# Mathematical Aspects of Program Obfuscation

Lukas Zobernig

A thesis submitted in fulfilment of the requirements for the degree of Doctor of Philosophy in Mathematics, the University of Auckland, 2020.

*Für meine Eltern — Bruno und Elisabeth.*

# Abstract

Program obfuscation is an active and heavily researched topic in theoretical and applied cryptography. *General purpose* program obfuscators would revolutionise the field and make designing new cryptosystems redundant. But as we shall see, there are various pitfalls surrounding general purpose program obfuscation. In this work we instead take a step back and consider *special purpose* program obfuscation for selected problems.

We briefly explore the state of applied program obfuscation and see whether we can bring more theoretical techniques into the field; for this we study the example of *evasive predicates*.

We construct an obfuscator for *fuzzy Hamming distance matching* which finds application in biometric authentication, fuzzy extractors, and secure sketches. The security of this obfuscator is based on a new computational assumption rooted in number theory, which we dubbed the *modular subset product problem* (MSP). We explain our approach to cryptanalysing this problem and give results and conjectures about its (post-quantum) hardness in various parameter ranges.

The Hamming distance obfuscator leads us to another application for special purpose obfuscation: *Conjunctions* or *pattern matching with wildcards*. Our conjunction obfuscator is conveniently based on the MSP hardness.

Finally, we give an obfuscator for a special class of *deterministic finite automata* (DFA). We consider what we call *evasive* DFAs which cannot be learned from oracle access. The obfuscator is based on techniques related to branching program obfuscation. It solves the problem of obfuscated substring matching where substrings can even be given by regular expressions.

# Acknowledgments & Danksagungen

In the following, I will attempt to tell you about the *dramatis personae* that helped me in one way or another during the time that went into this thesis.

I would like to give my deepest and most heartfelt thanks to Steven D. Galbraith, my supervisor. In the time of my doctoral studies at The University Of Auckland you were never short of answers to my countless questions. The freedom you gave me to pursue my own interests has made this journey a wonderful experience.

I cannot thank enough my colleagues Yan Bo Ti, Trey Li, Samuel Dobson, and my co-supervisor Giovanni Russello for being amazing friends and wonderful people to work with.

I am profoundly grateful to all the kind people here in Auckland and in the mathematics department that have made my stay so enjoyable. I have found many a true friend in you and, although there are far too many of you to list, I will try to eternalise at least some of you here: Nina, Boon, Werner, Irène, Natân, Valerie, Gray, Anton, Cris, Ielyaas, Oxana, Andrus, Elle, Ravindra, Stefan, Gina, Dana, Marcos, Elias, Chris, Gab, Anand, Demi, Liv; and all the other "I's & D's".

To all my office mates who came and went over these past three years, especially Shujie and Sana, thank you for making it a welcoming place every day. I wish we could have had cake more often.

I would like to thank the Marsden Fund of the Royal Society of New Zealand for funding my stay in this beautiful country.

*Schlussendlich kann ich nur noch eines hinzufügen: Der allergrößte Dank gebührt meinen Eltern — Elisabeth und Bruno — die mich die letzten drei Jahre aus der Ferne unterstützt haben. Danke, dass ihr es mir ermöglicht habt, in die Welt hinaus zu gehen und meinen Interessen zu folgen!*

# Contents

# 1 Humble Beginnings

Program obfuscation is one of the Holy Grails of *modern* cryptology research. Before we get into further details of the possible applications, pitfalls and problems, and our solutions, let us first give an example of what we understand by the term *program obfuscation*.

Assume that we are given a function (or circuit, or program, respectively) $f : \{0,1\}^n \to \{0,1\}^m$ for some $n, m \in \mathbb{N}$. We would like to construct an *obfuscation algorithm* $\mathcal{O}$ that takes such a function as an input and outputs a new function $\mathcal{O}(f) : \{0,1\}^n \to \{0,1\}^m$ which is equivalent to $f$ on inputs: $f(x) = \mathcal{O}(f)(x)$ for all $x \in \{0,1\}^n$. Say we present the function $f$ itself to Alice. Then Alice has complete information about $f$. The key idea of program obfuscation is the following: If we pass $f$ through the obfuscator $\mathcal{O}$, then we expect the output $\mathcal{O}(f)$ to be completely opaque to Alice. The obfuscated function should behave like a black-box in the following sense: Alice should not be able to learn anything more from the obfuscated function $\mathcal{O}(f)$ than she would already be able to learn from oracle access to $f$.

The astute reader might have noticed some oddities with this description. Especially the wish that nothing should be revealed about the obfuscated function very much depends on $f$ itself. Suppose that we restrict ourselves to the class of linear functions. If $f$ is a linear function $f(x) = kx + d$, with $k, d$ constants, for example, then there is a single unique representation which we can determine from any two distinct input/output pairs. We can find those pairs from black-box access. Hence, for our understanding of obfuscation, linear functions are not a sensible target. Similar ideas work for other classes of functions. The question we should ask is: "Which class of functions is sensible to obfuscate?". We will see other such considerations later.

Program obfuscation is considered a Holy Grail because of this very strong *functionality hiding* property we mentioned. It implies constructions of countless other cryptographic primitives as a by-product. Consider for example public key encryption. We could simply take the function $f_k : \{0,1\}^{128} \to \{0,1\}^{128}$ that encrypts a 128-bit block of data with the AES encryption algorithm under a fixed secret key $k$. If we then publish $\mathcal{O}(f)$, anyone could encrypt their message to us and we would be able to decrypt it since we know $k$. On the other hand, $\mathcal{O}(f)$ cannot decrypt ciphertexts. Furthermore, because of the black-box property of $\mathcal{O}$, an adversary can only learn as much about the secret key $k$ as they would already be able to learn from a chosen-plaintext oracle. It is easy to imagine other examples, even for exotic primitives such as functional encryption.

The informal definition of a program obfuscator given in the second paragraph is not the only one that research considers. Experience tells us that it is important to select good and workable definitions for cryptographic primitives, in our case program obfuscation. We find that different definitions of program obfuscation are possible, and their suitability depends on the specific application at hand. We will give the most common definitions here only in an informal fashion and defer stating the formal definitions until Chapter 2.

- The first definition that we care about is so-called *virtual black-box* (VBB) obfuscation. Barak et al. [Bar+01] introduced this notion in 2001. A VBB obfuscator preserves the functionality of any input program, either perfectly or otherwise up to some probability. It furthermore only imposes a polynomial slowdown on the obfuscated output program, compared to the

original input program. Finally, and this is what we call the *virtual black-box property*, any obfuscated program shall only reveal as much about the original program as black-box access to it would allow. This essentially covers the example description of program obfuscation that we gave in the second paragraph.

Note that VBB obfuscation is formally a very strong definition and we already mentioned that this definition exhibits several oddities when we consider applying it in a generic setting. Indeed, Barak et al. [Bar+01] showed that VBB obfuscation for general programs is impossible by constructing a family of unobfuscatable functions.

- To work around the aforementioned impossibility result, Barak et al. [Bar+12] introduced a weaker obfuscation definition which is now widely used in the theoretical cryptography literature. It is called *indistinguishability obfuscation* (iO) and informally the idea is as follows: Given two equivalent programs, by which we mean that they have the same input-output behaviour or in other words that they compute the same function, it should be hard to distinguish the obfuscations of these two programs. As a simple example, think of the programs $f_k, g_k : \{0,1\}^{128} \mapsto \{0,1\}^{128}$ that encrypt a 128-bit block of data with the AES encryption algorithm under a fixed key $k$. Thinking of $f_k$ and $g_k$ given as circuits for example, they need not be the same. Or in simple words, thinking of programming languages, their *implementations* need not be the same. Nevertheless, they both provide exactly the same functionality: $\forall x \in \{0,1\}^{128} : f_k(x) = g_k(x)$, and are equivalent in this way. Assume now that $O_i$ is an indistinguishability obfuscator. Then it should be hard to distinguish between $O_i(f_k)$ and $O_i(g_k)$.

  The simplest example of an indistinguishability obfuscator is the following: Consider the family of all circuits with $n$ inputs for some $n \in \mathbb{N}$. If we fix an order on these circuits, then for each possible circuit $C$ in the family we can find a canonical representative: Let it be the first circuit with respect to our order that computes the same function as $C$. On the input of an arbitrary circuit of this family, the obfuscator now simply outputs the canonical representative. It is clear that for two functionally equivalent circuits, the obfuscator will output identical canonical circuits and hence there exists no adversary that can distinguish between them.

  Even though the definition might seem innocuous at first, indistinguishability obfuscation promises, among other things, deniable encryption, public key encryption, signatures, non-interactive zero knowledge proofs, trapdoor functions, oblivious transfer, secure multi-party computation, multiparty key exchange, broadcast encryption, and traitor tracing [SW14]. In this light, the existence of such a powerful primitive almost sounds too good to be true. And indeed, we argue that in practice there are various problems with general purpose program obfuscation, see Section 1.1, Chapter 2, and Chapter 8.

- Lastly, in this work we will restrict to what we call *evasive functions*, which are those functions that are *almost* constant. For these it makes sense to consider an *input-hiding* or *perfect circuit-hiding* obfuscator. Barak et al. [Bar+14a] introduced these notions in 2014. Given an obfuscated program produced by an input-hiding obfuscator, it should be hard to find an input that is accepted by the original program. Similarly, given an obfuscated program produced by a circuit hiding obfuscator, it should be hard to answer any predicate about the circuit that represents the original program. We will already hint here that perfect circuit-hiding obfuscation is related to VBB obfuscation.

At the time of writing this, achieving feasible and secure general purpose obfuscation seems to be impossible. The impossibility result of Barak et al. [Bar+01] and thus non-existence of

a general purpose VBB obfuscator prevents us from obtaining cryptographic primitives in a simple fashion. All the great prospects and implications are out of reach in practice even though they are immensely interesting from a theoretical standpoint. Upon closer inspection, feasible schemes turn out to be insecure and secure schemes turn out to be infeasible, at least for generic programs. In an attempt to remedy this situation, we will take a step back and relax our requirements somewhat. But before we introduce our approach to *special purpose program obfuscation*, we want to give a brief overview of previous work. We hope that the reader will take away an idea about the techniques upon which previous work on program obfuscation is based, and possible pitfalls, sometimes in terms of security and sometimes in terms of feasibility.

## 1.1 Previous Work

In this section we will give a brief overview of previous work on program obfuscation. The first part concerns itself with approaches to general purpose program obfuscation. The second part will be about special purpose program obfuscation.

### 1.1.1 General Purpose Obfuscation

To set the stage, let us introduce a few technical notions. In previous work on general purpose program obfuscation, the authors usually consider programs represented by *(Boolean) circuits*. A Boolean circuit is a finite directed acyclic graph whose edges are labelled by $\{0,1\}$ and whose vertices could for example be AND, OR, and NOT gates. Another possibility is for example that a circuit is comprised solely of NAND gates. Whatever the choice, the possible gates at the vertices must form a functionally complete set of Boolean functions. The *depth* of a circuit is then the depth of the corresponding acyclic graph.

Because in general we cannot even obfuscate a circuit directly, we need to translate it first into a more useful representation called a *branching program*. A branching program is an algebraic representation of a circuit, see Definition 8.1, Definition 8.3, and Definition 8.4. An important tool for translating circuits into branching programs is *Barrington's Theorem*.

**Theorem 1.1** (Barrington's Theorem [Bar89])**.** *If $f : \{0,1\}^n \to \{0,1\}$ is computable by a circuit of depth $d$, then $f$ is computable by a branching program of width 5 and length $\ell \leq 4^d$. In particular, if $d = O(\log n)$ then $\ell = \mathrm{poly}(n)$.*

Hence obfuscation schemes that try to handle generic programs require input circuits of depth $O(\log n)$ such that the corresponding branching programs are of polynomial length. Giel [Gie01] studied ways to optimise the branching program length for certain formulae.

After the impossibility result for general purpose virtual black-box obfuscation by Barak et al. [Bar+01], work on program obfuscation became less prevalent. It was only with the work on general purpose indistinguishability obfuscation by Garg et al. [Gar+13] that the cryptographic community developed major interest in program obfuscation again. Their obfuscator achieves the weaker notion of iO as suggested by Barak et al. [Bar+12] and is based on multilinear maps as constructed by Garg et al. [GGH13], Coron et al. [CLT13], Gentry et al. [GGH15], and Ma and Zhandry [MZ18].

The idea is to represent circuits by branching programs and subsequently represent those branching programs as a sequence of matrices. The matrices are then encoded using a secure multilinear map. The homomorphic properties of multilinear maps allow us to evaluate the encoded branching program on any input. The iO property of the obfuscator is reduced to certain security assumptions of the underlying multilinear maps. The multilinear maps exploit

properties of *lattices* (see Section 4.3) to gain the required homomorphic properties. Hence, the security assumptions make claims about the hardness of various lattice problems. First, it is not clear in every case how to reduce those often exotic problems to standard lattice problems. Second, heuristically the lattice dimensions need to be large to achieve good hardness guarantees and so implementations end up non-performant, require large storage sizes, or plainly are infeasible for modern computing hardware.

Unfortunately, as we will see in Section 8.4, all but the scheme of Ma and Zhandry [MZ18] have been broken in some setting. Thus, any indistinguishability obfuscator based on these schemes is insecure as it is broken for some circuits.

In the meantime, Brakerski and Rothblum [BR14], and Barak et al. [Bar+14b] have constructed general purpose VBB obfuscators in a *generic model* of multilinear maps. To the best of our knowledge, up until this point, there are no implementations of such generic multilinear maps.

### 1.1.2 Special Purpose Obfuscation

Instead of focusing on general purpose obfuscation, a different approach — as taken by many others — is to limit the scope and focus on *special purpose obfuscation*. Instead of trying to obfuscate arbitrary programs, the idea is to select specific problems (and view them as *programs*) and obfuscate those in a sensible manner. Since we are not finding ourselves in a generic setting any longer, we can for example try to prove that a specific special purpose obfuscator is VBB, input-hiding, or perfect circuit-hiding.

This approach might not yield the all-powerful primitives to construct almost every possible cryptographic building block, but it will certainly provide solutions to specific interesting cryptographic problems.

**Evasive Programs** Let us introduce the true star of this work, *evasive programs*. The class of evasive programs, as we will see, is a specific class of programs that is tractable in terms of obfuscation. By an evasive function or evasive program we informally understand a program which is *almost constant*, i.e. if we fix a program $P : \{0,1\}^n \to \{0,1\}$ then we call $P$ evasive if, without loss of generality, it outputs 0 for *almost all* inputs in $\{0,1\}^n$.

**Definition 1.1** (Evasive Program Collection)**.** *Let $\mathcal{P} = \{P_n\}_{n\in\mathbb{N}}$ be a collection of polynomial time programs such that every $P \in P_n$ is a program $P : \{0,1\}^n \to \{0,1\}$. The collection $\mathcal{P}$ is called* evasive *if there exists a negligible function $\epsilon$ such that for every $n \in \mathbb{N}$ and for every $y \in \{0,1\}^n$:*

$$\Pr_{P \leftarrow P_n}[P(y) = 1] \leq \epsilon(n).$$

In short, Definition 1.1 means that a random program from an evasive collection $\mathcal{P}$ evaluates to 0 on a fixed input $y$ with overwhelming probability. Finally, we call a member $P \in P_n$ for some $n \in \mathbb{N}$ an *evasive program* or an *evasive function*.

There are some classes of evasive functions that are already quite efficiently obfuscated, such as point functions [Can97; Wee05], hyperplane membership [CRV10], logical formulae defined by many conjunctions [Bra+16; BR17], pattern matching with wild cards [Bis+18; Bar+19], finding roots of a polynomial system of low degree [Bar+14a], compute-and-compare programs [WZ17a; GKW17], and more [LPS04]. All of the aforementioned problems are evasive problems:

- Hyperplane membership is the problem of testing whether a point lies on a given hyperplane. Depending on the algebraic dimensions, this problem can be made evasive. Canetti et al. [CRV10] have constructed a VBB obfuscator for it based on the discrete logarithm problem.

- Conjunctions and pattern matching with wildcards require the user to present a certain binary input vector. For a given secret $s \in \{0,1,\star\}^n$, where $\star$ denotes a special wildcard symbol, the conjunction tester accepts an input vector $x \in \{0,1\}^n$ if the entries of $x$ match all of the non-wildcard entries of $s$ — the wildcards are allowed to be arbitrary. Depending on the number of non-wildcard entries of the secret, this problem is evasive. A similar argument holds for roots of low-degree polynomial systems. Brakerski et al. [Bra+16; BR17] gave obfuscators based on LWE and multilinear maps. Bishop et al. [Bis+18] and Bartusek et al. [Bar+19] constructed obfuscators for pattern matching with wildcards from generic groups.

- Let $m, n \in \mathbb{N}$ be parameters. A compute-and-compare program takes an input $x \in \{0,1\}^n$, computes several functions $f_i : \{0,1\}^n \to \{0,1\}$ on the input and compares the resulting vector $(f_1(x), \ldots, f_m(x)) \in \{0,1\}^m$ to a fixed secret $s \in \{0,1\}^m$. This is an evasive problem in the parameter $m$. Wichs and Zirdelis [WZ17a] and Goyal et al. [GKW17] gave obfuscators based on LWE.

## 1.2 Outline and Contributions

In this work, we also take the special purpose route. We will study applications of special purpose program obfuscation in areas such as *secure biometric matching*, computing secret *conjunctions* on public inputs, and even computing secret *deterministic finite automata* (DFA) on public inputs. The last part will allow us to evaluate secret regular expressions on public inputs and somewhat generalises biometric matching and conjunctions.

In all cases we will prove that the obfuscator is a VBB obfuscator. In the biometric matching and conjunctions setting, input-hiding will be the sensible obfuscation notion. It should be hard to find a matching fingerprint for example and it should be hard to find an input accepted by a conjunction. On the other hand, perfect circuit-hiding will be our definition of choice for obfuscated DFAs. We need to consider theoretical results about learning DFAs from oracle access and adjust our notions accordingly. We will consider evasive DFAs for which it is hard to find an arbitrary number of distinct accepting inputs such that no adversary is able to learn the DFA description from oracle access.

In summary, apart from giving a — hopefully coherent — introduction into the most important moving parts of the big framework that is program obfuscation, this work is structured as follows:

- Chapter 2 presents a formal introduction to the various notions of program obfuscation used in the theoretical cryptography literature. In particular, we will focus on the definitions which we saw informally in the first part of this introduction.

- In Chapter 3 we will present the first of our contributions, as appeared in [ZGR19]. It is an attempt to bring closer together the two distinct worlds of applied and theoretical program obfuscation. In practical software design, developers envision an adversary to be a (possibly malicious) reverse engineer. The possible goals of a reverse engineer are for example to clone proprietary program logic and algorithms, or extract assets such a designed graphics or cryptographic material. Other goals include finding and abusing program flaws or possible back-door functionality. In this chapter we will present our attempt to construct secure *opaque predicates*. An opaque predicate is either an evasive function for which it is hard to determine an accepting input, or it is a constant function

which is hard to distinguish from non-constant functions. We also determine the class of programs for which it is sensible to use opaque predicates as an obfuscation tool in the first place.

- Chapter 4 introduces the mathematics background required in the subsequent chapters. We will see *continued fractions*, the related *Diophantine approximation* theorem, and a short introduction to lattices. In the introduction to lattices we will give a brief summary of several standard lattice problems, for example the *shortest vector problem* and the *bounded distance decoding problem*. These lattice problems form the basis for more specific computational problems such as the *short integer solution problem* and the *learning with errors problem*.

- Chapter 5 is the next part of our contributions, as appeared in [GZ19]. In it we will introduce a new computational problem, called the *modular subset product problem*. We can think of it as a multiplicative version of the *modular subset sum problem*, which itself is a special case of the more general *short integer solution problem*. We will give conjectures about the hardness of our new problem in a *search* and a *decisional* setting. We will present our cryptanalysis and discuss algorithms and hardness of the new problem in different parameter areas as well as its applicability in a post-quantum setting.

- In Chapter 6 we continue presenting our contributions, as appeared in [GZ19]. We will explain our construction of a secure obfuscator for the problem of *fuzzy Hamming distance matching*. This problem is to determine whether a binary input vector is within a Hamming ball of some fixed radius around a secret binary vector. Our obfuscator is based on the modular subset product problem of Chapter 5. In a security analysis, we will show that the obfuscator is VBB and input-hiding under the assumption that the problem of Chapter 5 is hard.

- Chapter 7 is another part of our contributions, as appeared in [GZ19]. We consider obfuscating the problem of *pattern matching with wildcards* or *conjunctions*. This problem is related to the fuzzy Hamming distance matching problem of Chapter 6 and we will show that there is a natural obfuscation solution based on the modular subset product problem. Similarly, we will show that our conjunction obfuscator is VBB and input-hiding under the assumption that the problem of Chapter 5 is hard.

- In Chapter 8 we recall the theory of branching programs and give a short proof of Theorem 1.1. We will see a generalisation of *bilinear pairings* to *multilinear maps* and *graded encoding schemes*. We will give a review of previous constructions of graded encoding schemes base on lattices and their security. Finally, we will see how Barrington's theorem allows us to encode programs as sequences of matrices and how to cryptographically hide those using graded encoding schemes. We hope that Chapter 8 gives the reader a good idea of how one might approach more general program obfuscation. It is also imagined to be an introduction of the prerequisites of the following chapter.

- Finally, in Chapter 9 we give the last of our contributions, as will appear in [GZ20]. It is an obfuscator for a certain class of *deterministic finite automata* (DFAs). We will only consider obfuscating the class of *evasive* DFAs in light of several results on learning DFAs from the input/accept/reject behaviour. DFAs can represent problems such as *regular expressions* and conjunctions. Hence, we obtain an obfuscator for evasive regular expressions and consequently solve the problem of *obfuscated substring matching*. Given a plaintext input

$x \in \{0,1\}^n$, obfuscated substring matching is the problem of identifying whether $x$ contains a secret substring $s \in \{0,1\}^m$, for some $m < n \in \mathbb{N}$. We achieve something even more general, as the substring can be given by a regular expression. We also obtain a new obfuscator for arbitrary conjunctions. The techniques we use to obfuscate evasive DFAs are similar to the techniques used in branching program obfuscation, which we introduce in Chapter 8. In a security analysis, we will show that the obfuscator for evasive DFAs is VBB and perfect circuit-hiding under a new computational assumption.

# 2 Preliminaries on Obfuscation

In Chapter 1 we have already skimmed over informal descriptions of the most important notions of program obfuscation used in theoretical cryptography. In this chapter we want to give the full formal picture by stating definitions for: Virtual black-box obfuscation, input-hiding obfuscation, and indistinguishability obfuscation. Remember that there is a definition similar to input-hiding obfuscation called perfect circuit-hiding obfuscation. It will turn out that for evasive programs, perfect circuit-hiding obfuscation is more closely related to virtual black-box obfuscation than to input-hiding obfuscation.

## 2.1 Virtual Black-Box Obfuscation

Virtual black-box obfuscation is one of the most straightforward definitions in terms of underlying ideas: An obfuscated program should behave like a black-box; it should not reveal anything more to an adversary than would be revealed from black-box access to the original program. Denote by $|P|$ the size of a program $P$ and by $T(P)$ its running time.

**Definition 2.1** (Distributional Virtual Black-Box Obfuscator [Bar+01; Bar+14a])**.** *Let $\mathcal{P} = \{P_n\}_{n\in\mathbb{N}}$ be a collection of polynomial time programs with input size $n$ and let $\mathcal{O}$ be a PPT algorithm which takes as input a program $P \in \mathcal{P}$, a security parameter $\lambda \in \mathbb{N}$ and outputs a program $\mathcal{O}(P)$ (which itself is not necessarily in $\mathcal{P}$). Let $\mathcal{D}$ be a class of distribution ensembles $D = \{D_\lambda\}_{\lambda\in\mathbb{N}}$ that sample $P \leftarrow D_\lambda$ with $P \in \mathcal{P}$. The algorithm $\mathcal{O}$ is a* VBB obfuscator for the distribution class $\mathcal{D}$ *over the program family $\mathcal{P}$ if it satisfies the following properties:*

- *Functionality preserving: There exists a negligible function $\epsilon(\lambda)$ such that for all $P \in P_{n(\lambda)}$*

$$1 - \Pr\left[\forall x \in \{0,1\}^{n(\lambda)} : P(x) = \mathcal{O}(P)(x)\right] \leq \epsilon(\lambda)$$

  *where the probability is over the coin tosses of $\mathcal{O}$.*

- *Polynomial slowdown: For every $\lambda \in \mathbb{N}$ and $P \in \mathcal{P}$, we have $T(\mathcal{O}(P)) \leq \text{poly}(T(P), \lambda)$.*

- *Virtual black-box: For every (non-uniform) polynomial time adversary $\mathcal{A}$, there exists a (non-uniform) polynomial time simulator $\mathcal{S}$ with oracle access to $P$, such that for every $D = \{D_\lambda\}_{\lambda\in\mathbb{N}} \in \mathcal{D}$, and every (non-uniform) polynomial time predicate $\varphi : \mathcal{P} \to \{0,1\}$:*

$$\left| \Pr_{P\leftarrow D_\lambda, \mathcal{O}, \mathcal{A}} \left[\mathcal{A}(\mathcal{O}(P)) = \varphi(P)\right] - \Pr_{P\leftarrow D_\lambda, \mathcal{S}} \left[\mathcal{S}^P(|P|) = \varphi(P)\right] \right| \leq \epsilon(\lambda)$$

  *where $\epsilon(\lambda)$ is a negligible function.*

In simple terms, Definition 2.1 states that a VBB obfuscated program $\mathcal{O}(P)$ does not reveal anything more than would be revealed from having black-box access to the program $P$ itself.

We also force the obfuscator output to be functionally equivalent to the original program — up to negligible probability; some constructions might introduce errors, as we will see.

Note that in a more general (and weaker) definition we allow extra auxiliary information attached to a program which is not required to be hidden and is inherently public.

**Definition 2.2** (Distributional Virtual Black-Box Obfuscator with Auxiliary Input)**.** *Let $\mathcal{P} = \{P_n\}_{n\in\mathbb{N}}$ be a family of polynomial time programs with input size $n$ and let $\mathcal{O}$ be a PPT algorithm which takes as input a program $P \in \mathcal{P}$, a security parameter $\lambda \in \mathbb{N}$ and outputs a program $\mathcal{O}(P)$ (which itself is not necessarily in $\mathcal{P}$). Let $\mathcal{D}$ be a class of distribution ensembles $D = \{D_\lambda\}_{\lambda\in\mathbb{N}}$ that sample $(P, \alpha) \leftarrow D_\lambda$ with $P \in \mathcal{P}$ and $\alpha$ some auxiliary input. The algorithm $\mathcal{O}$ is a* VBB obfuscator for the distribution class *$\mathcal{D}$ over the program family $\mathcal{P}$ if it is functionality preserving, implies polynomial slowdown, and satisfies the following property:*

- *Virtual black-box: For every (non-uniform) polynomial time adversary $\mathcal{A}$, there exists a (non-uniform) polynomial time simulator $\mathcal{S}$ with oracle access to $P$, such that for every $D = \{D_\lambda\}_{\lambda\in\mathbb{N}} \in \mathcal{D}$, and every (non-uniform) polynomial time predicate $\varphi : \mathcal{P} \to \{0, 1\}$:*

$$\left| \Pr_{P\leftarrow D_\lambda, \mathcal{O}, \mathcal{A}} \left[\mathcal{A}(\mathcal{O}(P), \alpha) = \varphi(P)\right] - \Pr_{P\leftarrow D_\lambda, \mathcal{S}} \left[\mathcal{S}^P(|P|, \alpha) = \varphi(P)\right] \right| \leq \epsilon(\lambda)$$

  *where $\epsilon(\lambda)$ is a negligible function.*

We will call an obfuscator that satisfies either of Definitions 2.1 and 2.2 a VBB obfuscator.

A definition that is more convenient to work with for proving security is *distributional indistinguishability*. To make sense of this, we will need the following definition that tells when two distributions are indistinguishable in a computational sense.

**Definition 2.3** (Computational Indistinguishability)**.** *We say that two ensembles of random variables $X = \{X_\lambda\}_{\lambda\in\mathbb{N}}$ and $Y = \{Y_\lambda\}_{\lambda\in\mathbb{N}}$ are* computationally indistinguishable *and write $X \overset{c}{\approx} Y$ if for every (non-uniform) PPT distinguisher $\mathcal{A}$ it holds that*

$$|\Pr[\mathcal{A}(X_\lambda) = 1] - \Pr[\mathcal{A}(Y_\lambda) = 1]| \leq \epsilon(\lambda)$$

*where $\epsilon(\lambda)$ is some negligible function.*

**Definition 2.4** (Distributional Indistinguishability [WZ17a])**.** *An obfuscator $\mathcal{O}$ for the distribution class $\mathcal{D}$ over a family of programs $\mathcal{P}$ satisfies* distributional indistinguishability *if there exists a (non-uniform) PPT simulator $\mathcal{S}$ such that for every distribution ensemble $D = \{D_\lambda\}_{\lambda\in\mathbb{N}} \in \mathcal{D}$ the following distributions are computationally indistinguishable*

$$(\mathcal{O}(P), \alpha) \overset{c}{\approx} (\mathcal{S}(|P|), \alpha) \tag{2.1}$$

*where $(P, \alpha) \leftarrow D_\lambda$. Here $\alpha$ denotes some auxiliary information.*

Note that the sampling procedure for the left and right side of Equation (2.1) in Definition 2.4 is slightly different. For both we sample $(P, \alpha) \leftarrow D_\lambda$ and for the left side we simply output $(\mathcal{O}(P), \alpha)$ immediately. On the other hand, for the right side we record $|P|$, discard $P$ and finally output $(\mathcal{S}(|P|), \alpha)$ instead.

It can be shown that distributional indistinguishability implies VBB security under certain conditions. To see this, we first define the augmentation of a distribution class by a predicate.

