On the Complexity of the Circuit Obfuscation Problem for Split Manufacturing

by

Mohamed El Massad

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Applied Science
in
Electrical and Computer Engineering

Waterloo, Ontario, Canada, 2013

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

Recent work in the area of computer hardware security introduced a number of interesting computational problems in the context of directed acyclic graphs (DAGs). In this thesis, we pick one of these problems, circuit obfuscation — a combinatorial optimization problem — and study its computational complexity. First, we prove that the problem is **NP**-hard. Next, we show it to be in the class of $\mathbf{MAX-SNP}$ optimization problems, which means it is inapproximable within a certain constant (2.08) unless $\mathbf{P} = \mathbf{NP}$. We then use a reduction from the maximum common edge subgraph problem to prove a lower bound on the absolute error guarantee achievable for the problem by a polynomial-time algorithm. Given that the decision version of the problem is in **NP**, we investigate the possibility of efficiently solving the problem using a SAT solver and report on our results. Finally, we study a slightly modified version of the problem underlying a generalized hardware security technique and prove it to be **NP**-hard as well.

Acknowledgements

I would like to thank my supervisors, Dr. Mahesh Tripunitara and Dr. Siddharth Garg for their help and direction that made this thesis possible. I would also like to thank my colleague Frank Imeson for his help and assistance during the early stages of the project.

Dedication

To my parents, Nawal El Hussein and Bashir El Massad, with love, respect and a word of thanks they really deserve.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Hardware security is concerned with the protection of integrated circuits (ICs) from malicious attempts to change their functionality. The threat is usually in the form of an entity that is able to insert additional circuitry into the IC in either the design or the fabrication phase, and can therefore alter the circuit functionality in a malicious manner. The objectives of such an attacker can be wide-ranging; breaking encryption schemes implemented in hardware, to hijacking of satellites, weapons, and other crucial systems involving electronic circuits. The threat is made all the more imminent by the fact that IC design is nowadays done at sites scattered across the world, each with their own, potentially malicious interests. Further, IC manufacturing is typically done in fabrication facilities external to IC vendors, which makes it even more easier for an attacker in a fabrication facility to insert malicious circuits into the IC without the vendor knowing. The security literature abounds with techniques to combat such attacks on ICs, including modifications and additions to the design of the circuit itself, to post-fabrication testing and validation to detect and disable malicious circuits.

In a recent work [10], Imeson et al. propose a way to use an emerging IC manufacturing technology, known as 3D integration, to increase the security of ICs. In a 3D IC, two or more layers or *tiers* of active electronic components are integrated both horizontally and vertically into a single circuit. Although a few other ways exist to build 3D ICs, [10] consider a specific manufacturing technology in which the design is split into two tiers that are independently manufactured and then stacked on top of each other. The bottom tier in the technology consists of logic gates and metal wires used to interconnect the gates, whereas the top tier consists solely of metal wires that provide additional connections between gates on the bottom tier. The two tiers are interconnected together using vertical metal pillars to create the final 3D IC chip that is shipped to the vendor.

[10] discusses how manufacturing ICs in this way can potentially enhance hardware security. The premise is that each tier is fabricated in a separate facility. The bottom tier, referred to as the untrusted tier, is expensive to fabricate since it implements active transistor devices and passive metal, and is therefore sent to an external, untrusted facility for fabrication. The top tier, implementing only passive metal, can be manufactured at a trusted (possibly local) fabrication facility at a lower cost, and is hence referred to as the trusted tier. By distributing the design across two fabrication facilities, each facility now has an incomplete view of the circuit that the designer intends to fabricate, which reduces the ability of an attacker at any one foundry to compromise the security of the circuit by altering its functionality. A precise mathematical notion is provided by [10] that captures this increase in security level we get by using 3D manufacturing. The security measure involves the satisfying of certain structural properties by mathematical representations of the bottom tier and original circuits.

Achieving a certain security level in this way is shown by [10] to involve a number of computational problems in the context of graphs, which are objects of study in discrete mathematics that model pairwise relations between entities. Their work, however, leaves some questions open as to the computability and tractability properties of these problems, and whether methods exist that can be used to solve them efficiently. In this work, we study one such problem, a combinatorial optimization problem which they call circuit obfuscation. We analyze the computability properties of the problem and investigate some approaches to develop efficient solvers for it. This thesis reports our results.

The document is organized as follows. Chapter 2 gives a brief review of mathematical and computer science concepts that are necessary to understand our work. This includes concepts from graph theory and computational complexity theory. Chapter 3 gives a formal definition of the circuit obfuscation problem and outlines how it arises in the context of IC security and 3D manufacturing. Chapter 4 states the research questions that the thesis tackles, which are related to the computability properties of the obfuscation problem. Chapters 5 through 9 describe our answers to the research questions in Chapter 4. Chapter 10 concludes the thesis.

Chapter 2

A Brief Review of Graphs and Complexity Theory

In this chapter, we give a brief review of mathematical and computer science concepts necessary to to understand our work. We start by defining what a graph is and explain the structural property of isomorphism. We follow that by a quick introduction to decision problems and complexity classes. We also explain the concept of a reduction and the complexity classes that are defined based on it. An explanation of heuristics and approximation algorithms follows. We end the chapter by defining two computational problems that we use in our work, namely the boolean and circuit satisfiability problems.

2.1 Graphs and Isomorphism

A graph is a mathematical way to represent a set of objects where some pairs of objects are connected by links. A canonical example of such a set is guests at a party where two guests are connected by a link if they know each other. The interconnected objects in a graph are represented by *vertices* and the links that connect pairs of vertices are represented by *edges*. This is usually depicted in a diagram as a set of dots for the vertices, joined by lines or curves for the edges. An example is shown in Figure 2.1. We denote the set of vertices of a graph G as $V[G]$, and the set of edges as $E[G]$.

The edges in a graph may be directed or indirected. In an undirected graph the links represented by edges are symmetric, while in a directed graph they are not. An example of an undirected graph is the set of people at a party where the edges represent acquaintance

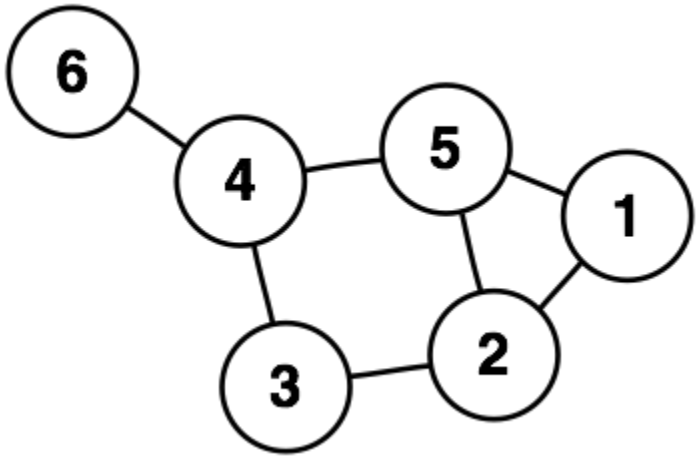

Figure 2.1: A drawing of a graph on 6 vertices and 7 edges. Credit: user:AzaToth / Wikimedia Commons / Public Domain.

relationships. An example of a directed graph is one where vertices represent gates in a combinational logic circuit and edges represent the wires. Here, the fact that the output of a gate is connected to the input of another does not imply the reverse, and should not, in fact.

An edge in a directed graph can be represented as an ordered pair (u, v) where u is the head vertex of the edge and v is the tail vertex. Two vertices in a graph are said to be adjacent if they are connected by an edge (directed or undirected). A directed acyclic graph (DAG) is a directed graph formed such that there is no way to start at some vertex v and follow a sequence of edges that eventually loops back to v again.

The vertices in a graph may be assigned colors to denote some attributes of the objects they represent. In the combinational logic circuit example, for instance, vertices may be assigned colors to distinguish types of gates (e.g., AND and OR) from one another. In this thesis, we sometimes express graphs as pairs $\langle V, E\rangle$ or tuples $\langle V, E, c\rangle$ where V is the set of vertices, E is the set of edges, and c is a function from V to $\mathbb{N}$ that maps each vertex to a natural number that denotes its color.

An **isomorphism** between two graphs is a bijective mapping between the vertices of the two graphs that preserves the edge relations. Formally, given two graphs $G_1 = \langle V_1, E_1, c_1\rangle, G_2 = \langle V_2, E_2, c_2\rangle$, we say that G_1 is isomorphic to G_2 if there exists a bijective mapping $\phi\colon V_1 \to V_2$ such that $\langle u, v\rangle \in E_1$ if and only if $\langle \phi(u), \phi(v)\rangle \in E_2$ and $c_1(u) =$

$c_2(\phi(u)), c_1(v) = c_2(\phi(v))$. In other words, renaming the vertices in G_1 according to ϕ gives us G_2. An **automorphism** is an isomorphism from a graph to itself.

A **subgraph** of a graph is a graph we obtain by removing some edges and/or vertices from the original graph. Formally, we say that $G_1 = \langle V_1, E_1, c_1 \rangle$ is a subgraph of $G_2 = \langle V_2, E_2, c_2 \rangle$ if $V_1 \subseteq V_2$, and $\langle u, v \rangle \in E_1$ only if $\langle u, v \rangle \in E_2$. A subgraph H is said to be a spanning subgraph of a graph G if it has the same vertex set as G. A subgraph H of a graph G is said to be induced if, for any pair of vertices x and y of H, (x, y) is an edge of H if and only if (x, y) is an edge of G. If a subgraph of G is isomorphic to H, we say that G is **subgraph isomorphic** to H. The corresponding mapping is called a **subgraph isomorphism**. Given two graphs G and H, the computational problem of deciding whether H is subgraph isomorphic to G is known as the subgraph isomorphism problem.

Finally, we define the disjoint union of graphs as follows: Given two graphs $G_1 = \langle V_1, E_1 \rangle$, $G_2 = \langle V_2, E_2 \rangle$ where V_1 and V_2 are disjoint (and hence $E1$ and $E2$ as well), the disjoint union of the two graphs is the graph $U = \langle V_1 \cup V_2, E_1 \cup E_2 \rangle$.

2.2 Decision Problems and Complexity Classes

A decision problem is a question in some formal system with a yes/no answer, depending on the values of some input parameters. An example is the problem "given an integer n, is it prime?". The answer can be either 'yes' or 'no' depending on the value of n. Decision problems usually appear in mathematical questions of decidability, that is, the question of the existence of a procedure to determine the existence of an object or its membership in a set. As it turns out, most of the important problems in mathematics are undecidable.

In the field of complexity theory, decision problems are categorized by how difficult they are to solve, where "difficult" is described in terms of the computational resources needed by the most efficient algorithm for the given problem. A set of problems of related resource-based complexity is referred to as a **complexity class**. For example, the set of problems that can be solved by an abstract computer in time quadratic in the input size constitutes a complexity class.

NP is one of the most fundamental complexity classes. The abbreviation **NP** refers to "nondeterministic polynomial time". Informally, **NP** is the set of all decision problems for which instances where the answer is "yes" have efficiently verifiable proofs or *certificates* of the fact that the answer is indeed "yes". Strictly Speaking, the proofs have to be verifiable

in polynomial time by what is referred to as a *deterministic Turing machine*, a hypothetical device that manipulates symbols on a strip of tape according to a table of rules.

Another fundamental complexity class is **P** also known as, **PTIME**, or **DTIME**$(n^{O(1)})$. This class contains decision problems that can themselves be solved using a polynomial amount of computation time by the deterministic Turing machine. Cobham's thesis [7], holds that problems in **P** are "efficiently solvable" or "tractable", that any problem which cannot be contained in **P** is not feasible, and that if a real-world problem can be solved by an algorithm existing in **P**, the algorithm will eventually be discovered. The reasoning for the thesis is the relatively mild rate of change in the running time of a polynomial time algorithm, as compared to an exponential time algorithm, where even for very small inputs the running time can be prohibitively large. Although the theory is not necessarily true in practice – some problems not known to be in **P** have practical solutions, and some that are in **P** do not – it is widely believed to be a good rule-of-thumb for real-life problems.

By definition, the class **P** is contained in **NP**. However, **NP** contains some problems whose solutions are sufficient to deal with any other **NP** problem in polynomial time. The hardest of these problems are called **NP**-complete problems, which we define in the following section. An open question in complexity theory asks whether polynomial time algorithms actually exist for **NP**-complete problems, and by extension, all **NP** problems. This is known as the **P** = **NP** problem and the answer for the question is widely believed to be no, that is, there are problems in **NP** that are harder to compute than to verify; they cannot be solved in polynomial time but the answer can be verified in polynomial time.

2.3 Polynomial Time Reductions, NP-hardness, and NP-completeness

A polynomial time reduction is a method of solving one problem by using a hypothetical subroutine for solving a different problem, which uses polynomial time excluding the time within the subroutine. The hypothetical routine is more commonly referred to as an *oracle*. A polynomial time reduction between two problems proves that the first problem is no more difficult than the second one, as a polynomial-time algorithm for the second problem can be used to efficiently solve the first problem. Depending on the details of how the subroutine for the second problem is used, there can be several different types of polynomial-time reductions, the most common being *polynomial-time many-to-one reductions* and *Turing reductions*.

