitly-triggered hardware trojans,” in *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS ’14. New York, NY, USA: ACM, 2014, pp. 153–166. [Online]. Available: <http://doi.acm.org/10.1145/2660267.2660289>
61. J.-B. Note and E. Rannaud, “From the bitstream to the netlist,” in *Proceedings of the 16th International ACM/SIGDA Symposium on Field Programmable Gate Arrays*, ser. FPGA ’08. New York, NY, USA: ACM, 2008, pp. 264–264. [Online]. Available: <http://doi.acm.org/10.1145/1344671.1344729>
62. C. Patterson, “High performance DES encryption in Virtex FPGAs using JBits,” in *Field-Programmable Custom Computing Machines, 2000 IEEE Symposium on*, 2000, pp. 113–121.
63. C. Patterson and S. A. Guccione, “JBits design abstractions,” in *Field-Programmable Custom Computing Machines, 2001. FCCM ’01. The 9th Annual IEEE Symposium on*, March 2001, pp. 251–252.
64. N. Steiner and P. Athanas, “An Alternate Wire Database for Xilinx FPGAs,” in *Field Programmable Custom Computing Machines, 2004. FCCM 2004. 12th Annual IEEE Symposium on*, April 2004, pp. 336–337.
65. E. Bergeron, L.-D. Perron, M. Feeley, and J. P. David, “Logarithmic-time FPGA bitstream analysis: A step towards JIT hardware compilation,” *ACM Trans. Reconfigurable Technol. Syst.*, vol. 4, no. 2, pp. 12:1–12:27, May 2011. [Online]. Available: <http://doi.acm.org/10.1145/1968502.1968503>

Bibliography

66. F. Benz, A. Seffrin, and S. A. Huss, "Bil: A tool-chain for bitstream reverse engineering," in *Field Programmable Logic and Applications (FPL)*, 2012 22nd International Conference on, Aug 2012, pp. 735–738.
67. Z. Ding, Q. Wu, Y. Zhang, and L. Zhu, "Deriving an NCD file from an FPGA bitstream: Methodology, architecture and evaluation." *Microprocessors and Microsystems Embedded Hardware Design*, vol. 37, no. 3, pp. 299–312, 2013. [Online]. Available: <http://dblp.uni-trier.de/db/journals/mam/mam37.html#DingWZZ13>
68. P. Swierczynski and M. Fyrbiak, "FPGA Trojans through detecting and weakening of cryptographic primitives," Cryptology ePrint Archive, Report 2014/649, 2014, <https://eprint.iacr.org/2014/649.pdf>.
69. S. Trimberger, "Trusted design in FPGAs," in *2007 44th ACM/IEEE Design Automation Conference*, June 2007, pp. 5–8.
70. J. Graf and A. Sohaghpurwala, "Private Verification for FPGA Bitstreams," in *GOMACTech 2017 Proceedings*, March 2017.
71. A. Huber and J. Scott, "The Role and Nature of Anti-Tamper Techniques in US Defense Acquisition," in *Acquisition Review Quarterly*, Fall 1999. Online: <https://apps.dtic.mil/dtic/tr/fulltext/u2/b250068.pdf>
72. IBM, *IBM 4758 PCI Cryptographic Coprocessor*, <http://www.ibm.com/security/cryptocards/>
73. R. Anderson and M. Kuhn, "Tamper Resistance-a Cautionary Note," *Proceedings of the Second Usenix Workshop on Electronic Commerce*, pp. 1–11, November 1996.
74. R. Anderson and M. Kuhn, "Low Cost Attacks on Tamper Resistant Devices," *International Workshop on Security Protocols*, 1997.
75. A. Huang, *Hacking the Xbox: An Introduction to Reverse Engineering*, 2003. Available online: http://bunniefoo.com/nostarch/HackingTheXbox_Free.pdf
76. Xilinx, "Design Security: Security Throughout the Product Lifecycle," 2019. Online: <https://www.xilinx.com/products/technology/design-security.html>