**Definition 2.5** (Predicate Augmentation [WZ17a])**.** *For a distribution class $\mathcal{D}$, its* augmentation under predicates $\mathrm{aug}(\mathcal{D})$ *is defined as follows: For any (non-uniform) polynomial time predicate $\varphi : \{0,1\}^* \to \{0,1\}$ and any $D = \{D_\lambda\}_{\lambda\in\mathbb{N}} \in \mathcal{D}$, the class* $\mathrm{aug}(\mathcal{D})$ *indicates the distribution $D' = \{D'_\lambda\}_{\lambda\in\mathbb{N}}$ where $D'_\lambda$ samples $(P,\alpha) \leftarrow D_\lambda$, computes $\alpha' = (\alpha, \varphi(P))$ and outputs $(P,\alpha')$. Here $\alpha$ denotes some auxiliary information.*

The idea is that we are allowed to attach a number of (sometimes "inevitable") predicates as auxiliary information to the obfuscated program. The following theorem shows that distributional indistinguishability for the larger augmented class $\mathrm{aug}(\mathcal{D})$ implies distributional VBB security for the class $\mathcal{D}$.

**Theorem 2.1** (Distributional Indistinguishability Implies VBB [WZ17a])**.** *For any family of programs $\mathcal{P}$ and a distribution class $\mathcal{D}$ over $\mathcal{P}$, if an obfuscator satisfies distributional indistinguishability (Definition 2.4) for the class of distributions* $\mathrm{aug}(\mathcal{D})$ *then it also satisfies distributional VBB security for the distribution class $\mathcal{D}$ (Definition 2.2).*

*Proof.* See [Bra+16, Lemma 2.2] and [WZ17b, Theorem 3.4]. □

Hence, if we can show that the (augmented) program class can be simulated according to Equation (2.1), then the obfuscator is a VBB obfuscator for the (non-augmented) program class.

## 2.2 Input and Perfect Circuit-Hiding Obfuscation

In this section, we will consider obfuscation definitions for targets that are evasive functions. Recall the notion of input-hiding obfuscation that we saw in Chapter 1. The obfuscated program should not reveal anything about possible inputs that are accepted by the original program. Similarly, perfect circuit-hiding obfuscation takes this idea and applies it to the actual representation of the programs. The obfuscated program should not reveal anything about the "implementation details" (circuit, algorithm description, source code) of the original program. We will model this by letting the adversary answer arbitrary predicates about the obfuscated program. Perfect circuit-hiding requires that the adversary cannot answer correctly with a better advantage than randomly guessing.

**Definition 2.6** (Input-Hiding Obfuscator [Bar+14a])**.** *An obfuscator $\mathcal{O}$ for a collection of evasive programs $\mathcal{P}$ is* input-hiding*, if for every PPT adversary $\mathcal{A}$ there exists a negligible function $\epsilon$ such that for every $n \in \mathbb{N}$ and for every auxiliary input $\alpha \in \{0,1\}^{\mathrm{poly}(n)}$ to $\mathcal{A}$:*

$$\Pr_{P\leftarrow P_n, \mathcal{O}, \mathcal{A}}[P(\mathcal{A}(\alpha, \mathcal{O}(P))) = 1] \leq \epsilon(n).$$

To summarise, Definition 2.6 states that given the obfuscated program $\mathcal{O}(P)$ it is hard to find an input that evaluates to 1.

**Definition 2.7** (Perfect Circuit-Hiding Obfuscation [Bar+14a])**.** *An obfuscator $\mathcal{O}$ for a collection of evasive programs $\mathcal{P}$ is* perfect circuit-hiding*, if for every PPT adversary $\mathcal{A}$ there exists a negligible function $\epsilon$ such that for every $n \in \mathbb{N}$, every balanced predicate $\varphi : P_n \to \{0,1\}$, and every auxiliary input $\alpha \in \{0,1\}^{poly(n)}$ to $\mathcal{A}$:*

$$\Pr_{P\leftarrow P_n, \mathcal{O}, \mathcal{A}}\left[\mathcal{A}(\alpha, \mathcal{O}(P)) = \varphi(P)\right] \leq \frac{1}{2} + \epsilon(n).$$

Compare Definition 2.7 to Definition 2.6. Both tell us that for the adversary $\mathcal{A}$ it should be hard to learn a particular detail about the program $P$ from the obfuscated program $\mathcal{O}(P)$. In the input-hiding case, it should be hard to learn an input that is accepted by $P$ (here we define *acceptance* by $P$ evaluating to 1 whereas we say the input was *rejected* if $P$ evaluates to 0). On the other hand, in the perfect circuit-hiding case it should be hard to learn an arbitrary predicate that takes as an input the program $P$. We can think of the different predicates as *measuring* something about the circuit comprising $P$ and that is why we use the terminology circuit-hiding.

By examining Definition 2.7, we can already guess that perfect circuit-hiding obfuscation is related to virtual black-box obfuscation. And indeed, Barak et al. [Bar+14a, Theorem 2.1] showed that for evasive programs Definition 2.1 is equivalent to Definition 2.7.

**Theorem 2.2.** *Let $\mathcal{O}$ by an obfuscator for an evasive program collection $\mathcal{P}$. Then $\mathcal{O}$ is perfect circuit-hiding if and only if $\mathcal{O}$ is a virtual black-box obfuscator.*

*Proof.* See [Bar+14a, Theorem 2.1]. □

## 2.3 Indistinguishability Obfuscation

Finally, we want to give the the last important obfuscation definition, namely indistinguishability obfuscation. Upon closer inspection it is much weaker than any of the Definitions 2.1, 2.6, and 2.7. We will now see that there is a good reason for this weak definition. Indistinguishability obfuscation was introduced because of one inherent problem with the VBB definition: A generic VBB obfuscator is impossible. Barak et al. [Bar+01] showed that conditional on the existence of a one-way function, there exists a family of *inherently unobfuscatable functions* $\mathcal{F}$ and a predicate $\pi : \mathcal{F} \to \{0,1\}$ such that

1. given any program that computes a function $f \in \mathcal{F}$, the value of $\pi(f)$ can be efficiently computed, yet
2. given oracle access to a (randomly selected) function $f \in \mathcal{F}$, no efficient algorithm can compute $\pi(f)$ much better than random guessing.

Hence, for any function $f$ of this unobfuscatable family $\mathcal{F}$, there cannot exist a hypothetical obfuscator $\mathcal{O}$ such that $\pi(\mathcal{O}(f))$ cannot be efficiently computed (as would be required by the virtual black-box assumption of Definition 2.1).

**Definition 2.8** (Indistinguishability Obfuscator (iO) [Bar+12])**.** *A uniform PPT algorithm $\mathcal{O}$ is called an* indistinguishability obfuscator *(iO) for a program class $\mathcal{P} = \{P_\lambda\}_{\lambda \in \mathbb{N}}$ if it satisfies the following properties:*

- *Functionality preserving: For all security parameters $\lambda \in \mathbb{N}$ and for all $P \in P_\lambda$ we have that*

$$\Pr_{P' \leftarrow \mathcal{O}(P)} \left[ \forall x \in \{0,1\}^{n(\lambda)} : P(x) = P'(x) \right] = 1$$

- *For any (not necessarily uniform) PPT distinguisher $\mathcal{D}$, there exists a negligible function $\epsilon(\lambda)$ such that the following holds: For all security parameters $\lambda \in \mathbb{N}$ and for all pairs of programs $P, Q \in P_\lambda$ we have that if $P(x) = Q(x)$ for all inputs $x \in \{0,1\}^{n(\lambda)}$, then*

$$|\Pr[\mathcal{D}(\mathcal{O}(P)) = 1] - \Pr[\mathcal{D}(\mathcal{O}(Q)) = 1]| \leq \epsilon(\lambda).$$

Recall the example we gave in Chapter 1. We considered $f_k, g_k : \{0,1\}^{128} \mapsto \{0,1\}^{128}$, two functions that encrypt a 128-bit block of data with the AES encryption algorithm under a fixed key $k$. Given an indistinguishability obfuscator $O$, it should be hard to distinguish between $O(f_k)$ and $O(g_k)$. A simple way to think about $O$ is in the form of an *optimiser* which, from the class of all possible circuits, selects a certain canonical one. This rather inefficient obfuscator outputs the same circuits for $f_k$ and $g_k$ and hence $O(f_k)$ and $O(g_k)$ are trivially indistinguishable.

# 3 Applied Program Obfuscation

Before fully touching the theoretical cryptography side of program obfuscation, it is interesting to explore briefly the state of the applied side. Today, computer software employs several obfuscation *techniques* to protect source code, program logic, cryptographic secrets, and other assets from *reverse-engineering*. Reverse engineering is a technique related to recovering these assets from *compiled* programs and other resources. The obfuscation techniques in use are somewhat connected to obfuscation in the cryptographical sense, albeit focusing more on heuristic security rather than formally provable security. As such, most research on applied program obfuscation does not provide rigorous modelling nor security reductions to hard computational problems. In some cases, the idea is to thwart reverse engineers long enough such that the software remains secured during a certain time window after it first enters the market. Other areas which employ similar protections are for example digital rights management schemes used in the film and digital games industry as well as *white box cryptography* implemented in smart cards and television set-top boxes.

Some of the research that went into the writing of this thesis had the goal to explore whether some of the techniques from theoretical cryptography could benefit this *applied* side of obfuscation and whether both extremes could be brought closer together. For this task we focused on one widely used technique of applied obfuscation: *Opaque predicates*. We will study them and their applicability in this chapter.

## 3.1 Opaque Predicates

Applied program obfuscation has the goal of obfuscating control flow. Opaque predicates [CTL97; CTL; Arb02; MC06; MT06; SS16] are commonly used to add complexity to control flow and to insert superfluous code or watermarks [GCT05; PCH16]. For example, a reverse-engineer may compute the *control flow graph* (CFG) of a program and then try to deduce something about the structure of the program from this information.

Opaque predicates are traditionally constant predicates (always true or always false) that have been obfuscated with the intention of hiding the fact that they are constant. One can add complexity to the CFG by introducing opaque predicates that appear to create extra branches and program blocks, or to add code that looks relevant to the program, even though these blocks never execute when the program runs.

There is a large literature on breaking control flow obfuscation based on opaque predicates, and we survey some of it in Section 3.3. Many proposals can be broken by using static analysis [Dal+06; Yad+15; MVD06; PAS17] and/or dynamic analysis [MR09; GG10; Bio+17; Min+15; Ban+16].

The principal goal is to hide the *true* control flow graph (CFG) of a program [Cho+01]. In general, a control flow transformation must ensure that the original control flow is contained in the newly generated CFG while introducing artificial complexity. Hence, control flow obfuscators often introduce superfluous and/or inert instructions interleaved with the original

ones. Special care must be taken so that the new instructions do not interfere with the original computation. This opens the schemes up for different types of attacks that filter the superfluous instructions from an obfuscated sequence by means of dynamic program analysis and *taint tracking* [UDM05; Wan+08; Sha+09; SAB10]. These attacks seek to remove the added instructions and restore the original sequence. Other dynamic approaches are based on symbolic or *concolic* (concrete + symbolic) execution using SMT solvers [YD15; Ban+16].

We survey attacks that detect opaque predicates and remove superfluous code. In particular, as we discuss in detail in Section 3.3.4, if the opaque predicates have no resemblance (either syntactically or semantically) to the naturally occurring predicates in a program then they can be easily removed by a pattern-matching attack. Such an attack is for example possible for programs obfuscated by *Obfuscator-LLVM* [Jun+15].

Another powerful type of attack tries to identify constant predicates by evaluating them on chosen inputs or by examining execution traces. A simple observation that seems to have been ignored in the obfuscation literature is that most naturally occurring predicates are easily confirmed to be non-constant using this approach. For example, we have performed an experiment using open source code (OpenSSL and Mbed TLS) to study predicates that naturally appear in programs. We found that approximately 91% (in the case of OpenSSL) and 83% (in the case of Mbed TLS) of all constant comparisons were comparisons with zero. Hence, simply evaluating a predicate at 0 is sufficient to detect an opaque (always false) predicate with high accuracy.

We will argue that opaque predicates can only be applied securely when the original program has certain features (in particular, that it contains predicates that are satisfied only for rare and hard-to-find inputs). We will give an integrated approach to control flow obfuscation that can be applied when the conditions which we present in Section 3.4 are satisfied. The main idea is to ensure that opaque predicates added to the program are *indistinguishable* from obfuscations of real predicates already in the program. For this we exploit the well-known and widely used obfuscation technique to obfuscate point functions using hash functions [Can97; CN09]. More importantly, we use these point function obfuscators to build opaque predicates from hash functions that are indistinguishable from obfuscations of real predicates.

## 3.2 Notation and Definitions

We will introduce some important notations and definitions that shall remain valid for this and only this chapter.

### 3.2.1 Classes of Predicates

Here we give definitions of certain classes of predicates commonly found in programs. Let $X$ be a finite set. We usually want $X$ to be of exponential size, for example $X = \{0,1\}^n$ for some $n \in \mathbb{N}$. A *predicate* on $X$ is then a function $P : X \to \{0,1\}$. The element 1 will usually represent the Boolean *true* and the element 0 the Boolean *false*. A predicate is *constant* if it is the constant function. We only consider the following types of predicates:

- We call a predicate $P(x) =$"$x == c$", where $c$ is a constant, a *constant comparison*.
- We call a predicate $P(x, y) =$"$ax + b == y$", where $a, b$ are constants and $x, y$ variables, a *variable comparison*. Here the domain $X$ of $P$ could for example be a product of any two of $\mathbb{N}, \mathbb{Z}, \mathbb{R}$.

**Definition 3.1** (Opaque Predicate Collection). *Let $\mathcal{P} = \{P_n\}_{n\in\mathbb{N}}$ be a collection of polynomial time predicates such that every $P \in P_n$ is a predicate $P : \{0,1\}^n \to \{0,1\}$. The collection $\mathcal{P}$ is called* opaque *if for every PPT adversary $\mathcal{A}$ and for every $n \in \mathbb{N}$, there exists a negligible function $\epsilon(n)$ such that the probability that $\mathcal{A}$ can decide whether $P \leftarrow P_n$ is a constant predicate, is bounded by $\epsilon(n)$.*

Recall Definition 1.1, which introduced the notion of an evasive program collection. Any predicate that belongs to such an evasive collection is called evasive; it evaluates to *false* for almost all inputs $x \in X$.

Since it is hard to find an input $x$ that satisfies an evasive predicate class, this class of predicates is a good candidate for obfuscation, and there is a large literature on the problem [Can97; Wee05; Bar+14a; CRV10].

### 3.2.2 Control Flow Graph

We assume that a general program is made up of many smaller building blocks, namely functions. In the following we will focus on obfuscating the *control flow graph* (CFG) of a function. The CFG $G = (V, E)$ is the graph consisting of the set of all *basic blocks* $V$ and the set of all *control flow edges* $E \subseteq V \times V$ of a program. A basic block $B_i \in V, i \in I$ is an ordered tuple $B_i = (\beta_j)_{j\in J}$ of *program instructions* $\beta_j$. A control flow edge can also be represented by the ordered pair $(i, j)$ with $i, j \in I$ modelling the control flow transfer from basic block $B_i$ to $B_j$.

A program instruction $\beta_j$ generally models the assignment of register or memory locations with the result of a function applied to several values taken from registers or memory locations. We shall denote this by writing $\mathbf{y} \leftarrow \mathbf{F}(\mathbf{x})$ where $\mathbf{x}$ is the vector of inputs and $\mathbf{y}$ is the memory location of assigned outputs. Additionally, there exists a special class of instructions, namely those that result in control flow transfers. These *branch* instructions terminate the basic block tuple of instructions and can never appear in any other position. A branch may additionally depend on the output value of a predicate $y = P(\mathbf{x})$. In this conditional case we write $\mathbf{BCOND}_y(B_0, B_1)$ which results in a branch to the target block $B_y$ with $y \in \{0,1\}$.

## 3.3 Attacks on Opaque Predicates

We now survey techniques to detect obfuscated constant predicates. The possible methods involve human interaction as well as automated algorithmic interaction [Dal+06; Min+15]. It is safe to say that generic human interaction is hard to model and hence we will consider automated attacks only.

Let $P : X \to \{0,1\}$ be a predicate. We wish to automate determining whether $P$ is constant or not. If we can solve this problem, we can build an automated reverse engineering tool that takes an obfuscated program, enumerates all its predicates, determines which are constant, and then removes the predicates and any unreachable blocks from the control flow graph. By iterating the process an adversary can try to recover the original version of the program or a close version of it.

### 3.3.1 Brute Force Search

If $X$ is a small enough set, then one can efficiently evaluate the predicate $P$ for all $|X|$ possible values $x \in X$ and so determine whether $P$ is a constant predicate. For example, if $x$ is a 32-bit integer then this requires $2^{32}$ executions of the program segment, which is non-trivial but feasible. On the other hand, if $x$ is a 64-bit integer then this requires $2^{64}$ executions of the

program segment and this is probably more work than anyone wants to spend on a simple reverse-engineering task. This topic is studied in [Dal+06, Section 4].

### 3.3.2 Evaluate at Zero

We have performed some measurements on real-world code. We have chosen the source code of two open source cryptographic libraries: OpenSSL 1.1.0f and Mbed TLS 2.5.1. OpenSSL and Mbed TLS are open source collections of routines implementing cryptographic algorithms and protocols. They consist of roughly 470K lines of code and 155K lines of code, respectively, and are widely used in practice [14]. Our analysis shows that typical programs exhibit a large number of constant comparisons. OpenSSL for example has 36855 integer comparisons of which 28890 are with constants. Approximately 91% of these constants are zero. Similary, for Mbed TLS, 17062 of 21038 integer comparisons are with constants. Here, roughly 83% of these constants are zero. Hence a simple strategy to distinguish a real predicate from an opaque constant (always false) predicate is to evaluate it at zero. If the predicate is true for 0 then it is not a constant false predicate; if the predicate is false for 0 then it can be flagged for further analysis. The power of this technique seems to not have been noticed by obfuscation researchers. The significance of the constant 0 stems from the common structure of most programs; 0 is traditionally used to indicate Boolean *false* and error conditions. Checks for those are common in most programs and usually translate into constant predicates.

### 3.3.3 Probabilistic Check

Instead of evaluating a predicate on all $x \in X$, an adversary could compute $P(x)$ for a number of randomly chosen $x \in X$. If $P(x)$ is constant for these random inputs, the adversary might suspect that $P$ is a constant predicate and hence flag it for removal from the program.

This type of attack is studied in detail in [Dal+06, Section 3]. They call a "false negative" the situation where a probabilistic or dynamic (see Section 3.3.7) attack incorrectly classifies a non-constant predicate as being constant. They conducted an experimental study using real programs and found false negative rates of between 20 and 40 percent.

In other words, computing predicates in real-world programs on random inputs is not a reliable method to determine if a predicate is constant, and the removal of code blocks based on such an attack is not safe. But this check is sensible as a pre-processing step before applying more sophisticated methods to determine if a predicate is constant or not.

### 3.3.4 Pattern Matching

We have surveyed the literature [CTL; Arb02; MC06; Jun+15], as well as studied samples of code produced by both free and commercial obfuscation solutions, to collect examples of constant predicates. Surprisingly, relatively few predicates have been proposed, and they are used over and over again. Figure 3.1 lists some of the most-used constant predicates.

A fundamental problem, mentioned in [CN09, Section 4.4], is that these predicates may not otherwise naturally occur in code. If the introduced predicates are of a different *kind* compared to the real code, then it is easy to detect and remove them purely by searching for that syntax in the code. This leads to a dictionary attack [Arb02; CN09; MC06], where one pattern-matches all predicates in a program against known constant predicates, such as the ones in Figure 3.1.

As an example, we consider the open source implementation of *Obfuscator-LLVM* [Jun+15]. It uses a unique constant predicate $P(x, y) = (y < 10) \vee (2|x(x + 1))$ where $x$ and $y$ are global

| |
|---|
| $7y^2 - 1 \neq x^2$ |
| $2 \mid x(x+1)$ |
| $3 \mid x(x+1)(x+2)$ |
| $x^2 > 0$ |
| $7x^2 + 1 \not\equiv 0 \mod 7$ |
| $x^2 + x + 7 \not\equiv 0 \mod 81$ |
| $x > 0$ for $x \in I$ random where $I \subset \mathbb{Z}_{>0}$ is a random interval |

Figure 3.1: List of constant predicates often found in literature on obfuscation solutions. These predicates have been constructed to always evaluate to the same result independent of the input value. Here the values $x, y$ are usually considered as an element of $\mathbb{Z}/2^n\mathbb{Z}$ for some $n \in \mathbb{N}$.

program variables. In this case we can simply apply pattern matching to detect all opaque predicates in the obfuscated program.

Of course, it is easy to create additional constant predicates that are not listed in Figure 3.1, but this does not seem to have been done in any large-scale way in current obfuscation solutions. One can also use the approach in [Arb02] to introduce a class of constant predicates that is parametrised by some parameter $n$ (a multiple of an algebraic identity for example). Even though this method yields a large set of different constant predicates, it is still possible to detect them using a pattern matching approach.

We believe that pattern matching attacks have been neglected in the obfuscation literature, and that they can be used to attack many recent obfuscation schemes that introduce predicates in a "non-organic" way. For example, [PCH16] introduces non-determinism and complexity of control flow by using particular branch instructions based on a random number generator. These predicates are of a very special form and are not likely to resemble the predicates in the original program. If an attacker knows that this "probfuscation" approach has been used then it is plausible that these branches can be removed using a pattern-matching attack.

As another example, [Pal+00; XMW16] describe dynamic opaque predicates of a very special form that rely on certain correlated variables. Once again, if an attacker knows that this approach has been used then it is plausible that a pattern-matching approach can identify these predicates and therefore simplify the control flow graph. Similar remarks apply to opaque predicates based on aliasing [CTL] or 3SAT [SS16]. In short, *predicates added to the program have to resemble the syntax of naturally occurring program code*. We even go as far as to say that they need to be *indistinguishable* from already existing predicates to prevent an attacker from applying the aforementioned techniques.

### 3.3.5 Automated Proving

Another approach to determining if a program segment computes a constant predicate is to run an SMT-solver. We shall call an obfuscated predicate $P : X \to \{0, 1\}$ SMT-solvable if a SMT solver is able to efficiently answer whether $P$ is constant or not. It is clear that this strongly depends on the size of the space $X$ and the *computational complexity* of $P$. *Symbolic execution* backed by SMT solvers is possible in a static and a dynamic context [CDE08; SS15].

### 3.3.6 Taint Analysis

Instead of considering a predicate itself, we can look at those basic blocks in the CFG that are potentially selected by the conditional expression. Figure 3.2 presents in (a) a basic block and in (b) an obfuscated constant predicate (that is always true) and a basic block that is never executed. If the program instructions in this superfluous block are chosen poorly then there may be no data dependency on the input variables of the obfuscated function. Taint analysis [Wan+08; SAB10] can then be used to identify basic blocks with no data dependencies. In this case, we may subsequently flag the preceding predicate as potentially constant. In the context of compiler construction, optimization and removal of dead code are important topics [KRS94a; KRS94b].

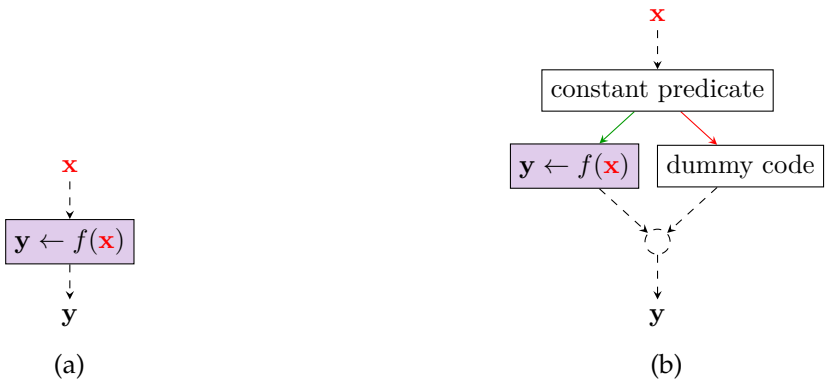



Figure 3.2: Example of extracting the original CFG from obfuscated CFG using taint analysis. The nodes that have no data dependence on the input can be ignored when extracting the logic that operates on the input. In (a) the input basic block produces an output **y** that depends on the input **x**. Note that the inserted basic block in (b) does not depend on the input **x**.

### 3.3.7 Execution Traces

This is essentially a dynamic version of the probabilistic check mentioned in Section 3.3.3. One executes the obfuscated program in a controlled environment and records the computed values of all predicates. Since the predicates are being evaluated on actual executions of the program, it is possible to correctly identify nearly constant predicates. This approach allows to determine that predicates are real predicates from the original program, however one cannot immediately conclude that program blocks that have not been executed are superfluous code. Such a strategy might incorrectly classify many predicates as opaque when they are not [Dal+06].

## 3.4 When Are Opaque Predicates Useful?

As already hinted, we conjecture that it only makes sense to use obfuscated predicates as an applied obfuscation tool under the following two conditions:

1. The program being obfuscated has a lot of conditional branches, including many constant comparisons ($x == c$) with "random" constants $c$, or variable comparisons ($ax + b == y$), again with "random" constants $a, b$.
2. There is an algorithm to generate superfluous basic blocks whose instructions and data dependencies are indistinguishable from real basic blocks in the program.

The first condition is needed to avoid the attacks mentioned in Section 3.3.2 and Section 3.3.3. The second condition is needed to prevent the attacks in Section 3.3.6.

Both these conditions depend on the program (or class of programs) being obfuscated. The second condition is implicit in all previous work on control flow obfuscation (and is made explicit in [PCH16]).

There are types of programs for which these two conditions do not hold. This statement is obvious but does not seem to have been clearly stated in the applied obfuscation literature. The implication is that control flow obfuscation using opaque predicates is not appropriate for some programs.

## 3.5 Obfuscating Predicates

For the remainder of this chapter we restrict to situations when the two conditions in Section 3.4 hold. We will give an obfuscation approach that avoids the attacks we discussed in Section 3.3.4 and Section 3.3.5.

The main idea is to simultaneously obfuscate the existing predicates in the program and introduce opaque predicates. The obfuscated real predicates need to be indistinguishable from the opaque predicates. We will not use constant predicates as opaque predicates but rather evasive predicates. For these we can prove indistinguishability if the constant predicates are from an appropriate distribution.

### 3.5.1 Obfuscating Constant Comparisons

It is standard to obfuscate a password check (constant comparison) "$x == \mathbf{pw}$" using a cryptographic hash function $H$ by computing $h = H(\mathbf{pw})$ and publishing the obfuscated predicate "$H(x) == h$" [LPS04; Cre+16].

We use the notation $u\|v$ for the concatenation of two binary strings $u, v$. The key ideas to obfuscate constant comparisons in a program and introduce new opaque predicates are as follows:

- Let $H$ be a cryptographic hash function. Let $C$ be the class consisting of all constant comparison predicates $P(x) =$ "$x == c$" where $x$ has $k$ bits. To obfuscate a comparison predicate "$x == c$" for some $k$-bit constant $c$, the obfuscator chooses a hash function $H$ with $n$-bit output (where $n > k$), chooses a random $t \in \{0,1\}^{n-k}$ and computes $h = H(c\|t)$. The obfuscated predicate $P(x)$, which is specified by the pair $(t, h)$, computes $y = H(x\|t)$ and then checks if $y = h$. It is clear that the obfuscated predicate outputs true on the correct input $x = c$. With overwhelming probability, it outputs false on all other inputs.

- Looking ahead we now explain how we will use the same construction to generate an opaque predicate: Choose a random $h \in \{0,1\}^n$ and $t \in \{0,1\}^{n-k}$ and publish the evasive predicate $P(x)$ that computes $y = H(x\|t)$ and checks if $y = h$. With overwhelming probability, the predicate is always false since there would be no $k$-bit value $x$ that satisfies the equation when $n > k$.

The following lemma and theorem are the basic tools in our security analysis.

**Lemma 3.1.** *Let $H : \{0,1\}^n \to \{0,1\}^n$ be a cryptographic hash function. Define an oracle $O$ that takes as input $(k, t \in \{0,1\}^{n-k}, y \in \{0,1\}^n)$, where $1 \le k \le n$, and returns 1 if there exists $x \in X_k$ such that $H(x\|t) = y$ and 0 otherwise. Then given $y \in \{0,1\}^n$ one can, using polynomially many calls to $O$, compute some $x \in \{0,1\}^n$ such that $H(x) = y$ or determine that no such $x$ exists.*

*Proof.* Calling $O(n, \epsilon, y)$, where $\epsilon$ is the empty string, decide if $x$ exists or not. If $x$ exists, set $t_0 = \epsilon$ and iterate the following process for $i = 0, 1, 2, \ldots$: Given that $O(n - i, t_i, y) = 1$ we call $O(n - (i + 1), 0 \| t_i, y)$. If the result is 1 then set $t_{i+1} = 0 \| t_i$, else set $t_{i+1} = 1 \| t_i$. On termination we set $x = t_n$. □

**Theorem 3.1.** *Let $X = \{0,1\}^k \subseteq \{0,1\}^n$ and let $H : \{0,1\}^n \to \{0,1\}^n$ be a preimage-resistant hash function. Let $C$ be the class of predicates $P(x) =$ "$x == c$" where $c$ is uniformly sampled from $X$. Then there does not exist any efficient adversary (i.e. running time polynomial in $k$) that, for any obfuscated predicate from $C$, can determine whether the predicate is constant or not.*

*Proof.* Let $A$ be an efficient adversary that, for all $k$, takes an obfuscated predicate on $X = \{0,1\}^k$ and determines if the predicate is constant or a constant comparison. Then $A$ performs the function of the oracle $O$ in Lemma 3.1. Hence one can use $A$ (executed at most $n$ times) to compute a preimage of $H$. But this contradicts the assumption that the hash function is preimage-resistant. □

### 3.5.2 Obfuscating Variable Comparisons

Similarly, we can obfuscate variable point comparisons $P(x, y) =$ "$ax + b == y$" using hash functions. Here $a$ and $b$ are constants and $x, y \in \mathbb{Z}$ are the variables. Our solution will hide $b$, which is enough to make it hard to find a solution $(x, y)$ to the predicate. Note that in practice, calculations are done using $k$-bit *machine integers*, so we will consider arithmetic modulo $2^k$.