A many-to-one reduction transforms inputs to some decision problem A, to inputs

to another decision problem B such that the transformed instances have the same answer (yes/no) as the original instances. An instance of problem A can be solved by applying this transformation to produce an instance of problem B, providing the latter to an algorithm for problem B and returning its output. Polynomial-time many-to-one reductions are also known as Karp reductions, and are usually denoted by the expression $A \leq_m^P B$, indicating problem A is polynomial-time many-to-one reducible to problem B. If the Karp reduction yields a way to efficiently transform a certificate for B to a certificate for A and vice versa, then the reduction is called a *Levin Reduction*[3].

A polynomial-time Turing reduction, on the other hand, solves problem A — which does not have to be a decision problem — using a polynomial number of calls to a subroutine for problem B — which does not have to be a decision problem either — and polynomial time outside these calls. Polynomial-time Turing reductions are also known as Cook reductions, and are usually denoted by the expression $A \leq_T^P B$, indicating problem A is polynomial-time Turing reducible to problem B.

The class **NP**-hard is defined as the class of problems (not necessarily decision) that are at least as hard as the hardest problems in **NP**. The hardest problems in **NP** are in turn referred to as **NP**-complete problems. Formally, a problem H is **NP**-hard if and only if there is an **NP**-complete problem L that is polynomial time Turing-reducible to H (that is, $L \leq_T^P H$). Following the definition of polynomial time Turing reductions, this means that L can be solved in polynomial time by an oracle machine with an oracle for H, meaning H is at least as hard as L. Since L is **NP**-complete, and hence the hardest in class **NP**, problem H is also at least as hard as any problem in **NP**.

As **NP**-hard is at least as hard as **NP**, a polynomial algorithm for any **NP**-hard problem implies there are polynomial algorithms for all problems in **NP**, and hence that $\mathbf{P} = \mathbf{NP}$. As a result of this, showing a decision problem to be **NP**-hard is a generally accepted way to prove it is intractable[11].

2.4 NP Optimization Problems

An optimization problem is the problem of finding the best solution from all feasible solutions. If the variables in an optimization problem are discrete, the problem is called a combinatorial optimization problem. Formally, a combinatorial optimization problem must consist of the following:

- a set of instances I;

- given an instance $x \in I$, a function $f(x)$ that determines the set of feasible solutions for x;
- given an instance x and a feasible solution y of x, another function $m(x, y)$ that determines the measure of y, which is usually a positive number.
- a goal function g, which is either min or max

The goal is to find for some instance x an *optimal solution*, which is a feasible solution y that satisfies

$$m(x, y) = g\{m(x, y') \mid y' \in f(x)\}$$

Each combinatorial optimization problem has a corresponding decision problem that asks whether there is a feasible solution with some particular measure m_0. As an example, the problem "Given a graph G and two vertices in it u and v, find a path from u to v that uses the fewest edges" is an optimization problem. A corresponding decision problem would be "is there a path from u to v that uses 10 or fewer edges?" This is a decision problem the answer for which can be either 'yes' or 'no'.

The decision problem is always polynomial-time Turing reducible to the optimization problem. In the example above, the decision problem can be solved using a single call to a subroutine for the optimization problem (If the answer of the optimization problem is less than or equal to 10, then the answer to the decision problem will be 'yes'; otherwise the answer will be 'no'). Thus, the optimization problem is always at least as hard as the decision problem and if the latter is proven to be **NP**-complete, then the former is proven to be **NP**-hard.

The definition for **NP**-optimization problems differs slightly from that of **NP** problems. A combinatorial optimization problem is said to be an **NP**-optimization problem or an **NPO** if it satisfies the following conditions.

- the size of every feasible solution $y \in f(x)$ is polynomially bounded in the size of the given instance x,
- The set of instances I and the set of feasible solutions $f(x)$ for a given instance x can be recognized in polynomial time, that is, it is decidable in polynomial time whether, for any x and for any y such that $|y| \leq p(|x|)$ where p is a polynomial, $y \in sol(x)$ and

- m is polynomial-time computable.

Note that the second condition implies the corresponding decision problem is in **NP**.

2.5 Heuristics, Approximation Algorithms and Approximation Classes

A heuristic is a method or algorithm that solves an optimization problem more quickly (compared to classic methods which would be too slow), but that is not certain to arrive at an optimal solution. The objective of a heuristic is usually to produce quickly enough a solution that is good enough for solving the problem at hand, typically when speed is more important than solution accuracy. Results about **NP**-hardardness often make heuristics the only viable option for a variety of complex optimization problems that need to be routinely solved in real-world applications. Deciding whether the solution found by a heuristic is good enough may be difficult, however, depending on the theory underlying that heuristic.

An approximation algorithm is also used to find approximate solutions to optimization problems, for which finding an exact solution is hard. Unlike heuristics, however, which usually only find reasonably good solutions reasonably fast, the solution quality and run-time bounds for an approximation algorithm are important. The run-time of an algorithm is usually restricted to be polynomial, while solution quality is specified in terms of an approximation ratio, which gives the largest value the ratio of the approximate solution to the optimal solution — or the inverse, for a minimization problem — can take. A combinatorial optimization problem is said to be approximable within ratio ρ, if a polynomial-time algorithm exists for solving the problem which has an approximation ratio of ρ.

An ideal situation is for an optimization problem to be approximable within every constant greater than 1 (with a different algorithm for every constant). A problem of this type is said to have a polynomial-time approximation scheme (or a **PTAS**) or to be in **PTAS**. Unfortunately, not all optimization problems have **PTAS**'s. In fact, some problems are not approximable within any constant or even polynomial factor unless $\mathbf{P} = \mathbf{NP}$.

This "hardness of approximation" notion is captured through a special type of reduction (referred to as a gap-preserving reduction) which we define in Chapter 7. Also, depending on what approximation ratios are hard to achieve for a specific optimization problem,

the problem is placed into one of several classes[4]. The class of interest to us here is $\mathbf{MAX-SNP}$, defined in [4] as the class of problems that are hard to approximate within a *certain* constant factor (hence not in **PTAS**). An example of a problem in this class is the minimum vertex cover problem, also defined in Chapter 7, and used to prove that the problem we study is also in $\mathbf{MAX-SNP}$.

Finally, an approximation algorithm is said to have an *absolute performance guarantee* c, if it has been proven for every instance x that

$$(\mathrm{OPT} - c) \leq f(x) \leq (\mathrm{OPT} + c).$$

where OPT is the measure of the optimum solution for x and $f(x)$ is the measure of the the approximate solution to x returned by the algorithm. The *performance guarantee*, $R(x, y)$, of a solution y to an instance x is similarly defined as

$$R(x, y) = \max\left(\frac{OPT}{m(y)}, \frac{m(y)}{OPT}\right),$$

where $m(y)$ is the measure of the solution y for the instance x.

2.6 The Boolean Satisfiability Problem, Circuit Satisfiability Problem and SAT Solvers

The boolean satisfiability problem (written or abbreviated SAT) is one of the most fundamental decision problems in complexity theory. Simply stated, given a boolean formula, it asks for whether there exists an assignment for the variables involved that would make the formula evaluate to TRUE. As an example, for the formula "a AND NOT b" the answer would be 'yes' (assign TRUE to a and FALSE to b), while for the formula "a AND NOT a" the answer would be 'no' (as the formula is identically FALSE). SAT was one of the first problems to be proven **NP**-complete by Richard Karp.

The circuit satisfiability problem (known as CSAT) is somewhat similar to SAT. Given a boolean combinational circuit, the question is whether there exists an assignment to the inputs of the circuit that would make the circuit output 1. Since any SAT instance can be polynomial-time Turing reduced to an instance of CSAT, CSAT is also **NP**-complete. A CSAT instance can also be reduced to a SAT instance using a reduction we describe in Chapter 6.

Despite being **NP**-complete, and therefore impossible to solve efficiently unless $\mathbf{P} = \mathbf{NP}$, a number of efficient and scalable algorithms for SAT were developed over the last

decade that solve a large enough subset of SAT instances. The algorithms have come to be referred to as *SAT solvers* and have been used in various practical areas in various fields by solving SAT instances made by transforming problems that arise in those areas. Extending the capabilities of SAT solvers is also an ongoing area of progress and a growing number of research and development groups at universities, research labs, and companies have started using SAT algorithms for solving different and decision and optimization problems[2].

In [10], the authors use a SAT solver as part of their approach to the circuit obfuscation problem, where they reduce the subgraph isomorphism problem to a SAT instance which they then feed to the SAT solver. In Chapter 6, we report the negative results of our efforts to develop an efficient SAT-based solver for our problem, and analyze why it does not work in this particular case.

It should be noted that miniSAT, the SAT solver we use, only accepts formulas that are in conjunctive normal form or CNF, which simply means the formula is a conjunction of clauses that are each a disjunction of literals. In other words, the formula is and AND of ORs (e.g., $\neg A \wedge (B \vee C)$).

Chapter 3

The Circuit Obfuscation Problem

In this chapter, we give a formal definition of the computational problem we study in this thesis. We also provide a brief background on how the problem arises in the context of hardware security and the 3D manufacturing process described in Chapter 1. We conclude the chapter with a brief summary of the results [10] obtained for the problem.

3.1 Definition of the Problem

Circuit obfuscation, introduced in [10], is defined as follows.

Circuit Obfuscation Given a colored DAG G representing some boolean combinational circuit and an integer $k \in (1, |V[G]|)$, the problem is to identify a set of edges $E' \subseteq |E[G]|$ with as few edges as possible such that the following property is satisfied for every vertex u in G:

If H is the graph we get by deleting the edges in E' from G, then there exist k distinct vertices $v_1, \ldots, v_k$ in G (and therefore in H), and mappings $\phi_1, \ldots, \phi_k$ from $V[G]$ to $V[H]$ such that every ϕ_i is a subgraph isomorphism from G to H, and for all $i \in [1, k]$, $\phi_i(u) = v_i$. A vertex that satisfies this property is said to be k-secure and the pair $\langle G, E' \rangle$ is said be k-secure. Note that no matter what edges are removed from G, at least one ϕ exists for each vertex that maps it to itself, that is, regardless of what edges are deleted from G, every vertex will still be 1-secure.

k is denoted as the *target security level*. The *actual* security level of the a solution is defined as the largest k such that $\langle G, E' \rangle$ is k-secure. The target and actual security levels of a solution are not necessarily the same since some vertices may be more than k-secure.

3.2 Context

If we think of the graph in question as a combinational logic circuit, where vertices represent logic gates, edges represent wires, and the color of vertex represents the type of the corresponding boolean function (e.g., AND, OR, or NOT), then the problem above essentially asks for a set of wires to be removed from the circuit, so that each each gate becomes indistinguishable from at least $k-1$ other gates. To see this, we note, as in [10], that the k-security property is a special case of the subgraph isomorphism problem. Whereas in the subgraph isomorphism problem we are given two graphs A and B and asked whether B is subgraph isomorphic to A, here, the two graphs restricted to be DAGs, and H is restricted to be a spanning subgraph of G. We know that H is subgraph isomorphic to G; the identity mapping from a vertex to itself serving as a certificate. But we require the existence of $k-1$ subgraph isomorphisms that are different from the identity mapping, and, on top of that, that each of them maps u to a distinct vertex v_i.

Now assume that the circuit represented by H is provided to a malicious attacker that wishes to alter the original circuit functionality in a certain, targeted manner by inserting some covert, malicious circuitry in the circuit. Assume, further, that to effect its attack, the attacker must first identify the specific logic gates or wires in the circuit that implement the functionality that it wishes to modify, i.e. the attacker wants to attack a specific gate u in the circuit, and not just any gate. Since there are k mappings under which u in the original circuit is indistinguishable from $k-1$ other gates in circuit represented by H, the attacker does not know whether u corresponds to v_1,v_2,..., or v_k in H. The attacker will now have two choices: either to randomly pick one of the v_i's to attack; or to attack all k v_i's with a larger malicious circuit and risk the exposure of the attack. In either case, the attacker's ability to effect a targeted attack on the circuit is hindered; and hence the term "k-secure gate".

[10] refer to this technique as **circuit obfuscation**. The malicious entity is assumed to operate from the IC fabrication facility responsible for manufacturing the chip that implements the circuit functionality. This facility, they show, is often external to the IC vendor, which makes it easier for the malicious attacker to to insert covert circuitry in the IC. The process of edge (or wire) removal is referred to as **lifting**, in reference to the stage in the 3D IC manufacturing process in which wires are "lifted" from the bottom untrusted

tier to the top, trusted one. The number of lifted edges is chosen as a simple cost metric that captures the cost we incur by choosing to fabricate the lifted wires on a separate tier.

3.3 Summary of Previous Results

The decision version of circuit obfuscation is shown to be in **NP** by [10]. A related decision problem, k-Security-Dec, defined below is also shown to be **NP**-complete.

k-Security-Dec Given a DAG G', $E' \subseteq E[G]$, and $k \in [1, |V[G]|]$, determine whether lifting the edges in E' results in k-security.