Bibliography

77. Intel, “Anti-Tamper Capabilities in FPGA Designs,” 2008. <https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/wp/wp-01066-anti-tamper-capabilities-fpga.pdf>
78. Syphermedia library. [Online]. Available: http://www.smi.tv/syphermedia_library_circuit_camouflage_technology.html
79. R. Krueger, “Using high security features in Virtex-II series FPGAs,” Xilinx, CA, Tech. Rep. XAPP766(v.1.0), Jul. 2004. [Online]. Available: http://www.xilinx.com/support/documentation/application_notes/xapp766.pdf
80. A. Moradi, A. Barengi, T. Kasper, and C. Paar, “On the vulnerability of FPGA bitstream encryption against power analysis attacks: Extracting keys from Xilinx Virtex-II FPGAs,” in *Proceedings of the 18th ACM Conference on Computer and Communications Security, ser. CCS '11*. New York, NY, USA: ACM, 2011, pp. 111–124. [Online]. Available: <http://doi.acm.org/10.1145/2046707.2046722>
81. A. Moradi, D. Oswald, C. Paar, and P. Swierczynski, “Side-channel attacks on the bitstream encryption mechanism of altera Stratix II: Facilitating black-box analysis using software reverse-engineering,” in *Proceedings of the ACM/SIGDA International Symposium on Field Programmable Gate Arrays, ser. FPGA '13*. New York, NY, USA: ACM, 2013, pp. 91–100. [Online]. Available: <http://doi.acm.org/10.1145/2435264.2435282>
82. P. Swierczynski, A. Moradi, D. Oswald, and C. Paar, “Physical security evaluation of the bitstream encryption mechanism of Altera Stratix ii and Stratix iii FPGAs,” *ACM Trans. Reconfigurable Technol. Syst.*, vol. 7, no. 4, pp. 34:1–34:23, Dec. 2014. [Online]. Available: <http://doi.acm.org/10.1145/2629462>
83. DPA contest. (2016) [Online]. Available: <http://www.dpacontest.org/home/>
84. DPA workstation. (2016) [Online]. Available: <https://www.rambus.com/security/dpa-countermeasures/dpa-workstation-platform/>
85. Virtex UltraScale plus. (2016) [Online]. Available: <http://www.xilinx.com/products/silicon-devices/fpga/virtex-ultrascale-plus.html>
86. T. Lu, R. Kenny, S. Atsatt, “Secure Device Manager for Intel® Stratix® 10 Devices Provides FPGA and SoC Security,” 2015. Online:

Bibliography

- <https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/wp/wp-01252-secure-device-manager-for-fpga-soc-security.pdf>
87. Microsemi, UG0753 User Guide PolarFire FPGA Security, 2019. Online: https://www.microsemi.com/document-portal/doc_view/136534-ug0753-polarfire-fpga-security-user-guide
 88. B. Barak, O. Goldreich, R. Impagliazzo, S. Rudich, A. Sahai, S. Vadhan, and K. Yang, “On the (im)possibility of obfuscating programs,” Cryptology ePrint Archive, Report 2001/069, 2001, <http://eprint.iacr.org/>.
 89. T. Kirkland and M. R. Mercer, “Algorithms for automatic test-pattern generation,” *IEEE Design and Test of Computers*, vol. 5, no. 3, pp. 43–55, May 1988.
 90. S. Haider, C. Jin and M. van Dijk, “Hatch: A formal framework of hardware trojan design and detection,” Cryptology ePrint Archive, Report 2014/943, 2014, <https://eprint.iacr.org/2014/943.pdf>.
 91. A. Kimura and S. Bibyk, “Quantifying error payload and error implementation cost for hardware assurance,” in *GOMACTech 2016 Proceedings*, March 2016.
 92. S. Roy, C. Ellis, S. Shiva, D. Dasgupta, V. Shandilya, and Q. Wu, “A Survey of Game Theory as Applied to Network Security,” in *2010 43rd Hawaii International Conference on System Sciences*, 2010, pp. 1–10.
 93. C. A. Kamhoua, M. Rodriguez, and K. A. Kwiat, “Testing for Hardware Trojans: A Game-Theoretic Approach,” in *Decision and Game Theory for Security*, 2014, pp. 360–369.
 94. A. M. Smith, J. R. Mayo, V. Kammler, R. C. Armstrong, and Y. Vorobeychik, “Using computational game theory to guide verification and security in hardware designs,” in *2017 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, 2017, pp. 110–115.
 95. M. Galiardi *et al.*, “On Modeling Detection for Quantitative Trust Analysis,” in *GOMACTech*, 2018.
 96. W. Saad, A. Sanjab, Y. Wang, C. A. Kamhoua, and K. A. Kwiat, “Hardware Trojan Detection Game: A Prospect-Theoretic Approach,” *IEEE Transactions on Vehicular Technology*, vol. 66, no. 9, pp. 7697–7710, Sep. 2017.