- Suppose that $x$ and $y$ are at most $k$ bits in length and let $X = [0, 2^k)$ be the set of $k$-bit integers. So $x, y \in X$. Let $n > k$ and let $H : \{0,1\}^n \to \{0,1\}^n$ be a preimage-resistant hash function. The obfuscator chooses a random $t \in \{0,1\}^{n-k}$ and a random $r \in X$, sets $h = H(r \| t)$ and $u = r + b \mod 2^k$ and publishes $a, u, h$. On input $x, y$ the obfuscated program computes $H(ax + u - y \mod 2^k \| t)$ and then checks if this equals $h$. Note that $h$ hides the value $r$ and so $u$ hides the real value $b$. It is clear that the obfuscated program is correct: any input $(x, y)$ for which $P$ is true will also evaluate to true. The probability of a false positive can be made negligible by taking $n$ large enough (e.g. $n = 3k$).

- Again, we explain how to make an opaque predicate of the same form. We choose a random $t \in \{0,1\}^{n-k}$, random $a, u \in X$, and a random $h \in \{0,1\}^n$ and publish the obfuscated predicate $P(x, y)$ that checks whether $h = H(ax + u - y \mod 2^k \| t)$. With high probability there is no input $(x, y)$ for which this predicate evaluates to true.

### 3.5.3 Control Flow Graph Modification

We start with a high-level overview. The approach is to first obfuscate all point-comparison and variable-comparison predicates in the original program. The second step is to introduce superfluous control flow, by using constant predicates. The key idea is to obfuscate random evasive predicates so that they are indistinguishable from the obfuscated original predicates in the program. To make the control flow more complex, we take a full or partial CFG and prepend it with a constant predicate. The inserted CFG can be of arbitrary complexity as it will never be executed. The situation is depicted in Figure 3.3.

This way of introducing opaque predicates avoids the pattern-matching attack from Section 3.3.4: an attacker cannot simply remove all predicates that match the pattern of opaque predicates. Some of them are real comparisons and their removal breaks the correctness of the program.

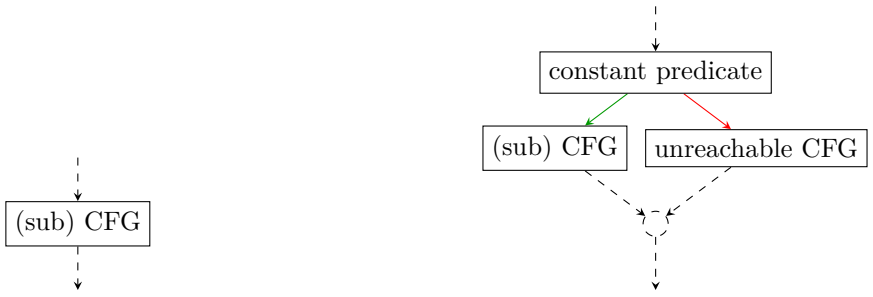



Figure 3.3: Obfuscating control flow graph using a constant predicate. The input graph depicted in (a) is prepended by a constant predicate and random superfluous code is inserted in the branch that is never taken. This produces the output graph depicted in (b).

We now detail the first stage of the obfuscator. Consider a constant point comparison $P(x) =$"$x == c$". The obfuscator $O$ transforms this predicate according to

$$\begin{bmatrix} x \leftarrow \cdots \\ y \leftarrow \mathbf{CMP}(x, c) \\ \mathbf{BCOND}_y(B_0, B_1) \end{bmatrix} \overset{O}{\mapsto} \begin{bmatrix} x \leftarrow \cdots \\ h \leftarrow \mathbf{H}(x \| t) \\ y \leftarrow \mathbf{CMP}(h, h_c) \\ \mathbf{BCOND}_y(B_0, B_1) \end{bmatrix}$$

where $h_c = H(c\|t)$ and $\mathbf{CMP}(a, b)$ is the operation that compares $a$ and $b$ and returns true or false accordingly.

The analogous process for variable comparisons follows the construction of Section 3.5.2. Consider the simple variable comparison $P(x, y) =$"$x == y$". First, we generate random integers $r, t$ and compute the hash $h_r = H(r\|t)$. The comparison predicate is then obfuscated as follows:

$$\begin{bmatrix} x \leftarrow \cdots \\ y \leftarrow \cdots \\ z \leftarrow \mathbf{CMP}(x, y) \\ \mathbf{BCOND}_z(B_0, B_1) \end{bmatrix} \overset{O}{\mapsto} \begin{bmatrix} x \leftarrow \cdots \\ y \leftarrow \cdots \\ h \leftarrow \mathbf{H}(x - y + r \| t) \\ z \leftarrow \mathbf{CMP}(h, h_r) \\ \mathbf{BCOND}_z(B_0, B_1) \end{bmatrix}.$$

### 3.5.4 Security Analysis

Recall that Theorem 3.1 tells us that a preimage-resistant hash function gives us a strong obfuscated constant predicate. This analysis also depends on the distribution of constant comparison predicates that appear in a program. We have informally stated this in Section 3.4. Formally, we require that the distribution of constant comparison predicates has high enough entropy. In practice, computers use 64-bit processors, although in some cases 128-bit and higher are possible. This means that constants generally are 64-bits in size. We do not know of any work that has determined the expected distribution and its entropy of constants in typical programs. We measured the number of the constant 0 appearing in constant comparisons in two example codebases, see Section 3.3.2 for the results.

As explained in Sections 3.3.3 and 3.3.7, simple probabilistic and dynamic attacks are likely to incorrectly flag real predicates as opaque and result in real code being removed by an attacker. It is true that an adversary can identify some predicates (such as loop terminations) that will be

seen to be not constant. We require that the real program has sufficiently complex control flow for our solution to resist dynamic attacks. We furthermore claim that the other static attacks cannot distinguish between the obfuscations of the original and inserted opaque predicates. Using an SMT solver (see Section 3.3.5) will not help an adversary to identify the constant predicates, assuming the hash function is preimage-resistant. Due to our assumption, the superfluous basic blocks have a dependency on the input variable(s) $\mathbf{x}$ in the inert CFG in Figure 3.2b as well as generate an output dependency for the output variable(s) $\mathbf{y}$. This way both possible execution paths will depend on $\mathbf{x}$ and generate dependencies for $\mathbf{y}$ and so an adversary will not be able to identify the superfluous path without having to solve the predicate. Hence our approach gives protection against taint analysis, as described in Section 3.3.6.

# 4 Continued Fractions, Lattices, and All That

In Chapter 5 we will introduce a new computational problem that is rooted in number theory which we call the *modular subset product problem* (MSP) over a base ring $\mathbb{Z}/q\mathbb{Z}$ for some prime $q$. Before we introduce this new problem, we need to recall some important number theoretical facts. In the easy parameter range, solving MSP involves computing continued fraction expansions. In the most basic case, we approximate elements in $\mathbb{R}$ by continued fractions. But we can also imagine a generalised version of the modular subset product problem away from $\mathbb{Z}/q\mathbb{Z}$ over a different base ring, such as a polynomial quotient ring $k[z]/(q(z))$ for some field $k$ and an irreducible polynomial $q(z) \in k[z]$. In this case we need to know the theory of continued fraction expansions of Laurent series. Because this is the most general version of the theory that we require and because most of the results regarding $\mathbb{R}$ follow from the more general theory for Laurent series, we will only provide proof for the theorems in this case.

A portion of the cryptanalysis of the modular subset product problem is based on understanding *lattices*. Furthermore, the obfuscation techniques we will study in Chapter 8 and Chapter 9 are based on different lattice problems. Hence, in the second part of this chapter, we will also give a brief overview of the theory of lattices and remind the reader of several associated hard problems.

## 4.1 Continued Fractions

The background can be found in any number theory textbook, such as [HW75]. Consider a rational number $x \in \mathbb{Q}$. It has a finite *continued fraction* representation of the form

$$x = a_0 + \cfrac{1}{a_1 + \cfrac{1}{\ddots + \cfrac{1}{a_N}}}$$

for $a_i \in \mathbb{N}$, $i = 0, \ldots, N$. We define the notation $x = [a_0; a_1, a_2, \ldots, a_N]$ for such a representation, where $a_0 = \lfloor x \rfloor$ is the integer part of $x$. In the more general case of $x \in \mathbb{R}$ such a representation also exists, though it is not necessarily finite. In this case $x = [a_0; a_1, a_2, \ldots]$ which can be contracted into $x = [a_0; a_1, a_2, \ldots, \alpha_n]$ where $\alpha_n \in \mathbb{R}$.

Define the integers $h_i, k_i$ for all $i = -2, \ldots, N$ by the following recursion

$$\begin{aligned} h_{-2} &= 0, \quad h_{-1} = 1, \quad h_i = h_{i-2} + a_i h_{i-1}, \\ k_{-2} &= 1, \quad k_{-1} = 0, \quad k_i = k_{i-2} + a_i k_{i-1}. \end{aligned} \tag{4.1}$$

We call the fractions $h_i/k_i$ the *convergents* of $x$.

There is a deeper relation between the convergents induced by a continued fraction expansion of an element $x \in \mathbb{R}$ and the element $x$ itself. This *approximation* relation is detailed by the *Diophantine approximation theorem*.

**Theorem 4.1** (Diophantine Approximation [Hur91]). *Let $x \in \mathbb{R}$ then there exist fractions $p/q \in \mathbb{Q}$ such that*

$$\left| x - \frac{p}{q} \right| < \frac{1}{\sqrt{5}q^2}.$$

*If, on the other hand, there exist $p/q \in \mathbb{Q}$ such that*

$$\left| x - \frac{p}{q} \right| < \frac{1}{2q^2},$$

*then $p/q$ is a convergent of $x$.*

*Proof.* The proof of the second half of this theorem is analogous to the proof of a generalised version given by Theorem 4.2. □

To find the continued fraction representation it is useful to review the extended Euclidean algorithm first. The coefficients calculated in the intermediate steps of the extended Euclidean algorithm are related to convergents.

### 4.1.1 Extended Euclidean Algorithm

For a pair of integers $a, b \in \mathbb{Z}$, the extended Euclidean algorithm finds integers $x, y \in \mathbb{Z}$ such that $ax + by = \gcd(a, b)$. If $\gcd(a, b) = 1$, then the algorithm essentially computes the inverse of $a \mod b$ and vice versa. The algorithm proceeds as follows: First, it initialises variables $r_i, s_i, t_i$ for $i = 0, 1$ as

$$\begin{aligned} r_0 &= a, \quad r_1 = b, \\ s_0 &= 1, \quad s_1 = 0, \\ t_0 &= 0, \quad t_1 = 1. \end{aligned}$$

Then it iteratively produces the sequence

$$\begin{aligned} r_{i+1} &= r_{i-1} - q_i r_i, \\ s_{i+1} &= s_{i-1} - q_i s_i, \\ t_{i+1} &= t_{i-1} - q_i t_i. \end{aligned} \tag{4.2}$$

Here we use Euclidean division ($r_{i-1} = q_i r_i + r_{i+1}$) to find $r_{i+1}$ and $q_i$ such that $0 \leq r_{i+1} < |r_i|$. Finally, the algorithm stops when $r_{i+1} = 0$.

Assuming that $a < b$ it can be shown that the worst-case runtime of the extended Euclidean algorithm is of the order $O(\log(a))$; the average runtime is of a similar order [Dix70; Hen94].

### 4.1.2 Finding Convergents

Compare Equation (4.1) with Equation (4.2). We immediately see that the convergents of a fraction $p/q \in \mathbb{Q}$ are exactly produced by the integers $s_i, t_i$ (up to signs) in the steps of the extended Euclidean algorithm applied to $p$ and $q$. Thus, the runtime for computing the continued fraction representation of $p/q$ is essentially the same as that of the extended Euclidean algorithm applied to $p$ and $q$. Here we assumed that $p$ and $q$ coprime, and $p < q$. Furthermore, we see from the analysis of Section 4.1.1 that the number of convergents is linear in the input size $\log(p)$.

## 4.2 Diophantine Approximation of Laurent Series

In this section we will give a proof for the Diophantine approximation theorem in a more general form. Instead of approximating elements in $\mathbb{R}$ with fractions in $\mathbb{Q}$, we will be approximating formal Laurent series over some field $k$ with fractions of polynomials in $k[z]$. This generalisation of continued fractions and the theory of Diophantine approximation from $\mathbb{R}$ to formal Laurent series is an established theory and has been studied many times; for a survey on the matter see for example [Las00].

Consider the field of fractions $k(z)$. Remember, that a possible valuation on $k(z)$ is given by $v(f) = \deg q - \deg p$ for elements $f = p/q$ with $p, q \in k[z]$. This induces a norm on $k(z)$, namely

$$|f| = \begin{cases} c^{-v(f)} & , f \neq 0 \\ 0 & , f = 0 \end{cases} \tag{4.3}$$

for some fixed constant $c \in \mathbb{R}$ with $c > 1$. This norm is non-Archimedean, i.e. it satisfies

$$|f + g| \leq \max\{|f|, |g|\} \tag{$\star$}$$

with equality when $f \neq g$ (cf. definition of an *ultrametric*) [Lan02, Chapter XII], [Sti09, Proposition 1.2.1].

**Definition 4.1** (Formal Laurent Series)**.** *Let $k$ be a field, then the* formal Laurent series *over $k$ are given by*

$$L = k((z^{-1})) = \left\{ \sum_{i \leq N} a_i z^i \,\middle|\, N \in \mathbb{Z}; a_i \in k, i \in (-\infty, N] \right\}.$$

One can show that $L$ is a field, since for a non-zero $\alpha = \sum_{i \leq N} a_i z^i$ with $a_N \neq 0$ there exists its inverse $\alpha^{-1} = \sum_{j \leq -N} a'_j z^j$ with coefficients

$$\begin{aligned} a'_{-N} &= a_N^{-1}, \\ a'_{j-N} &= -a_N^{-1} \sum_{i=N+j}^{N-1} a_i a'_{j-i} \end{aligned}$$

where $j < 0$ [Mal17, Section 1.2].

Furthermore, one can show that $L$ is the completion of $k(z)$ with respect to the norm given by Equation (4.3). The valuation on $k(z)$ can be extended to a valuation on $L$ in a natural way; for $\alpha = \sum_{i \leq N} a_i z^i \in L$ we set $v(\alpha) = -N$ and so $|\alpha| = c^N$.

### 4.2.1 Continued Fractions

We want to consider continued fractions over $L$. Given an element $f \in k(z)$, we can define a *continued fraction expansion* of the form

$$f = a_0 + \cfrac{1}{a_1 + \cfrac{1}{\ddots + \cfrac{1}{a_N}}}$$

with a sequence of polynomials $a_i \in k[z]$, $i = 0, \dots, N$. As a shorthand notation we again use $f = [a_0; a_1, \dots, a_N]$. An element $\alpha \in L$ has a similar expansion, though it is not necessarily finite: $\alpha = [a_0; a_1, a_2, \dots]$. Again, a continued fractions expansion may be contracted in the following sense:

$$[a_0; a_1, \dots, a_n, a_{n+1}, \dots] = [a_0; a_1, \dots, \alpha_n]$$

where now $\alpha_n \in L$. We can immediately verify that for the convergents defined by Equation (4.1) the following identities hold:

$$\begin{aligned} [a_0; a_1, \dots, \alpha_n] &= \frac{\alpha_n h_{n-1} + h_{n-2}}{\alpha_n k_{n-1} + k_{n-2}}, \\ (-1)^n &= h_{n-1}k_n - h_n k_{n-1}. \end{aligned} \tag{4.4}$$

Furthermore, let $\alpha = [a_0; a_1, \dots, a_n, \alpha_{n+1}]$, then by Equation (4.4) we have

$$\left| \alpha - \frac{h_n}{k_n} \right| = \frac{1}{|k_n(\alpha_{n+1}k_n - k_{n-1})|}. \tag{4.5}$$

Much like in the rational case, we find that the convergents of a *fraction* $p/q \in k(z)$ are exactly produced by the polynomials $s_i, t_i \in k[z]$ (see Equation (4.2)) in the steps of the extended Euclidean algorithm applied to the polynomials $p$ and $q$.

Inspecting degrees yields that for convergents $h_n/k_n \in k(x)$ we have

$$|k_{n-1}| < |k_n| = |a_1| \cdots |a_n|$$

for all $n \geq 0$. Hence, from Equation (4.5), we find for $\alpha = [a_0; a_1, \dots, a_n, \alpha_{n+1}]$ that

$$\left| \alpha - \frac{h_n}{k_n} \right| = \frac{1}{|\alpha_{n+1}k_n^2|},$$

as by the ultrametric property ($\star$) it holds that

$$|k_n(\alpha_{n+1}k_n - k_{n-1})| = |\alpha_{n+1}k_n^2|.$$

Now since $|\alpha_n| = |a_n|$ and $|k_n| < |k_{n+1}|$, we arrive at the following useful (in)equality

$$\left| \alpha - \frac{h_n}{k_n} \right| = \frac{1}{|k_n k_{n+1}|} < \frac{1}{|k_n|^2}. \tag{4.6}$$

The first equality holds since the metric on $L$ is non-Archimedean. In the case of an ordinary metric we would have to replace = with < in Equation (4.6).

### 4.2.2 Diophantine Approximation

Finally, we want to show a statement that can be thought of as a generalisation of Theorem 4.1, albeit in the case that the norm in question is non-Archimedean.

**Theorem 4.2** (Diophantine Approximation for Laurent Series)**.** *Let $k$ be a field. Let $L$ be the formal Laurent series over $k$. Let $\alpha \in L$ and $p, q \in k[z]$ with* $\gcd(p(z), q(z)) = 1$ *such that*

$$\left| \alpha - \frac{p}{q} \right| < \frac{1}{|q|^2}.$$

*Then $p/q$ is a convergent of $\alpha$.*

*Proof.* Consider the convergents of $\alpha$. Let $n \in \mathbb{N}$ be the index such that $|k_n| \leq |q| < |k_{n+1}|$. Then by our assumption we have

$$\left|\alpha - \frac{p}{q}\right| < \frac{1}{|q|^2} \leq \frac{1}{|q||k_n|}.$$

On the other hand, by Equation (4.6), we have

$$\left|\alpha - \frac{h_n}{k_n}\right| = \frac{1}{|k_n k_{n+1}|} < \frac{1}{|q||k_n|}.$$

Hence, we find

$$\left|\frac{p}{q} - \frac{h_n}{k_n}\right| < \frac{1}{|q||k_n|} \tag{4.7}$$

by the ultrametric property ($\star$).

Now assume that $p/q \neq h_n/k_n$ for all $i \geq 0$, i.e. $pk_n - qh_n \neq 0$. Then, by inspection of degrees, $|pk_n - qh_n| \geq 1$ for all $n \geq 0$. Thus, we end up with the following inequality

$$\left|\frac{p}{q} - \frac{h_n}{k_n}\right| = \frac{|pk_n - qh_n|}{|q||k_n|} \geq \frac{1}{|q||k_n|},$$

a contradiction to Equation (4.7). See also [Ste92, Satz I.3.21]. □

## 4.3 Introduction to Lattices

A discrete subgroup of the vector space $\mathbb{R}^n$ for some $n \in \mathbb{N}$ is called a *lattice*. If we consider such a discrete group along with the restriction of the Euclidean metric, then these *Euclidean* lattices are the prototypical picture to have in mind when thinking about more general lattices. Pick $m \leq n$ linearly independent vectors $v_i \in \mathbb{R}^n$, $i = 1, \ldots, m$. We can represent the lattice generated by the set of *basis* vectors $\{v_1, \ldots, v_m\}$ as follows

$$\Lambda_{\{v_1,\ldots,v_m\}} = \left\{ \sum_{i=1}^{m} a_i v_i \,\middle|\, a \in \mathbb{Z}^m \right\}.$$

The integer $m \in N$ denotes the *rank* or *dimension* of the lattice $\Lambda$, if $m = n$ we call $\Lambda$ *full-rank*. A *basis matrix* for $\Lambda_{\{v_1,\ldots,v_m\}}$ is given by the matrix $B \in \mathbb{R}^{n \times m}$ whose rows are the basis vectors $\{v_1, \ldots, v_m\}$.

From now on we assume that any lattice we consider is full-rank. Important invariants of a lattice are its *volume* and *shortest vector length*.

**Definition 4.2** (Lattice Volume)**.** *Let $\Lambda$ be a lattice of dimension $n$. The* volume *(or* determinant*)* $\mathrm{vol}(\Lambda)$ *is given by*

$$\mathrm{vol}(\Lambda) = |\det(\Lambda)| = |\det(B)|,$$

*for any basis $B$ generating the lattice $\Lambda$.*

The volume of $\Lambda$ is an invariant since one can show that two basis matrices $B_1$ and $B_2$ generate the same lattice if and only if they are related by a unimodular transformation $U$: $B_1 = UB_2$. Since for a unimodular matrix $U$ we have that $\det(U) = \pm 1$, invariance of $\mathrm{vol}(\Lambda)$ follows immediately.

**Definition 4.3** (Shortest Lattice Vector Length)**.** *We define the* length of the shortest vector *in the lattice $\Lambda$ as follows*

$$\lambda_1(\Lambda) = \min_{x \in \Lambda, x \neq 0} \|x\|.$$

We can bound the length of the shortest lattice vector as follows, see also [Cas59, Chapter VIII].

**Theorem 4.3** (Hermite, Minkowski)**.** *Let $\Lambda$ be a lattice of dimension $n$. Then it holds that $\lambda_1(\Lambda) \leq \sqrt{n}\,\mathrm{vol}(\Lambda)^{1/n}$.*

The *Gaussian heuristic* (see [HPS14, Section 7.5.3] and [Gal12, Chapter 16]) states that the shortest vector in a "randomly chosen" lattice $\Lambda$ of rank $n$ satisfies

$$\lambda_1(\Lambda) \approx \sqrt{\frac{n}{2\pi e}}\,\mathrm{vol}(\Lambda)^{1/n}.$$

We can also ask for the shortest distance of a point in the ambient vector space to the embedded lattice.

**Definition 4.4.** *The* distance *of a vector $x \in \mathbb{R}^n$ to a lattice $\Lambda \subset \mathbb{R}^n$ is given by*

$$\mathrm{dist}(x, \Lambda) = \min_{y \in \Lambda} \|x - y\|.$$

Closely related to a lattice is its dual lattice. It is defined as follows.

**Definition 4.5** (Dual Lattice)**.** *The* dual *of a lattice $\Lambda \subset \mathbb{R}^n$ is defined as*

$$\Lambda^* = \{x \in \mathbb{R}^n \mid x \cdot \Lambda \subseteq \mathbb{Z}\}.$$

This means that the dual lattice $\Lambda^*$ to a lattice $\Lambda$ is the set of all points whose inner product with the elements in $\Lambda$ is an integer. Consider a full-rank lattice $\Lambda$ with a given basis $B$. One can show that $\left(B^T\right)^{-1}$ is a basis of the dual lattice $\Lambda^*$.

### 4.3.1 Discrete Gaussians

Define the *Gaussian function* $\psi_s(x) : \mathbb{R}^n \to \mathbb{R}$ for some parameter $s \in \mathbb{R}_{>0}$ by

$$\psi_s(x) = e^{-\pi \frac{\|x\|^2}{s^2}}.$$

From this Gaussian function we can now define a probability distribution on $\mathbb{Z}$ which is the discrete version of the Gaussian distribution on $\mathbb{R}$.

**Definition 4.6** (Discrete Gaussian Distribution)**.** *Denote with $\psi_s(\mathbb{Z})$ the quantity*

$$\psi_s(\mathbb{Z}) = 1 + 2\sum_{i=0}^{\infty} \psi_s(i).$$

*The* discrete Gaussian distribution *$D_s$ on $\mathbb{Z}$ with parameter $s$ is the distribution with probability density*

$$f_s(x) = \frac{\psi_s(x)}{\psi_s(\mathbb{Z})}.$$

Note that for a general lattice $\Lambda \subset \mathbb{R}^n$, $\psi_s(\Lambda)$ is computed by summing $\psi_s(x)$ over all the lattice points $x \in \Lambda$. We can then analogously define the discrete Gaussian distribution on $\Lambda$.

### 4.3.2 Computational Problems

Let us recall the most important computational problems arising from lattices. This and the following sections closely follow [Pei16].

**Problem 4.1** (Shortest Vector Problem (SVP))**.** *Given a basis $B$ of a lattice $\Lambda$, find a lattice vector $x$ such that $\|x\| = \lambda_1(\Lambda)$.*

**Problem 4.2** (Approximate Shortest Vector Problem ($\mathrm{SVP}_\gamma$))**.** *Given a basis $B$ of a $n$-dimensional lattice $\Lambda$, find a nonzero lattice vector $x$ such that $\|x\| \leq \gamma(n)\lambda_1(\Lambda)$, where $\gamma(n) \geq 1$ is a real number.*

**Problem 4.3** (Decisional Approximate SVP ($\mathrm{GapSVP}_\gamma$))**.** *Given a basis $B$ of a $n$-dimensional lattice $\Lambda$, such that either $\lambda_1(\Lambda) \leq 1$ or $\lambda_1(\Lambda) > \gamma(n)$, determine which is the case.*

**Problem 4.4** (Bounded Distance Decoding Problem ($\mathrm{BDD}_\gamma$))**.** *Given a basis $B$ of a $n$-dimensional lattice $\Lambda$ and a target point $t \in \mathbb{R}^n$ such that $\mathrm{dist}(t, \Lambda) < d = \lambda_1(\Lambda)/(2\gamma(n))$, find the unique lattice vector $x$ such that $\|t - x\| < d$.*

One approach to solving the aforementioned lattice problems is commonly called *lattice basis reduction*. The idea is to find a *reduced* basis consisting of short(est) lattice vectors. Given this particularly nice basis, the problems become easy. For example, to solve SVP we select the shortest vector from the reduced basis. The other problems have similar algorithms. Hence, common lattice-based cryptosystems have as a part of the public parameters a random looking non-reduced basis. The lattice properties are chosen in a way such that lattice basis reduction is hard.

### 4.3.3 Related Lattice Problems

In this section we are concerned with what we call $q$-ary lattices. Suppose we are given a matrix $A \in (\mathbb{Z}/q\mathbb{Z})^{m\times n}$ for some modulus $q$. If we consider $A$ to define a map $A : \mathbb{Z}^n \to (\mathbb{Z}/q\mathbb{Z})^m$, we can show that the kernel of $A$ over $\mathbb{Z}^n$ actually defines the lattice

$$\Lambda^\perp(A) = \ker A = \left\{z \in \mathbb{Z}^n | Az \equiv 0 \mod q\right\}$$

where $q\mathbb{Z}^n \subseteq \Lambda^\perp(A)$ is a sublattice. The dual of this lattice (up to a scaling factor $1/q$) is given by

$$\Lambda^T(A) = \left\{z \in \mathbb{Z}^n \,\middle|\, \exists s \in (\mathbb{Z}/q\mathbb{Z})^n : z \equiv A^T s \mod q\right\},$$

i.e. we have that (recall Definition 4.5)

$$\Lambda^\perp(A)^* = \frac{1}{q}\Lambda^T(A).$$

To see this for the full-rank case, let $B$ be a basis of $\ker A$, i.e. $AB \equiv 0 \mod q$. The only full-rank solution to this is given by $B = qA^{-1}$, thus the dual basis is given by $q^{-1}A^T$. This is exactly a basis for the lattice $\Lambda^T(A)$ (up to scaling by $1/q$).

We will summarise several important computational problems that can be reduced to worst-case lattice problems described in Section 4.3.2.

**Short Integer Solution.** The first lattice problem we want to introduce is related to solving a (homogeneous) system of linear equations with a twist: The norm of the solution vector is constrained.

**Problem 4.5** (Short Integer Solution ($\mathrm{SIS}_{n,q,\beta,m}$))**.** *Given $m$ uniformly random vectors $a_i \in (\mathbb{Z}/q\mathbb{Z})^n$, forming the columns of a matrix $A \in (\mathbb{Z}/q\mathbb{Z})^{n\times m}$, find a nonzero integer vector $z \in \mathbb{Z}^m$ of norm $\|z\| \leq \beta$ such that $Az = 0$.*

Ajtai [Ajt99] gave an average-case to worst-case reduction from approximate SVP to SIS which makes the problem attractive for cryptographic use. An average-case instance of SIS is secure if the worst-case of $\mathrm{SVP}_\gamma$ is hard.

We obtain a special case of SIS called the *modular subset sum problem* by letting $n = 1$. The subset sum problem is to specify a set of random integers $S = \{a_i \in \mathbb{Z}\}_{i=1,\ldots,m}$, taking a random subset $R \subset S$, and setting $b = \sum_{r\in R} r$. The subset sum problem is to find $R$ given $(S, b)$. If we take the base ring as $\mathbb{Z}/q\mathbb{Z}$ instead of $\mathbb{Z}$ we get the aforementioned modular version. It is equivalent to SIS with $n = 1$ by forming the vector $(a_1, \ldots, a_m, -b)$ from the set $S$ and finding a short ($\beta = m + 1$) vector $z \in \{0, 1\}^{m+1}$ such that $a \cdot z = 0$.

**Learning With Errors.** The second lattice problem is related to solving an inhomogeneous system of linear equations. Again, the norm of the solution vector is constrained but the individual linear equations have also been modified by a *small* error term.

**Definition 4.7** (LWE Distribution)**.** *For a vector $s \in (\mathbb{Z}/q\mathbb{Z})^n$ and a distribution $\chi$ over $\mathbb{Z}/q\mathbb{Z}$, the* LWE distribution *$A_{s,\chi}$ over $(\mathbb{Z}/q\mathbb{Z})^n \times (\mathbb{Z}/q\mathbb{Z})$ is sampled by choosing $a \in (\mathbb{Z}/q\mathbb{Z})^n$ uniformly at random, choosing $e \leftarrow \chi$, and outputting $(a, b = s \cdot a + e \mod q)$.*

We represent $\mathbb{Z}/q\mathbb{Z}$ by the elements $\{\lceil -q/2\rceil, \ldots, \lfloor (q-1)/2\rfloor\}$. Typically, for LWE, the noise distribution $\chi$ is a discrete Gaussian. Given a lattice dimension $n \ll q$, consider for example $D_s$ on $\mathbb{Z}^n$ analogously to Definition 4.6, with parameter $s = \sqrt{n}$. The norm $\|x\|$ of a sampled vector $x \leftarrow D_s$ lies in a small interval centered around 0 with high probability.