The decision problem simply asks whether a candidate set of edges E' is a feasible solution for circuit obfuscation. [10] also devises an approach to addressing circuit obfuscation comprising a greedy heuristic to identify E', and the use of a SAT solver to compute the security level of a circuit (by deciding k-Security-Dec). The approach is empirically assessed on benchmark circuits, including a case-study of a Data Encryption Standard (DES) circuit, to illustrate the inability of an attacker to effectively attack circuits secured using their technique.

Chapter 4

Research Questions

This thesis tackles the following questions relating to the computational complexity of the circuit obfuscation problem defined in Chapter 3:

- How hard is it to solve the problem exactly? [10] shows the decision version of the problem to be in **NP**, which is not enough to conclude the optimization version is intractable. We show that the decision problem is also **NP**-hard, which, combined with the fact that it is in **NP**, means it is **NP**-complete, and therefore intractable unless **P** = **NP**. Showing that the problem is **NP**-hard justifies using other, suboptimal approaches for solving it.
- Can we use a SAT solver to efficiently solve the problem?. In [10], the authors use a SAT solver as part of an approach to address the obfuscation problem. In particular, they reduce the subgraph isomorphism problem to a SAT instance and use the SAT solver to decide the instance. The same approach has been used elsewhere to develop somewhat efficient solvers for otherwise intractable problems [1]. We as well pursue an efficient solver to the problem in the form of a reduction to SAT followed by a SAT solver. It turns out that for circuit obfuscation, even relatively small instances (in terms of the number of gates in the circuit to be obfuscated), can result in prohibitively large SAT formulas, at least when transformed using our reduction. The SAT instances are difficult to solve even by a state-of-the-art SAT solver running on a high-performance computing cluster with ample computing resources. We therefore forgo SAT solvers as a viable approach to our problem.
- How hard is it to approximate the problem? The solution measure for circuit obfuscation is $|E'|$ (if we adopt the same cost metric as [10]). Although [10] shows that

the decision version of the problem is **NP**, they do not tackle the approximability properties of the optimization problem. As the optimization problem is **NP**-hard, it follows that it is quite unlikely they will ever be an efficient polynomial-time exact algorithm solving it. A provably hard-to-achieve approximation ratio for the problem can serve as a baseline against which proposed approximation algorithms can be compared, or it can stop efforts to devise approximation algorithms with lower approximation ratios.

- How good is the heuristic used by [10]? [10] proposes a greedy heuristic for circuit obfuscation but they do not address its performance in terms of solution quality. As is the case with any approximate algorithm, it helps to know what kind of performance guarantees one can expect from running the algorithm on a specific problem instance.

- How hard is the problem of obfuscating a circuit netlist when we are allowed to introduce new gates into the netlist, in addition to removing wires? [10] considers the case where we are only allowed to lift edges (wires) from a circuit netlist. We observe for their notion of security, that introducing additional gates into the netlist, while lifting certain wires at the same time, can also help in circuit obfuscation by enabling more mappings between the original and lifted netlists. This poses a whole new set of underlying computational problems. As is the case in their work, it pays to characterize the computational complexity of the underlying problems as a first step to proposing concrete approaches for them.

Chapter 5

NP-completeness

In this chapter, we show that the optimization problem of circuit obfuscation as defined in Chapter 3, is **NP**-hard. As explained in Chapter 2, this means the problem is intractable (no algorithm can solve it in polynomial time) unless $\mathbf{P} = \mathbf{NP}$. We show the problem is **NP**-hard by showing the decision version of it to be **NP**-complete.

The decision version of the problem is as follows: given a tuple $\langle G, k, \eta \rangle$ where G is a DAG, $k \in [1, |V[G]|]$, $\eta \in [1, |E[G]|]$, determine whether whether we can get k-security by removing at most η edges from G. We refer to this problem from now on as Lifting-Dec, and to the optimization version as Lifting-Opt.

Suppose the answer to a specific Lifting-Dec is "yes". To prove that the answer is indeed "yes", it suffices to provide:

1. A graph H such that $|V[H]| = |V[G]|$ and $|E[G]| - |E[H]| \leq \eta$, and
2. k mappings each of which is a subgraph isomorphism from G to H, for each verex $u \in |V[G]|$, such that u is mapped to a distinct vertex in each of the k mappings that correspond to it.

Veirfying (1) can definitely be done in polynomial time, and verifying (2) can be done in time that is $O(|V[G]|^3)$, which is polynomial in the number of vertices in G. Hence, the instance has a proof that can be efficiently checked, and therefore Lifting-Dec is in **NP**.

To prove that Lifting-Dec is **NP**-hard, we describe a polynomial-time Turing reduction (in fact a polynomial time many-to-one reduction) from the Gate-Subiso problem defined in [10] to Lifting-Dec. Gate-Subiso is defined as follows:

Gate-Subiso Given as input $\langle G, E', u, v\rangle$, where G is a DAG, $E' \subseteq E[G]$, and two distinct vertices $u, v \in V[G]$, let H be the graph we get by removing the edges that are in E' from G. Then, Gate-Subiso is the problem of determining whether there exists a mapping $\phi\colon V[G] \to V[H]$ that is a subgraph isomorphism from G to H such that $\phi(u) = v$.

Since Gate-Subiso is **NP**-complete [10], the existence of a plynomial-time Turing reduction from it to Lifting-Dec shows that Lifting-Dec is **NP**-hard. This reduction is as follows:

1. Create a copy of G and call it H
2. Remove the edges that are in E' from H
3. Add two sets VH and VG of $|E'| + 1$ vertices each and color those vertices using a color that is not assigned by G's or H's coloring functions to any of G's or H's vertices
4. Add two more sets CH and CG of $\eta + 1$ vertices each and assign them their own unique color as well
5. Construct H' and G' as follows. $V[G'] \leftarrow V[G] \cup VG \cup CG$. $V[H'] \leftarrow V[H] \cup VH \cup CH$. $E[G'] \leftarrow E[G]$. $E[H'] \leftarrow E[H]$
6. Add an edge from u to each vertex in VG and edge from v to each vertex in VH and denote by DG and DH those sets of newly added edges
7. Add an edge from each vertex in G' that was in G to each vertex in CG and do the same for H' (add edges to vertices in CH)
8. Construct $Final \leftarrow H' \cup G'$
9. Provide $\langle Final, 2, |E'|\rangle$ to the oracle for Lifting-Dec and return the output from the oracle as the output of the algorithm

It can be seen that the oracle for Lifting-Dec is called only once by the algorithm (at then end), and that the algorithm spends polynomial time outside the call. All we have to do now to show the validity of the reduction is prove that the algorithm above indeed solves Lifting-Dec, i.e., an instance of Gate-Subiso transformed by the algorithm will have the same answer as the original instance. To prove this, first observe the following:

1. Removing less than $|E'|$ of the edges ending at vertices in CG or CH does not help in increasing the security level of the circuit.

2. As long as none of the edges ending at the vertices in CG or CH is removed, G' and H' will both be (weakly) connected and it would need at least $|E'| + 1$ edges to be removed for either to become disconnected.

3. The edges in DG and DH distinguish u and v as the the existence of any of them in an unlifted netlist means the starting vertex is either u or v.

4. An automorphism in G' that maps u to a vertex other than itself cannot exist. This is since u is the only vertix in G' adjacent to a vertex in DG, and those vertices are of a unique color. The same holds for H' and v.

Now assume the answer to some Gate-Subiso instance $\langle G, E', u, v\rangle$ is "yes". By defintion, this means that there exists a mapping $\phi : V[G] \to V[H]$ that is a subgraph isomorphism from from G to H such that $\phi(u) = v$. If that is the case, then removing the edges in E' from $Final$ will give us 2-security, as the result will be a graph with an automorphism that maps each vertex to a vertex other than itself. Hence, the anawer to the Lifting-Dec instance will be "yes" as well (since $|E'| \leq |E'|$).

Now assume that the answer to the same Gate-Subiso instance was "no"; i.e., there is no subgraph isomorphism from G to H that maps u to v. We show that the answer to the Lifting-Dec instance must be "no" as well. Assume the answer was "yes", that is, there exits an edge set $R \subseteq E[Final]$ with size at most E' that when removed from $Final$ renders the circuit 2-secure. Let $FinalLifted$ be the graph we get by remvoing the edges in R from $Final$. By Items (1) and (2) above, we know that $FinalLifted$ must consist of two connected components. Let us call these components GL and HL. Denote by u' and v' the pair of vertices in $FinalLifted$ corresponding to u and v, respectively, in $Final$. As R cannot contain all the edges in DH or DG (the sets have more than $|E'|$ edges each); any subismorphic mapping from $FinalLifted$ to $Final$ will have to map u' and v' to either u or v (by Item (3) above) and there must be 2 such subisomorphic mappings for both u' and v' (one mapping it to to u and the other to v) (by Item (4)); otherwise $FinalLifted$ can not be 2-secure. Now as both HL and GL are (weakly) connected, those subisomorphic mappings will have to map HL and GL to either G' or H' (i.e., if a vertex in HL is mapped to a vertex in H' then all other vertices of HL will have to be mapped to vertices in H' as well, and the same goes for GL). So now we know there must exist a subisomorphic mapping from H' to GL. But such a mapping cannot exist unless H' has at least as many edges as GL (by definition of subgraph isomorphism). Since $|E[G']| - |E[H']| = |E'|$,

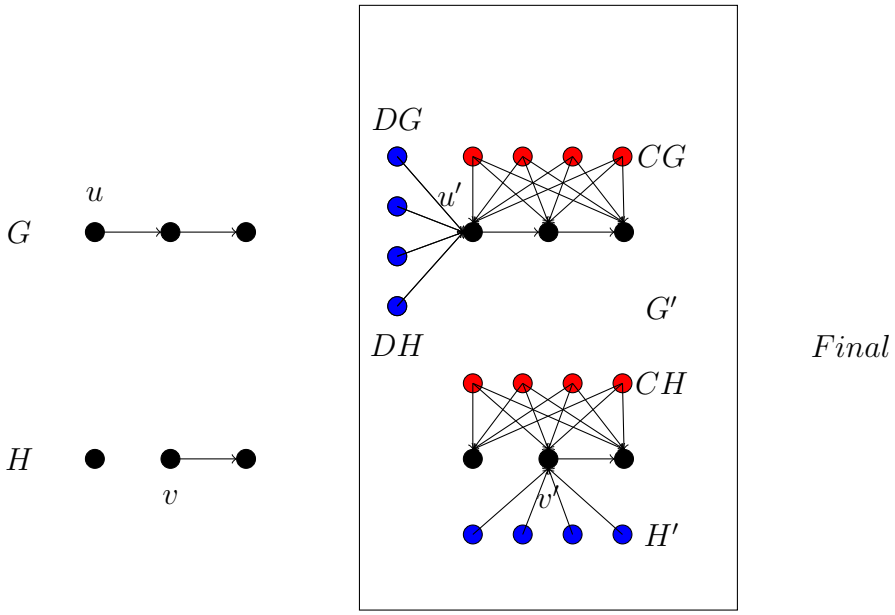

Figure 5.1: An example of the reduction from Gate-Subiso to Lifting-Dec. E' contains a single edge, the one starting at u. $\langle G, E', u, v\rangle$ is a true instance of Gate-Subiso, which means $\langle Final, 2, |E'|\rangle$ is a true instance of Lifting-Dec.

it means that all of R's edges must be from G', and that the mapping is actually an isomorphosm from GL to H'. But we know that GL is subgraph ismorphic to G' (since GL is actually G' with some edges removed). Composing the two mappings together, we get a subismorphism from G' to H' that maps u to v. The subisomorphic mapping is valid even when the vertices of G' and H' that are extra to G and H, respectively, are removed. This means that there exists a subisomorphism from G to H that maps u to v, which is a contradiction. Hence, the answer to Lifting-Dec instance must be "no" as well.

We have therefore shown that the answer to a Gate-Subiso instance remains the same after applying the reduction. Hence, the reduction is valid and Lifting-Dec must be **NP**-hard. This, combined with the fact that it is is **NP** means it is **NP**-complete, which in turn means that Lifting-Opt is **NP**-hard. The example in Figure 5 illustrates the validity of the reduction.

We point out that Lifting-Opt is not in **NPO**, i.e., it is not an **NP** optimization problem. Although the problem satisfies the first and third conditions of an **NPO** described in Chapter 2 (the size of every feasible solution cannot be greater than $|E[G]$ and is

at the same time polynomial-time computable), deciding whether a solution is feasible (second part of the second condition) is not easy. In fact, it is the same as deciding the k-Security-Dec problem, which is proven to be **NP**-complete and thus infeasible by [10].