Bibliography

97. Charles River Analytics, “How to Avoid Malice Using Linguistics-Inspired Exploit Testing,” 2018. [Online]. Available: <https://www.cra.com/work/case-studies/hamlet>.
98. C. A. Kamhoua, H. Zhao, M. Rodriguez, and K. A. Kwiat, “A Game-Theoretic Approach for Testing for Hardware Trojans,” *IEEE Transactions on Multi-Scale Computing Systems*, vol. 2, no. 3, pp. 199–210, Jul. 2016.
99. A. G. Kimura and S. B. Bibyk, “Quantifying error payload and error implementation cost for hardware assurance,” in *GOMACTech*, 2016.
100. R. Anderson and T. Moore, “The Economics of Information Security,” *Science*, vol. 314, no. 5799, pp. 610–613, Oct. 2006.
101. L. A. Gordon and M. P. Loeb, “The Economics of Information Security Investment,” *ACM Trans. Inf. Syst. Secur.*, vol. 5, no. 4, pp. 438–457, 2002.
102. Y. Shoham, “Computer Science and Game Theory,” *Communications of the ACM*, vol. 51, no. 8, pp. 74-79. August 2008. DOI: <https://doi.org/10.1145/1378704.1378721>
103. S.N. Durlauf and L.E. Blume, “Computer Science and Game Theory,” *Game Theory*, Springer New Palgrave Economics Collection, pp. 48-65, 2010.
104. J.Y. Halpern, “Computer Science and Game Theory: A Brief Survey,” 2007. Available online: <https://arxiv.org/abs/cs/0703148>
105. Moshe Tennenholtz, “Game Theory and Artificial Intelligence,” *UKMAS Workshop on Foundations and Applications of Multi-Agent Systems*, Springer-Verlag, pp 49-58, 2002.
106. E. Elkind and K. Leyton-Brown, “Algorithmic Game Theory and Artificial Intelligence,” *AI Magazine*, vol 31, no 3, Winter 2010.
107. V. Jalaparti, G. Nguyen, I. Gupta, and M. Caesar, “Cloud Resource Allocation Games,” University of Illinois Whitepaper, 2010. Available Online: <http://hdl.handle.net/2142/17427>
108. J. Yang, B. Jiang, Z. Lv, and K.R. Choo, “A task scheduling algorithm considering game theory designed for energy management in cloud computing,” in *Future Generation Computer Systems*, March 2017. Available Online: <https://doi.org/10.1016/j.future.2017.03.024>