Another possible noise distribution is the uniform distribution on an interval $[-B, \ldots, B]^n$ for some appropriate $B \ll q$. Sometimes binary or trinary errors are considered, here the noise distribution is the uniform distribution over $\{0, 1\}^n$ or $\{-1, 0, 1\}^n$, respectively.

**Problem 4.6** (Search-$\mathrm{LWE}_{n,q,\chi,m}$)**.** *Given $m$ independent samples $(a_i, b_i)$ drawn from $A_{s,\chi}$ for a uniformly random $s \in (\mathbb{Z}/q\mathbb{Z})^n$, find $s$.*

**Problem 4.7** (Decision-$\mathrm{LWE}_{n,q,\chi,m}$)**.** *Given $m$ independent samples $(a_i, b_i)$ where every sample is distributed according to either:*

- *$A_{s,\chi}$ for a uniformly random $s \in (\mathbb{Z}/q\mathbb{Z})^n$, or*
- *the uniform distribution,*

*distinguish which is the case (with non-negligible advantage).*

Regev [Reg05] gave a quantum reduction from GapSVP to Decision-LWE and Peikert [Pei09] gave a classical reduction from a weaker version of GapSVP to Decision-LWE. Regev [Reg05] also gave an average-case to worst-case reduction from BDD to Search-LWE.

**Ring-SIS.** We want to generalise Problem 4.5 to a general ring $R$. Different choices for $R$ are possible. Commonly used is the polynomial quotient ring over the integers $R = \mathbb{Z}[X]/(f(X))$ for some modulus $f(X)$. We want to use this problem to build cryptographic schemes in a finite setting, hence we should restrict from the integers $\mathbb{Z}$ to the finite ring $(\mathbb{Z}/q\mathbb{Z})$. Then the ring we will work over becomes $R_q = R/qR$.

**Problem 4.8** (Ring-SIS$_{q,\beta,m}$)**.** *Let $R_q$ be a ring. Given $m$ uniformly random elements $a_i \in R_q$, defining a vector $a \in R_q^m$, find a nonzero vector $a \in R^m$ of norm $\|z\| \leq \beta$ such that $a \cdot z = 0$.*

**Ring-LWE.** Here we want to generalise Problems 4.6 and 4.7. Again, we work over a ring $R$ and specifically over $R_q = R/qR$ for some integer modulus $q$.

**Definition 4.8** (Ring-LWE Distribution)**.** *For $s \in R_q$ and a distribution $\chi$ over $R_q$, the* ring-LWE distribution *$A_{s,\chi}$ over $R_q \times R_q$ is sampled by choosing $a \in R_q$ uniformly at random, choosing $e \leftarrow \chi$, and outputting $(a, b = s \cdot a + e \mod q)$.*

**Problem 4.9** (Search-LWE$_{n,q,\chi,m}$)**.** *Given $m$ independent samples $(a_i, b_i)$ drawn from $A_{s,\chi}$ for a uniformly random $s \in R_q$, find $s$.*

**Problem 4.10** (Decision-Ring-LWE$_{n,q,\chi,m}$)**.** *Given $m$ independent samples $(a_i, b_i)$ where every sample is distributed according to either:*

- *$A_{s,\chi}$ for a uniformly random $s \in R_q$, or*
- *the uniform distribution,*

*distinguish which is the case (with non-negligible advantage).*

**NTRU.** We will now introduce the last lattice problem that we are interested in. We say that a polynomial is *small* if its coefficients are of small norm in the base ring. The notion of small norm needs to be specified on a case-by-case basis, depending on the base ring.

**Problem 4.11** (Search-NTRU)**.** *Let $n, q \in \mathbb{N}$, $\phi \in \mathbb{Z}[X]$ monic of degree $n$, and set $R_q = (\mathbb{Z}/q\mathbb{Z})[X]/(\phi)$. Let $f \in R_q^\times$, $g \in R_q$ be small polynomials and set $h = f^{-1}g \mod q$. The* search-NTRU *problem is then to recover $f$ or $g$ given $h$.*

In the original NTRU schemes, the modulus polynomial would be chosen as $\phi(x) = x^n - 1$. The reason for choosing a modulus of this form is that polynomial multiplication in the ring $R_q$ becomes easier than it would be for a general modulus.

The NTRU problem induces a certain lattice, called the *NTRU lattice*. Assuming no other attacks, then it is the hardness of lattice basis reduction in the NTRU lattice that determines the hardness of NTRU itself.

# 5 Subset Products

In this chapter we introduce a new computational problem that is rooted in number theory and somewhat related to the problems of factoring and computing discrete logarithms. In a nutshell, the new problem (which we call the *modular subset product problem*) can be thought of as a multiplicative version of the well-known subset sum problem over the ring $\mathbb{Z}/q\mathbb{Z}$, for a suitable prime modulus $q$. We will also introduce new computational assumptions based on the modular subset product problem upon which we eventually want to base new obfuscation constructions in Chapter 6 and Chapter 7. Related ideas have been considered by Contini et al. [CLS06], Naccache [Nac16] and Ducas and Pierrot [DP19]. We conjecture that our new assumptions are hard in appropriate parameter areas. We provide classical and post-quantum cryptanalysis of the problem in cryptographically interesting parameter areas to back up our conjecture.

## 5.1 Preliminaries

Before we introduce and analyse the modular subset product problem, let us mention that it is connected to a certain problem involving Hamming distance. While we will venture deeper into this matter in Chapter 6, we want to provide the reader with the most important facts already here. We believe that several statements are worded more naturally in the language of Chapter 6.

Fix some dimension $n \in N$ and consider the following problem: Determine a vector $y \in \{0,1\}^n$ that is contained within a Hamming ball of a fixed radius $r \in \mathbb{N}$ around a fixed target vector $x \in \{0,1\}^n$. This problem is easy if either $r > n/2$ (since then any random vector is contained in the ball with high probability), or if we are given the vector $x$ itself. But we can instead imagine a setting where we are only given access to an oracle that takes as an input $y$ and answers with whether $y$ is contained in the Hamming ball around $x$. One can show that if $x \in \{0,1\}^n$ was sampled uniformly and $r < n/2 - \sqrt{\log(2)n\lambda}$, then this problem is evasive in $\lambda \in \mathbb{N}$ (see Lemma 6.3).

On the other hand, we would like to allow a more general target vector $x \in \{0,1\}^n$ that need not be sampled uniformly from $\{0,1\}^n$. For this, assume that $D$ is some distribution over $\{0,1\}^n$. The aforementioned problem is evasive if $D$ is a so-called *Hamming distance evasive distribution* (see Definition 6.6). In this case, we say that $D$ has Hamming ball min-entropy at least $\lambda$ (see Definition 6.5).

## 5.2 Modular Subset Products

Before we introduce the new problem, we start with the definition of a safe prime.

**Definition 5.1** (Safe Prime, Sophie Germain Prime)**.** *A prime $q$ is called a* safe prime *if $q$ is of the form $q = 2p + 1$ for a prime $p$. The prime $p$ is then called a* Sophie Germain prime*.*

As is customary in cryptography, we give a search version of the problem first. Independently of the problem setting (search or decision) we will let the *secret* $x \in \{0,1\}^n$ be from some distribution $D$ over $\{0,1\}^n$. This allows for a more general statement rather then assuming uniformly distributed secrets.

**Problem 5.1** (Distributional Modular Subset Product Problem)**.** *Let $r < n \in \mathbb{N}$ and $D$ be a distribution over $\{0,1\}^n$. Let $\rho$ be a distribution that outputs a sequence of distinct primes $(p_i)_{i=1,\dots,n}$ and a safe prime $q$ such that*

$$\prod_{i \in I} p_i < \frac{q}{2} < (1 + o(1)) \max\{p_i\}^r \text{ for all } I \subset \{1, \dots, n\} \text{ with } |I| \leq r. \tag{5.1}$$

*Given an integer*

$$X = \prod_{i=1}^{n} p_i^{x_i} \mod q \tag{5.2}$$

*for some vector $x \leftarrow D$, the $(r, n, D, \rho)$*-distributional modular subset product problem ($\text{MSP}_{r,n,D,\rho}$) *is to find $x$.*

We also state a decisional version of the problem, where we ask to distinguish between a real instance and a uniformly random element.

**Problem 5.2** (Decisional Distrib. Modular Subset Product Problem)**.** *Let $r < n \in \mathbb{N}$ and $D$ be a distribution over $\{0,1\}^n$. Let $\rho$ be a distribution of primes $(p_i)_{i=1,\dots,n}$ and $q$ as in Equation (5.1). Define the distribution*

$$D_0 = ((p_i)_{i=1,\dots,n}, q, X)$$

*where $X$ satisfies Equation (5.2) for some vector $x \leftarrow D$. Define the distribution*

$$D_1 = ((p_i)_{i=1,\dots,n}, q, X')$$

*where $(p_i)_{i=1,\dots,n}$ and $q$ are as in $D_0$, but $X' \leftarrow (\mathbb{Z}/q\mathbb{Z})^*$ uniformly.*

*Then the $(r, n, D, \rho)$*-decisional distributional modular subset product problem ($\text{D-MSP}_{r,n,D,\rho}$) *is to distinguish $D_0$ from $D_1$. In other words, given a sample from $D_b$ for uniform $b \in \{0,1\}$, the problem is to determine $b$.*

**Definition 5.2** (Minimal Prime Distribution)**.** *Let $r < n \in \mathbb{N}$. We define the* minimal prime distribution *$\rho_{r,n}$ as follows: Sample distinct primes $p_i \leftarrow [2, O(n \log(n))]$ uniformly for all $i = 1, \dots, n$, to obtain a sequence $(p_i)_{i=1,\dots,n}$. Finally, sample a safe prime $q$ uniformly such that for all $I \subset \{1, \dots, n\}$ with $|I| \leq r$ it holds that $q/2 \in \left[\prod_{i \in I} p_i, (1 + o(1)) \max\{p_i\}^r\right]$. The distribution $\rho_{r,n}$ outputs $((p_i)_{i=1,\dots,n}, q)$.*

Note that for a pair of parameters $r < n \in \mathbb{N}$, a sample $((p_i)_{i=1,\dots,n}, q) \leftarrow \rho_{r,n}$ has the property that $p_i \approx n \log(n)$ for all $i = 1, \dots, n$ and that $q \approx (n \log(n))^r$. For the remainder of this work we will implicitly assume that the prime distribution $\rho$ in $\text{MSP}_{r,n,D,\rho}$ and $\text{D-MSP}_{r,n,D,\rho}$ is the minimal prime distribution $\rho = \rho_{r,n}$ given by Definition 5.2.

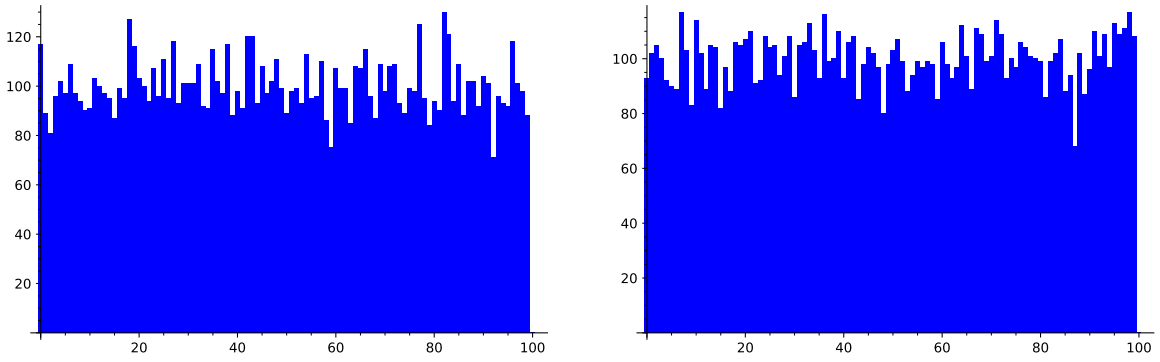
We believe these computational problems are hard whenever the Hamming distance matching problem we described in Section 5.1 itself is evasive. The vector $x \in \{0,1\}^n$ plays the role of a secret target and $r \in \mathbb{N}$ is the Hamming ball radius. For example, solving the search problem $\text{MSP}_{r,n,D,\rho}$ could involve finding a vector $y \in \{0,1\}^n$ in the Hamming ball of radius $r$ around the target $x$. Precisely we make the following conjecture that covers all possible distributions $D$.

**Conjecture 1.** *Fix $r < n/2 \in \mathbb{N}$. If $D$ is a distribution on $\{0,1\}^n$ with Hamming ball min-entropy at least $\lambda$ (i.e. $D$ is a Hamming distance evasive distribution in the sense of Definition 6.6) then solving D-MSP$_{r,n,D,\rho}$ (Problem 5.2) with probability 1 requires $\Omega(\min\{2^\lambda, 2^{n/2}\})$ operations.*

Note that Problem 5.2 only makes sense if the two distributions $D_0$ and $D_1$ are different. In the case $q \ll 2^n$ we conjecture that the values $X = \prod_{i=1}^n p_i^{x_i} \mod q$ are distributed close to uniformly if $x$ is sampled uniformly, and so it makes no sense to ask for a distinguisher between this distribution and the uniform distribution. For the proof of Theorem 5.1 we need a more precise version of this statement, so we make the following conjecture that we believe is very reasonable. y

**Conjecture 2.** *Let $r, n, (p_i)_{i=1,\ldots,n}, q$ be as in Problem 5.1, with the extra condition that $q \leq 2^n$. Let $D$ be the uniform distribution on $\{0,1\}^n$. Then the statistical distance of the distribution $\prod_{i=1}^n p_i^{x_i} \mod q$ over $x \leftarrow D$ and the uniform distribution on $(\mathbb{Z}/q\mathbb{Z})^*$ is negligible.*

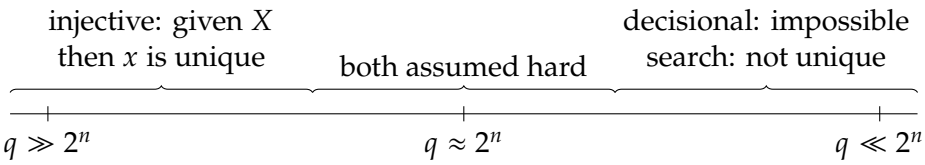
We have used a computer algebra system to simulate sampling from a modified distribution $D_0$ (where we fixed $q$) and compared it numerically to the uniform distribution on $\mathbb{Z}/q\mathbb{Z}$ for different values of $q \leq 2^n$ and different values of $r, n \in \mathbb{N}$. Numerical evidence suggests that Conjecture 2 holds. See Figure 5.1 for example results. The plotted data gives an indication about the true behaviour of the probability density function for the distribution $D_0$ of Problem 5.2.




(a) $r = 16$, $n = 128$ (b) $r = 50$, $n = 500$

Figure 5.1: Bar plots for two different numerical experiments as evidence that Conjecture 2 holds. We have chosen $r, n \in \mathbb{N}$ such that $q \leq 2^n$ for $q \approx (n\log(n))^r$. For each plot, we have partitioned the interval $[0, q)$ into 100 equally sized subintervals. Each plot depicts the recorded number of $X = \prod_{i=1}^n p_i^{x_i} \mod q$ falling into a subinterval, for 10000 samples of random $(p_i)_{i=1,\ldots,n}$ such that $p_i \approx n\log(n)$ and uniformly random $x \in \{0,1\}^n$.

The situation is summarised in the following diagram. In this diagram we assume that the exponent vector $x$ is sampled from the uniform distribution on $\{0,1\}^n$. The left-hand side with $q \gg 2^n$ is the *low-density* case. In this range the distributions $D_0$ and $D_1$ of Problem 5.2 are very different and $x$ is uniquely determined by $X$. The right-hand side with $q \ll 2^n$ is the *high-density* case where the distributions $D_0$ and $D_1$ are close and for every value $X$ there are likely (multiple) solutions. As can be seen in Figure 6.2 our interest reaches over all density cases.




We suspect that a rigorous "search-to-decision" reduction in the low-density or *density one* case (i.e. $2^n < q$) is possible by borrowing techniques from [IN96; MM11]. We leave as an open problem to work out whether a decision oracle for Problem 5.2 in this case can be used to solve the search problem Problem 5.1 in polynomially many queries to the decision oracle. This would give further evidence that Problem 5.2 should be hard (recall that the assumption makes no sense in the high-density case).

## 5.3 Algorithms

We now consider algorithms for Problem 5.1. If the Hamming weight of $x$ is too small (this happens if $w_H(x) < r$), or if $q$ is too large (this happens if $(n\log(n))^n < q$ when we take $p_i \approx n\log(n)$), then we will necessarily have that $\prod_{i=1}^n p_i^{x_i} < q$. This means that we are actually looking at a factorisation problem over the integers and hence Problem 5.1 can be solved by factoring $X$ over the integers. This is the extreme low-density case and is easy since we are given the exact list of possible factors.

More generally, an approach to Problem 5.1 is to guess some $x_i$ for $i \in I$ and try to factor $X\prod_{i\in I} p_i^{-x_i} \mod q$. We will now argue that this approach does not lead to an efficient attack. By inspecting Problem 5.1, we may assume that $q \approx (n\log(n))^r$ and that $x$ is sampled from a distribution with large Hamming ball min-entropy. To simplify our analysis, we can consider the uniform distribution on $\{0,1\}^n$ as described in Section 5.1. For fixed parameters $r < n/2 \in \mathbb{N}$, the gap $n/2 - r$ yields a certain security parameter $\lambda$ such that the Hamming ball membership problem is then evasive in $\lambda$. Hence, by guessing $\ell = |I|$ many bits of the target vector $x$, we obtain an a priori easier problem by considering $X\prod_{i\in I} p_i^{-x_i} \mod q$ instead of $X$ with a smaller gap $(n-\ell)/2 - r$. But note that we had to trade an easier instance for trying to guess $\ell$ many bits of the vector $x$. If the Hamming ball min-entropy is large enough, then hitting a correct guess will take exponentially many tries in $\ell$. Assuming a uniform distribution, this will take $2^\ell$ many guesses in the worst case. We see that this type of attack is still combinatorial in nature. In short, the requirement that the Hamming ball membership problem is evasive already implies that such an attack requires exponential time.

We now consider algorithms that are appropriate in general. There is an obvious meet-in-the-middle algorithm: Let $m = \lfloor n/2 \rfloor$. Given $X$ we compute a list $L$ of pairs $(z, Z)$ where $Z = \prod_{i=1}^m p_i^{z_i} \mod q$ for all $z \in \{0,1\}^m$. Then for all $z' \in \{0,1\}^{n-m}$ compute

$$Z' = X\prod_{i=1}^{n-m} p_{m+i}^{-z_i'} \mod q$$

and check if $Z'$ is in $L$. If there is a match, then we have found $x = z\|z'$. This attack requires $O(2^{n/2})$ operations. It follows that $n$ must be sufficiently large for the problem to be hard.

As we will see in Section 5.7, we can rewrite a MSP instance (by solving multiple discrete logarithms) as a modular subset sum problem. Becker et al. [BCJ11] gave an algorithm for solving such knapsacks in $O(2^{0.291n})$ time and $O(2^{0.256n})$ space. Solving modular subset sum problems can also be done by finding shortest vectors in certain lattices. Becker et al. [Bec+] gave a sieving algorithm for SVP which requires $O(2^{0.292n})$ time and space.

## 5.4 Hardness

We now give evidence that Problem 5.1 is hard in the high-density case. Our argument is based on ideas from index calculus algorithms in finite fields. We prove that if one can solve Problem 5.1 (in the medium-/high-density case) in time $T$ then one can solve the discrete logarithm problem (DLP) in $(\mathbb{Z}/q\mathbb{Z})^*$ in time $poly(T)$. Note that this result gives at best a subexponential hardness guarantee and does not say anything about post-quantum security. Contini et al. [CLS06] considered a similar computational assumption called the *very smooth number discrete log*. They give a similar reduction to the discrete logarithm problem.

**Theorem 5.1.** *Fix $r, n \in \mathbb{N}$ such that $r < n/2$. Let $q$ be prime such that $q \leq 2^n$ and $(p_i)_{i=1,\ldots,n}$ be a sequence of distinct primes such that $p_i \in [2, O(n\log(n))]$. Assume Conjecture 2 holds and suppose $MSP_{r,n,D,\rho}$ (Problem 5.1) can be solved with probability 1 in time $T$. Then there is an algorithm to solve the DLP in $(\mathbb{Z}/q\mathbb{Z})^*$ with expected time $\tilde{O}(nT)$.*

*Proof.* Let $g_0, h \in (\mathbb{Z}/q\mathbb{Z})^*$ be a DLP instance and let $\mathcal{A}$ be an oracle for Problem 5.1 that runs in time $T$ and succeeds with probability 1. Let $g$ be a generator of $(\mathbb{Z}/q\mathbb{Z})^*$ so that its order is $M = q - 1$. Choose random $0 < a < q - 1$ and compute $C = g^a \mod q$. Call $\mathcal{A}$ on $C$. Due to the assumptions in the theorem, with probability bounded below by a constant, $\mathcal{A}$ succeeds and outputs a solution $x$. Store $(a, x)$. Note that each relation implies a linear relation

$$a \equiv \sum_{i=1}^{n} x_i \log_g(p_i) \mod M.$$

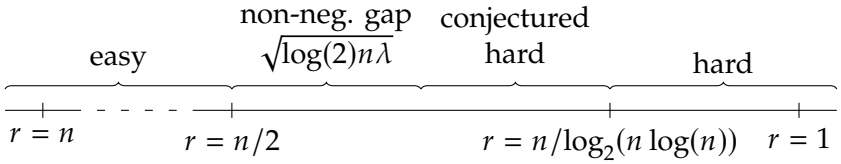
Repeat until we have $n$ linearly independent relation vectors $x$, and hence use linear algebra to solve for $\log_g(p_i)$. Finally, choose a random $b$ and set $C = hg^b \mod q$. Call $\mathcal{A}$ on $C$ to get, with high probability, one more relation $(b, y)$. Knowing $\log_g(p_i)$ we now compute

$$\log_g(h) = -b + \sum_i y_i \log_g(p_i).$$

We now perform the same algorithm to compute $\log_g(g_0)$ and then finally obtain $\log_{g_0}(h)$. □

The above proof generalises to any group whose order is known. When $q \approx (n\log(n))^r$ then the condition $2^n \geq q$ boils down to $r < n/\log_2(n\log(n))$. Hence, when $r < n/\log_2(n\log(n))$ the hardness of Problem 5.1 follows from the discrete logarithm assumption.

It follows that Problem 5.1 has a spectrum of difficulty, ranging from easy in the extreme low-density case to hard in the medium-/high-density case. We visualise the situation below for the case $q \approx (n\log(n))^r$.




All index calculus algorithms for factoring and discrete logarithms are based on smoothness. A typical situation is to generate certain random elements $x$ modulo $N$ (or $p$), and check if they are *equal* to $\prod_i p_i^{e_i}$ for primes $p_i$ less than some bound (and exponents $e_i \in \mathbb{N}$). If one

could efficiently compute a smooth product $\prod_i p_i^{e_i}$ that is *congruent* to $x$ modulo $N$ (or $p$) then factorization and discrete logarithm algorithms would be revolutionised (and classical public key cryptography broken).

Our subset product problem is slightly different, since we impose the restriction $e_i \in \{0, 1\}$. But we still believe any fundamentally new algorithmic approach to the problem would likely lead to major advances. The only algorithms we know for this problem are "combinatorial" (in other words, requiring some kind of brute-force search), apart from when the density is extremely low and we can just factor. Note that our parameter choices (e.g. in Figure 6.2) are very far from such low density (as we require $r < n/2$ by Lemma 6.3).

We briefly discuss the relation with lattice problems in Section 5.6 and in Section 6.4.3. Our feeling is that the subset product problem is not really a lattice problem but a number-theoretical problem. As evidence, Ducas and Pierrot [DP19] and Naccache [Nac16] use similar number theory ideas to solve coding/lattice type problems, rather than using lattice techniques to solve the number-theoretical problems. Nevertheless, any new algorithms to solve Problem 5.1 would have implications in lattices, such as giving an improvement on the work of Ducas and Pierrot [DP19].

Ultimately, we are making a new assumption based on our experience and knowledge. Contini et al. [CLS06] made a similar assumption. We hope this work will inspire further study of these problems.

## 5.5 General Distributions

So far, in our discussion of the hardness of Problems 5.1 and 5.2, we have assumed that the the target vector $x$ is sampled from the uniform distribution on $\{0, 1\}^n$. In practice, *biometric features* induce an important example of distributions one could consider. It is important that the biometric features are distributed with high enough entropy, i.e. we require them to be Hamming distance evasive. Entropy of biometric features was studied by Li et al. [LSM06] and Sutcu et al. [SLM07].

## 5.6 The Relation Lattice

This section uses some definitions and the notation from [DP19]. Recall that in Section 5.4 we mentioned that index calculus algorithms for factoring and discrete logarithms work with *relations* of the form $x_j \equiv \prod_i p_i^{e_i}$, where the equivalence relation could be given by congruence modulo $N$, see for example [Buc88]. A collection of such relations is often called *factor base*. For a single relation, we call the vector $e = (e_1, \ldots, e_n)$ whose entries are the exponents $e_i$ a *relation vector*. It so happens that the relation vectors $e = (e_1, \ldots, e_n)$ actually form a group under addition (as already suggested by their name), and even more: They form a sub-lattice of $\mathbb{Z}^n$. This is one of the key observations that make index calculus algorithms work.

**Definition 5.3** (Relation Lattice)**.** *Let $n \in \mathbb{N}$ and $G$ be a group. For a group morphism $\phi : \mathbb{Z}^n \to G$, the* relation lattice $\Lambda_\phi$ *is given by $\Lambda_\phi = \ker \phi$.*

We can now construct the analogous lattice attached to the problems of Section 5.2. Let $n \in \mathbb{N}$ and let $(p_i)_{i=1,\ldots,n}$ be a sequence of distinct primes. Let $q$ be a prime distinct to $p_i$ for all

$i = 1, \ldots, n$ and consider the following group morphism

$$\phi : \mathbb{Z}^n \to (\mathbb{Z}/q\mathbb{Z})^*,$$
$$(x_1, \ldots, x_n) \mapsto \prod_{i=1}^{n} p_i^{x_i} \mod q.$$

If any of the $p_i$ is a primitive root modulo $q$ then $\phi$ is surjective, otherwise it need not necessarily be. Whether $\phi$ is surjective in the latter case is not immediately clear and depends on whether every element of $(\mathbb{Z}/q\mathbb{Z})^*$ can be written as such a product. Hence we are asking if $\{p_1, \ldots, p_n\}$ generates $(\mathbb{Z}/q\mathbb{Z})^*$.

The kernel of $\phi$ defines the relation lattice

$$\Lambda = \ker \phi = \left\{ x \in \mathbb{Z}^n \;\middle|\; \prod_{i=1}^{n} p_i^{x_i} = 1 \mod q \right\}.$$

Note that $\Lambda$ is indeed a lattice in the usual sense since the kernel of $\phi$ is in particular a $\mathbb{Z}$-module. This lattice encodes the multiplicative relation modulo $q$ of the primes in the sequence $(p_i)_{i=1,\ldots,n}$. For every $x \in \mathbb{Z}^n$ such that $X = \phi(x)$ it holds trivially that $X = \phi(x + v)$ for every $v \in \Lambda$. Thus, the relation lattice contains all the *shifts* $v$ that leave a fixed vector in $\mathbb{Z}^n$ invariant when mapped through $\phi$. This fact about $\Lambda$ will become important later.

We would also like to say something about the number of elements contained in a single fundamental parallelepiped of $\Lambda$, i.e. the volume $\mathrm{vol}(\Lambda)$ of the lattice. By the first isomorphism theorem it holds that $\mathrm{im}\,\phi = \mathbb{Z}^n/\ker\phi$ and so $|\mathrm{im}\,\phi| = |\mathbb{Z}^n/\Lambda|$. Hence, we have that the volume of the lattice is given by $|\mathrm{im}\,\phi|$ and thus bounded by

$$\mathrm{vol}(\Lambda) \leq q - 1$$

since $\varphi(q) = q - 1$ (here $\varphi$ is *Euler's totient function*). Note that equality holds if and only if $\{p_1, \ldots, p_n\}$ generates $(\mathbb{Z}/q\mathbb{Z})^*$.

If the sequence of primes $(p_i)_{i=1,\ldots,n}$ is sufficiently *random* (and thus the generated lattice is sufficiently *random*), then we may employ the Gaussian heuristic to estimate the size of the shortest lattice vector

$$\lambda_1 \approx \sqrt{\frac{n}{2\pi e}}\,\mathrm{vol}(\Lambda)^{1/n} \leq \sqrt{\frac{n}{2\pi e}}(q-1)^{1/n}, \tag{5.3}$$

where again equality holds if the $p_i$ generate $(\mathbb{Z}/q\mathbb{Z})^*$.

## 5.7 Post-Quantum Hardness

To the best of our knowledge there exists no classical nor quantum algorithm that efficiently solves either of Problem 5.1 or Problem 5.2 in general.