Chapter 6

Reduction to SAT

In this chapter, we report the negative results of our efforts to develop an efficient solver for Lifting-Opt based on miniSAT, an open-source SAT solver. These efforts were motivated by the recent advances in the speed and efficiency of SAT solvers and the fact that Lifting-Dec, the decision version of Lifting-Opt, is in **NP**, meaning it is polynomial-time Turing reducible to SAT. [10] uses a SAT solver as part of their approach to circuit obfuscation, but in their case, the solver is used to compute security, where the subgraph isomorphism problem is reduced to a SAT instance which is then fed to the SAT solver. We instead reduce Lifting-Dec to a SAT instance which we then feed to the solver. We start by presenting the algorithm we use to reduce Lifting-Dec to SAT, which involves an intermediate reduction to the circuit satisfiability problem (CSAT). We then describe how the algorithm can be used to obtain the optimal solution for a Lifting-Opt instance. Finally, we analyze why the approach, despite being valid, does not work very efficiently on large Lifting-Opt instances.

6.1 Reduction of Lifting-Dec to SAT

We reduce Lifting-Dec to SAT in two steps. The first is a Levin reduction from Lifting-Dec to CSAT. We use a Levin reduction to ensure we will have an efficient way to transform a certificate for the CSAT instance (a satisfying assignment to the boolean combinational circuit) to a certificate for Lifting-Dec (a candidate edge set which, when lifted, gives us k-security). The second step is a textbook reduction from CSAT to SAT. The textbook reduction is also a Levin reduction.

6.1.1 Reduction of Lifting-Dec to CSAT

The reduction from Lifting-Dec to CSAT works as the following. Given a prospective instance of Lifting-Dec in the form of a tuple $\langle G, k, \eta \rangle$, we construct a boolean combinational circuit for which the output can be 1 if and only if there exists an edge set $E' \subseteq E[G]$ such that $\langle G, k \rangle$ is k-secure and $|E'| \leq \eta$. The circuit has the following inputs:

1. For each edge of G, the circuit has an input e_i. A satisfying assignment for the circuit with a value of 1 for an e_i input translates to an E' that includes the corresponding edge in G. A satisfying assignment with a value of 0 translates to an E' that does not include the edge.

2. For every two pairs of vertices (u, v) and (i, j) in G that each have the same color, i.e. $c(u) = c(v)$ and $c(i) = c(j)$, the circuit has an input $\phi_{uv,ij}$. The semantics for these inputs are as follows. Assume we have an assignment for the circuit with values for the e_i inputs corresponding to some $E' \subseteq E[G]$, and let H be the graph we get by deleting the edges in E' from G. For a given pair of vertices (u', v'), the assignment for inputs $\phi_{u'v',ij}$ corresponds to a mapping from G to H that maps u' to v'. The mapping maps vertex i in G to vertex j in H if and only if the assignment assigns a value of 1 to input $\phi_{u'v',ij}$ (Note that this means that $\phi_{u'v',u'v'}$ is always equal to 1 and is not really an input of the circuit per se).

The definition of a k-secure gate requires the existence of k such mappings that are each a subisomorphism and that each map u' to a distinct gate v' in H. Hence, for the reduction to be valid, we must enforce this condition on every satisfying assignment for the CSAT instance. To do this, we build a circuit that outputs 1 if and only if the mapping represented by inputs $\phi_{u'v',ij}$,$i, j \in V[G]$, and $c(i) = c(j)$ is a subisomorphic. The circuit consists of three subcircuits, the outputs of which are connected by an AND gate. The first two subcircuits output 1 if and only if the given mapping is bijective (i.e., it maps each i to only one j and vice-versa). The subcircuits have the following Boolean functions (We drop the $u'v'$ part of the subscript in $\phi_{u'v',ij}$ for simplicity).

$$F_1 = \prod_{j}^{|V[G]|} \sum_{i}^{|V[H]|} \left(\phi_{ij} \prod_{k \neq i}^{|V[H]|} \neg \phi_{kj} \right) \tag{6.1}$$

$$F_2 = \prod_{j}^{|V[G]|} \sum_{i}^{|V[H]|} \left(\phi_{ij} \prod_{k \neq i}^{|V[H]|} \neg \phi_{kj} \right) \tag{6.2}$$

The products here correspond to AND gates and the summations correspond to OR gates. The third subcircuit outputs 1 if and only if the edge relations of the vertices in H are satisfied. It takes and the e inputs along with the ϕ inputs and applies the following Boolean function (again, the $u'v'$ part of the subscript in $\phi_{u'v',ij}$ is dropped for simplicity):

$$F_3 = \prod_{k}^{|E[G]|} e_k \vee \sum_{l}^{|E[G]|} \phi_{src(e_k),src(e_l)} \wedge \phi_{dest(e_k),dest(e_l)}$$

Here *src* and *dest* are functions that take an e input and return respectively, the head and tail of the corresponding edge in G. What this essentially means is if an edge e_i is lifted in G (and hence $e_i = 1$) the corresponding head and tail vertices in H do not need to map to adjacent vertices in G; otherwise they do. Here again, the products correspond to AND gates and the summations to OR gates. By connecting the outputs of the three subcircuits to an AND gate, we make sure the final circuit outputs 1 if and only if the mapping is bijective *and*itand it preserves the edge relations of the vertices of H, i.e., is subisomorphic. We create one copy of the circuit for each pair of vertices (u, v) in G that has the same color, and denote the resulting circuit by C_{uv}. We point out the similarity between the formulas for the three subcircuits and the ones used in [10] to compute security.

To make sure that at least k subisomorphic mappings exist for u', we take the outputs of the Cuv circuits that have $u = u'$ and feed them an adder, and then use a comparator to ensure the addition result is greater than or equal to k. We perform the summation using $|S'_u| - 1$ $\lceil \log_2 |S'_u| \rceil$-bit adders, where S_u is the number of vertices in G with the same color as u, and perform the comparison using a single similarly-sized comparator. We denote the comparator output for a vertex u by $kSAT_u$.

We also have to make sure that an assignment satisfies the circuit if and only if the corresponding E' in G has η edges at most. We do this by feeding the e inputs to an adder followed by a comparator that outputs 0 if the addition result is greater than η. Addition is performed using $|E[G]| - 1$ $\lceil \log_2 |E[G]| \rceil$-bit adders and comparison using a single similarly-sized comparator. We finally connect the comparator output and the $kSAT_u$ outputs to an AND gate to make sure both k-security and cost conditions are satisfied by the satisfying assignment. The output of this AND gate will be the output of the circuit. Therefore, denoting the output of the comparator by $etaSAT$, the output of the circuit can be expressed as:

$$F = etaSAT \prod_{i}^{|V[G]|} kSAT_i \tag{6.3}$$

6.1.2 Reduction from CSAT to SAT

We follow the reduction from Lifting-Dec to CSAT with a textbook polynomial-time (in fact linear) reduction from CSAT to SAT. The reduction is usually attributed to Tseitin[13] although the construction has been independently discovered, in different variations, many times since then[9]. The reduction works as follows.

Given a circuit C to be reduced to a SAT formula, for each wire x_i in the circuit, the formula ϕ has a variable x_i. The operation of each gate in the circuit C is then expressed using a small formula (clause) involving the variables of its incident wires. The clause for a gate that applies a Boolean function f to its inputs $x_1, \ldots, x_n$ is $x_o \iff f(x_1, \ldots, x_n)$, where x_o is the output of the gate. The formula ϕ produced by the reduction algorithm is the AND of the circuit-output variable with the conjunction of the clauses that describe the operation of each gate. The reason the reduction works is that a satisfying assignment for the formula ϕ must assign a value of 1 to the output of the circuit while ensuring correct operation for every gate in the C, as implied by the last ANDing. A satisfying assignment for C similarly satisfies ϕ, hence the transformation is a reduction.

It is worth pointing out that if all the gates in our CSAT instance had a fan-out of at most 1, then the textbook reduction would not have been necessary; we could have simply expressed the circuit in the CSAT instance as a boolean formula. In fact for the three subcircuits that constitute the subisomorphism circuit in the redcution from Lifting-Dec to CSAT, we do not use the textbook reduction; rather, we express them as boolean formulas directly, as the gates in each of them have a maximum fan-out of 1. However, using such a straightforward method on the rest of the circuit would have resulted in exponentially-sized formulas, as some of the the gates in the rest of the circuit have fan-outs of 2 and more. We therefore use the textbook reduction to ensure the final SAT instance is polynomially sized (in the size of the Lifting-Dec instance that is).

Finally, since the SAT solver we use does not accept formulas in forms other than conjunctive-normal form (CNF), we convert the formula generated by the Tseitin reduction into an equisatisfiable formula that is in CNF.

6.2 Lifting-Opt Solver

The reduction from Lifting-Dec to SAT can easily be used to find the optimal solution for a Lifting-Opt instance. Given such an instance with a graph G and a security level k to be achieved by lifting the minimum number of edges, we can do binary search using

the Lifting-Dec solver to find the smallest η such that $\langle G, k, \eta \rangle$ is a true instance of Lifting-Dec. We know η to be in $[0, |E[G]|]$ so we can start with $\eta = \lceil |E[G]|/2 \rceil$ for instance. Since the Lifting-Dec-to-SAT reduction is a Levin reduction, when the search terminates, we simply transform the certificate for the SAT instance, i.e. the satisfying assignment for the formula returned by the SAT solver, to a an edge set to be lifted in G to get k-security. The transformation is simply by looking at the first $|E[G]|$ elements of the satisfying assignment for the SAT formula, lifting the corresponding edge if an element has a value of 1 and leaving the edge in G if the element has a value of 0.

6.3 Results

Although our approach of reducing Lifting-Dec to a SAT instance and feeding the instance to a SAT solver is valid, the SAT instances generated by the reduction can be too large for even a state-of-the-art SAT solver running on a high performance computing (HPC) cluster to handle efficiently. This is due to the relation between the the size of the SAT formula generated by our algorithm and the size of the graph in Lifting-Dec. Although the size of the SAT formula is linear in the size of the CSAT circuit, the size of the CSAT circuit is in fact $O(|V[G]|^5)$ where G is the graph in the Lifting-Dec instance. To show this, consider the circuits corresponding to the formulas in Equations 6.1 and 6.2. It can be seen that each circuit has $O(|V[G]|^3)$ wires and since we need $O(|V[G]|^2)$ copies of each, one for each pair of vertices in G, the total number of wires will be $O(|V[G]|^5)$. This means that, for a circuit with ≈ 200 circuits such as the c432 benchmark from ISCAS-85 benchmark suite [5] (a benchmark bus-controller), the size of the generated CSAT circuit will be $\approx 200^5 = 320$ billion gates. Assuming we need a single byte to store the data relating to each wire (which is an underestimation), this calls for at least 320 GB of memory *just* to store the circuit. This amount of memory exceeds the per-process memory limits of the HPC cluster we used to run our experiments, which means the instance of the SAT solver runs out of memory *before* it even begins trying to solve the CNF formula.

In fact, even tring to solve the special case of Lifting-Opt where the k-security target is a single gate is hard. The only case we were able to solve on the HPC cluster was when k in Lifting-Dec is restricted to be equal to 2 *and* the target of security is reduced to a single gate. Restricting Lifting-Opt in this way enables us to replace the adder-comparator combinations following the C_{uv} circuits in our CSAT circuit with a simple OR of the outputs of the circuits (that does not include the circuits where the two vertices in the pair are the same vertex). This is since, when k is 2, only one subisomorphism other than the identity mapping is required for the vertex. But even that would take about 75%

of the available per-job memory available on the cluster.

6.4 Conclusion

We conclude that a straightforward reduction from Lifting-Opt to SAT is not a viable apparoch for developing an efficient solver for Lifting-Opt. This is largely due to the Pseudo-Boolean (PB) constraints involved in Lifting-Opt (namely, the constraints on the number of edges that can be lifted as well as the number of subisomorphic mappings that must exist for each vertex in the unlifted netlist). Although it seems like our approach for translating these PB constraints to circuits is the same as that used by SAT solvers that handle PB constraints[9], we point out that this still may not be the most efficient way to handle the problem. This is since it is quite possible that a more efficient reduction from Lifting-Dec to SAT exists. Another possible approach for developing an efficient solver for Lifting-Dec could be to reduce it to a k-Security-Dec instance (this should be possible since k-Security-Dec is **NP**-complete), and then use the solver developed by [10] to solve the k-Security-Dec instance. A third approach is to express Lifting-Opt as $0-1$ integer linear programming (ILP) problem with Pseudo-Boolean constraints and then use a commercial ILP or a free open-source PB solver to solve the ILP problem. We leave an investigation of the feasibility of these approaches as topics for future work.

Chapter 7

Approximability Properties

In this chapter, we report two results relating to the approximability of Lifting-Opt, that is, how hard it is to achieve certain approximation ratios for it. The first result follows from a gap-preserving reduction from the minimum vertex cover problem to Lifting-Opt, which proves that Lifting-Opt is in $\mathbf{MAX - SNP}$, and therefore no polynomial-time approximation scheme is likely to exist for it. The other result is through a Cook reduction from a special case of the maximum common edge subgraph problem, which proves that a certain absolute error guarantee is **NP**-hard to achieve for Lifting-Opt.