Bibliography

109. A. Nezarat and G. Dastghaibifard, "Efficient Nash equilibrium resource allocation based on game theory mechanism in cloud computing by using auction," *International Conference on Next Generation Computing Technologies (NGCT)*, pp. 1-5, 2015.
110. Y. Wang, Y. Wang, J. Liu, Z. Huang and P. Xie, "A Survey of Game Theoretic Methods for Cyber Security," *2016 IEEE First International Conference on Data Science in Cyberspace (DSC)*, Changsha, pp. 631-636, 2016.
111. X. Liang and Y. Xiao, "Game Theory for Network Security," in *IEEE Communications Surveys & Tutorials*, vol. 15, no. 1, pp. 472-486, First Quarter 2013.
112. J. Jormakka and J.V.E. Molsa, "Modeling Information Warfare as a Game," *Journal of Information Warfare*, vol 4, no 2, pp 12-25, 2005.
113. M. H. Manshaei et al, "Game theory meets network security and privacy," *ACM Computing Surveys*, vol 45, no 3, Article 25, July 2013. DOI=<http://dx.doi.org/10.1145/2480741.2480742>Ya
114. W. He, C. Xia, H. Wang, C. Zhang and Y. Ji, "A Game Theoretical Attack-Defense Model Oriented to Network Security Risk Assessment," *2008 International Conference on Computer Science and Software Engineering*, Hubei, pp. 498-504, 2008.
115. D. P. Wilt, R. C. Meitzler, and J. DeVale, "Metrics for trust in integrated circuits," in *GOMACTech*, 2008.
116. DARPA, "Trusted Integrated Circuits (TRUST) (Archived)," 2006. [Online]. Available: <https://www.darpa.mil/program/trusted-integrated-circuits>.
117. D. R. Collins, "DARPA 'Trust in IC's' Effort," 2007. [Online]. Available: <https://apps.dtic.mil/dtic/tr/fulltext/u2/a503809.pdf>.
118. N. Tsagourias, "Cyber attacks, self-defence and the problem of attribution," *Journal of Conflict and Security Law*, vol. 17, no. 2, pp. 229–244, 2012.
119. USDOT, "Revised Departmental Guidance on Valuation of a Statistical Life in Economic Analysis," 2016. [Online]. Available: <https://www.transportation.gov/office-policy/transportation-policy/revised-departmental-guidance-on-valuation-of-a-statistical-life-in-economic-analysis>.

Bibliography

120. USEPA, “Mortality Risk Valuation,” 2018. [Online]. Available: <https://www.epa.gov/environmental-economics/mortality-risk-valuation>.
121. R. D. McKelvey and T. R. Palfrey, “Quantal Response Equilibria for Normal Form Games,” *Games and Economic Behavior*, vol. 10, no. 1, pp. 6–38, Jul. 1995.
122. R. Selten, “Reexamination of the perfectness concept for equilibrium points in extensive games,” *International Journal of Game Theory*, vol. 4, no. 1, pp. 25–55, 1975.
123. H. R. Varian, “Revealed Preference.” [Online]. Available: <http://people.ischool.berkeley.edu/~hal/Papers/2005/revpref.pdf>.
124. P. Honner, “Why Winning in Rock-Paper-Scissors (and in Life) Isn’t Everything,” 2018. [Online]. Available: <https://www.quantamagazine.org/the-game-theory-math-behind-rock-paper-scissors-20180402/>.
125. J. Ondráček, “Extending game-theoretic models to account for subrational adaptive behavior,” Czech Technical University in Prague, 2018.
126. D. Kar, F. Fang, F. Delle Fave, N. Sintov, and M. Tambe, “‘A Game of Thrones’: When Human Behavior Models Compete in Repeated Stackelberg Security Games,” in *Proceedings of the 2015 International Conference on Autonomous Agents and Multiagent Systems*, Istanbul, Turkey, 2015, pp. 1381–1390.
127. B. Johnson, A. Laszka, J. Grossklags, M. Vasek, and T. Moore, “Game-Theoretic Analysis of DDoS Attacks Against Bitcoin Mining Pools,” in *Financial Cryptography and Data Security*, Berlin, Heidelberg, 2014, pp. 72–86.
128. Digilent, “Arty Z7: APSoC Zynq-7000 Development Board for Makers and Hobbyists,” 2018. [Online]. Available: <https://store.digilentinc.com/artzy-z7-apsoc-zynq-7000-development-board-for-makers-and-hobbyists/>.
129. Digilent, “NetFPGA-SUME Virtex-7 FPGA Development Board,” 2018. [Online]. Available: <https://store.digilentinc.com/netfpga-sume-virtex-7-fpga-development-board/>.
130. 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>.