Consider an adversary that has access to a quantum computer for computing discrete logarithms in $(\mathbb{Z}/q\mathbb{Z})^*$ and consider Problem 5.1. Given an encoding $((p_i)_{i=1,\ldots,n}, q, X)$ of a secret $x \in \{0,1\}^n$, the adversary may turn it into a modular subset sum instance

$$\log_g(X) = \sum_i x_i \log_g(p_i) \mod (q-1)$$

by computing the discrete logarithms $\log_g(X)$ and $\log_g(p_i)$ with respect to some primitive root $g \mod q$. Such a modular subset sum problem may be classified by its density $d$, see [LO85;

Cos+92]. For a subset sum problem with *weights* $a_1, \ldots, a_n$ (i.e. $S = \sum_{i=1}^{n} x_i a_i$), the density $d$ is defined as follows:

$$d = \frac{n}{\log_2(\max_{i=1,\ldots,n} a_i)}.$$

Lagarias and Odlyzko [LO85] gave a polynomial time algorithm for low-density subset sum instances where $d < 0.645$. Coster et al. [Cos+92] later gave an improved algorithm which allows $d < 0.941$. This algorithm requires access to a perfect lattice oracle; just using LLL [LLL82] is not enough in general. Helm and May [HM18] gave a quantum algorithm for such a lattice oracle (solving the SVP problem) which requires $O(2^{0.226n})$ time and space.

In our case, the density is $d = n/\log_2(q)$ since we can expect $\log_g(p_i) \approx q$. Hence, we can give an estimate for when we expect post-quantum security. We have $q \approx (n \log n)^r$. Thus we can estimate the density by $d \approx n/(r \log_2(n \log n))$. To ensure a density of $d > 1$ we require

$$r < \frac{n}{\log_2(n \log n)} = r_{\mathrm{PQ}}(n). \tag{5.4}$$

Thus, we conjecture post-quantum hardness of the modular subset product problem when $r < r_{\mathrm{PQ}}(n)$, and potentially even for slightly larger values for $r$.

# 6 Obfuscated Hamming Distance

One very natural class of evasive functions is fuzzy matching for the Hamming metric: Define the program $P_x(y)$, parametrised by $x \in \{0,1\}^n$ and a threshold $0 < r < n/2$, that determines whether or not the $n$-bit input $y$ has Hamming distance at most $r$ from $x$. Let $D$ be a distribution on $\{0,1\}^n$. Then $D$ determines a program collection $\mathcal{P} = \{P_x : x \leftarrow D\}$. For $\mathcal{P}$ to be an evasive program collection it is necessary that the distribution $D$ has high Hamming ball min-entropy (see Definition 6.5; this notion is also known as fuzzy min-entropy in [FRS16]). The uniform distribution on $\{0,1\}^n$ is an example of such a distribution.

We are interested in obfuscating the membership program $P_x(y)$, so that the description of $P_x$ does not reveal the value $x$. Note that once an accepting input $y \in \{0,1\}^n$ is known then one can easily determine $x$ by a polynomial length sequence of chosen executions of $P_x$.

Indeed, as with most other solutions to this problem, the scheme we will introduce in the following is essentially an error correcting code, and so the computation recovers the value $x$.

Fuzzy matching has already been treated by many authors and there is a large literature on it. For example, Dodis et al. [DRS04; DS05; Dod+08] introduced the notion of *secure sketch* and a large number of works have built on their approach. They also show how to obfuscate proximity queries.

One drawback of the secure sketch approach is that the parameters are strongly constrained by the need for an efficient decoding algorithm. As discussed by Bringer et al. [Bri+08] this leads to "a trade-off between the correction capacity and the security properties of the scheme". Since the security analysis by Dodis and Smith [DS05] is information-theoretic, there is no discussion about the "form" of the leakage, although they prove that the secure sketch preserves entropic security. A related issue with secure sketches and fuzzy extractors is reusability [Boy04]. In general, fuzzy extractors are not secure if the same or correlated values are encoded multiple times. In contrast, our scheme is based on computational hardness rather than information-theoretic hardness and can be implemented for a much wider range of parameters.

Karabina and Canpolat [KC16] gave a different solution to fuzzy matching, based on computational assumptions related to the discrete logarithm problem. We sketch their scheme in Section 6.9. Note that they do not mention obfuscation or give a security proof. Wichs and Zirdelis [WZ17a] noted that fuzzy matching can be obfuscated using an obfuscator for compute-and-compare programs.

In practice our scheme improves upon all previous solutions to this problem: It handles a wider range of parameters than secure sketches; it is 20 times faster than [KC16]; it is many orders of magnitude more compact than [Che+18; WZ17a]; for full discussion see Section 6.7. Our solution is related to [KC16], but we think our approach is simpler and furthermore we give a complete security analysis.

We consider two variants of our scheme. One is based only on the subset-product assumption but when $r$ is very small, it admits the possibility of accepting an input $y$ that is not within the correct Hamming ball. We will only present the second variant which assumes the existence of a *dependent auxiliary input point function obfuscator* [LPS04; Wee05; BS16; BP12; Bit+14] and is

perfectly correct. The key idea is to use the point function obfuscator to verify the Hamming ball center after error correction, see Section 6.4.3 for details.

## 6.1 Hamming Distance

Consider the function that determines if an input binary vector is close to a fixed target. In our setting, the fixed target vector will be a secret and the other input vectors will be arbitrary. To make sense formally of such a function, we first need to specify which notion of *distance* we want to use. For binary vectors the so-called Hamming distance is a natural choice. The problem is then to construct a special purpose obfuscator for that function: Testing whether an arbitrary binary input vector is within a *Hamming ball* (of a fixed radius) around a secret binary target vector. A natural property of a binary vector is its *Hamming weight* which is the number of non-zero elements of the vector. For $x \in \{0,1\}^n$, we will denote this by $w_H(x)$.

**Definition 6.1** (Hamming Weight)**.** *Let $x \in \{0,1\}^n$ be a binary vector. Then the* Hamming weight *of $x$ is given by $w_H(x) = |\{i \mid x_i \neq 0\}|$.*

This weight function induces a natural notion of distance between two vectors. The *Hamming distance* between two binary vectors $x, y \in \{0,1\}^n$ is then given by $d_H(x,y) = w_H(x-y)$.

**Definition 6.2** (Hamming Distance)**.** *Let $x, y \in \{0,1\}^n$ be two binary vectors. Then the metric given by*

$$\begin{aligned} d_H : \{0,1\}^n \times \{0,1\}^n &\longrightarrow \mathbb{N}, \\ (x,y) &\longmapsto w_H(x-y) \end{aligned}$$

*is called the* Hamming distance *between $x$ and $y$.*

Finally, a *Hamming ball* $B_{H,r}(x) \subset \{0,1\}^n$ of radius $r$ around a point $x \in \{0,1\}^n$ is the set of all points with Hamming distance at most $r$ from $x$.

**Definition 6.3** (Hamming Ball)**.** *A* Hamming ball *$B_{H,r}(x) \subset \{0,1\}^n$ of radius $r$ around a point $x \in \{0,1\}^n$ is given by*

$$B_{H,r}(x) = \left\{ y \in \{0,1\}^n \mid d_H(x,y) \leq r \right\}.$$

*We denote by $B_{H,r}$ the Hamming ball around an unspecified point.*

**Remark 1.** *Note that Definition 6.1, Definition 6.2, and Definition 6.3 are given for binary vectors with entries in $\mathbb{Z}/2\mathbb{Z}$ but they make sense also for $q$-*ary *vectors in $(\mathbb{Z}/q\mathbb{Z})^n$ for some prime $q$.*

### 6.1.1 Hamming Ball Membership over Uniformly Chosen Centers

We stated in the introduction that we are interested in programs that determine if an input binary vector $y \in \{0,1\}^n$ is contained in a Hamming ball of radius $r$ around some secret value $x$, i.e. if $y \in B_{H,r}(x)$. This problem is only interesting if it is hard for a user to determine such an input $y$, because if it is easy to determine values $y$ such that $y \in B_{H,r}(x)$ and also easy to determine values $y$ such that $y \notin B_{H,r}(x)$ then an attacker can easily learn $x$ by binary search. So, the first task is to find conditions that imply it is hard to find a $y$ that is accepted by such a program. In other words, we need conditions that imply Hamming ball membership is an *evasive* problem (i.e. the program that tests whether an input is inside the Hamming ball around a fixed target should be evasive as in Definition 1.1). As we will see in Figure 6.1, there are essentially three ways that this problem can become easy: if the Hamming balls are too big; if there are too few possible centers $x$; or if the centers $x$ are clustered together.

### 6.1.2 Hamming Ball Program Collection.

Let $r, n \in \mathbb{N}$ with $0 < r < n/2$. For every binary vector $x \in \{0,1\}^n$ there exists a polynomial time program $P_x : \{0,1\}^n \to \{0,1\}$ that computes whether the input vector $y \in \{0,1\}^n$ is contained in a Hamming ball $B_{H,r}(x)$ and evaluates to 1 in this case, otherwise to 0. Any distribution on $\{0,1\}^n$ therefore gives rise to a distribution $P_n$ of polynomial time programs.

We first consider the uniform distribution on $\{0,1\}^n$, so that sampling $P \leftarrow P_n$ means choosing $x$ uniformly in $\{0,1\}^n$ and setting $P = P_x$. Since the condition $y \in B_{H,r}(x)$ is equivalent to $x \in B_{H,r}(y)$ we need to determine the probability that a random element lies in a Hamming ball. This is done in the next two lemmata. Note that if $r \geq n/2$ then a random element lies in the Hamming ball with probability $\geq 1/2$, which is why we are always taking $r < n/2$.

**Lemma 6.1.** *Let $n \in \mathbb{N}$, $x \in \{0,1\}^n$. The number of elements in a Hamming ball $B_{H,r}(x) \subseteq \{0,1\}^n$ of radius $r$ is given by*

$$h_r = |B_{H,r}| = \sum_{k=0}^{r} \binom{n}{k}.$$

*Proof.* This can be readily seen from the fact that for each $k \in [0, r]$ a vector has $\binom{n}{k}$ possible ways to be at Hamming distance of $k$ from the origin point. □

Next we show that the probability for a randomly chosen element in $\{0,1\}^n$ to be contained in a Hamming ball $B_{H,r}$ is negligible if the parameters $r < n/2$ are chosen properly. In order to prove this, we state the following fact first.

**Proposition 6.1** (Chernoff Bound for Binomial Distribution Tail [Che+52; AS92])**.** *Let $X$ be a random variable following a binomial distribution with probability $p$ and $n$ repeats and let $r \leq np$. Then the cumulative binomial distribution with the same parameters is bounded by*

$$\Pr(X \leq r) = \sum_{k=0}^{r} \binom{n}{k} p^k (1-p)^{n-k} \leq \exp\left(-\frac{1}{2p}\frac{(np-r)^2}{n}\right).$$

**Lemma 6.2.** *Let $\lambda \in \mathbb{N}$ be a security parameter and let $r, n \in \mathbb{N}$ such that*

$$r \leq \frac{n}{2} - \sqrt{n\lambda \log(2)}.$$

*Fix a point $x \in \{0,1\}^n$. Then the probability that a randomly chosen vector $y \in \{0,1\}^n$ is contained in a Hamming ball of radius $r$ around $x$ satisfies*

$$\Pr_{y \leftarrow \{0,1\}^n}\left[y \in B_{H,r}(x)\right] \leq \frac{1}{2^\lambda}.$$

*Proof.* The total number of points in $\{0,1\}^n$ is given by $2^n$. By Lemma 6.1 we thus have the probability of a randomly chosen vector $y \in \{0,1\}^n$ to be contained in a Hamming ball of radius $r$ around a point $x$ given by

$$\Pr_{y \leftarrow \{0,1\}^n}\left[y \in B_{H,r}(x)\right] = \frac{h_r}{2^n} = \frac{1}{2^n}\sum_{k=0}^{r}\binom{n}{k}.$$

On the other hand, by Proposition 6.1 we have

$$\sum_{k=0}^{r} \binom{n}{k} p^k (1-p)^{n-k} \leq \exp\left(-\frac{1}{2p}\frac{(np-r)^2}{n}\right).$$

Substitute $p = 1/2$ to find $\Pr(X \leq r) = h_r/2^n$. Hence, for $r < n/2 - \sqrt{n\lambda \log(2)}$ we have

$$\frac{h_r}{2^n} \leq \exp\left(-(n/2 - r)^2/n\right) \leq \frac{1}{2^\lambda}$$

and the result follows. □

This result shows that Hamming ball membership over the uniform distribution is evasive when $r$ is small enough.

**Lemma 6.3.** *Let $\lambda(n)$ be such that the function $1/2^{\lambda(n)}$ is negligible. Let $r(n)$ be a function such that*

$$r(n) \leq n/2 - \sqrt{\log(2) n \lambda(n)}.$$

*Let $P_n$ be the set of programs that tests Hamming ball membership in $B_{H,r(n)}(x) \subseteq \{0,1\}^n$ over uniformly sampled $x \in \{0,1\}^n$. Then $\mathcal{P} = \{P_n\}_{n \in \mathbb{N}}$ is an evasive program collection.*

*Proof.* We need to show that, for every $n \in \mathbb{N}$ and for every $y \in \{0,1\}^n$, $\Pr_{P \leftarrow P_n}[P(y) = 1]$ is negligible. Note that

$$\Pr_{P \leftarrow P_n}[P(y) = 1] = \Pr_{x \leftarrow \{0,1\}^n}[y \in B_{H,r(n)}(x)] = \Pr_{x \leftarrow \{0,1\}^n}[x \in B_{H,r(n)}(y)]$$

and this is negligible by Lemma 6.2. □

### 6.1.3 Hamming Ball Membership over General Distributions

Biometric templates may not be uniformly distributed in $\{0,1\}^n$, so it is important to have a workable theory for fuzzy matching without assuming that the input data is uniformly sampled binary strings. For example, in the worst case, there is only a small number of possible values $x \in \{0,1\}^n$ that arise, in which case taking $y$ to be one of these $x$-values will show that Hamming ball membership is not evasive. More generally, as pictured in the right-hand panel of Figure 6.1, one could have many centers but if they are too close together then there might be values for $y$ such that $\Pr_{P \leftarrow P_n}[P(y) = 1]$ is not negligible.

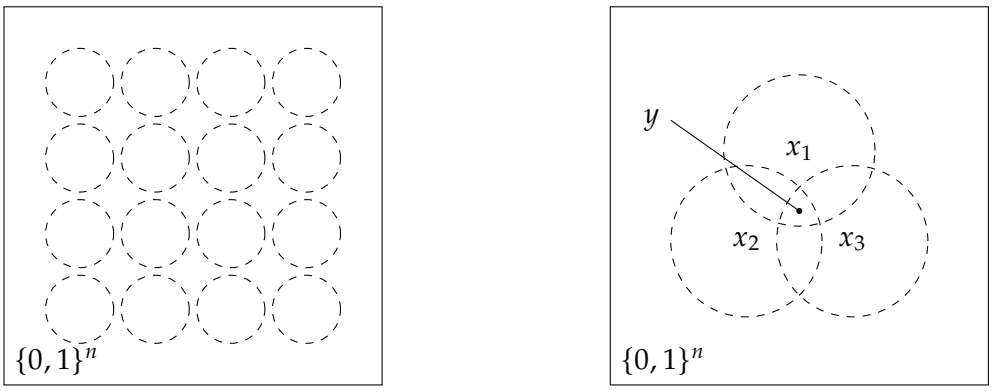



Figure 6.1: Two example cases of Hamming ball distributions. The left side depicts the ideal distribution of Hamming ball centers. The right one shows what happens if the balls overlap.

Hence, for Hamming ball membership to be evasive, the centers $x$ must be chosen from a "reasonably well spread" distribution. Before treating this in detail we give some definitions related to entropy of distributions in the computational sense. The following definition is taken from Wichs and Zirdelis [WZ17a].

**Definition 6.4** (Min-Entropy)**.** *The* min-entropy *of a random variable $X$ is defined as*

$$\mathrm{H}_\infty(X) = -\log\left(\max_x \Pr[X = x]\right).$$

*The* (average) conditional min-entropy *of a random variable $X$ conditioned on a correlated random variable $Y$ is defined as*

$$\mathrm{H}_\infty(X|Y) = -\log\left(\mathop{\mathrm{E}}_{y\leftarrow Y}\left[\max_x \Pr[X = x|Y = y]\right]\right).$$

Now suppose we have a distribution $D_n$ on $\{0,1\}^n$, which defines a distribution $P_n$ of Hamming ball membership programs. For the program collection to be evasive (i.e. to satisfy Definition 1.1), it is necessary that for any $y \in \{0,1\}^n$ we have $\Pr_{P\leftarrow P_n}[P(y) = 1]$ being negligible. But note that

$$\Pr_{P\leftarrow P_n}[P(y) = 1] = \Pr_{x\leftarrow D_n}[y \in B_{H,r}(x)] = \Pr_{x\leftarrow D_n}[x \in B_{H,r}(y)].$$

So the requirement for evasiveness is that this probability is negligible. In other words, we need that $D_n$ has large min-entropy in the following sense.

**Definition 6.5** (Hamming Ball Min-Entropy)**.** *Let $r < n \in \mathbb{N}$. The* Hamming ball min-entropy *of a random variable $X$ on $\{0,1\}^n$ is defined to be*

$$\mathrm{H}_{H,\infty}(X) = -\log\left(\max_{y\in\{0,1\}^n} \Pr[X \in B_{H,r}(y)]\right).$$

Note that Definition 6.5 is also known as *fuzzy min-entropy* [FRS16, Definition 3].

For convenience, we give some necessary conditions to have Hamming ball min-entropy at least $\lambda$. Let $|D_n| = \{x \in \{0,1\}^n : \Pr(x \leftarrow D_n) > 0\}$ be the support of $D_n$. If for any $y \in \{0,1\}^n$

$$\frac{\left|\bigcup_{x\in|D_n|} B_{H,r}(x)\right|}{|B_{H,r}(y)|} < 2^\lambda$$

then we certainly do not have min-entropy at least $\lambda$. Hence at the very least it is required that points in $D_n$ are well-spread-out, as pictured in the left-hand panel of Figure 6.1.

Intuitively, we can say that if there are enough points in $|D_n|$ and if they are spread out such that the overlap between the Hamming balls is relatively small, then Hamming ball membership is an evasive problem.

**Definition 6.6** (Hamming Distance Evasive Distribution)**.** *Consider an ensemble of distributions $D_\lambda$ over $\{0,1\}^{n(\lambda)}$, call it $\mathcal{D} = \{D_\lambda\}_{\lambda\in\mathbb{N}}$. Let $r(\lambda) < n(\lambda)/2$ be some function. We say that $\mathcal{D}$ is* Hamming distance evasive *if the Hamming ball min-entropy of $D_\lambda$ for Hamming balls in $\{0,1\}^{n(\lambda)}$ of radius $r(\lambda)$ (as in Definition 6.5) is at least $\lambda$.*

## 6.2 Hamming Distance With Auxiliary Information

We eventually want to prove that our Hamming distance obfuscator is VBB secure. For this we need to consider a slightly more general setting, where we additionally allow auxiliary information.

**Definition 6.7** (Conditional Hamming Ball Min-Entropy)**.** *Let $r < n \in \mathbb{N}$. The* Hamming ball min-entropy *of a random variable $X$ on $\{0,1\}^n$ conditioned on a correlated variable $Y$ is defined to be*

$$\mathrm{H}_{H,\infty}(X|Y) = -\log\left(\mathop{\mathrm{E}}_{y\leftarrow Y}\left[\max_{z\in\{0,1\}^n}\Pr[X \in B_{H,r}(z)|Y=y]\right]\right).$$

**Definition 6.8** (Hamming Distance Evasive Distribution With Auxiliary Information)**.** *Consider an ensemble of distributions $D_\lambda$ over $\{0,1\}^{n(\lambda)}\times\{0,1\}^{\mathrm{poly}(n(\lambda))}$, call it $\mathcal{D} = \{D_\lambda\}_{\lambda\in\mathbb{N}}$. Let $r(\lambda) < n(\lambda)/2$ be some function. We say that $\mathcal{D}$ is* Hamming distance evasive *if the Hamming ball min-entropy of $(x,\alpha) \leftarrow D_\lambda$ for Hamming balls in $\{0,1\}^{n(\lambda)}$ of radius $r(\lambda)$ conditioned on the auxiliary information $\alpha$ is at least $\lambda$.*

We consider an *entropic* version of Problem 5.2, where we additionally allow some auxiliary information. We believe this computational problem stays hard whenever the conditional Hamming ball min-entropy of the secrets $x$ given the auxiliary information $\alpha$ is large enough.

**Problem 6.1** (Entropic Decisional Distrib. Modular Subset Product Problem)**.** *Let $r < n \in \mathbb{N}$ and $D$ be a distribution over $\{0,1\}^n \times \{0,1\}^{\mathrm{poly}(n)}$. Let $\rho$ be a distribution of primes $(p_i)_{i=1,\dots,n}$ and $q$ as in Equation (5.1). Define the distribution*

$$D_0 = ((p_i)_{i=1,\dots,n}, q, X, \alpha)$$

*where $X$ satisfies Equation (5.2) for some vector $x$ where $(x,\alpha) \leftarrow D$. Define the distribution*

$$D_1 = ((p_i)_{i=1,\dots,n}, q, X', \alpha)$$

*where $(p_i)_{i=1,\dots,n}$, $q$, and $\alpha$ are as in $D_0$, but $X' \leftarrow (\mathbb{Z}/q\mathbb{Z})^*$ uniformly. Here $\alpha$ is some auxiliary information.*

*Then the* entropic $(r,n,D,\rho)$-decisional distributional modular subset product problem (entropic D-MSP$_{r,n,D,\rho}$) *is to distinguish $D_0$ from $D_1$. In other words, given a sample from $D_b$ for uniform $b \in \{0,1\}$, the problem is to determine $b$.*

Consider now for example the uniform distribution over $\{0,1\}^n$ as in Lemma 6.2. For a security parameter $\lambda \in \mathbb{N}$ and a Hamming ball radius $r \in \mathbb{N}$, it shows that if $r \leq n/2 - \sqrt{\log(2)n\lambda}$, then Hamming balls with uniformly distributed centers have at least $\lambda$ min-entropy. If we now also output auxiliary information $\alpha = \varphi(x)$ consisting of a one-bit predicate $\varphi : \{0,1\}^n \to \{0,1\}$ depending on the Hamming ball center $x$, then this lowers the min-entropy by at most one. Hence, to keep the security parameter $\lambda$, we require in this example that $r \leq n/2 - \sqrt{\log(2)n(\lambda+1)}$. We find that a slightly smaller maximal Hamming ball radius $r$ is possible compared to the case without auxiliary information.

## 6.3 Obfuscating Hamming Distance

In this section we describe our new obfuscator for fuzzy Hamming distance testing. The key idea is to encode a target vector in $\{0,1\}^n$ as a modular subset product as introduced in Section 5.2.

Let $r, n \in \mathbb{N}$ with $r < n/2$. Choose a random sequence of small distinct primes $(p_i)_{i=1,\dots,n}$ (i.e. $p_i \neq p_j$ for $i \neq j$). By the prime number theorem it suffices to randomly sample each $p_i$ from the interval $[2, O(n\log(n))]$. Choose then a safe prime $q$ such that $\prod_{i\in I} p_i < q/2$ for all $I \subset \{1,\dots,n\}$ with $|I| \leq r$. The prime $q$ should be sampled to satisfy the bound $q/2 < (1+o(1))\max\{p_i\}^r$ as in Equation (5.1). This corresponds to sampling from the minimal prime distribution given

by Definition 5.2. We refer to the discussion regarding Equation (7.3) to justify why we may assume that such a suitable safe prime exists.

To encode an element $x \in \{0,1\}^n$, publish

$$X = \prod_{i=1}^{n} p_i^{x_i} \mod q \tag{6.1}$$

along with the list of primes $(p_i)_{i=1,\dots,n}$ and $q$. Note that, for this encoding to hide $x$, we require that $w_H(x) > r$ and $\prod_{i=1}^{n} p_i^{x_i} > q$.

Given another element $y \in \{0,1\}^n$ we can now check if $y \in B_{H,r}(x)$ using the encoding $X$. First we compute $Y = \prod_{i=1}^{n} p_i^{y_i} \mod q$ from which we can find

$$E = XY^{-1} \mod q = \prod_{i=1}^{n} p_i^{x_i - y_i} \mod q = \prod_{i=1}^{n} p_i^{\epsilon_i} \mod q$$

where $\epsilon_i \in \{-1, 0, 1\}$. We show in Lemma 6.4 that if $y \in B_{H,r}(x)$ then we are able to recover the errors $\epsilon_i$ using continued fraction decomposition and factoring.

### 6.3.1 Why a Safe Prime.

Recall the following group homomorphism called the *Legendre symbol* [Leg98]

$$\left(\frac{\cdot}{q}\right) : \mathbb{F}_q^* \to \{\pm 1\}.$$

The Legendre symbol $(\frac{a}{q})$ is $+1$ if $a$ is a non-zero square modulo $q$ and $-1$ if $a$ is a non-zero non-square. Hence, the Legendre Symbol indicates whether an element in $\mathbb{F}_q^*$ is a quadratic residue or not. It is multiplicative in the sense that

$$\left(\frac{ab}{q}\right) = \left(\frac{a}{q}\right)\left(\frac{b}{q}\right).$$

For $X$ as in Equation (6.1) the Legendre symbol $(\frac{X}{q})$ is equal to the product $\prod_i (\frac{p_i}{q})^{x_i}$, which reveals a linear equation in the secret $(x_i)_{i=1,\dots,n}$. In other words, the encoding $X$ leaks one bit of information about $x$. Note that this does not violate the definition of VBB security: since the primes $p_i$ are chosen randomly by the obfuscator, we cannot fix in advance a predicate and compute it using the Legendre symbol.

If there were other small prime divisors of $q-1$ then one could extend this idea to get further linear equations. Hence we choose $q$ to be a safe prime to ensure that only the single bit of leakage arises. An alternative solution would be to square $X$, but this would mean we need to use larger parameters to do fuzzy matching with Hamming weight $r$, so we prefer to use the minimal parameters.

## 6.4 Obfuscator and Obfuscated Program

To be precise, for every pair of integers $r < n/2 \in \mathbb{N}$ and every binary vector $x \in \{0,1\}^n$ there exists a polynomial time program $P_x : \{0,1\}^n \to \{0,1\}$ that computes whether the input vector $y \in \{0,1\}^n$ is contained in a Hamming ball $B_{H,r}(x)$ and evaluates to 1 in this case, otherwise to 0. Denote the family of all such programs with $\mathcal{P}$.

The Hamming distance obfuscator $\mathcal{O}_H : \mathcal{P} \to \mathcal{P}'$ takes one such program $P_x \in \mathcal{P}$ and uses Algorithm 6.2 to output another polynomial time program in a different family denoted by $\mathcal{P}'$. In our case this is the decoding algorithm along with the polynomial size elements $(p_i)_{i=1,\ldots,n}, q$ and $X \in (\mathbb{Z}/q\mathbb{Z})^*$.

We furthermore require a dependent auxiliary input point function obfuscator [BP12; Bit+14] that we call $\mathcal{O}_{PT}$. Let $R_z : \{0,1\}^n \to \{0,1\}$ be a program that takes an input $y \in \{0,1\}^n$ and outputs 1 if and only if $y = z$. The point function obfuscator outputs an obfuscated version $\mathcal{O}_{PT}(R_z)$ of $R_z$. In addition to the output of Algorithm 6.2, our obfuscator $\mathcal{O}_H$ also outputs $Q \leftarrow \mathcal{O}_{PT}(R_x)$.

As the decoding algorithm is a universal algorithm, we will simply denote the *obfuscated program* $\mathcal{O}_H(P)$ with the tuple $((p_i)_{i=1,\ldots,n}, q, X, Q)$. During the execution of the obfuscated program, Algorithm 6.4 is run on $(n, (p_i)_{i=1,\ldots,n}, q, X, y)$ and returns either $\perp$ (in which case the program returns 0) or a candidate value $x'$. The obfuscated program then outputs $Q(x')$, which is 1 if and only if $x' = x$. Formally, the obfuscated program is given in Algorithm 6.1.

**Algorithm 6.1** Obfuscated Program (with embedded data $(p_i)_{i=1,\ldots,n}, q, X, Q$)

**procedure** EXECUTE($y \in \{0,1\}^n$)
  $x' =$ DECODE($n, (p_i)_{i=1,\ldots,n}, q, X, y$)
  **if** $x' = \perp$ **then return** 0
  **return** $Q(x')$
**end procedure**

The encoder (Algorithm 6.2) receives as an input the distance threshold $r$, the vector size $n$ and the target vector $x$. It then outputs the encoding represented by a triple $((p_i)_{i=1,\ldots,n}, q, X)$. The primes $(p_i)_{i=1,\ldots,n}$ and $q$ in the output of Algorithm 6.2 fit the minimal prime distribution $\rho_{r,n}$, see Definition 5.2.

**Algorithm 6.2** Encoding (Obfuscation)

**procedure** ENCODE($r < n/2 \in \mathbb{N}$; $x \in \{0,1\}^n$)
  Sample a random sequence of distinct primes $(p_i)_{i=1,\ldots,n}$ from $[2, O(n \log(n))]$.
  Sample small safe prime $q$ such that $\forall I \subset \{1,\ldots,n\}$ with $|I| \leq r$, $\prod_{i \in I} p_i < q/2$.
  Compute $X = \prod_{i=1}^{n} p_i^{x_i} \mod q$.
  **return** $((p_i)_{i=1,\ldots,n}, q, X)$
**end procedure**

The constrained factoring algorithm (Algorithm 6.3) factors an input number using a fixed list of primes and outputs the factors on success. It fails if the input is composite with factors that are not in the list of primes.

The decoder (Algorithm 6.4) receives as an input an encoding in the form of a triple $((p_i)_{i=1,\ldots,n}, q, X)$ and a test vector. It then attempts to decode the triple and outputs the original target vector or fails if the test vector was not within the required distance threshold.