7.1 Lifting-Opt is in MAX − SNP

We prove that Lifting-Opt is in $\mathbf{MAX - SNP}$, and therefore inapproximable within a certain constant factor, by describing a gap-preserving reduction from the minimum vertex cover problem to it. Minimum vertex cover is defined as the problem of determining the vertex cover of a given graph that has the minimum cardinality, where a vertex cover of a graph is a set of vertices such that each of edge of the graph is incident to at least one vertex in the set. The problem is known to be inapproximable within a factor of 1.04 unless $\mathbf{P} = \mathbf{NP}$[12]. A gap-preserving reduction from it to Lifting-Opt proves that the same constant approximation factor is hard to achieve for Lifting-Opt.

Gap-preserving reductions work as follows. Given a minimization problem I with a known **NP**-hard approximation ratio ρ, a polynomial-time algorithm is described that transforms an instance of I to an instance of another minimization problem I' for which an **NP**-hard approximation ratio is unknown, such that the following property is satisfied.

$$OPT(I) \leq c \implies OPT(I') \leq c'$$

$$OPT(I) \geq c/\rho \implies OPT(I') \geq c'/\rho'$$

where OPT(I) and OPT(I') are the optima of I and I' respectively, c is some function of $|I|$, the size of instance I and c', ρ' are some functions of $|I'|$. The existence of such a reduction then proves that achieving an approximation ratio of ρ' for I' is also **NP**-hard [4]. When one (or both) of the optimization problems involve maximization, the appropriate $\leq$ and $\geq$ signs in the above implications are reversed.

Our reduction from minimum vertex cover to Lifting-Opt works as the following. Let G be the graph in the minimum vertex cover instance. We create a new graph G' as follows. For each vertex v in G, we creates two edges, e_{v1} and e_{v2}, in G'. As well, for each edge e in G, we create four vertices v_{e1}, v_{e2}, v_{e3}, v_{e4}, with a unique color, in G'. Further, if an edge e is covered by vertices v_1 and v_2 in G (note that every edge in G has to be covered by exactly 2 vertices), we create edges from both v_{e1} and v_{e2} (i.e., the first two vertices in G' corresponding to e) to the head vertices of $e_{v_1,1}$ and $e_{v_1,2}$ (the two edges in G' corresponding to $v1$) and edges from both v_{e3} and v_{e4} to the tail vertices of $e_{v_1,1}$ and $e_{v_1,2}$. We also create edges from both v_{e3} and v_{e4} to the head vertices of $e_{v_2,1}$ and $e_{v_2,2}$ (the two edges in G' corresponding to $v2$) and edges from both v_{e2} and v_{e4} to the tail vertices of $e_{v_2,1}$ and $e_{v_2,2}$. The Lifting-Opt instance will ask for the minimum number of edges that can be removed from G' to make it 2-secure. Figure 7.1 illustrates the operation of the algorithm.

It can be seen that the algorithm runs in polynomial time. Specifically, the run-time is $O(|V[G]|)$. We now show that if the size of the minimum vertex cover for G is k, then the optimal solution to the Lifting-Opt instance will be of size $2k$. To show this, we observe first that the head and tail vertices of all edges in G' corresponding to vertices in G are 2-secure. This is since the reduction ensures each of these vertices has the same connectivity as at least one other vertex in G'. We also observe that lifting any of the edge pairs in G' corresponding to vertices in G will make all vertices connected to them 2-secure. More specifically, for the example in Figure 7.1, lifting the red edge pair will make vertices v_{e1} and v_{e2} indistinguishable from each other, and vertices v_{e3} and v_{e4} indistinguishable from each other. Similarly, lifting the blue edge pair will also make all vertices 2-secure. Therefore, lifting the edge pair e_{v1} and e_{v2} in G' makes 2-secure all the vertices v_{ei} in G' such that $i \in [1, 4]$ and e is incident to v in G. Hence, if a set of vertices of size k exists that covers all edges in G, a corresponding set of edges of size $2k$ exists that when lifted from G' makes all vertices 2-secure.

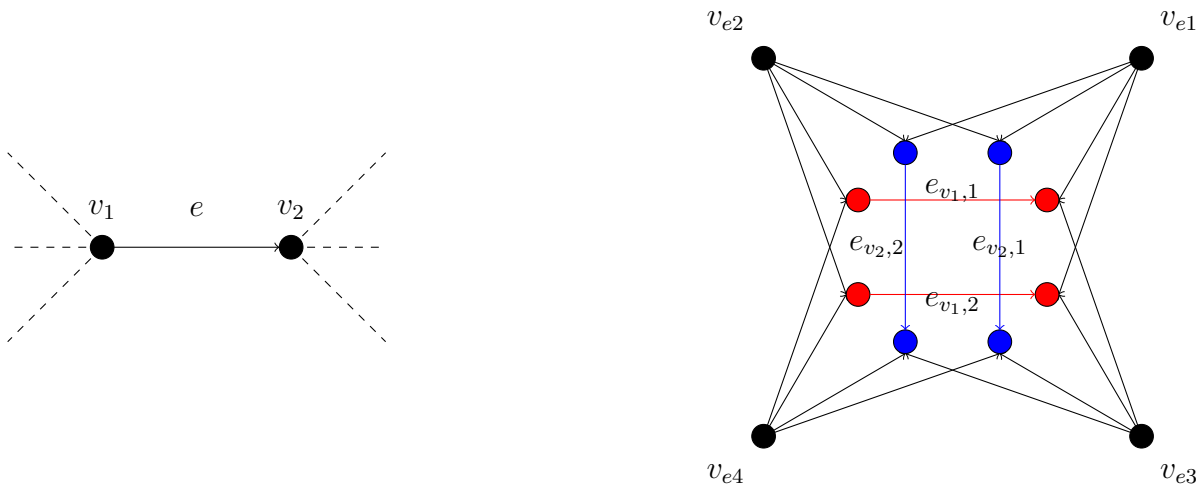

Figure 7.1: Operation of the reduction from minimum vertex cover to Lifting-Opt. On the left depicted is an edge in G, the input graph to the minimum vertex cover instance. On the right is depicted the corresponding subgraph in G', the input graph to the Lifting-Opt instance.

Now assume a cover of size k does not exist for G, that is, no set of vertices of size k covers all the edges of G. We show that a solution of size $2k$ cannot exist for Lifting-Opt. First, we note that for any vertex v_{ei} in G', $i = 1, \ldots, 4$, making it 2-secure requires lifting at least one of the edge pairs in the set $\{(e_{v1}, e_{v2}) : e \text{ is incident to } v \text{ in } G\}$, i.e., one of the edge pairs corresponding to a vertex in G that is incident to e. Hence, any feasible solution for Lifting-Opt, i.e., an edge set that makes all vertices in G' 2-secure when lifted, must include at least once such pair for each vertex. This edge set corresponds to a vertex cover in G. If a solution for Lifting-Opt exited with at most $2k$ edges, it means a vertex cover exits for G that is of size k, which is a contradiction, hence, a solution for Lifting-Opt with size $2k$ cannot exist.

The following relation therefore holds between the optimas of the Lifting-Opt and minimum set cover instances of the reduction.

$$OPT(\text{Lifting-Opt}) = 2OPT(\text{Vertex-Cover})$$

We can therefore write

$$OPT(\text{Vertex-Cover}) \leq c \implies OPT(\text{Lifting-Opt}) \leq c/2$$

$$OPT(\text{Vertex-Cover}) \geq c/\rho \implies OPT(\text{Lifting-Opt}) \geq c/2\rho$$

By the definition of a gap-preserving reduction, this means that if an approximation ratio ρ is **NP**-hard to achieve for minimum vertex cover, then an approximation ratio of 2ρ is **NP**-hard to achieve for Lifting-Opt. Since the minimum vertex cover problem is inapproximable within a factor of 1.04 unless $\mathbf{P} = \mathbf{NP}$, it follows that Lifting-Opt is inapproximable within a factor of 2.08. Thus, Lifting-Opt is itself in $\mathbf{MAX} - \mathbf{SNP}$, and as such, no polynomial-time approximation scheme can exist for it.

7.2 Reduction from the Maximum Common Edge Subgraph Problem

We show another result that proves an absolute error of $\left(\frac{|V[G]|}{2}\right)^{\epsilon}$ is hard to achieve for Lifting-Opt. Stated formally, the result is as follows.

Theorem 7.2.1 *There is no polynomial-time approximation algorithm for the undirected and not necessarily acyclic version of Lifting-Opt that can guarantee an absolute error of* $\left(\frac{|V[G]|}{2}\right)^{\epsilon}$ *where* ϵ *is some constant that lies between* 0 *and* 1.

Note the condition placed on Lifting-Opt that it is undirected and not necessarily acyclic. We conjecture that the result holds even when Lifting-Opt is restricted to DAGs. We leave a investigation into the validity of this claim as a topic for future work.

The result follows from a reduction to Lifting-Opt from the maximum common edge subgraph problem, which is the problem of determining, given two graphs G and H, a graph g with a maximal number of edges that is isomorphic to a subgraph of G and a subgraph of H. We consider the special case of the problem where the two graphs in question have the same number of vertices, which [14] shows is not approximable within addend $|V[G]|^{\epsilon}$, for some $\epsilon \in (0, 1)$, unless $NP = P$.

The reduction we use is a polynomial-time many-to-one reduction that has the following property.

$$|E'| = |E(G)| + |E(H)| - 2k \tag{7.1}$$

Here, E' is the optimal solution for the Lifting-Opt instance generated by the reduction, G and H are the two graphs in the maximum common edge subgraph instance, and k is the maximum size of a common subgraph of G and H. The reduction works as follows.

Let G and H be the two graphs in the maximum common edge subgraph instance. The reduction adds $|E(G)| + |E(H)|$ new vertices each of a distinct color to each graph and connects each new vertex to each of the original vertices of the graphs. If we call the resulting graphs G' and H', it can be seen that G' and H' are weakly connected and it would need at least $|E(G)| + |E(H)|$ edges to be removed for any to become disconnected. The instance of Lifting-Opt generated by the reduction asks for the minimum number of edges to be removed from $G' \cup H'$ to make it 2-secure.

To prove that the reduction has the property in Equation 7.1, recall that E' is the smallest subset of the edge set of $G' \cup H'$ such that $\langle G' \cup H', E' \rangle$ is 2-secure. Let us denote by S the graph we get by deleting the edges in E' from $G' \cup H'$. It can be seen that $G' \cup H'$ can always be made 2-secure by deleting G's and H's edges from it, and so $|E'| \leq |E[G]| + |E[H]|$. As G' and H' both have an edge-connectivity of at least $|E(G)|+|E(H)|$, S must consist of two connected components that are subgraph isomorphic to both G and H (the possibility of an automorphism existing in either G or H that provides 2-security has been eliminated by the addition of the new vertices with distinct colors). Also $|E'|$ cannot contain any of the edges connecting the new vertices in G' and H' to G's and H's original vertices, as removing less than $|E(G)| + |E(H)|$ of these edges does not help in increasing security. The following two lemmas prove that if k is the maximum size of a common subgraph of G and H, then $|E'| = |E(G)| + |E(H)| - 2k$.

Lemma 7.2.2 *There is an edge set with size $|E(G)| + |E(H)| - 2k$ which when removed from $G' \cup H'$ gives us 2-security.*

Proof Since G and H have a common subgraph of size k, we can get 2-security by removing from $G' \cup H'$ the edges of G and H that are not part of this subgraph. The result will be a graph that is 2-isomorphic (has 2 components that are isomorphic to each other), and therefore 2-secure by the definition of k-security.

Lemma 7.2.3 *There is no edge set with size $|E(G)| + |E(H)| - 2k - 1$ that can give us 2-security when removed from $G' \cup H'$.*

Proof Assume there was. It means either G' and H' in the unlifted graph must have at least $k+1$ edges, which cannot be as the maximum size of a common subgraph of G and H is k.