Bibliography

131. W. C. J. A. A. C. A. R. LaFleur, “2017 Government Contractor Survey,” 2017. [Online]. Available: <https://www.grantthornton.com/-/media/content-page-files/public-sector/pdfs/surveys/2018/2017-government-contractor-survey>.
132. Amazon.com, “Amazon EC2 Pricing,” 2018. [Online]. Available: <https://aws.amazon.com/ec2/pricing/on-demand/>.
133. H. Salmani, M. Tehranipour, and R. Karri, “On design vulnerability analysis and trust benchmarks development,” in *2013 IEEE 31st International Conference on Computer Design (ICCD)*, 2013, pp. 471–474.
134. B. Shakya, T. He, H. Salmani, D. Forte, S. Bhunia, and M. Tehranipour, “Benchmarking of Hardware Trojans and Maliciously Affected Circuits,” *Journal of Hardware and Systems Security*, vol. 1, no. 1, pp. 85–102, Mar. 2017.
135. GAO, “Weapon System Cybersecurity: DOD Just Beginning to Grapple with Scale of Vulnerabilities,” 2018. [Online]. Available: <https://www.gao.gov/products/GAO-19-128>.
136. S. M. Slayback, “A Computer Scientist’s Evaluation of Publically Available Hardware Trojan Benchmarks’,” Naval Postgraduate School, 2015.
137. DARPA, “CEP - Common Evaluation Platform (v1.2),” 2018. [Online]. Available: <https://github.com/mit-ll/CEP>.
138. A. Tang, S. Sethumadhavan, and S. Stolfo, “CLKSCREW: Exposing the Perils of Security-Oblivious Energy Management,” in *26th USENIX Security Symposium*, 2017.
139. A. Tang, S. Sethumadhavan, and S. Stolfo, “Motivating Security-Aware Energy Management,” *IEEE Micro*, vol. 38, no. 3, pp. 98–106, 2018.
140. H. Salmani, “COTD: Reference-Free Hardware Trojan Detection and Recovery Based on Controllability and Observability in Gate-Level Netlist,” *Trans. Info. For. Sec.*, vol. 12, no. 2, pp. 338–350, 2017.
141. R. Marlow, S. Harper, W. Batchelor, and J. Graf, “Hardware Trojan Detection using Xilinx Vivado,” in *NAECON 2018 - IEEE National Aerospace and Electronics Conference*, 2018, pp. 86–91.

Bibliography

142. S. C. Seth, V. D. Agrawal, and H. Farhat, “A theory of testability with application to fault coverage analysis,” in *[1989] Proceedings of the 1st European Test Conference*, 1989, pp. 139–143.
143. F. Pedregosa *et al.*, “Scikit-learn: Machine Learning in Python,” *J. Mach. Learn. Res.*, vol. 12, pp. 2825–2830, 2011.
144. A. Dabrowski, P. Fejes, J. Ullrich, K. Krombholz, H. Hobel, and E. Weippl, “Poster: Hardware Trojans – Detect and React?,” in *Network and Distributed System Security (NDSS) Symposium*, 2014.
145. A. Dabrowski, H. Hobel, J. Ullrich, K. Krombholz, and E. Weippl, “Towards a Hardware Trojan Detection Cycle,” in *2014 Ninth International Conference on Availability, Reliability and Security*, 2014, pp. 287–294.
146. “Jenkins: Build great things at any scale,” 2018. [Online]. Available: <https://jenkins.io/>.
147. D. Avis and C. Jordan, “Comparative computational results for some vertex and facet enumeration codes,” *CoRR*, vol. abs/1510.02545, 2015.
148. SageMath, [Online]. Available: <http://www.sagemath.org/>
149. R. D. McKelvey, A. M. McLennan, and T. L. Turocy, “Gambit: Software Tools for Game Theory, Version 16.0.1,” 2016. [Online]. Available: <http://www.gambit-project.org>.
150. Gambit Project, “For contributors: Ideas and suggestions for Gambit-related projects,” accessed April 22, 2019. <http://www.gambit-project.org/gambit13/ideas.html>
151. G. D. Rosenberg, “Enumeration of All Extreme Equilibria of Bimatrix Games with Integer Pivoting and Improved Degeneracy Check,” London School of Economics and Political Science, 2005.
152. D. Avis, “lrs home page,” 2018. [Online]. Available: <http://cgm.cs.mcgill.ca/ avis/C/lrs.html>.
153. D. Avis, “LRS: A Revised Implementation of the Reverse Search Vertex Enumeration Algorithm,” 1999. [Online]. Available: <http://cgm.cs.mcgill.ca/ avis/doc/avis/Av98a.pdf>.