**Algorithm 6.3** Constrained Factoring

**procedure** CFactor($n, (p_i)_{i=1,\dots,n}, x \in \mathbb{N}$)
  Set $F = \{\}$.
  **for** $i = 1, \dots, n$ **do**
    **if** $p_i \mid x$ **then** append $p_i$ to $F$ and reduce $x \leftarrow x/p_i$
  **end for**
  **return** $F$ **if** $x = 1$ **else** $\perp$
**end procedure**

**Algorithm 6.4** Decoding (Executing the obfuscated program)

**procedure** Decode($n, (p_i)_{i=1,\dots,n}, q \in \mathbb{N}$; $X \in (\mathbb{Z}/q\mathbb{Z})^*$; $y \in \{0,1\}^n$)
  Compute $Y^{-1} = \prod_{i=1}^{n} p_i^{-y_i} \mod q$.
  Compute $E = XY^{-1} \mod q$.
  Compute the continued fraction representation of $E/q$, with convergents $C$.
  **for all** $h/k \in C$ **do**
    $F \leftarrow$ CFactor($n, (p_i)_{i=1,\dots,n}, k$), $F' \leftarrow$ CFactor($n, (p_i)_{i=1,\dots,n}, kE \mod q$)
    **if** $F \neq \perp$ and $F' \neq \perp$ **then**
      Let $m = (0, \dots, 0) \in \{0,1\}^n$ be the zero vector.
      **for** $i = 1, \dots, n$ **do**
        **if** $p_i \in F \cup F'$ **then** set $m_i = 1$
      **end for**
      **return** $y \oplus m$
    **end if**
  **end for**
  **return** $\perp$
**end procedure**

### 6.4.1 Decoding

In this section we will analyse decoding complexity and efficiency. For decoding we have to factor the product

$$E = \prod_{i=1}^{n} p_i^{\epsilon_i} \mod q, \tag{6.2}$$

where $\epsilon_i \in \{-1, 0, 1\}$. First, we note that Equation (6.2) can be written as $ND^{-1}$ modulo $q$, or in other words $ED = N + sq$, $N = \prod_{i=1}^{n} p_i^{\mu_i}$, and $D = \prod_{i=1}^{n} p_i^{\nu_i}$ for some $s \in \mathbb{Z}$ and where now $\mu_i, \nu_i \in \{0,1\}$, $\mu_i \nu_i = 0$ for all $i$. By expanding $E/q$ into a continued fraction we are then able to recover $s/D$ from one of the convergents $h_i/k_i$ for some $i \in \mathbb{N}$ under the condition that $ND < q/2$. Hence decoding always succeeds since we have chosen the primes $(p_i)_{i=1,\dots,n}$ and $q$ such that $ND = \prod_{i \in I} p_i < q/2$ for some $I \subset \{1, \dots, n\}$ with $|I| \leq r$.

**Lemma 6.4** (Correctness). *Let $O_{PT}$ be a dependent auxiliary input point function obfuscator. Consider the algorithms* Encode *(Algorithm 6.2) and* Execute *(Algorithm 6.1). For every $r < n/2 \in \mathbb{N}, x \in \{0,1\}^n$, for every*

$$Q \leftarrow O_{PT}(R_x)$$
$$((p_i)_{i=1,\dots,n}, q, X) \leftarrow \text{Encode}(r, n, x)$$

*and for every* $y \in \{0,1\}^n$ *such that* $d_H(x,y) \leq r$ *it holds that*

$$Q(\textsc{Decode}(n, (p_i)_{i=1,\ldots,n}, q, X, y)) = 1.$$

*Proof.* To see why we require $ND < q/2$, note that there exists an $s \in \mathbb{Z}$ such that $ED - sq = N$. Therefore

$$\left|\frac{E}{q} - \frac{s}{D}\right| = \frac{N}{qD}.$$

Now Theorem 4.1 asserts that $s/D$ is a convergent of $E/q$ if

$$\left|\frac{E}{q} - \frac{s}{D}\right| < \frac{1}{2D^2}$$

and so we find the requirement $ND < q/2$.

For each convergent $h_i/k_i$ of $E/q$, $k_i$ and $k_iE \mod q$, respectively, can be factored separately using the $p_i$ to recover the $\nu_i$ and $\mu_i$ from which $x \in \{0,1\}^n$ can then finally be recovered using $y \in \{0,1\}^n$. This all works assuming $y \in B_{H,r}(x)$ since then the factors of $N$ and $D$ will be unique (of multiplicity 1) and contained in the sequence $(p_i)_{i=1,\ldots,n}$. If now $y \notin B_{H,r}(x)$ then with high probability (dependent on $r, n$) the factors of $N$ and $D$ will not be unique and/or not contained in $(p_i)_{i=1,\ldots,n}$ in which case the decoding fails. Finally correctness follows from the point obfuscation $Q$ of $x$. □

**Remark 2.** *Note that our decoding algorithm can also be used to solve the problem of matching distance in* $\mathbb{Z}^n$ *under the* $\ell_1$ *norm. If* $X = \prod_i p_i^{x_i} \mod q$ *is an encoding of* $x \in \mathbb{Z}^n$ *and if* $y \in \mathbb{Z}^n$ *is such that* $\|x - y\|_1 \leq r$ *then computing continued fractions and factoring still reveals the error vector* $e = x - y \in \mathbb{Z}^n$.

### 6.4.2 Decoding Efficiency.

We will now argue that decoding $E$ is efficient. Assuming that $E = XY^{-1}$ for some $x, y \in \{0,1\}^n$ such that $d_H(x,y) < r$, one of the convergents $h_i/k_i$ will yield $s/D$. From Section 4.1 we know that in our case the number of convergents we have to factor is of the order $O(\log(q))$ in the worst case. Because we fixed a list of small primes $p_i$ beforehand, we can test for a proper convergent $h_i/k_i$ and simultaneously factor $N$ and $D$ efficiently. Thus, decoding is of the order $O(n \log(q))$ in the number of (modular) multiplications/divisions. By the prime number theorem we may take $q \approx (n \log n)^r$ and thus decoding is also of the order $O(nr \log(n \log n))$.

### 6.4.3 Avoiding False Accepts

Recall the relation lattice we have introduced in Section 5.6. It can happen that it contains short vectors which interfere with the correctness of the Hamming distance obfuscator in the following way: There is a possibility of incorrectly accepted inputs. We define a *false accept* to be an input $y$ that is far from $x$ but such that $E$ has a smooth product representation.

**Definition 6.9** (False Accept)**.** *Fix* $r < n/2 \in \mathbb{N}$, $(p_i)_{i=1,\ldots,n}, q$ *as in Section 6.3, and* $x \in \{0,1\}^n$. *Set* $X = \prod_{i=1}^n p_i^{x_i} \mod q$. *A* false accept *is a vector* $y \in \{0,1\}^n$ *such that* $d_H(x,y) > r$ *and yet* $XY^{-1} = \prod_{i=1}^n p_i^{\epsilon_i} \mod q$ *can be factored where* $\epsilon \in \{-1,0,1\}^n$ *with* $w_H(\epsilon) \leq r$.

Recall that the obfuscator $O_H$ defined in Section 6.4 additionally outputs $Q = O_{PT}(R_x)$ which is used to prevent such false accepts. We will explain in this section that the additional step can be omitted if $r$ is chosen such that

$$r > \log(2\sqrt{2\pi e})\frac{n}{\log(n\log(n))} = r_f(n) \tag{6.3}$$

where we assume that $q \approx (n\log(n))^r$. Such an $r$ implies that the relation lattice will not contain short enough vectors that generate false accepts.

**Lemma 6.5.** *Let $r < n/2 \in \mathbb{N}$ and $q \approx (n\log(n))^r$. Assume that the* Gaussian heuristic *holds and that $r > r_f(n)$ (where $r_f(n)$ is defined by Equation (6.3)). Then the Hamming distance obfuscator with parameters $r, n, q$ does not admit false accepts.*

*Proof.* Let $y$ be a false accept, which means $XY^{-1} \equiv \prod_i p_i^{\epsilon_i} \mod q$ as in Equation (6.2), where $\epsilon_i \in \{-1, 0, 1\}$. Then

$$\prod_{i=1}^{n} p_i^{x_i - y_i - \epsilon_i} = 1 \mod q$$

where $-2 \leq x_i - y_i - \epsilon_i \leq 2$. It follows that there is a non-zero vector in the lattice

$$\Lambda = \left\{ x \in \mathbb{Z}^n \;\middle|\; \prod_{i=1}^{n} p_i^{x_i} = 1 \mod q \right\}$$

with norm bounded by $2\sqrt{n}$. For more details about this lattice see Section 5.6.

We now explain why Equation (6.3) implies that $\Lambda$ is not expected to have a short enough vector to admit a false accept. Assuming the Gaussian heuristic holds, Equation (5.3) tells us that the expected length of the shortest vector in $\Lambda$ is

$$\lambda_1 \approx \sqrt{\frac{n}{2\pi e}} q^{1/n} = \sqrt{\frac{n}{2\pi e}} (n\log(n))^{r/n}.$$

Hence, if we choose $r, n \in \mathbb{N}$ such that $\lambda_1 > 2\sqrt{n}$, i.e. such that

$$(n\log(n))^{\frac{r}{n}} > 2\sqrt{2\pi e}, \tag{6.4}$$

then the Hamming distance obfuscator with parameters $r, n, q$ does not admit false accepts. □

**Remark 3.** *Equation (6.4) assumes that $q \approx (n\log(n))^r$, i.e. the size of the primes $p_i$ is as small as possible. If we want to be able to use a smaller $r$ we may also choose the primes $p_i > n\log(n)$.*

## 6.5 Relation to Code-Based Constructions

Dodis et al. [DS05; Dod+08] introduced the notion of a *secure sketch*. Let $H$ be a secure cryptographic hash function. The first step is to construct a random liner error-correcting code over $\mathbb{F}_2$ of length $n$ with generator matrix $G$. To encode a vector $x \in \{0,1\}^n$ one chooses a random vector $s$ (of dimension equal to the dimension of the code) and encodes $x$ as the pair $(X, h)$ where $X = x \oplus Gs$ and $h = H(s)$. Note that the target vector $x$ needs to be of high enough entropy in order to sufficiently hide all information about it in the encoding $X$. To test whether a vector $y$ is close, one can compute $X \oplus y$, decode the result (using the decoding algorithm

for the code) to find a binary vector $s'$, and test whether $H(s') = h$. If $y = x \oplus e$ for some error vector $e$ then $X \oplus y = e \oplus Gs$ and so the process is correct. The hard problem in this case is: Given $(G, x \oplus Gs)$, find $x$. Since a decoding algorithm is part of the secure sketch, an attacker can efficiently compute a parity check matrix $P$ such that $PG = 0$. Then, given $X$, one can compute $PX = Px$, which is a system of linear equations in the secret vector $x$. This leaks partial information about $x$.

Using the parity check matrix $P$, we can mount a *dual* scheme to the above construction. In this scheme we would output $(P, Px)$ for a target vector $x \in \{0,1\}^n$. The hard problem for the dual scheme is: Given $(P, Px)$, find x.

In light of Section 5.6, we can view the vectors in the relation lattice $\Lambda$ for a sequence of primes $(p_i)_{i=1,\ldots,n}$ modulo $q$ as codewords, and the morphism $\phi(x) = \prod_{i=1}^{n} p_i^{x_i} \mod q$ as the parity check map: It holds that $\phi(x) = 1$ if and only if $x$ is a codeword. Then we can view our Hamming distance obfuscator as the dual scheme to a theoretical secure sketch with respect to the code based on $\Lambda$, cf. Ducas and Pierrot [DP19].

## 6.6 Example Parameters

The parameters of the Hamming distance obfuscator can be chosen flexibly. We want to emphasise that a priori any vector size $n \in \mathbb{N}$ is possible. The actual security level of the obfuscator depends on the error parameter $r < n/2$ which we expect to be fixed by the demands of the application. Note that it is naturally bounded by Lemma 6.2. Assuming a uniform distribution of possible target vectors $x$, the bit-security (meaning logarithm to base 2 of the expected number of operations to find an accepting input) of a parameter set $(r, n)$ can be calculated using

$$\lambda_{r,n} = -\log_2\left(\frac{h_r}{2^n}\right)$$

where $h_r$ is defined in Lemma 6.1. We give some example parameter sets along with their bit-security in Figure 6.2.

| $r$ | $\lfloor \lambda_{r,n} \rfloor$ | $\lfloor \log_2(q) \rfloor$ |
|---|---|---|
| 155 | 64 | 1804 |
| 128 | 102 | 1490 |
| 32 | 346 | 372 |

(a) $n = 512$ ($r_f(n) = 134$, $r_{\mathrm{PQ}}(n) = 44$)

| $r$ | $\lfloor \lambda_{r,n} \rfloor$ | $\lfloor \log_2(q) \rfloor$ |
|---|---|---|
| 306 | 128 | 3915 |
| 256 | 199 | 2546 |
| 64 | 686 | 818 |

(b) $n = 1024$ ($r_f(n) = 244$, $r_{\mathrm{PQ}}(n) = 80$)

Figure 6.2: Example parameter sets for obfuscated Hamming distance with $r < n/2$ and bit-security parameter $\lambda_{r,n}$. We estimate the size of $q$ by $q \approx (n \log n)^r$. When $r > r_f(n)$ (see Equation (6.3)) we do not expect false accepts, and so do not need to use the point obfuscator. When $r < r_{\mathrm{PQ}}(n)$ (see Equation (5.4)) then the scheme is conjectured to be post-quantum secure (as long as the point obfuscator is post-quantum secure).

## 6.7 Performance

For completeness, we have implemented (an unoptimised version of) the Hamming distance obfuscator using the C programming language and conducted experiments on a desktop computer (Intel(R) Core(TM) i7-4770 CPU 3.40 GHz). We take $n = 511$ and $r = 85$ (i.e.

$\lfloor \lambda_{85,511} \rfloor = 185$ and $\lfloor \log_2(q) \rfloor = 989$) to allow comparison with [Tuy+05; KC16]. We measured the time to produce and decode 1000 obfuscation instances. We found an average encoding time of 52 ms and an average decoding time of 14 ms. In comparison Karabina and Canpolat [KC16] found 100 ms for encoding and 350 ms for decoding respectively on a similar computer (Intel(R) Xeon(R) CPU E31240 3.30 GHz). It is possible to further speed up encoding by choosing a good safe prime generating algorithm and decoding can be parallelised in the factoring steps instead of attempting to factor after computing each new convergent. Note that the data $((p_i)_{i=1,\dots,n}, q, X)$ can be stored in less than one kilobyte.

An interactive model of program obfuscation called *token-based obfuscation* was first considered by Goldwasser et al. [Gol+13] and an LWE based implementation was presented by Chen et al. [Che+18]. They found experimentally that "For the case of the Hamming distance threshold of 3 and 24-bit strings, the TBO construction requires 213 GB to store the obfuscated program." Obfuscation took 72.6 minutes. Of course, the parameters $(n, r) = (24, 3)$ are much too small for the function to be evasive. Furthermore, this problem is easily solved using a secure sketch. In comparison our scheme can be implemented with realistic parameters like $(n, r) = (511, 85)$ and requires less than a kilobyte of storage and less than a second to run. Clearly the ring-LWE approach by Chen et al. [Che+18] is orders of magnitude worse than our scheme. Also for comparison, the scheme of Bishop et al. [Bis+18] (using the optimised variant of Bartusek et al. [Bar+19]) requires $n + 1$ elements of a group with a hard discrete logarithm problem. With $n = 511$ and a group of size $2^{256}$ this would require at least 16 kilobytes to store the program. To deflect any criticism of our performance analysis, we remark that none of the works [Bis+18; Bar+19; Che+18] give any performance comparison with previous proposals.

Fuller et al. [FMR13] gave a code-based construction for a fuzzy extractor (see Section 6.5) and Huth et al. [Hut+17] gave performance results of an implementation. While the performance is comparable to our solution, their solution is based on decoding LWE instances and hence only allows for a maximal error bound of $O(\log(n))$.

## 6.8 Polynomial Ring Variant

There is a variant of our Hamming distance obfuscator that uses a polynomial ring over a finite field to encode binary vectors. Note that this variant works analogously for the conjunction obfuscator.

Consider a field $k$ and the ring of polynomials $k[z]$ respectively the field of fractions $k(z)$. One can show that a similar statement to Theorem 4.1 holds; see Section 4.2 for a summary of the relevant theory and especially Theorem 4.2 for the generalised statement.

Let $R = k[z]$, then the idea is to replace $\mathbb{Z}/q\mathbb{Z}$ by $R/Q$ where the ideal $Q = (q(z))$ is generated by some suitable irreducible polynomial $q \in R$. For the ground field $k$ we may take a finite field of suitable order.

Given $r < n/2 \in \mathbb{N}$, encoding a target vector $x \in \{0,1\}^n$ follows the same process as before. We choose a random sequence of small distinct irreducible polynomials $(p_i)_{i=1,\dots,n}$ in $R$ and an irreducible polynomial $q$ such that

$$\sum_{i \in I} \deg(p_i) < \deg(q)$$

for all $I \subset \{1, \dots, n\}$ with $|I| \leq r$. To encode $x \in \{0,1\}^n$, publish $X = \prod_{i=1}^n p_i^{x_i} \mod q$ along with the polynomials $(p_i)_{i=1,\dots,n}$ and $q$. Given another element $y \in \{0,1\}^n$ we can check if $y \in B_{H,r}(x)$ using the encoding $X$. Again, to recover the errors $\epsilon_i$ we use continued fraction decomposition and factoring, though now in $R$. We are asserted that this works by Theorem 4.2.

### 6.8.1 Decoding Condition

Again, we find that Equation (6.2) can be written as $ED = N + sq$, $N = \prod_{i=1}^{n} p_i^{\mu_i}$, and $D = \prod_{i=1}^{n} p_i^{\nu_i}$ for some polynomial $s \in R$. Hence, we have that $\left|\frac{E}{q} - \frac{s}{D}\right| = \left|\frac{N}{qD}\right|$. Theorem 4.2 asserts that $s/D$ is a convergent of $E/q$ if $\left|\frac{E}{q} - \frac{s}{D}\right| < \frac{1}{|D|^2}$, i.e. for correctness in the polynomial case we find the relaxed requirement $|ND| < |q|$.

### 6.8.2 Comparison to $\mathbb{Z}/q\mathbb{Z}$-case

Using polynomials has several advantages: The ground field $k$ can be of small order since the order of $R/Q$ is given by $|R/Q| = |k|^{\deg(q)}$ and thus controllable by the size of $q \in R$. We may furthermore choose a compact representation of the irreducibles $(p_i)_{i=1,\dots,n}$ and $q$ to shrink encoding size and speed up computation.

## 6.9 The Scheme of Karabina and Canpolat

Karabina and Canpolat [KC16] give a secure fuzzy matching scheme that uses a particular subgroup of a finite field that has a representation as an algebraic torus. We briefly describe their scheme here, changing the notation to be consistent with our work and making some simplifications to the presentation.

Let $(r, n)$ be such that we want to do fuzzy matching on $x \in \{0,1\}^n$ up to Hamming distance $r < n/2$. First Karabina and Canpolat choose $p > 2n$ and $m = 2r$ and work in the subgroup $G$ of $\mathbb{F}^*_{p^{2m}}$ of order $p^m + 1$ (technically they work with a representation of this group as the algebraic torus $T_2(\mathbb{F}_{p^m})$).

To encode a string $x \in \{0,1\}^n$ they choose $n$ elements $g_i$ in a special subset $S$ of $G$ that has size $p$. It is also important that if $g_i \in S$ then $g_i^{-1} \notin S$. The encoding of $x$ is then

$$X = \prod_{i=1}^{n} g_i^{-2x_i+1}.$$

Given an input $y \in \{0,1\}^n$ one can determine if $y$ is within Hamming distance $r$ of $x$ by computing $Y = \prod_i g_i^{-2y_i+1}$ and then trying to write $XY^{-1}$ as a product of $r$ elements. This is done in [KC16] using some techniques from discrete logarithms in algebraic tori (namely, Weil descent and reduction to solving a system of polynomial equations).

We remark that there is no difference in security between the encoding $X = \prod_i g_i^{-2x_i+1}$ and the encoding $\tilde{X} = \prod_i g_i^{x_i}$, as one can convert between them: given $\tilde{X}$ one has $X = \tilde{X}^2(\prod_i g_i)^{-1}$; given $X$ one has $\tilde{X} = \sqrt{X(\prod_i g_i)}$, and it is easy to compute square roots in finite fields and to make the correct choice of sign.

Karabina and Canpolat [KC16] do not discuss obfuscation or give a formal security analysis. Their analysis is for uniform distributions on $\{0,1\}^n$, although they mention in [KC16, Remark 1] that in real situations the entropy of $x$ may be lower. We also note that they do not consider the possibility of false accepts, which follows from a similar analysis of an analogous relation lattice as in Section 6.4.3.

In terms of size, the encoding in their scheme is an element of $\mathbb{F}_{p^m}$ where $p^m \approx (2n)^{2r}$. For our scheme, an encoding is an element of $\mathbb{Z}_q$ where $q \approx 2(n \log(n))^r$. Hence, their scheme is more compact than ours. We can obtain a more compact scheme by considering the polynomial ring

variant. On the other hand, their scheme is more complex to understand and implement and the performance is weaker, see Section 6.7.

### 6.9.1 Security Analysis.

We define two computational problems in order to formalise the security of the scheme. Under the assumption that these problems are hard, the VBB and input-hiding security of the obfuscator follows immediately by the same arguments as used to prove Theorem 6.1 and Theorem 6.2.

**Problem 6.2** (Distributional Torus Subset Product Problem)**.** *Let $r < n \in \mathbb{N}$ and $D$ be a distribution over $\{0,1\}^n$. Set $m = 2r$. Given a prime $p > 2n$, the subgroup $G < \mathbb{F}^*_{p^{2m}}$ of order $p^m + 1$, $S \subset G$ as in [KC16], a sequence $(g_i)_{i=1,\dots,n}$ with $g_i \in S$, and an element*

$$X = \prod_{i=1}^{n} g_i^{-2x_i+1}$$

*for some vector $x \leftarrow D$, the* distributional torus subset product problem *is to find $x$.*

**Problem 6.3** (Decisional Distributional Torus Subset Product Problem)**.** *Let $r < n \in \mathbb{N}$ and $D$ be a distribution over $\{0,1\}^n$. Given parameters as in Problem 6.2, the* decisional distributional torus subset product problem *is to distinguish between the two distributions on tuples $((g_i)_{i=1,\dots,n}, X)$ and $((g_i)_{i=1,\dots,n}, X')$, respectively, corresponding to*

$$X = \prod_{i=1}^{n} g_i^{-2x_i+1}$$

*for some vector $x \leftarrow D$, and uniformly random*

$$X' \leftarrow G.$$

## 6.10 Security

Here we analyse the security of our Hamming distance obfuscator. We will show distributional VBB security and that the obfuscator is input-hiding. Our results will depend on the hardness of the *distributional modular subset product problem* that was introduced in Chapter 5, and some variants that we introduced in this chapter.

To show that the Hamming distance obfuscator is a distributional VBB obfuscator, we need to show that it satisfies all the properties of Definition 2.1. Note that Definition 2.1 for VBB obfuscation is given in asymptotic terms with respect to a security parameter $\lambda$. On the other hand, Problem 5.1 and Problem 6.1 are given in terms of explicit parameters $r, n \in \mathbb{N}$. Thus in the following, let the parameters $r, n \in \mathbb{N}$ be implicitly dependent on the security parameter $\lambda$, i.e. $r = r(\lambda), n = n(\lambda)$.

There are various ways to construct the dependent auxiliary input distributional VBB point function obfuscator $\mathcal{O}_{PT}$, see [LPS04; Wee05; BS16; BP12; Bit+14]. Here we will construct it from a *preimage resistant hash function* $H : \{0,1\}^n \to \{0,1\}^n$, which we model as a *random oracle*. In Section 6.4 we described the obfuscated program $Q = \mathcal{O}_{PT}(R_x)$ for a point $x \in \{0,1\}^n$. Here it is simply given by the hash value $h = H(x)$. To evaluate $Q$ on another point $y \in \{0,1\}$ we simply test whether $H(y) = h$ and output 1 in this case, otherwise we output 0.

We remind the reader about our assumption that the prime distribution $\rho$ in $\mathrm{MSP}_{r,n,D,\rho}$ and $\text{D-MSP}_{r,n,D,\rho}$ is the minimal prime distribution $\rho = \rho_{r,n}$ given by Definition 5.2.

**Theorem 6.1.** *Let $(n(\lambda), r(\lambda))$ be a sequence of parameters for $\lambda \in \mathbb{N}$. Let $\mathcal{D} = \{D_\lambda\}_{\lambda\in\mathbb{N}}$ be an ensemble of Hamming distance evasive distributions with auxiliary information (as in Definition 6.8). Suppose that entropic D-MSP$_{r,n,D,\rho}$ (Problem 6.1) is hard. Then the Hamming distance obfuscator $\mathcal{O}_H$ is a distributional VBB obfuscator for $\mathcal{D}$ in the random oracle model.*

*Proof.* Let $H : \{0,1\}^n \to \{0,1\}^n$ be a random oracle. On an input $z \in \{0,1\}^n$, if the oracle $H$ has been queried before on $z$, $H$ outputs the same answer as before. Otherwise, $H$ samples a random $h \leftarrow \{0,1\}^n$, records this as the answer to the input $z$, and outputs $h$. The point function obfuscator $\mathcal{O}_{PT}$ outputs $H(x)$ on the input of a point $x \in \{0,1\}^n$.

The obfuscator $\mathcal{O}_H$ is functionality preserving by Lemma 6.4. It is also clear that the obfuscator causes only a polynomial slowdown when compared to an unobfuscated Hamming distance calculation since the evaluation algorithm runs in time polynomial in all the involved parameters.

Let $\mathcal{D} = \{D_\lambda\}_{\lambda\in\mathbb{N}}$ now be an ensemble of Hamming distance evasive distributions with auxiliary information as in Definition 6.8. By Theorem 2.1 it is sufficient to show that there exists a (non-uniform) PPT simulator $\mathcal{S}$ such that, for the distribution ensemble $\mathcal{D}$, it holds that (where $\alpha$ denotes any auxiliary information)

$$(\mathcal{O}_H(P), \alpha) \overset{c}{\approx} (\mathcal{S}(|P|), \alpha).$$

We will construct the simulator $\mathcal{S}$. First the simulator $\mathcal{S}$ takes as input $|P|$ and determines the parameters $r, n \in \mathbb{N}$. Then it runs Algorithm 6.5 which will generate the first half of the eventual output. Lastly, the simulator $\mathcal{S}$ samples a uniformly random $h'$ from the codomain

**Algorithm 6.5** Encoding Simulator

**procedure** SimulateEncode($r < n/2 \in \mathbb{N}$)
  Sample random sequence of distinct primes $(p_i)_{i=1,\dots,n}$ from $[2, O(n\log(n))]$.
  Sample small safe prime $q$ such that $\forall I \subset \{1,\dots,n\}$ with $|I| \le r$: $\prod_{i\in I} p_i < q/2$.
  Sample $X' \leftarrow \mathbb{Z}/q\mathbb{Z}$ uniformly random.
  **return** $((p_i)_{i=1,\dots,n}, q, X')$
**end procedure**

$\{0,1\}^n$ of the random oracle $H$. Denote the simulator output by the tuple $((p_i)_{i=1,\dots,n}, q, X', h')$. It is clear that $\mathcal{S}$ is polynomial time since Algorithm 6.5 is too.

Consider now a real obfuscation $((p_i)_{i=1,\dots,n}, q, X, h)$ obtained from the Hamming distance obfuscator $\mathcal{O}_H$ described in Section 6.4. Since $H$ is a random oracle, $((p_i)_{i=1,\dots,n}, q, X)$ is independent of $h$. By construction $((p_i)_{i=1,\dots,n}, q, X')$ is independent of $h'$, and $h$ is computationally indistinguishable from $h'$. Finally, assuming that Problem 6.1 is hard, a real obfuscation and the simulator output are computationally indistinguishable:

$$((p_i)_{i=1,\dots,n}, q, X, h, \alpha) \overset{c}{\approx} ((p_i)_{i=1,\dots,n}, q, X', h', \alpha). \tag{6.5}$$

This completes the proof. □

**Remark 4.** *As noted in Section 6.4.3, the obfuscator $\mathcal{O}_H$ can be modified to omit the point obfuscation step. Hence Theorem 6.1 can be restated without requiring a distributional VBB obfuscator $\mathcal{O}_{PT}$ by assuming an ensemble of Hamming distance evasive distributions $\{D_\lambda\}_{\lambda\in\mathbb{N}}$ that satisfy Equation (6.3).*

Next, we will show that the Hamming distance obfuscator is input-hiding according to Definition 2.6.

**Theorem 6.2.** *Let $(n(\lambda), r(\lambda))$ be parameters satisfying $r > r_f(n)$ (recall Equation (6.3)). Let $\mathcal{D} = \{D_\lambda\}_{\lambda\in\mathbb{N}}$ be an ensemble of Hamming distance evasive distributions (as in Definition 6.6). Suppose that $MSP_{r,n,D,\rho}$ (Problem 5.1) is hard. Then the Hamming distance obfuscator $\mathcal{O}_H$ is input-hiding.*

*Proof.* The ensemble $\{D_\lambda\}_{\lambda\in\mathbb{N}}$ of Hamming distance evasive distributions induces an ensemble of programs $\{P_n\}_{n\in\mathbb{N}}$ (see Section 6.4). Recall Definition 2.6 of an input-hiding obfuscator. Suppose there exists a PPT adversary $\mathcal{A}$ such that the success probability is given by

$$g(n) = \Pr_{P\leftarrow P_n}\left[P(\mathcal{A}(\mathcal{O}_H(P))) = 1\right].$$

We will now construct an algorithm $\mathcal{A}'$ that solves Problem 5.1 given $\mathcal{A}$ with success probability $g(n)$. Let $((p_i)_{i=1,\dots,n}, q, X)$ be an instance of Problem 5.1. Since $r > r_f(n)$, this instance uniquely defines $x \in \{0,1\}^n$ such that $X = \prod_{i=1}^n p_i^{x_i} \mod q$, and hence defines a program $P$. Then $((p_i)_{i=1,\dots,n}, q, X)$ is a correct obfuscation of $P$. The algorithm $\mathcal{A}'$ runs the adversary $\mathcal{A}$ on $((p_i)_{i=1,\dots,n}, q, X)$ and $\mathcal{A}$ outputs a vector $y \in \{0,1\}^n$ that is accepted by $P$ with probability $g(n)$. Note that in Definition 2.6 the adversary outputs a valid input for $P$, not $\mathcal{O}_H(P)$. Hence, $y$ is close to $x$ as $r > r_f(n)$. Finally, $\mathcal{A}'$ decodes $X$ given $y$ using Algorithm 6.4 and thus outputs $x$ with probability $g(n)$.