We now derive a relation between $|V[G]|$ and $|V[G' \cup H']|$ as follows. First, we note that, by definition of the reduction, the following relations hold between the vertex and edge counts of the graphs.

$$\|V[G' \cup H']| = 2|V[G] + 2(|E(G)| + |E(H)|)| \tag{7.2}$$

$$\|E[G' \cup H']| = |E(G)| + |E(H)| + 2|V[G]|(|E(G)| + |E(H)|) \tag{7.3}$$

This is of course assuming that $|V[G]| = |V[H]|$, which is a necessary condition for our reduction. Solving Equations 7.2 and 7.3 simultaneously for $|V[G]|$, we get:

$$|V[G]| = \frac{|V[G' \cup H']| - 1 + \sqrt{(|V[G' \cup H']|)^2 + 8\left(\frac{|V[G' \cup |H']]|}{2} - |E[G' \cup H']|\right)}}{4}$$

The second root is invalid if we take into account the nature of the variables. Moreover, the expression under the root cannot be 0 (again, taking into account the nature of the variables), which means we can write.

$$V[G] \geq \frac{V[G' \cup H']}{4} \tag{7.4}$$

As stated previously, [14] proves that for some $\epsilon \in (0, 1)$, there is no polynomial-time algorithm that can achieve an absolute error guarantee of $|V[G]|^\epsilon$ for the maximum common subgraph problem unless $\mathbf{P} = \mathbf{NP}$. Assume a polynomial-time algorithm existed that guarantees an absolute error of $\left(\frac{|V[G]|}{2}\right)^\epsilon$ for Lifting-Opt. For such an algorithm, the solution size, η_o for any Lifting-Opt instance will satisfy:

$$\eta_o - \eta \leq \left(\frac{|V[G]|}{2}\right)^\epsilon \tag{7.5}$$

where η is the size of the optimal solution for that specific Lifting-Opt instance and G is the graph. Assume an instance of the maximum common edge subgraph problem is transformed to an instance of Lifting-Opt according to the our reduction. If k is the size of the optimal solution for the maximum common edge subgraph instance, then combining (7.1) and (7.5) we get:

$$\eta_o - (|E[G]| + |E[H]| - 2k) \leq \left(\frac{|V[G' \cup H']|}{2}\right)^{\epsilon} \tag{7.6}$$

or, since $0 < \epsilon < 1$

$$(\eta_o - |E[G]| + |E[H]|)/2 - k \leq \frac{1}{2}\left(\frac{|V[G' \cup H']|}{2}\right)^{\epsilon} < \left(\frac{|V[G' \cup H']|}{4}\right)^{\epsilon} \tag{7.7}$$

where G, H, G' and H' are as in the reduction. Combining (7.7) with (7.4) we get:

$$(\eta_o - |E[G]| + |E[H]|)/2 - k \leq |V[G]|^{\epsilon}$$

meaning we will always get a value $((\eta_o - (|E[G]| + |E[H]|/2))$ that is within $|V[G]|^{\epsilon}$ of the size of the optimal solution for the maximum common edge subgraph instance. In other words, the algorithm can be used to approximate the maximum common subgraph problem within addend $|V[G]|^{\epsilon}$, by first transforming the maximum common subgraph problem instance to a Lifting-Opt instance and then applying the algorithm. This contradicts [14]'s result, which means a polynomial-time algorithm that guarantees an absolute error of $\left(\frac{|V[G]|}{2}\right)^{\epsilon}$ for Lifting-Opt cannot exist, unless $\mathbf{P} = \mathbf{NP}$, and that is our result.

7.3 Conclusion

We have shown that Lifting-Opt is in $\mathbf{MAX - SNP}$, meaning there is no **PTAS** for it, and also that it is inapproximable with addend $\frac{|V[G]|}{2}^{\epsilon}$, where ϵ is some number between 0 and 1, unless $\mathbf{P} = \mathbf{NP}$. Although the latter can be seen as a somewhat stronger result, we suspect Lifting-Opt to be even harder to approximate. In fact, we conjecture it to be in Class II of [4], i.e. we suspect it be inapproximable within ratio $O(\log n)$ unless **NP** has quasipolynomial-time deterministic algorithms. We leave a proof of this as a topic for future work.

Chapter 8

A Greedy Heuristic

[10] use a greedy wire lifting procedure as part of their approach to determine the security-code trade-off involved in circuit obfuscation but they do not address its performance in terms of solution quality. As is the case for any heuristic or approximation algorithm, it is of interest to know what performance guarantees one can expect from running the algorithm on a specific instance. In this chapter, we show that for [10]'s heuristic there is no performance guarantee better than $O(n^2)$, where n is the number of vertices in the graph in the Lifting-Opt instance. This is the worst any algorithm can do considering as the solution measure for the problem is the number of edges in the solution, which is limited by n^2. We start by giving a brief description of how the heuristic works and discuss why it is not optimal. We follow that by describing two example instances where the greedy heuristic may return solutions with performance guarantees of $O(n)$ and $O(n^2)$. Given that a slight modification to the heuristic can fix the performance guarantee for one of the examples, we address that and give example cases where even the new heuristic may return solutions with performance guarantees of $O(n)$ and $O(n^2)$. We conclude the chapter by comparing the heuristic to an algorithm we propose for achieving k-security that was inspired by recent work in the context of privacy-preserving data publication.

8.1 How the Heuristic Works

[10]'s heuristic works as follows. Given a graph G representing a circuit to be obfuscated, the algorithm starts by lifting every edge in G. This, naturally, gives us the best possible security. The algorithm then checks to see if edges exist that can be added back to G without causing the security level of the circuit to go below the target. If such edges exist,

the edge that will have the least effect on the security level is added back (ties are broken randomly). This step is repeated until no more edges can be added back without causing the security level to drop below the target. The set of edges that are still lifted is then returned as the solution. The heuristic is shown as Algorithm 1, where E' maintains the set of edges that are still lifted, and $\sigma(G, E')$ is the security level of G given E'. The security level is computed using a SAT solver, as mentioned previously.

```
E' ← E[G];
while |E'| > 0 do
    s ← 0;
    foreach e ∈ E' do
        E' ← E' − {e};
        if σ(G, E') > s then
            s ← σ(G, E');
            e_b ← e;
        end
        E' ← E' ∪ {e};
    end
    if s < k then return E';
    E' ← E' − {e_b};
end
return E';
```

Algorithm 1: lift_wires(G, k)

The algorithm essentially makes the locally optimal choice at each stage, hence the term "greedy", but in doing so, it runs the risk of not yielding an optimal set of edges on return. As an example, consider the graph in Figure 8.1. Assume the graph represented a circuit to be obfuscated with a target security level of 2. The security target can achieved by lifting the middle edge. The heuristic, however, starting with an empty graph, may decide to put any of three edges back; as far as it is concerned, putting any of them back gives the same security level of 3. If the heuristic decides to put the middle edge back, however, it will be stuck, as putting any of the other two edges back now will make each vertex uniquely identifiable. The heuristic will then output a solution with two edges when the optimal edge set consists of a single edge.

Figure 8.1: The greedy heuristic does not necessarily yield an optimal set of edges.

8.2 Performance Guarantees

The graph in Figure 8.1 belongs to the family of graphs depicted in Figure 8.2, with n vertices in each of the upper and lower rows of vertices. Assume a graph in this family is given to the heuristic algorithm with a target security level of 2. When every edge in the graph is lifted, the security level is $n + 2$. It can be seen that adding any edge back will give us the same security level (i.e., $n + 1$). But once more, if the greedy heuristic chooses to add the middle edge back, then adding any more edges back will make both "hub" vertices uniquely identifiable, thereby reducing the security level of the whole circuit to 1. The heuristic will return an edge set with $2n$ edges as the solution, when the same security level can be achieved by lifting a single edge (the middle one), meaning the performance guarantee of the solution will be $2n = O(n)$. Note that the heuristic also has a chance of skipping the middle edge at every iteration, in which case it is going to output the optimal solution when it terminates.

Now assume we modify the heuristic so that instead of starting with an empty graph, it starts with the original graph, and then tries to progressively add edges to E', i.e. lift edges from the graph. This is shown as Algorithm 2.

That is, the new heuristic iterates while G still has edges to remove. If this is the case, the heuristic identifies the best edge that can be removed, which is the one that gives the greatest security if added to E'. As soon as the target security level is reached, the heuristic stops.

Assume this heuristic is applied to the graph in Figure 8.2. At the first iteration, the heuristic finds that no edge other than the middle one will have any effect on the security

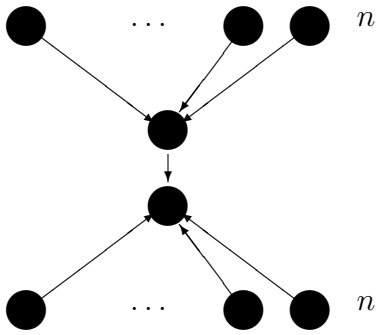

Figure 8.2: Performance guarantee of the greedy heuristic solution can be as bad as $2n$.

$E' \leftarrow \phi$;
while $|E'| < |E[G]|$ **do**
  $s \leftarrow 0$;
  foreach $e \in E[G] - E'$ **do**
    $E' \leftarrow E' \cup \{e\}$;
    if $\sigma(G, E') > s$ **then**
      $s \leftarrow \sigma(G, E')$;
      $e_b \leftarrow e$;
    end
    $E' \leftarrow E' - \{e\}$;
  end
  if $s > k$ **then return** E';
  $E' \leftarrow E' \cup \{e_b\}$;
end
return E';

Algorithm 2: lift_wires2(G, k)

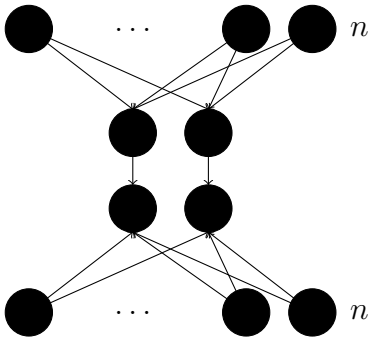

Figure 8.3: Performance guarantee of the new heuristic solution can be as bad as $2n$.

level if lifted. The middle edge, on the other hand, gives a security level of 2 when lifted. The algorithm chooses that edge for lifting, and afterwards it stops, since the target security level of 2 has been reached. Thus, by changing the heuristic in this way, the algorithm is now guaranteed to always return the optimal edge set for this graph family.

Now consider the graph family shown in Figure 8.3, where the upper and lower rows have n vertices each. For a graph in this family, a security level of 3 can be achieved by lifting the two middle edges. However, starting with the original circuit, the heuristic in Algorithm 2 has no way of knowing that it should select one of two middle edges for lifting, as lifting any single edge will give us the same security level of 2. This remains true as long as at least one "non-hub" vertex is connected to two hub vertices. But the heuristic can end up lifting $4n - 1$ edges before every non-hub vertex becomes connected to connected no more than one hub vertex, meaning the performance guarantee of the solution can still be as bad as $4n/2 = O(n)$.

We have therefore shown that both ways of applying the heuristic can output solutions with performance guarantees of $O(n)$. We now show that they can also output solutions with performance guarantees of $O(n^2)$ for some instances. The example family of instances for [10]'s heuristic is shown in Figure 8.2, where there are $6n$ white vertices, for a natural number n. In this case, 2-security can be achieved by lifting 4 edges (the ones connecting the blue vertices to black vertices). However, starting with the version of the graph where every edge is lifted, the heuristic may choose to place the edges connecting the blue vertices to the black vertices as well as those connecting the black vertices to the white vertices (the security level will still be 2). The heuristic will then be stuck (adding back any of the white-to-white edges will expose the identities of the blue vertices). The output edge set will then have $O(n^2)$ edges compared to the optimal set which has $O(1)$ edges, meaning

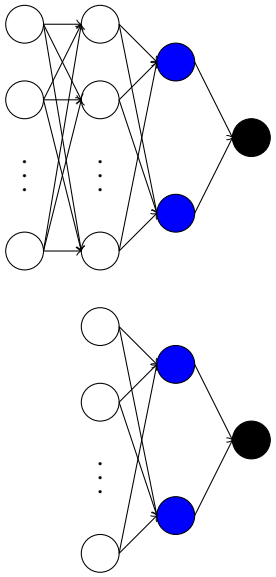

Figure 8.4: Performance guarantee of the greedy heuristic solution can be as bad as $O(n^2)$.

the performance guarantee of the solution will be $O(n^2)$.

The heuristic in Algorithm 2 can also output a solution with a performance guarantee that is $O(n^2)$. The example family of instances is shown in Figure 8.2, where we have $2n$ white vertices, exactly two of which are connected to the black vertex. For a graph in this family, 2-security can be achieved by lifting two edges, namely, the ones incident to the black vertex. But again, starting with a full graph, the heuristic has no way of knowing that it should select one of these edges first (as long as least one of them exist, the colored vertex will be uniquely identifiable). The heuristic can end up lifting all other edges before lifting the two that are incident to the lower black vertex, thus providing a solution with $O(n^2)$ edges, compared to the optimal solution that has only 2 edges.

Thus, we have shown that both heuristics can output solutions with performance guarantees that are as bad as $O(n^2)$. In the next section, we compare [10]'s heuristic to an algorithm we develop for solving Lifting-Opt that was inspired by recent work in the context of privacy-preserving data publication.

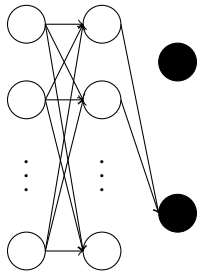

Figure 8.5: Performance guarantee of the heuristic in Algorithm 2 can be as bad as $O(n^2)$.

8.3 K-Isomorphism

In a somewhat recent work, Cheng et al. [6] introduced an algorithm that transforms a graph representing a social network into an arbitrary number of pairwise isomorphic graphs, by performing elementary operations on the original graph. The objective of the transformation is to protect the privacy of individuals in the network represented by the graph from certain types of attacks that leverage information in graph representations. The transformation is demonstrated as being both necessary and sufficient for the protection of the network. The algorithm they propose involves partitioning the given graph into into k subgraphs with the same number of vertices, then augmenting those with edge addition and deletion to ensure pairwise graph isomorphism. In an effort to keep the main characteristics of the original graph in the final k-isomorphic graph, so that the latter may be useful for data analysis, the algorithm tries to keep the total number of edge additions and deletions to a minimum.