Bibliography

154. R. S. Chakraborty and S. Bhunia, "HARPOON: An Obfuscation-Based SoC Design Methodology for Hardware Protection," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 28, no. 10, pp. 1493–1502, 2009.
155. R. S. Chakraborty and S. Bhunia, "RTL Hardware IP Protection Using Key-Based Control and Data Flow Obfuscation," in *2010 23rd International Conference on VLSI Design*, 2010, pp. 405–410.
156. R. S. Chakraborty and S. J. Bhunia, "Security Against Hardware Trojan Attacks Using Key-Based Design Obfuscation," *Journal of Electronic Testing*, vol. 27, no. 767, 2011.
157. J. Rajendran, Y. Pino, O. Sinanoglu, and R. Karri, "Security analysis of logic obfuscation," in *DAC Design Automation Conference 2012*, 2012, pp. 83–89.
158. L. Li and H. Zhou, "Structural transformation for best-possible obfuscation of sequential circuits," in *2013 IEEE International Symposium on Hardware-Oriented Security and Trust (HOST)*, 2013, pp. 55–60.
159. K. Vaidyanathan, B. P. Das, E. Sumbul, R. Liu, and L. Pileggi, "Building trusted ICs using split fabrication," in *2014 IEEE International Symposium on Hardware-Oriented Security and Trust (HOST)*, 2014, pp. 1–6.
160. S. Garg, C. Gentry, S. Halevi, M. Raykova, A. Sahai, and B. Waters, "Candidate indistinguishability obfuscation and functional encryption for all circuits," *SIAM Journal on Computing*, vol. 45, no. 3, pp. 882–929, 2016.
161. J. Zhang, "A Practical Logic Obfuscation Technique for Hardware Security," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 24, no. 3, pp. 1193–1197, 2016.
162. J. Tian, G. R. Reddy, J. Wang, W. Swartz, Y. Makris, and C. Sechen, "A field programmable transistor array featuring single-cycle partial/full dynamic reconfiguration," in *Design, Automation Test in Europe Conference Exhibition (DATE), 2017*, 2017, pp. 1336–1341.
163. Q. Yu, J. Dofe, Z. Zhang, and S. Kramer, "Hardware Obfuscation Methods for Hardware Trojan Prevention and Detection," in *The Hardware Trojan War*, B. S. and T. M., Eds. Springer, 2018, pp. 291–325.

Bibliography

164. "Trusted and Assured MicroElectronics Forum," 2018. [Online]. Available: <https://tameforum.org/>.
165. V. Jyothi, P. Krishnamurthy, F. Khorrami, and R. Karri, "TAINT: Tool for Automated INsertion of Trojans," in *2017 IEEE International Conference on Computer Design (ICCD)*, 2017, pp. 545-548.
166. J. Cruz, Y. Huang, P. Mishra, and S. Bhunia, "An Automated Configurable Trojan Insertion Framework for Dynamic Trust Benchmarks", in *23rd Design Automation and Test in Europe (DATE) Conference*, Dresden, Germany, March 19 - 23, 2018.
167. S. Farashahi, H. Azab, B. Hayden, and A. Soltani, "On the Flexibility of Basic Risk Attitudes in Monkeys," *Journal of Neuroscience*, vol. 38, no. 18, pp. 4383–4398, 2018.
168. M. Edwards, "Trusted Silicon Stratus (TSS) Workshop," 2010. [Online]. Available: <https://apps.dtic.mil/dtic/tr/fulltext/u2/a540791.pdf>.
169. Nimbis, "Trusted Silicon Stratus: Joint Federated Assurance Center Distributed Transition Environment." [Online]. Available: <https://www.trustedstratus.org/>.
170. D. Prelec, "The Probability Weighting Function," *Econometrica*, vol. 66, no. 3, pp. 497–527, 1998.
171. V. Rao and N. Price, "Hardware Trojan Horse Detection (Poster)," in *Military and Aerospace Programmable Logic Devices (MAPLD) Workshop*, 2018.
172. B. Schneier, "The Psychology of Security," Online: https://www.schneier.com/essays/archives/2008/01/the_psychology_of_se.html