But we assumed that Problem 5.1 is hard and hence $g(n)$ is negligible. □

# 7 Obfuscated Conjunctions

Another class of evasive functions are conjunctions. A conjunction on Boolean variables $b_1, \ldots, b_n$ is $\chi(b_1, \ldots, b_n) = \bigwedge_{i=1}^{k} c_i$ where each $c_i$ is of the form $b_j$ or $\neg b_j$ for some $1 \leq j \leq n$.

An alternative representation of a conjunction is called *pattern matching with wildcards*. Consider a vector $x \in \{0, 1, \star\}^n$ of length $n \in \mathbb{N}$ where $\star$ is a special *wildcard* symbol. Such an $x$ then corresponds to a conjunction $\chi : \{0,1\}^n \to \{0,1\}$ which, using Boolean variables $b_1, \ldots, b_n$, can be written as $\chi(b) = \bigwedge_{i=1}^{n} c_i$ where $c_i = \neg b_i$ if $x_i = 0$, $c_i = b_i$ if $x_i = 1$, and $c_i = 1$ if $x_i = \star$.

Conjunction obfuscators have been considered before [Bra+16; BR17; Bis+18; Bar+19]. See Section 7.7 for a summary of these. It is clear that, if the number $r$ of wildcards is sufficiently smaller than $n/2$, one can reduce pattern matching with wildcards to fuzzy Hamming matching. Hence our solution also gives an alternative approach to obfuscating conjunctions that can be used for certain parameter ranges. We give a full security analysis and comparison to existing schemes.

## 7.1 Conjunctions

**Definition 7.1** (Conjunction/Pattern Matching With Wildcards)**.** *Let $n \in \mathbb{N}$ and let $x \in \{0, 1, \star\}^n$ where $\star$ is a special* wildcard *symbol. Such an $x$ then corresponds to a* conjunction *$\chi : \{0,1\}^n \to \{0,1\}$ which, using a vector of Boolean variables $b = (b_1, \ldots, b_n)$, can be written as $\chi(b) = \bigwedge_{i=1}^{n} c_i$ where $c_i = \neg b_i$ if $x_i = 0$, $c_i = b_i$ if $x_i = 1$, and $c_i = 1$ if $x_i = \star$. Denote by $W_x = \{i | x_i = \star\}$ the set of all wildcard positions and let $r = |W| \in \mathbb{N}$ be the number of wildcards.*

Note that a priori the input is considered a plaintext and directly visible to the evaluating party. The wildcard positions of an obfuscated conjunction are only secret as long as no matching input is known. Once such an input is presented to the evaluator, it is straightforward to work out all wildcard positions in time linear in the input length: Simply flip each input bit and check whether this changed input still matches, in which case the flipped position must be a wildcard.

**Lemma 7.1.** *Let $\lambda \in \mathbb{N}$ be a security parameter and let $r < n/2 \in \mathbb{N}$ such that $r \leq n - \lambda$. Fix a conjunction $\chi$ corresponding to a vector $x \in \{0, 1, \star\}^n$ such that $r = |\{i | x_i = \star\}|$. Then the probability that $\chi$ evaluates to true on a randomly chosen vector $y \in \{0,1\}^n$ satisfies $\Pr_{y \leftarrow \{0,1\}^n}\left[\chi(y) = 1\right] \leq 1/2^{\lambda}$.*

*Proof.* The total number of points in $\{0,1\}^n$ is given by $2^n$. We thus have the probability of a randomly chosen vector $y \in \{0,1\}^n$ to be matched by $\chi$ to be $\Pr_{y \leftarrow \{0,1\}^n}\left[\chi(y) = 1\right] = 2^r/2^n$. This probability is upper-bounded by $1/2^{\lambda}$ if $r \leq n - \lambda$. □

Lemma 7.1 shows that all conjunctions which have their non-wildcard values uniformly distributed over $\{0,1\}^{n-r}$ are evasive. The following definition considers general distributions.

**Definition 7.2** (Conjunction Evasive Distribution)**.** *Consider an ensemble $\mathcal{D} = \{D_\lambda\}_{\lambda \in \mathbb{N}}$ of distributions $D_\lambda$ over $\{0, 1, \star\}^{n(\lambda)}$ with $r(\lambda)$-many wildcards for functions $r(\lambda) < n(\lambda)$. We say that $\mathcal{D}$ is* conjunction evasive *if the min-entropy of $D_\lambda$ is at least $\lambda$.*

**Definition 7.3** (Conjunction Evasive Distribution With Auxiliary Information)**.** *Consider an ensemble $\mathcal{D} = \{D_\lambda\}_{\lambda\in\mathbb{N}}$ of distributions $D_\lambda$ over $\{0,1,\star\}^{n(\lambda)}\times\{0,1\}^{\mathrm{poly}(n(\lambda))}$ with $r(\lambda)$-many wildcards for functions $r(\lambda) < n(\lambda)$. We say that $\mathcal{D}$ is* conjunction evasive *if the min-entropy of $(x,\alpha) \leftarrow D_\lambda$ conditioned on the auxiliary information $\alpha$ is at least $\lambda$.*

## 7.2 Reduction to Hamming Distance

We first give a generic reduction of pattern matching with wildcards to Hamming distance. Let $x \in \{0,1,\star\}^n$ be a pattern and let $r$ be the number of wildcards. Let $x' \in \{0,1\}^n$ be any string such that $x'_i = x_i$ for all non-wildcard positions $1 \le i \le n$. Then it is clear that any $y \in \{0,1\}^n$ that satisfies the pattern has Hamming distance at most $r$ from $x'$. The problem is that there are many other vectors $y$ that have Hamming distance at most $r$ from $x'$ but which do not satisfy the pattern. Furthermore, pattern matching with wildcards can be evasive with $r$ as large as $n-\lambda$ where $\lambda$ is a security parameter (e.g. $n = 1000$ and $r = 900$), while Hamming distance is not evasive if $r > n/2$. So it is clear that this is not a general reduction of obfuscating conjunctions to fuzzy matching.

However, in certain parameter ranges (where $r < n/2$) one can consider using fuzzy matching to give an approach to obfuscating conjunctions. As we will explain in this section, our scheme has some advantages over the generic reduction because inputs $y$ that match the pattern are more easily identified than vectors $y$ that are close to $x'$ in the Hamming metric but do not match the pattern. Indeed, we will explain that, for certain parameter ranges, our approach is much more compact than other solutions to the conjunction problem.

## 7.3 Obfuscating Conjunctions

In this section we describe a new obfuscator for conjunctions, based on the Hamming distance obfuscator that we introduced in Chapter 6.

Let $n \in \mathbb{N}$ and $x \in \{0,1,\star\}^n$. Choose a random sequence of small distinct primes $(p_i)_{i=1,\ldots,n}$ (i.e. $p_i \neq p_j$ for $i \neq j$). It suffices to randomly sample each $p_i$ from the interval $[(n\log(n))^2, ((n+1)\log(n+1))^2]$. Denote by $W_x = \{i \mid x_i = \star\}$ the set of indices such that $x_i$ is a wildcard. Assume we can choose a safe prime $q$ such that

$$\prod_{i\in W_x} p_i < \frac{q}{2} < \prod_{i\in W_x\cup\{j\}} p_i \tag{7.1}$$

for all $j \in \{1,\ldots,n\} \setminus W_x$. Set $r = |W_x|$; we furthermore require that $r < n/2$ as we will see shortly.

To encode $x$, consider the map $\sigma : \{0,1,\star\} \to \{-1,0,1\}$ that acts in the following fashion

$$0 \mapsto -1, \quad 1 \mapsto 1, \quad \star \mapsto 0.$$

Publish

$$X = \prod_{i=1}^{n} p_i^{\sigma(x_i)} \mod q$$

along with the list of primes $(p_i)_{i=1,\ldots,n}$ and the modulus $q$. Note that, for this encoding to hide $x$, we require $\prod_{i=1}^n p_i^{\sigma(x_i)} > q$.

Given a vector $y \in \{0,1\}^n$ such that $\chi(y) = 1$, we compute $Y = \prod_{i=1}^{n} p_i^{\sigma(y_i)} \mod q$ from which we can immediately find

$$E = XY^{-1} \mod q = \prod_{i=1}^{n} p_i^{\sigma(x_i)-\sigma(y_i)} \mod q = \prod_{i=1}^{n} p_i^{\epsilon_i} \mod q \tag{7.2}$$

where $\epsilon_i \in \{-1, 0, 1\}$. We then recover the errors $\epsilon_i$ using continued fraction decomposition and factoring. The errors $\epsilon_i$ directly correspond to the wildcard positions $W_x$.

If $\chi(y) \neq 1$ then $y_i \neq x_i$ in some non-wildcard positions, i.e. Equation (7.2) includes values $\epsilon_i \in \{-2, 2\}$ and so decoding fails with high probability. The fact that incorrect inputs give factors $p_i^{\pm 2}$ in the product (while wildcard positions introduce simply $p_i$) is a nice feature that makes our scheme more secure than the generic transformation of conjunctions to Hamming matching. It means we are not reducing conjunctions to Hamming distance, but to a weighted $\ell_1$-distance on $\mathbb{Z}$, where the non-wildcard positions are weighted double. Hence, even if an attacker guesses some wildcard positions (and so does not include the corresponding $p_i$ in their product $Y$), the value $XY^{-1} \mod q$ has $p_i^{\pm 2}$ terms for each incorrect non-wildcard position and so the attacker still needs to correctly guess the correct bits in most non-wildcard positions.

## 7.4 Obfuscator and Obfuscated Program

The conjunction obfuscator works as follows: For every conjunction $x \in \{0, 1, \star\}^n$ with $|W_x| < n/2$ there exists a polynomial time program $P : \{0,1\}^n \to \{0,1\}$ that computes whether the input vector $y \in \{0,1\}^n$ matches $x$ and evaluates to 1 in this case, otherwise to 0. Denote the family of all such programs with $\mathcal{P}$.

Again, let $\mathcal{O}_{PT}$ be a dependent auxiliary input point function obfuscator [BP12; Bit+14]. The conjunction obfuscator $\mathcal{O}_C : \mathcal{P} \to \mathcal{P}'$ takes one such program $P \in \mathcal{P}$ and uses Algorithm 7.2 to output another polynomial time program in a different family $\mathcal{P}'$. In our case this is the decoding algorithm along with the polynomial size elements $(p_i)_{i=1,\ldots,n}$, $q$ and $X \in (\mathbb{Z}/q\mathbb{Z})^*$. The obfuscator also outputs $Q = \mathcal{O}_{PT}(R_{x'})$ where $x'$ denotes the vector $x$ with the wildcards replaced with 0.

We again identify the obfuscated program with the tuple $((p_i)_{i=1,\ldots,n}, q, X, Q)$. The obfuscated program outputs 1 if Algorithm 7.3 succeeds for an input $y \in \{0,1\}^n$ and if the program $Q$, when executed on the decoded conjunction with its wildcards replaced with 0, outputs 1, else the output is 0. Formally, the obfuscated program is given in Algorithm 7.1.

**Algorithm 7.1** Obfuscated Program (with embedded data $(p_i)_{i=1,\ldots,n}, q, X, Q$)

```
procedure Execute(y ∈ {0,1}^n)
    x' = Evaluate(n, (p_i)_{i=1,...,n}, q, X, y)
    if x' = ⊥ then return 0
    return Q(x')
end procedure
```

The encoder (Algorithm 7.2) receives as an input a vector $x \in \{0, 1, \star\}^n$ corresponding to a conjunction $\chi : \{0,1\}^n \to \{0,1\}$. It then outputs the encoding represented by a triple $((p_i)_{i=1,\ldots,n}, q, X)$.

The evaluator (Algorithm 7.3) receives as an input an encoding in the form of a triple $((p_i)_{i=1,\ldots,n}, q, X)$ and a test vector. It then attempts to decode the triple and outputs the

**Algorithm 7.2** Encoding

**procedure** ENCODE($n \in \mathbb{N}; x \in \{0,1,\star\}^n$)
Sample a random sequence of distinct primes $(p_i)_{i=1,\dots,n}$ from $[(n\log(n))^2, ((n+1)\log(n+1))^2]$.
Let $W_x = \{i | x_i = \star\}$.
Sample safe prime $q$ such that $\prod_{i\in W_x} p_i < q/2 < \prod_{i\in W_x\cup\{j\}}$ for all $j \in \{1,\dots,n\} \setminus W_x$.
Compute $X = \prod_{i=1}^n p_i^{\sigma(x_i)} \mod q$.
**return** $((p_i)_{i=1,\dots,n}, q, X)$
**end procedure**

conjunction with the wildcards replaced with 0 if $\chi(y) = 1$. The evaluator further uses Algorithm 6.3 for constrained factoring.

**Algorithm 7.3** Evaluation

**procedure** EVALUATE($n, (p_i)_{i=1,\dots,n}, q \in \mathbb{N}$; $X \in (\mathbb{Z}/q\mathbb{Z})^*$; $y \in \{0,1\}^n$)
Compute $Y^{-1} = \prod_{i=1}^n p_i^{-\sigma(y_i)} \mod q$.
Compute $E = XY^{-1} \mod q$.
Determine the continued fraction representation of $E/q$, with convergents $C$.
**for all** $h/k \in C$ **do**
$F \leftarrow$ CFACTOR($n, (p_i)_{i=1,\dots,n}, k$), $F' \leftarrow$ CFACTOR($n, (p_i)_{i=1,\dots,n}, kE \mod q$)
**if** $F \neq \perp$ and $F' \neq \perp$ **then**
Let $m = (1,\dots,1) \in \{0,1\}^n$ the vector of all ones.
**for** $i = 1,\dots,n$ **do**
**if** $p_i \in F \cup F'$ **then** set $m_i = 0$
**end for**
**return** $y \wedge m$
**end if**
**end for**
**return** $\perp$
**end procedure**

### 7.4.1 Decoding

We once again argue about the correctness of the obfuscator.

**Lemma 7.2** (Correctness)**.** *Consider the algorithms* ENCODE *(Algorithm 7.2) and* EVALUATE *(Algorithm 7.3). For every $r < n/2 \in \mathbb{N}$, $x \in \{0,1,\star\}^n$ corresponding to the conjunction $\chi : \{0,1\}^n \to \{0,1\}$, for every*

$$((p_i)_{i=1,\dots,n}, q, X) \leftarrow \text{ENCODE}(n, x)$$

*and for every $y \in \{0,1\}^n$ such that $\chi(y) = 1$ it holds that*

$$\text{EVALUATE}(n, (p_i)_{i=1,\dots,n}, q, X, y) = 1.$$

*Proof.* We find the requirement $ND < q/2$ analogously to the proof of Lemma 6.4.

The $k_i$ respectively $(k_i E \mod q)$ can be factored separately using the $p_i$ to recover the $\nu_i$ and $\mu_i$ from which the wildcard positions can the immediately be recovered. This all works

assuming $\chi(y) = 1$ since then the factors of $N$ and $D$ will be unique (of multiplicity 1) and contained in the sequence $(p_i)_{i=1,\dots,n}$. If now $\chi(y) = 0$ then with high probability the factors of $N$ and $D$ will not be unique and/or not contained in $(p_i)_{i=1,\dots,n}$. □

## 7.5 Relation to Hamming Distance

Our conjunction obfuscator construction is related to our Hamming distance obfuscator (see Chapter 6) and thus exhibits several limitations.

- Firstly, the construction limits the number of wildcards such that $|W_x| < n/2$.
- Secondly, due to the construction, the problem of finding a match to $x \in \{0, 1, \star\}^n$ reduces to the problem of finding a vector $y \in \{-1, 0, 1\}^n \subset \mathbb{Z}^n$ such that $\|\sigma(x) - y\|_1 < |W_x|$. Note that we took the representatives of $\mathbb{Z}/3\mathbb{Z}$ to be $\{-1, 0, 1\}$ such that the wildcard primes never appear as factors of $X$.

  We may compute the number of possible vectors in an $\ell_1$-ball of radius $r$ ($0 \leq r \leq n(q-1)$ for $\mathbb{Z}/q\mathbb{Z}$) using

  $$|B_{1,q,r}| = \sum_{k=0}^{r} \left\langle {n \atop k} \right\rangle_q$$

  where the *q-nomial triangle* $\left\langle {n \atop k} \right\rangle_q$ is defined as

  $$\left\langle {n \atop k} \right\rangle_q = \sum_{i=0}^{n} (-1)^i \binom{n}{i} \binom{k + n - 1 - iq}{n - 1}.$$

  The upper limit of the sum may actually be taken as $\lfloor (k + n - 1)/q \rfloor$ instead of $n$. The symbol $\left\langle {n \atop k} \right\rangle_q$ counts the number of compositions of $k$ into $n$ parts $p_i$ such that $0 \leq p_i \leq q - 1$ for each $p_i$ [FA91].
- Finally, an input conjunction $x \in \{0, 1, \star\}^n$ needs to be evasive. Assuming a uniform conjunction, this will be the case if

  $$\frac{|B_{1,3,r}|}{3^n} < \frac{1}{2^\lambda}$$

  for $1/2^\lambda$ negligible.

## 7.6 Parameters

The same considerations regarding the use of a safe prime and decoding efficiency as in Section 6.4 apply here. Let us now argue that a safe prime $q$ which is bounded as in Equation (7.1) exists. We use the following heuristic: The density of Sophie Germain primes is given by $\pi_{\mathrm{SG}}(n) \sim 2Cn/\log^2(n)$ for a constant $2C \approx 1.32032$ [Sho09, Section 5.5.5]. An asymptotic inverse is given by $n \log^2(n)$ and so we can expect the $m$-th Sophie Germain prime to be of size approximately $m \log^2(m)$. Hence, assuming that the $p_i$ are sampled from $[(n \log(n))^2, ((n+1)\log(n+1))^2]$, we require that there exists an index $m \in \mathbb{N}$ such that

$$((n+1)\log(n+1))^{2r} < m \log^2(m) < (n \log(n))^{2(r+1)} \tag{7.3}$$

which, heuristically, we may convince ourselves to hold by considering the exponential nature of the bounding expressions in $r$.

As an example, consider the explicit family of parameters $n = n(\lambda) = 6\lambda$ and $r = r(\lambda) = \lambda$ that admits a security parameter of at least $\lambda$. The left plot in Figure 7.1 exhibits how we can fit $m \log^2(m)$ for some $m \in \mathbb{N}$ between the bounds for an example interval for $\lambda = 30, \ldots, 45$. Denote with $\pi_{\mathrm{SG}}[r, r+1]$ the number of Sophie Germain primes in the interval

$$[((n+1)\log(n+1))^{2r}, (n\log(n))^{2(r+1)}]$$

where the number of Sophie Germain primes in the interval $[a, b]$ is given by $\pi_{\mathrm{SG}}(b) - \pi_{\mathrm{SG}}(a)$. The right plot of Figure 7.1 shows $\pi_{\mathrm{SG}}[r, r+1]$.

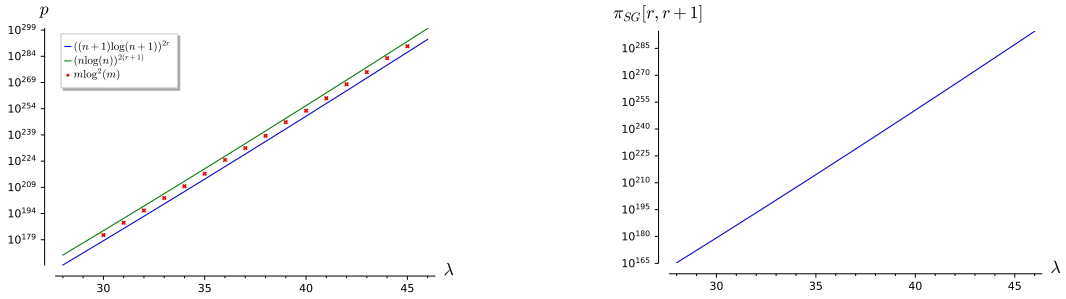



Figure 7.1: A plot of the heuristic given by Equation (7.3) for the explicit family $n = n(\lambda) = 6\lambda$ and $r = r(\lambda) = \lambda$ and the expected number of Sophie Germain primes in the interval $[((n+1)\log(n+1))^{2r}, (n\log(n))^{2(r+1)}]$.

Hence, compared to the Hamming distance obfuscator, we see that the possible parameter choices of the conjunction obfuscator are more limited, see Section 6.6. Assuming a uniform and evasive conjunction distribution, we find from Lemma 7.1 and Section 7.5 that the bit-security is given by

$$\lambda_{r,n} = \min\left\{n - r, -\log_2\left(\frac{|B_{1,3,r}|}{3^n}\right)\right\}.$$

On the other hand, the conjunction obfuscators given in [Bis+18; Bar+19] allow for a wider range of $r < n - O(\log(n))$ at the cost of assuming a generic group model, see Section 7.7.2. Brakerski et al. [Bra+16] give no estimate of encoding size or security parameters for their graded encoding scheme based obfuscator, see Section 7.7.1. Chen et al. [Che+18] found experimentally that "The TBO of 32-bit conjunctions is close to being practical, with a total evaluation runtime of 11.6 milliseconds, obfuscation runtime of 5.1 minutes, and program size of 11.6 GB for a setting with more than 80 bits of security." This program size and obfuscation time is orders of magnitude worse than the encoding size of our scheme or the schemes of [Bis+18; Bar+19].

## 7.7 Other Conjunction Obfuscators

Conjunction obfuscators have been considered before [Bra+16; BR17; Bis+18; Bar+19]. Let us quickly remind the reader of these schemes. We are interested in their parameter areas and their security properties.

### 7.7.1 LWE-based

One way to tackle the problem of constructing a conjunction obfuscator is to use a *graded encoding scheme* [BR17]. Brakerski et al. [Bra+16] gave an explicit construction of such a conjunction obfuscator along with a security reduction to *entropic ring LWE*. See Section 8.3 for an introduction to graded encoding schemes.

Before we summarise this obfuscation construction, we need to briefly introduce a derived definition. A *directed encoding scheme* is a weaker variant of a multilinear map. Let $\mathcal{R}$ be a ring such that $\mathcal{R}$ has a notion of *small* elements. Let $m \in \mathbb{N}$. For public keys $a_1, a_2 \in \mathcal{R}^m$, define the encoding of a small secret $s \in \mathcal{R}$ along the path $(1 \to 2)$ as a matrix $C \in \mathcal{R}^{m \times m}$ whose entries are small elements, such that

$$a_1 C = s a_2 + e$$

where $e \in \mathcal{R}^m$ is a small Gaussian error vector. Two such encodings $C_1$ and $C_2$ can be multiplied if the end point of $C_1$ coincides with the start point of $C_2$. Encodings along the same path may furthermore be added and subtracted, and we are also able to test whether an encoding is the encoding of $0 \in \mathcal{R}$. This also allows us to test whether two encodings with the same start- and endpoints are equal.

If we now wish to obfuscate a conjunction $\chi : \{0,1\}^n \to \{0,1\}$ represented by a vector $x \in \{0,1,\star\}^n$ for $n \in \mathbb{N}$, we first sample a collection of random vectors $a_i$ for all $i = 0, 1, \ldots, n, n+1$ representing the paths $(0 \to 1), \ldots, (n \to n+1)$. Next, we sample random small elements $s_{i,b}, r_{i,b} \in \mathcal{R}$ for all $i = 1, \ldots, n$ and all $b \in \{0,1\}$ such that $s_{i,0} = s_{i,1}$ if $x_i = \star$. We then sample encodings $R_{i,b}$ of $r_{i,b}$ and $S_{i,b}$ of $s_{i,b} r_{i,b}$ using $a_{i-1}$ and $a_i$, respectively. Finally, sample a random small element $r_{n+1} \in \mathcal{R}$ and sample the encodings $R_{n+1}$ of $r_{n+1}$ and $S_{n+1}$ of

$$r_{n+1} \prod_{i=1}^{n} s_{i,x_i}$$

where we set $s_{i,\star} = s_{i,0} = s_{i,1}$ when $x_i = \star$. These two encodings are sampled along the path $(n \to n+1)$ using $a_n$ and $a_{n+1}$.

The obfuscated program consists of the tuple

$$((a_i)_{i=0,\ldots,n+1}, (S_{i,b})_{i=1,\ldots,n;b\in\{0,1\}}, (R_{i,b})_{i=1,\ldots,n;b\in\{0,1\}}, S_{n+1}, R_{n+1}).$$

To evaluate it on an input $y \in \{0,1\}^n$, compute

$$S = \left( \prod_{i=1}^{n} S_{i,y_i} \right) R_{n+1}, \quad R = \left( \prod_{i=1}^{n} R_{i,y_i} \right) S_{n+1}.$$

If $\chi(y) = 1$, then $S$ and $R$ are encodings of the same value along the path $(0 \to n+1)$. Otherwise, if $\chi(y) = 0$, they do not encode the same value with overwhelming probability.

The obfuscated program consists of $2(2n+1)m^2 + (n+2)m = O(nm^2)$ elements of $\mathcal{R}$. Brakerski et al. [Bra+16] did not give a proper analysis of the security parameter in terms of $\mathcal{R}$, though it is apparent that the elements of the base ring and the dimension $m$ need to be quite large. We need a large dimension to guarantee a suitable security parameter and to make sure that the accumulated error with respect to the directed encoding scheme is appropriately bounded.

**Security.** The security of this scheme is based on the security assumptions about the underlying directed encoding scheme. In this case, the entropic ring LWE assumption says that for random short ring elements $s_1, \ldots, s_n \in R$ and a secret vector $x \in \{0,1\}^n$, the ring LWE assumption holds for the secret $\prod_{i=1}^{n} s_i^{x_i}$ given that $x$ is sampled from a high entropy distribution, cf. [Bra+16, Section 3.1].

### 7.7.2 Generic Group/DLP-based

Here we summarise the scheme presented by Bishop et al. [Bis+18]. We note that Bartusek et al. [Bar+19] gave a generalisation and present a *dual* scheme.

Suppose we are given a conjunction $\chi : \{0,1\}^n \to \{0,1\}$ represented by a vector $x \in \{0,1,\star\}^n$ for $n \in \mathbb{N}$. To obfuscate this conjunction, select a random prime $p$ and sample $a_1, \ldots, a_{n-1}$ from the uniform distribution on $R = \mathbb{Z}/p\mathbb{Z}$. These coefficients determine the polynomial

$$f(X) = \sum_{i=1}^{n-1} a_i X^i \in R[X].$$

Let now $G$ be a group, which will be modelled as a *generic group* in the security analysis. Let $g$ be a generator of prime order $p > 2^n$. The obfuscated program consists of the elements $h_{i,j}$ (for all $i = 1, \ldots, n$ and all $j \in \{0,1\}$) where $h_{i,j} = g^{f(2i+j)}$ if $x_i = j$ or $x_j = \star$. Otherwise $h_{i,j}$ is a randomly sampled element of $G$. Thus, the obfuscated program is given by $2n$ group elements.

To evaluate the obfuscated program on an input $y \in \{0,1\}^n$, compute the *Lagrange interpolation* coefficients

$$C_i = \prod_{j \neq i} \frac{-2j - y_j}{2i + y_i - 2j - y_j}$$

and then evaluate the interpolation polynomial using

$$T = \prod_{i=1}^{n} (h_{i,y_i})^{C_i}.$$

If $\chi(y) = 1$, then we have $T = g^0$ and otherwise if $\chi(y) = 0$ then $T$ will be a random element of $G$ with overwhelming probability.

Assuming the size of one group element is $\log_2(q)$ ($q$ the order of G), then the size of the obfuscated program is $2n \log_2(q)$ in the case of Bishop et al. [Bis+18] and $(n+1)\log_2(q)$ for the dual scheme of Bartusek et al. [Bar+19].

**Security.** The security proof of this scheme works in the generic group model to show that the pattern matching with wildcards obfuscator is a distributional VBB obfuscator for $x \in \{0,1,\star\}^n$ if the number of wildcards satisfies $|W_x| < 0.774n$ [Bis+18]. The dual scheme attains distributional VBB security in the generic group model and replaces the limitation on the number of wildcards with the natural bound $|W_x| < n - \omega(\log(n))$ [Bar+19].

## 7.8 Security

Showing security of the conjunction obfuscator works in essentially the same way as for the Hamming distance obfuscator in Section 6.10. Note that Problem 5.1 respectively Problem 5.2 also makes sense when the distribution $D$ is considered to be over $\{-1,0,1\}^n$ instead of $\{0,1\}^n$. Note that Remark 4 applies to Theorem 7.1 as well.

As in Section 6.10, we construct the dependent auxiliary input distributional VBB point function obfuscator $O_{PT}$ from a *preimage resistant hash function* $H : \{0,1\}^n \to \{0,1\}^n$, which we model as a *random oracle*. The obfuscated program $Q = O_{PT}(R_x)$ for a point $x \in \{0,1\}^n$ is given by the hash value $h = H(x)$. As before, to evaluate $Q$ on another point $y \in \{0,1\}$ we test whether $H(y) = h$ and output 1 in this case, otherwise we output 0.

Again, we remind the reader that the prime distribution $\rho$ in $\mathrm{MSP}_{r,n,D,\rho}$ and $\text{D-MSP}_{r,n,D,\rho}$ is assumed to be the minimal prime distribution $\rho = \rho_{r,n}$ given by Definition 5.2.