Now assume an algorithm existed that achieves the same end result, i.e. k-isomorphism, but it does so *only* by deleting edges from the original graph, that is, no edge additions are performed. The following proposition states that the resulting graph will be k-secure with respect to the original graph.

Proposition 8.3.1 *Given a DAG G, and a DAG H we get from G by removing the edges in a set $E' \subseteq E[G]$, if H is k-isomorphic then $\langle G, E' \rangle$ is k-secure.*

Proof A k-isomorphic graph H, as defined in [6], consists of k disjoint subgraphs $h_1, \ldots, h_k$, i.e. $H = \{h_1, \ldots h_k\}$ such that h_i and h_j are isomorphic for $i \neq j$. Such a graph, by definition, has $k-1$ automorphisms for each component graph h_i, that each map the vertices in h_i to the vertices of a distinct component graph h_j, such that $i \neq j$ for all mappings. Assume

the identity mapping from G to H maps vertex u in G to some vertex v in h_i (we know we have such a mapping since H is subgraph isomorphic to G). This mapping, combined with an automorphism of H that maps the vertices of h_i to the vertices of another component h_j gives us a subisomorphic mapping from G to H that maps u to some other vertex v_j in h_j. This vertex is different for each automorphism provided that automorphism maps h_i's vertices to another component's vertices, which means we have k subisomorphisms from G to H that each map u to a distinct vertex. This means u is k-secure and the whole graph by extension is also k-secure. ∎

Of course, while k-isomorphism is a sufficient condition for k-security, it is not a necessary condition, i.e., $\langle G, E' \rangle$ can be k-secure even if H is not k-isomorphic. Still, it seems like an efficient algorithm for making a graph k-isomorphic by deleting as few edges as possible could act as an approximation algorithm for Lifting-Opt, maybe even with an approximation ratio better than that of the greedy heuristic.

Unfortunately it turns out that such an algorithm is highly unlikely to exist, and that if it did, just like the heuristic, the performance guarantee can be no better than $O(n^2)$. The reason for the first claim is that the problem of finding the optimum edge set remains **NP**-hard, and therefore unsolvable exactly in an efficient way unless $\mathbf{P} = \mathbf{NP}$. The proof of **NP**-hard ardness is the same as the one used to be prove that Lifting-Dec is **NP**-complete, if we note that the unlifted netlist in the reduction's Lifting-Dec instance is always 2-isomorphic. Moreover, an algorithm that transforms a graph into k pairwise-isomorphic components by edge deletion can return a solution with a size that is $O(n^2)$ times the size of the optimum solution for Lifting-Opt, as shown in the example in Figure 8.2. For the graph in the figure, n-security can be achieved by lifting a single edge (the one ending at the black vertex). However, to transform the graph into n pair-wise isomorphic components, at least $n(n-1)+1$ edges will need to be lifted, meaning the performance guarantee of the solution will be $n(n-1)+1 = O(n^2)$.

But the example in Figure 8.3 is unlikely to correspond to a real-life circuit, so we investigate the relative performance of the k-isomorphism approach compared to [10]'s greedy heuristic on a more realistic benchmark, that is, the c432 circuit from the ISACS benchmark suite[5]. We choose the circuit for comparison since [10] already uses it to to investigate security-cost trade-offs obtained from the heuristic. We modify [6]'s algorithm so that the operations it performs on the input graph consist *only* of edge deletions, instead of both edge additions and deletions, hence guaranteeing the generated graph will be subgraph isomorphic to the original graph. The modified algorithm works as follows.

1. Vertices are added to the input graph to ensure the number of vertices of each color in the graph is a multiple of k (since otherwise no algorithm can make the graph

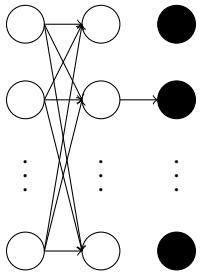

Figure 8.6: Performance guarantee of the solution provided by an algorithm that transforms a graph into k pairwise-isomorphic components by lifting as few edges as possible can be as bad as $2n$.

k-isomorphic merely by deleting some edges). We note that this can be done by adding no more than $c(k-1)$ vertices where c is the number of distinct colors in the input graph.

2. We then locate what [6] refers to as Potential Anonymization Graphs (PAGs). A PAG is simply any connected subgraph of the input graph with an edge count that lies within a certain threshold, $maxPAGsize$. In [10], $maxPAGsize$ is set to the average degree of the input graph, with the intuition being that many vertices in the input graph have this degree and each forms a PAG with their immediate neighbors. The PAGs are determined by traversing the given graph from each vertex in a depth-first manner, and enumerating all connected subgraphs with $mazPAGsize$ edges. The graph-traversal also determines the *embeddings* for each PAG, where an embedding of a PAG is simply another subgraph of the input graph that is isomorphic to the PAG in question.

3. For each PAG, it is then determined whether the PAG has k vertex-disjoint embeddings. As [6] shows, this poses an **NP**-hard decision problem that can be transformed to a problem of finding an independent set with size k in a graph representation of the embeddings and their relations. [6] seem to do without a polynomial time algorithm for the independent set problem. For better efficiency, we use the polynomial time algorithm described in [8] instead.

4. We select one of the PAGs that was determined in the previous step to have k vertex-disjoint embeddings, and assign the vertices of each of the k embeddings to a separate partition. Since a vertex may only be assigned to a partition once, this step may affect the formation of vertex-disjoint embeddings for the remaining PAGs. [6] intuit

that PAGs that contain vertices with the highest degrees may incur greater distortion and that there is a chance of reducing the overall distortion if treated earlier. We do the same. We also maintain the isomorphic mappings between the embeddings in a table, which we refer to later in the algorithm.

5. Steps 3 to 4 are repeated until no PAG remains with size $maxPAGsize$ that has at least k vertex-disjoint embeddings.

6. In [6] the remaining PAGs are "anonymized", a process which involves adding new vertices and edges to the graph to form new vertex-disjoint embedding for a PAG. Since we do not have the liberty to do this for Lifting-Opt, we simply look for PAGs with smaller sizes that have at least k vertex disjoint embeddings. That is, we decrement $maxPAGsize$ by 1 and repeat Steps 2 to 6 until all vertices in the graph have been assigned to partitions or $maxPAGsize$ becomes 1, in which case we simply divide the remaining vertices into groups such that the number of vertices of each color within a group in the same, and assign each group to a separate partition.

7. Edges that cross partitions are deleted, so that we end up with k components. For every pair of components, we use the mappings we generated earlier between the vertices of the two components to determine which edges should be deleted, if any, to make the two components isomorphic. This is another difference from [6]'s algorithm, where they can add edges that did not exist originally in the input graph.

The results of running our algorithm on the c432 benchmark for security levels 2 through 10 are reported in Table 8.1.

For security levels higher than 10, the algorithm deletes all edges in the c432 circuit. The algorithm thus does a significantly worse job than [10]'s heuristic, which for the first 10 security levels, deletes at most 200 wires from the circuit. The relatively bad performance for higher security levels can be due to the isolated vertices introduced into the circuit in Step 1, which necessitate the isolation of some other vertices in the circuit to achieve k-isomorphism.

8.4 Conclusion

We have shown that for [10]'s greedy heuristic, no performance guarantee better than $O(n^2)$ can be expected. The heuristic performs favorably however compared to a state-of-the-art algorithm that provides a feasible solution for Lifting-Opt by transforming the

Target security level	Number of added vertices	Number of deleted edges
2	1	229
3	4	258
4	3	279
5	6	278
6	7	291
7	8	303
8	7	303
9	16	294
10	11	293

Table 8.1: Performance of [6]'s modified algorithm on the c432 benchmark for security levels 2 to 10.

input graph to a k-isomorphic one, where k is the target security level in the Lifting-Opt instance.

Chapter 9

Circuit Obfuscation by Augmentation

In this chapter, we give a couple of examples to illustrate how augmenting a circuit by introducing new gates and wires can help in increase the security level, provided it is coupled with the right wires being lifted to the trusted tier. We give a new notion of security appropriate in this context and characterize the computational complexity of the problem of achieving security as a first step to proposing concrete approaches for it.

9.1 Example

Assume the target of the defender is the make the full adder circuit in Figure 9.1 from [10] 2-secure. The defender can select the wires connecting the adder's inputs and outputs to the rest of the circuit for lifting, then request that the foundry fabricates two isolated copies of the unlifted netlist on the untrusted tier. That is, the defender can send the attacker the netlist in Figure 9.2. Since it has access to the original netlist, the foundry knows that all the designer needs is a single full adder circuit on the chip, but it has no way of knowing which of the two circuits will be activated when the final 3D chip is created. An attacker at the foundry can then either randomly pick one of the two circuits to attack and only succeed with probability 50%, or it can attack the two circuits with a larger malicious circuit and risk the detection of the attack. The attacker's ability to conduct a malicious attack is, therefore, again somewhat hindered.

The circuit duplication technique of course has the disadvantage that implementing two copies of a circuit on a tier requires an IC that is approximately two times larger. This cost can be mitigated by lifting more wires from the original netlist to make some gates

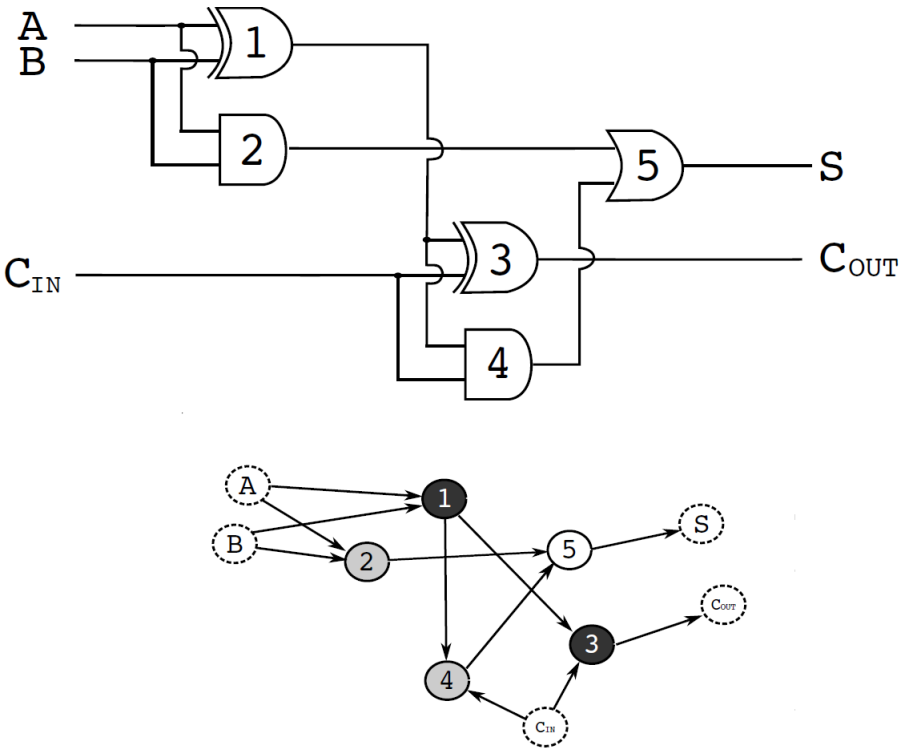

Figure 9.1: A full adder circuit and its graph representation.

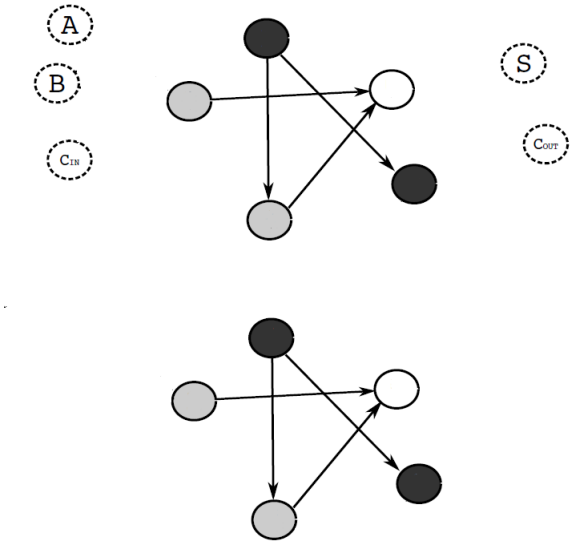

Figure 9.2: The defender can select the wires connecting the adder's inputs and outputs to the rest of the circuit for lifting, then request that the foundry fabricates two isolated copies of the unlifted netlist on the untrusted tier.

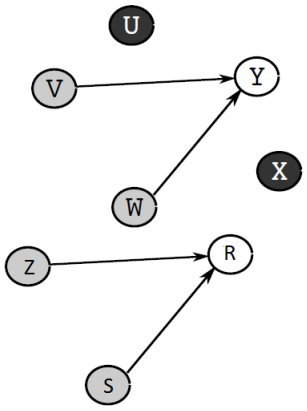

Figure 9.3: Possible bottom tier netlist for the full adder circuit. The netlist consists of 8 gates.

more secure, even without duplication. The graph representation of an example bottom tier netlist is depicted in Figure 9.3. The netlist has 8 gates as opposed the one in Figure 9.2 which has 10, but at the same time more wires need to be implemented on the top tier to create the final 3D circuit. In the example in Figure 9.4, the bottom tier netlist has even less gates, but in order to activate Gate r in the final circuit, a total of 7 wires will need to be implemented on the top tier.

For the netlist in Figure 9.4, note that if Gate r is chosen for activation as the OR gate in the final circuit, then we will have a dead gate in the circuit, namely Gate y. In this chapter, we only consider the case where the final active circuit is a replica of the original circuit, i.e., without any extraneous logic. This includes logic that is not dead yet introduced for the mere purpose of obfuscation (e.g., a pair of inverters placed in a row on an internal wire of the circuit).

To conclude this section, we note the following two points about the circuit augmentation technique. First, if there are less than k gates of a certain type in a given circuit, then

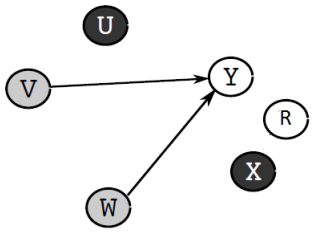

Figure 9.4: Possible bottom tier netlist for the full adder circuit. The netlist consists of 5 gates.

circuit obfuscation by wire lifting alone cannot make these gates k-secure, e.g. without introducing a new OR gate into the full adder netlist, it is impossible to make Gate 5 2-secure, as it is the only OR gate in the full adder circuit. Second, obfuscation by circuit augmentation implicitly assumes an underlying 3D IC manufacturing process. This is since for it to work, at least the wires connecting the circuit I/O to the rest of the circuit will need to be implemented on a separate tier than the one where the rest of the circuit is implemented.

9.2 Notion of Security

The notion of security here is also based on the existence of multiple one-to-one mappings between gates in the original netlist and gates in the netlist sent out for fabrication. However, in this case the mappings do not need to be surjective. i.e., not every gate in the bottom tier netlist will need to be map to a gate in the original netlist. As we mentioned previously, we consider the special case where the subgraph of the bottom tier netlist is disconnected from the rest of the circuit. We define a special type of mapping, which we call a **subcircuit isomorphism**, to capture this notion.

Definition (Subcircuit isomorphism). A DAG G is subcircuit isomorphic to a DAG H if a *spanning* subraph of G is isomorphic to an *induced* subgraph of H that is connected to no additional vertex in H. The corresponding mapping is called a subcircuit isomorphism.

Note that we restrict the subgraph of H to be an induced subgraph. The *cost* of a subcircuit isomorphism from a DAG G to a DAG H is defined as the difference in edge count between the spanning subgraph of G and the induced subgraph of H. Based on the notion of subcircuit isomorphism, we define the notion of a k-secure gate for obfuscation by circuit augmentation as follows.

Definition (k-secure gate). Given a DAG G, a vertex u in it, and another graph H such that there is at least one subcircuit isomorphism from G to H, we say that u is k-secure if there exists k distinct vertices $v_1, \ldots, v_k$ in H and mappings $\phi_1, \ldots, \phi_k$ from $V[G]$ to a subset of $V[H]$ such that every ϕ_i is a subcircuit isomorphism from G to H and for all $i \in [1, k]$, $\phi_i(u) = v_i$.

This is similar to [10]'s definition of a k-secure gate, except here subgraph isomorphism is replaced by subcircuit isomorphism. We denote by S_u the set of vertices in H that u can be mapped to via a subcircuit isomorphism from G to H. By definition, for u to be k-secure, $|S| \geq k$. If v_i is a vertex in S_u, it is possible that u can be mapped to v_i via many different subcircuit isomorphisms from G to H. Let ϕ_{u,v_i} denote the mapping that has the lowest cost among all subcircuit isomorphisms that map u to v_i. We define the *lift cost* of the k-secure gate u, σ_u, as the cost of the *kth* least expensive ϕ_{u,v_i}, where $v_i \in S_u$.

The intuition behind the cost definition is that for the gate to be truly k-secure, the defender must be willing to incur the cost of each of the k mappings corresponding to every gate. In fact, to make the attacker's job as difficult as possible, the selection of the final mapping should be done uniformly at random from the set of all possible mappings. We choose the pessimistic approach of using the cost of the most expensive mapping as a cost metric. Another approach would be to use the expected cost instead.

Based on the definition of a k-secure gate, we define a k-secure graph as follows.

Definition (k-security with lift cost σ). Given two DAGs G and H such that there is at least one subcircuit isomorphism from G to H, we say that $\langle G, H \rangle$ is k-secure with lift cost σ if every vertex in G is k-secure and $\max_{u \in V[G]} \sigma_u = \sigma$. The cost of the k-secure pair $\langle G, H \rangle$ is defined as $\max(c_1\sigma, c_2|V[H]|)$ where c_1 and c_2 are positive constants. An *optimum k-secure version* of G is defined as a graph that has the minimum cost among all graphs H such that $\langle G, H \rangle$ is k-secure.

Here, $c_2|V[H]|$ is an estimate of the area consumed by the cells in the bottom tier of the 3D IC while $c_1\sigma$ is an estimate of the area consumed by the bond-points required to lift wires to the top tiers (According to [10], the bond-density is limited by technology and therefore, more lifted wires correspond to increased area). The area of a 3D IC is determined by the larger of the two areas, hence the cost definition.

9.3 Computational Complexity

We now show that finding an optimum k-secure version of a given DAG G as defined above is **NP**-hard. We show this by showing the corresponding decision problem, Obfuscate, to be **NP**-complete.

Definition (Obfuscate). Given a tuple $\langle G, c, k\rangle$, where G is a DAG, c is a real number, and k is a natural number, determine if there exists a DAG H such that $\langle G, H\rangle$ is k-secure and the cost of the k-secure pair is at most c.

The problem is **NP**-complete by the following simple reduction from Lifting-Dec. Given a prospective instance of Lifting-Dec, $\langle G, \eta, k\rangle$, add a number of vertices to G equal to $\max(n, k)$ where n is the smallest integer that satisfies:

$$c_2(|V[G]| + n) > c_1\eta$$

The new vertices need to be of a color that is not assigned by G's coloring function to any of G's vertices. Call the new graph G'. The instance of Obfuscate becomes $\langle G', c_1\eta\rangle$. To see that the reduction is valid, observe first that a solution for the Lifting-Dec instance always yields a solution for Obfuscate, as, if E' is the edge set in the certificate for the Lifting-Dec instance and H' is the graph we get by removing the edges in E' from G', then $\langle G', H'\rangle$ is k-secure with cost $c_2\eta$. Moreover, since $c_2|V[G']| > c_1\eta$, a solution for the Obfuscate instance implies the existence of an edge set with size at most η which when lifted from G' makes it k-secure. The same edge set will make G k-secure when lifted from it. Hence, $\langle G, \eta, k\rangle$ can only be a true instance of Lifting-Dec if $\langle G, c_1\eta\rangle$ is a true instance of Obfuscate, and since we showed Lifting-Dec to be **NP**-complete in Chapter 5, it follows that Obfuscate is **NP**-hard.

To show that Obfuscate is in **NP**, note that to prove that the answer to a specific Obfuscate instance is 'yes', it suffices to provide the graph H, the size of which will be limited by c/c_1, as well as k subcircuit isomorphisms for each vertex in G that each map the vertex to a distinct vertex in H. The proof can be efficiently checked and hence Obfuscate is in **NP**.

We have therefore shown that Obfuscate is **NP**-complete, which in turn means the optimization problem is **NP**-hard. Thus, finding an optimal k-secure version of a DAG G as defined here is also unlikely to be computationally intractable.

9.4 Conclusion

Characterizing the computational complexity of a problem usually serves as a first step to proposing concrete approaches for it. Here, we have shown that the problem of circuit obfuscation by augmentation is **NP**-hard which means it cannot be solved exactly in polynomial time, and that other, suboptimal algorithms for solving it must be pursued. As a first step in this direction, we conjecture that any optimal solution for the problem will be a subgraph of the graph we get by repeating G k times. We leave a proof and procedure for achieving k-security based on it as topics for future work.

Chapter 10

Conclusions

10.1 Conclusions

The questions stated in Chapter 4 have been answered. As shown in Chapters 5 to 9:

1. The computational problem of circuit obfuscation, as defined in Chapter 3, is highly unlikely to be tractable.
2. SAT solvers are unlikely to be a viable approach for solving the problem.
3. The problem is inapproximable within ratio 2.08 unless $\mathbf{P} = \mathbf{NP}$. Moreover, it is highly unlikely that a polynomial time approximation algorithm exists for solving the problem that has an absolute error guarantee of $\left(\frac{|V[G]|}{2}\right)^{\epsilon}$.
4. The heuristic used by [10] to obtain the security-cost trade-offs involved in circuit obfuscation can output solutions with relative guarantees of $O(n^2)$ for certain Lifting-Opt instances, where n is the vertex count of the input graph. This is the worst the heuristic can do.
5. The problem of circuit obfuscation is still hard when we are allowed to introduce new gates and wires into the circuit to enhance the security level.

10.2 Summary of Contributions

1. Proved that Lifting-Opt, as defined in Chapter 3, is **NP**-hard.

2. Proved that the same problem is in $\mathbf{MAX-SNP}$ and that it cannot be approximated within addend $\left(\frac{|V[G]|}{2}\right)^{\epsilon}$ for some $\epsilon \in (0,1)$ unless $\mathbf{P} = \mathbf{NP}$.

3. Showed that no performance guarantee better than $O(|V[G]|^2)$ can be provided by the heuristic proposed by [10] for solving the problem. Yet the heuristic does better than a solution based on a state-of-the-art algorithm that achieves k-isomorphism.

4. Developed a reduction from Lifting-Opt to SAT.

5. Showed that the problem of circuit obfuscation remains **NP**-hard when we are allowed to introduce gates and wires into the netlist. This is assuming that we adopt the area of the final 3D circuit as a cost metric. The area of the bottom tier is assumed to be directly proportional to the number of cells implemented in it.

10.3 Future Research

1. Try to place Lifting-Opt into one of the higher classes of [4]. We showed Lifting-Opt to be in Class I ($\mathbf{MAX-SNP}$), but we suspect it to also be in Class II, that is, to be inapproximable within ratio $O(\log(n))$ unless **NP** has quasipolynomial-time deterministic algorithms.

2. Investigate the possibility of developing an efficient exact solver for Lifting-Opt using a reduction from Lifting-Dec to k-Security-Dec, followed by [10]'s efficient solver for k-Security-Dec.

3. Investigate the possibility of developing an efficient exact solver for Lifting-Opt using an ILP or PB solver.

4. Try to come up with a more efficient reduction from Lifting-Dec to SAT, or prove that ours is the most efficient.

5. Develop a concrete algorithm for approximating the problem of determining the optimum k-secure version of a DAG G, as defined in Chapter 9. We conjecture that any k-secure version of G will be subgraph isomorphic to the graph we get by repeating G k times, which can serve as a starting point for developing concrete algorithms for the problem.

References

[1] Algorithmic problems in access control.

[2] Fadil A. Aloul. Sat 101: Solving challenging optimization problems using advanced boolean satisfiability and ilp techniques, November 2013.

[3] Sanjeev Arora and Boaz Barak. *Computational Complexity: A Modern Approach*. Cambridge University Press, 2009.

[4] Sanjeev Arora and Carsten Lund. Hardness of approximations. In *Approximation Algorithms for NP-hard problems*. PWS Publishing, 1996.

[5] F Brglez. Neutral netlist of ten combinational benchmark circuits and a target translator in fortran. In *Special session on ATPG and fault simulation, Proc. IEEE International Symposium on Circuits and Systems, June 1985*, 1985.

[6] James Cheng, Ada Wai chee Fu, and Jia Liu. K-isomorphism: Privacy preserving network publication against structural attacks. In *SIGMOD '10 Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*, pages 459–470. ACM, 2010.

[7] Alan Cobham. The intrinsic computational difficulty of functions. In *Proc. Logic, Methodology, and Philosophy of Science II,*, 1965.

[8] A. Dharwadker. The independent set algorithm, November 2013.

[9] Nilas Een and Niklas Sorensson. Translating pseudo-boolean constraints into sat. *Journal on Satisability, Boolean Modeling and Computation*, 2006.

[10] Frank Imeson, Ariq Emtenan, Siddharth Garg, and Mahesh V. Tripunitara. Securing computer hardware using 3d integrated circuit (ic) technology and split manufacturing for obfuscation. In *USENIX Security '13*, 2013.

[11] Viggo Kann. On the approximability of the maximum common subgraph problem. In *STACS 92*, pages 375–388. Springer, 1992.

[12] C. Papadimitriou and M. Yannakakis. Optimization, approximation, and complexity classes. *Journal of Computer and System Sciences*, 1991.

[13] G. Tseitin. On the complexity of derivation in propositional calculus. *Studies in Constr. Math. and Math. Logic*, 1968.

[14] O Verbitksy. On the largest common subgraph problem. Unpublished manuscript, 1994.