**Theorem 7.1.** *Let $\mathcal{D} = \{D_\lambda\}_{\lambda\in\mathbb{N}}$ be an ensemble of conjunction evasive distributions with auxiliary information (as in Definition 7.3). Suppose that entropic $\text{D-MSP}_{r,n,D,\rho}$ (Problem 6.1) with the distribution $D$ over $\{-1,0,1\}^n$ is hard. Then the conjunction obfuscator $\mathcal{O}_C$ is a distributional VBB obfuscator for $\mathcal{D}$ in the random oracle model.*

*Proof.* Same as the proof of Theorem 6.1. □

**Theorem 7.2.** *Let $\mathcal{D} = \{D_\lambda\}_{\lambda\in\mathbb{N}}$ be an ensemble of conjunction evasive distributions (as in Definition 7.2). Suppose that $MSP_{r,n,D,\rho}$ (Problem 5.1) with the distribution $D$ over $\{-1,0,1\}^n$ is hard for $r > r_f(n)$ (recall Equation (6.3)).Then the conjunction obfuscator $\mathcal{O}_C$ is input-hiding.*

*Proof.* Same as the proof of Theorem 6.2. □

# 8 Branching Programs

In this chapter we want to study ways of approaching general purpose obfuscation. Instead of focusing on specific, well-defined problems, we want to consider arbitrary functions, circuits, and programs. Recall Section 2.3 in which we described the impossibility result by Barak et al. [Bar+01] for general purpose VBB obfuscation. We also saw that since VBB is too strong for certain general programs, they later suggested a weaker obfuscation notion — indistinguishability obfuscation [Bar+14a].

The most common generic approach is now to reinterpret a Boolean circuit as a so-called *branching program*. We will see that we should consider certain classes of circuits that have limited length. Given a branching program, it is then (hopefully) securely encoded using a *multilinear map*. An example of a bilinear map is a cryptographic pairing and a multilinear map is a generalisation from two to multiple inputs. We will additionally need a primitive to determine how to translate the result of encoded branching program evaluation into a plaintext answer.

Unfortunately, secure algebraic constructions of multilinear maps still evade us [BS03] and for now the best that we can hope for is a weaker construction called *graded encoding scheme*. There exist different candidates for graded encoding schemes from lattices and their security depends on certain lattice problems and assumptions. Graded encoding schemes have fully homomorphic properties and usually come with a so-called *zero-testing* primitive. We will see that zero-testing is well suited to check the result of encoded branching program evaluation.

## 8.1 Branching Programs

In this section we want to explain ideas on obfuscating general programs. We will see that it is useful to represent programs as finite directed acyclic graphs; we call an element in this representation a *branching program*.

**Definition 8.1** (Branching Program)**.** *A* branching program $B$ *is a finite directed acyclic graph* $B = G(V, E)$ *such that* $G$ *has exactly one source vertex* $v^{\text{in}}$ *and multiple sink vertices* $V^{\text{out}} = \{v_1^{\text{out}}, \ldots, v_m^{\text{out}}\}$ *for some* $m \in \mathbb{N}$*. The set* $V^{\text{out}}$ *is the disjoint union of into two sets* $V^{\text{out}} = V^{\text{accept}} \cup V^{\text{reject}}$*. Each vertex in* $V \setminus V^{\text{out}}$ *has exactly two outgoing edges labelled 0 or 1.*

The *length* $\ell$ of a branching program $B$ is the longest path in $G$. If additionally $G$ is layered, we call $B$ a *layered branching program* and define its *width* as the maximal layer width. Branching program evaluation works as follows. A concrete input $x \in \{0,1\}^*$ induces a path from the source vertex to one of the sink vertices by following the edge labels according to the entries of $x$. The input is accepted if the path ends at any node contained in $V^{\text{accept}}$ otherwise it is rejected by the branching program.

Recall Barrington's theorem (Theorem 1.1) [Bar89]. It states that any circuit of depth $d$ is computable by a branching program of width 5 and length at most $4^d$. In particular, if $d = O(\log n)$ then $\ell = \text{poly}(n)$. Usually, we denote the class of all circuits with $d = O(\log n)$ by

$\mathbf{NC}^1$. This is important since we are only able to efficiently work with polynomial size branching programs and thus need to consider circuits in $\mathbf{NC}^1$ when talking about obfuscating general programs.

It is instructive to study the proof of Theorem 1.1 since it is easy to understand and involves only a few properties of finite groups. The key step involves examining the conjugacy classes of a certain permutation group. Recall that the commutator of two group elements $g, h \in G$ of a group $G$ is given by $[g, h] = ghg^{-1}h^{-1}$.

A priori, the constants in Barrington's theorem appear to be random. We will see that they stem from the curious fact that the symmetric group $S_5$ is non-solvable. We can find two elements $a, b \in S_5$ that are conjugate to each other and to their commutator $[a, b]$. Barrington's construction uses these elements to rewrite an input circuit consisting solely of AND and NOT gates as a branching program.

### 8.1.1 Computing Circuits

Let us now follow Barrington's original construction to see how a circuit consisting of AND and NOT gates can be computed using a branching program. To start we recall the definition of a symmetric group.

**Definition 8.2** (Symmetric Group)**.** *Let $X$ be a set. The* symmetric group $S_X$ *is the group formed by all permutations acting on $X$. The group operation is function composition. For $X = \{1, \ldots, n\}$ we denote the symmetric group of $n$ elements by $S_n$.*

Denote by $\mathbb{1} \in S_n$ the identity permutation. Elements in $S_n$ can be decomposed into transpositions $\tau_{i,j}$ that swap symbols $i$ and $j$. Alternatively, they can be decomposed into cycles, written $(i_1 \ldots i_k)$. A cycle maps the symbols $i_j$ to $i_{j+1}$ for $j \in \{1, k-1\}$ and the symbol $i_k$ to $i_1$. Note that the decomposition of an element in $S_n$ into cycles is independent of the order of cycles, as they act on disjoint subsets of symbols. It can be shown that the conjugacy classes of $S_n$ are exactly the sets of elements with the same cycle structure.

In the next steps we focus on the symmetric group $S_5$. Let $C_5 \subset S_5$ be the conjugacy class of 5-cycles. It it possible to find two elements $g, h \in C_5$ such that $[g, h] \in C_5$, for example through exhaustive search.

We make the following definitions of *group programs* and how to evaluate them on an input vector $x \in \{0,1\}^n$. We can think of a group program as a sequence of 2-tuples of group elements. The individual 2-tuples are indexed by the entries of the input vector.

**Definition 8.3.** *(Group Program) A* group program $P = (\varphi_i)_{i \in I}$ *over a group $G$ is a sequence of maps $\varphi_i(x) : \{0,1\}^n \to G$.*

**Definition 8.4.** *(Group Program Evaluation) Let $P$ a group program and $x \in \{0,1\}^n$ an input vector. Evaluating the program on the input is then a matter of computing*

$$y = \prod_{i \in I} \varphi_i(x).$$

Suppose that we are now given an arbitrary circuit $C : \{0,1\}^n \to \{0,1\}$. Recall that $C$ is a directed acyclic graph whose edges are labelled by $\{0,1\}$ and whose vertices are Boolean functions, for example AND, OR, NOT, NAND, XOR, etc.

**Definition 8.5** (Circuit Computability)**.** *We say that a group program $P$* $g$-computes *a circuit $C$ for $g \in C_5$ if*

$$\forall x \in \{0,1\}^n : P(x) = g^{C(x)},$$

*i.e.* $P(x) = \mathbb{1}$ *if* $C(x) = 0$ *and* $P(x) = g$ *if* $C(x) = 1$.

If a group program $g$-computes a circuit $C$, then there exists a group program $P'$ of the same length that $h$-computes the same circuit. To see this, conjugate the evaluation equation by an appropriate element $k$ such that $h = kgk^{-1}$. This element $k$ exists since $g$ and $h$ are in the same conjugacy class $C_5$.

Denote now by $\varphi_{i,g,h}(x) = g(1 - x_i) + hx_i$ the map that evaluates to $g$ if $x_i = 0$ and $h$ if $x_i = 1$. Similarly, consider the map $\varphi_{c,g}(x) = g^c$ for a constant $c \in \{0, 1\}$, which is used to lift $\{0, 1\}$ to group programs. These maps will be the elementary building blocks in the following construction.

**Lemma 8.1** (Identity and Constant Group Programs)**.** *Suppose $C$ is the constant circuit outputting a variable $x_i$ or a constant $c \in \{0, 1\}$. Then $C$ is $g$-computable by $P = (\varphi_{i,\mathbb{1},g})$ or $P = (\varphi_{c,g})$, respectively.*

*Proof.* This can be seen immediately from Definition 8.3. □

**Lemma 8.2** (Group Program Negation)**.** *Suppose $C$ is $g$-computable, then the circuit $C' = \neg C$ is also $g$-computable.*

*Proof.* The associated group program that $g$-computes the circuit $C$ can be turned into a group program $P'$ that $g^{-1}$-computes $C$ by conjugation with appropriate elements. Finally, redefine the last map in the program's sequence as $\varphi'_I(x) = \varphi_I(x)g$. □

**Lemma 8.3** (Group Program Conjunction)**.** *Suppose $C_1$ is $g$-computable and $C_2$ is $h$-computable, then the circuit $C' = C_1 \wedge C_2$ is $[g, h]$-computable.*

*Proof.* Assume $P_1$ and $P_2$ are the associated group programs. From these we can generate group programs $P'_1$ and $P'_2$ that $g^{-1}$-compute $C_1$ respectively $h^{-1}$-compute $C_2$. Concatenating $P_1P_2P'_1P'_2$ then yields the group program that $[g, h]$-computes $C'$. □

*Proof of Theorem 1.1.* By using induction in combination with Lemma 8.1, Lemma 8.2 and Lemma 8.3 it follows that a circuit $C$ consisting only of AND and NOT gates is $g$-computable over the group $S_5$. This is possible because $S_5$ contains two elements $g \in C_5$ and $h \in C_5$ such that $[g, h] \in C_5$. □

### 8.1.2 The Symmetric Group $S_6$

One downside we encounter by following Barrington's original construction is that the input circuit must solely consist of AND and NOT gates. This requirement increases the corresponding branching program's length. Unfortunately, using $S_5$ does not allow for a more interesting structure. One way to overcome this hurdle, is to study different groups.

Let us now have a closer look at the symmetric group $S_6$. Note that it contains two elements $g$ and $h$ that are conjugate to each other and their commutator. This fact again allows us to use this group in combination with Barrington's theorem. Additionally, the two elements can be chosen to be involutions, i.e. $g^2 = h^2 = \mathbb{1}$. We will see how that lets us evaluate more complex circuits with fewer group operations. Denote by $C_1 \oplus C_2$ the *exclusive or* (XOR) of two circuits $C_1$ and $C_2$.

**Lemma 8.4** (Group Program XOR)**.** *Suppose $C_1$ and $C_2$ are $g$-computable over $S_6$, then the circuit $C' = C_1 \oplus C_2$ is $g$-computable.*

*Proof.* Assume $P_1$ and $P_2$ are the associated group programs. Concatenating $P_1P_2$ then yields the group program that $g$-computes $C'$. To see this, suppose $P_1$ evaluates to $g$ and $P_2$ evaluates to $\mathbb{1}$ or vice versa, then the concatenation evaluates to $g$. On the other hand, if both $P_1$ and $P_2$ evaluate to $\mathbb{1}$ or $g$, then the concatenation evaluates to $\mathbb{1}$. Thus, the resulting program computes the XOR of both circuits. □

From Lemma 8.4 we can see that choosing different underlying groups with a richer structure allows us to generalise the types of circuits we are able to compute.

**Example.** Let $C$ be the circuit that computes $(x_1 \oplus x_2) \wedge (x_3 \oplus x_4)$ for an input $x = (x_1, x_2, x_3, x_4) \in \{0,1\}^4$. Let $g, h \in S_6$ such that they and $[g,h]$ are conjugate to each other and such that $g^2 = h^2 = \mathbb{1}$. Then the program given by

$$P = (\varphi_{1,\mathbb{1},g}, \varphi_{2,\mathbb{1},g}, \varphi_{3,\mathbb{1},h}, \varphi_{4,\mathbb{1},h}, \varphi_{1,\mathbb{1},g^{-1}}, \varphi_{2,\mathbb{1},g^{-1}}, \varphi_{3,\mathbb{1},h^{-1}}, \varphi_{4,\mathbb{1},h^{-1}})$$

$[g,h]$-computes $C$.

## 8.2 Matrices and Programs

In Section 8.1.1 we have seen how the proof of Barrington's theorem makes use of the symmetric group $S_5$. This group is known to admit a faithful irreducible representation called the natural permutation representation over the matrix space $k^{5\times5}$ where $k$ is a large enough field. It allows us to translate Barrington's original construction to the space of 5 by 5 matrices in a natural way.

Hence, instead of talking about group programs whose objects consist of abstract group elements, we can specify to a matrix representation. Then a branching program can be described using matrices and we shall call such a representation a *matrix branching program*.

In Section 8.1.2 we studied the properties of the symmetric group $S_6$ and saw that it allows us to represent a more general circuit as a group program. We can again specify to a matrix representation and obtain a matrix branching program from a circuit whose vertices can be AND, NOT, and XOR.

Continuing this we can consider arbitrary matrix branching programs whose matrices need not represent elements of $S_5$ or $S_6$, respectively.

**Definition 8.6** (Matrix Branching Program)**.** *A* matrix branching program $P = (M_i)_{i=1,\dots,n}$ *over a matrix ring $R$ is a sequence of tuples $M_i = (M_{i,0}, M_{i,1})$.*

**Definition 8.7** (Matrix Branching Program Evaluation)**.** *Let $P$ a matrix branching program and $x \in \{0,1\}^n$ an input vector. Evaluating the program on the input is then a matter of computing*

$$M = \prod_{i=1}^{n} M_{i,x_i}.$$

*The result of $P$ evaluated on an input $x \in \{0,1\}^n$ is 0 if $M = \mathrm{id}$ and 1 otherwise.*

Definition 8.7 is for a branching program which reads each input bit exactly once. We can also consider more general *multi-input* branching programs for which a single input bit can be used multiple times.

To hide the matrices from an adversary and thus the encoded program, we need a multilinear map or graded encoding scheme. It is also important to think about which obfuscation definition

we want to use. We argued in Section 2.3 that VBB obfuscation is impossible for generic programs. Hence, we can either try to achieve VBB for a specific class of branching programs or focus on a different obfuscation notion. Indistinguishability obfuscation is commonly considered for generic branching programs.

Different schemes further modify the branching program matrices; one widely used technique is *Kilian randomisation* [Kil88]. The idea is to consider a sequence of randomisation matrices $(R_i)_{i=1,\dots,n+1}$. Set $R_1 = R_{n+1} = \mathrm{id}$, sample random invertible matrices $R_2, \dots, R_n$, and modify the matrix branching program as follows: Replace the matrices $M_{i,b}$ with $R_i M_{i,b} R_{i+1}^{-1}$ for all $b \in \{0,1\}$ and all $i = 1, \dots, n$. If the matrices are multiplied in the correct order, the random matrices cancel out and we effectively compute the same product as in Definition 8.7.

Finally, to obtain the program output, we need to be able to compare the encoded matrix with an encoding of the identity matrix. If the graded encoding scheme used to encode the matrices admits a zero-test, this is a readily solved problem. We can homomorphically compute the difference between the encoded result and an encoding of the identity matrix and check whether the difference is zero using the zero-test.

## 8.3 Multilinear Maps and Graded Encoding Schemes

In cryptography the following definition of a bilinear pairing is usually used.

**Definition 8.8** (Bilinear Pairing)**.** *Let $G_1$, $G_2$ and $G_T$ be cyclic groups. A map $f : G_1 \times G_2 \to G_T$ is called a* bilinear pairing *if the following requirements hold for any $g_1$ of $G_1$ and $g_2$ of $G_2$:*

- $\forall a, b \in \mathbb{Z} : f(g_1^a, g_2^b) = f(g_1, g_2)^{ab}$ *and*
- *$f$ is not degenerate in the sense that $f(g_1, g_2)$ generates $G_T$ if $g_1$ and $g_2$ generate their respective groups.*

Note that a bilinear pairing takes two inputs. We generalise this to a multilinear map that takes an arbitrary number of inputs.

**Definition 8.9** (Multilinear Map [BS03; Rot13])**.** *Let $I = \{1, \dots, n\}$. Let $G_1, \dots, G_n$ and $G_T$ be cyclic groups. A map $f : G_1 \times \cdots \times G_n \to G_T$ is called an $n$-*multilinear map *if the following requirements hold for any sequence $(g_i \in G_i)_{i \in I}$:*

- $\forall a \in \mathbb{Z}^n : f(g_1^{a_1}, \dots, g_n^{a_n}) = f(g_1, \dots, g_n)^{a_1 \cdots a_n}$ *and*
- *$f$ is not degenerate in the sense that $f(g_1, \dots, g_n)$ generates $G_T$ if the $g_i$ generate their respective groups.*

If we want to use multilinear maps in cryptographic applications, we need a security notion. The most natural approach is to generalise the *decision Diffie-Hellman* (DDH) problem to the multilinear map setting.

**Problem 8.1** (Multilinear Decision Diffie-Hellman (MDDH))**.** *For a symmetric multilinear map $f$ with generator $g_1, \dots, g_n$ the* multilinear decision Diffie-Hellman *problem is to distinguish between the following two distributions:*

- $\mathcal{D}_0 = \left((g_i)_{i=1,\dots,n}, g_1^{a_1}, \dots, g_n^{a_n}, f(g_1, \dots, g_n)^{a_1 \cdots a_n}\right)$*, and*
- $\mathcal{D}_1 = \left((g_i)_{i=1,\dots,n}, g_1^{a_1}, \dots, g_n^{a_n}, f(g_1, \dots, g_n)^{r}\right)$ *where $r$ is a uniformly random exponent.*

A notion that is very similar to but slightly weaker than a multilinear map is that of a graded encoding scheme. A graded encoding scheme exhibits a certain *maximal grading* and fully homomorphic properties up to that maximal grading. The reason for such a definition is that known constructions are based on lattices and that for such lattice-based schemes the homomorphisms increase the noise level on every application.

**Definition 8.10** (Graded Encoding System)**.** *A* $\kappa$-Graded Encoding System *consists of a ring* $R$ *and a system of sets*

$$\mathcal{S} = \left\{ S_i^{(\alpha)} \subset \{0,1\}^* \middle| \alpha \in R, i \in [0,\kappa] \right\},$$

*with the following properties:*

1. *For every fixed index* $i$*, the sets* $\left\{ S_i^{(\alpha)} \middle| \alpha \in R \right\}$ *are disjoint.*
2. *There is an associative binary operation '*$+$*' and a self-inverse unary operation '*$-$*' such that for every* $\alpha_1, \alpha_2 \in R$*, every index* $i \leq \kappa$*, and every* $u_1 \in S_i^{(\alpha_1)}$ *and* $u_2 \in S_i^{(\alpha_2)}$*, it holds that*

$$u_1 + u_2 \in S_i^{(\alpha_1+\alpha_2)}$$

*and*

$$-u_1 \in S_i^{(-\alpha_1)}.$$

3. *There is an associative binary operation '*$\times$*' such that for every* $\alpha_1, \alpha_2 \in R$*, every* $i_1, i_2$ *with* $i_1 + i_2 \leq \kappa$*, and every* $u_1 \in S_{i_1}^{(\alpha_1)}$ *and* $u_2 \in S_{i_2}^{(\alpha_2)}$*, it holds that*

$$u_1 \times u_2 \in S_{i_1+i_2}^{(\alpha_1\alpha_2)}.$$

## 8.4 Graded Encoding Schemes from Lattices

It is possible to explicitly construct graded encoding schemes using lattices. The first constructions were simply parametrised by the maximal grading $\kappa$ and encodings of different grading can be multiplied freely as long as the sum of the two gradings does not exceed $\kappa$. The later graph induced schemes achieve a finer grained control of which encodings can be multiplied. The non-graph-based encoding schemes by Garg et al. [GGH13], Coron et al. [CLT13] and Ma and Zhandry [MZ18] encode scalar elements. We will see that the graph-based scheme by Gentry et al. [GGH15] naturally deals with matrices. On the one hand this comes at the cost of a non-commutative plaintext space. On the other hand, matrices are better suited for obfuscation. For an excellent summary on the use of graded encoding schemes for indistinguishability obfuscation, see [Pel19, Chapter 6].

### 8.4.1 GGH13

Here we follow the construction of the graded encoding scheme by Garg et al. [GGH13]. As in Section 4.3, let $q$ be a modulus and fix a ring $R$, for example $R = \mathbb{Z}[X]/(X^m + 1)$ for some $m \in \mathbb{N}$. The idea is to encode elements of a quotient ring $R/I$ where $I$ is the principal ideal generated by a *short* element $g \in R$, i.e. $I = \langle g \rangle$. The encoded elements are cosets $e + I$ for some plaintext element $e$. In the terms of Definition 8.10, the level-zero encodings of a coset $e + I$ are given by

$$S_0^{(e+I)} = \left\{ c \in R_q \middle| c \in e + I, \|c\| < q^{1/8} \right\}.$$

Generally, to move to higher-level encodings we need an additional parameter $z \in R_q$ which is chosen at random. A level-$i$ encoding of a coset $e + I$ is then given by

$$S_i^{(e+I)} = \left\{ \frac{c}{z^i} \in R_q \middle| c \in e + I, \|c\| < q^{1/8} \right\}.$$

In this scheme the elements $g$ and $z$ are kept secret.

To encode cosets at higher level, we need to generate and publish a level-one encoding of $1 + I$, call it $y = [a/z]$. Then given a level-zero encoding $d$ we can use $y$ to get $[yd] = [ad/z]$ which is a level-one encoding of $d$. To get a level-$i$ encoding we raise $y$ to the $i$-th power and find $[y^i d] = [a^i d/z^i]$.

Suppose we are given two encodings, namely $u_1 = [v_1/z]$ and $u_2 = [v_2/z]$. Then $u_1 + u_2 = [(v_1 + v_2)/z]$ is an encoding of the sum of the two. Similarly, we can multiply encodings and get $u_1 u_2 = [v_1 v_2/z^2]$.

This scheme is additively homomorphic and so we can test equality between two elements by subtracting them and comparing the result to zero. For this however, we need to provide a special zero-testing element. It is given by

$$p_{\text{zt}} = \left[ \frac{hz^n}{g} \right]$$

where $h \in R_q$ is small enough. Suppose we are now given a level-$n$ encoding $u = [v/z^n]$ and we wish to know whether it is an encoding of zero. For this we simply compute $[p_{\text{zt}} u]$ and check whether this result is short, else $u$ was not an encoding of zero.

The MDDH hardness was broken in [HJ16]. Generic attacks in certain settings are given in [ABD16; CJL16].

### 8.4.2 CLT13

This scheme by Coron et al. [CLT13] requires $n$ secret primes $p_i$ that form the public modulus $x_0 = p_1 \cdots p_n$. We furthermore generate $n$ small secret primes $g_i$ and a random secret integer $z$ mod $x_0$. A message is given by an element in $R = (\mathbb{Z}/p_1\mathbb{Z}) \times \cdots \times (\mathbb{Z}/p_n\mathbb{Z})$. The level-$k$ encoding $c$ of a message $(m_i) \in R$ is an integer such that for all $i \in \{1, \ldots, n\}$ we have that

$$c \equiv \frac{r_i g_i + m_i}{z^k} \mod p_i$$

for small random integers $r_i$. The encoding $c$ modulo $x_0$ can be calculated using CRT. Two encodings $c_1$ and $c_2$ can be added and multiplied. Suppose $c_1$ is a level-$i$ encoding and $c_2$ is a level-$j$ encoding, multiplication of them yields a level-$(i + j)$ encoding.

The MDDH hardness was broken in [Che+15]. The follow-up construction [CLT15] was broken in [Che+16].

### 8.4.3 MZ18

This construction by Ma and Zhandry [MZ18] improves upon CLT13. To this day, there are no efficient attacks that break the MDDH hardness of their graded encoding scheme.

### 8.4.4 GGH15

Fix a lattice dimension $m \in \mathbb{N}$, let $q$ be a modulus and consider the ring $R = \mathbb{Z}/q\mathbb{Z}$. Gentry et al. [GSW13] describe a homomorphic cryptosystem which encrypts a message $\mu \in R$ as a matrix $C \in R^{m \times m}$ with small entries such that

$$Ca = \mu a + e$$

where $a \in R^m$ is the secret key and $e \in R^m$ is a small error vector.

A modification of this scheme would be to choose a secret matrix $A \in R^{n \times m}$ for some $n \in \mathbb{N}$ instead of a secret vector $a \in R^m$. A secret matrix $S \in R^{n \times n}$ is then encrypted as a matrix $C \in R^{m \times m}$ with small entries such that

$$AC = SA + E \tag{8.1}$$

for some small error matrix $E \in R^{n \times m}$.

This construction is additively and multiplicatively homomorphic. Take two encryptions such that $AC_1 = S_1A + E_1$ and $AC_2 = S_2A + E_2$, then we have

$$A(C_1 + C_2) = (S_1 + S_2)A + E' \tag{8.2}$$

for some small $E'$. Obviously, we can only add a finite number of such ciphertexts before the error grows too big and decryption eventually fails. Similarly, for the multiplication of two ciphertexts we have

$$AC_1C_2 = (S_1A + E_1)C_2 = S_1(S_2A + E_2) + E_1C_2 = S_1S_2A + E' \tag{8.3}$$

for some small $E'$.

We are also able to compare two ciphertexts by using the additive homomorphism, that is if $S_1 = S_2$ then

$$A(C_1 - C_2) = E' \tag{8.4}$$

which is small.

Assume we are now given a graph $G = (V, \mathcal{E})$. To turn this into a graded encoding scheme, the idea of Gentry et al. [GGH15] is to modify the encoding Equation (8.1) to

$$A_uC = SA_v + E,$$

where this equation represents the encoding of $S$ along the edge $(u, v) \in \mathcal{E}$ between two vertices $u, v \in V$. Here each vertex $v \in V$ has a random matrix $A_v \in R^{n \times m}$ associated to it. Only ciphertexts along the same edge can now be added, i.e. given two encryptions such that $A_uC_1 = S_1A_v + E_1$ and $A_uC_2 = S_2A_v + E_2$ then we have

$$A_u(C_1 + C_2) = (S_1 + S_2)A_v + E'.$$

Multiplication is restricted too, we can only multiply such ciphertexts, whose associated edges form a possible path in the graph. That is, given two encryptions such that $A_uC_1 = S_1A_v + E_1$ and $A_vC_2 = S_2A_w + E_2$ we have

$$A_uC_1C_2 = (S_1A_v + E_1)C_2 = S_1(S_2A_w + E_2) + E_1C_2 = S_1S_2A_w + E',$$

an encoding of $S_1S_2$ along the path from $u$ to $w$ (through $v$).

Equality testing can be performed for encodings that end at the same sink vertex. The underlying primitive is the zero-test Equation (8.4). Given $C_1$ along $(u, w)$ and $C_2$ along $(v, w)$ we can compute

$$A_u C_1 - A_v C_2 = (S_1 - S_2) A_w + E'.$$

If $S_1 = S_2$, then $A_u C_1 - A_v C_2 = E'$ and we can check whether this result is smaller than some predetermined threshold.

The MDDH hardness was broken in [Cor+16].

## 8.5 Graph Induced Multilinear Maps

In Section 8.4.4 we saw how to construct a (*graph induced*) graded encoding scheme from a more fundamental encoding scheme. Note, however, that an actual construction still requires a *lattice trapdoor* (see [Ajt99]) in order to generate the encoding $C$ in the defining equation

$$A_u C = S A_v + E \tag{8.5}$$

for an edge $(u, v)$ in the associated graph. The trapdoor construction should be *efficient* in the sense that it should be fast to realise on actual computing hardware. As an example, the branching program obfuscation schemes described by Wichs and Zirdelis [WZ17a] and Halevi et al. [Hal+17] require such efficient trapdoors.

Micciancio and Peikert [MP12] introduced one such possible trapdoor construction. Halevi et al. [Hal+17] generalised the underlying *gadget matrix* from a binary representation to a faster mixed-radix representation. We will briefly state these ideas here since we will require them in Chapter 9. Let $q \in \mathbb{N}$ again be a modulus and consider the ring $R = \mathbb{Z}/q\mathbb{Z}$.

**Definition 8.11** (Gadget Matrix)**.** *Let $k, n \in \mathbb{N}$. For the vector $g \in R^k$ given by*

$$g = \left(2^0, 2^1, \cdots, 2^{k-1}\right),$$

*the* gadget matrix *$G \in R^{n \times nk}$ is defined as*

$$G = g \otimes \mathrm{id}_n .$$

Let us look at the lattice induced by this special matrix $G$. Denote with $S_k \in \mathbb{Z}^{k \times k}$ the following basis of the lattice $\Lambda^\perp(g)$.

**Definition 8.12** (Gadget Lattice, Power-of-Two Modulus)**.** *Let $q = 2^k$ be a power of 2 modulus. A basis $S_k \in \mathbb{Z}^{k \times k}$ for the lattice $\Lambda^\perp(g)$ for $g$ as in Definition 8.11 is given by*

$$S_k = \begin{pmatrix} 2 & & & & \\ -1 & 2 & & & \\ & -1 & \ddots & & \\ & & \ddots & 2 & \\ & & & -1 & 2 \end{pmatrix}.$$

The matrix $S_k$ in Definition 8.12 satisfies $\det S_k = q$ and so is full-rank. We further see that $g S_k \equiv 0 \mod q$ and hence it is a basis for the lattice $\Lambda^\perp(g)$.

We can generalise this basis to an arbitrary modulus $q$ by considering the binary expansion of the modulus. This leads to the following definition.

**Definition 8.13** (Gadget Lattice, Arbitrary Modulus)**.** *Let $q$ be an arbitrary modulus and $(a_0, \ldots, a_{k-1})$ its binary expansion such that*

$$q = \sum_{i=0}^{k-1} a_i 2^i.$$

*Here $k = \lceil \log_2 q \rceil$. A basis $S_k \in \mathbb{Z}^{k \times k}$ for the lattice $\Lambda^\perp(g)$ for $g$ as in Definition 8.11 is given by*

$$S_k = \begin{pmatrix} 2 & & & & a_0 \\ -1 & 2 & & & a_1 \\ & -1 & \ddots & & \vdots \\ & & \ddots & 2 & a_{k-2} \\ & & & -1 & a_{k-1} \end{pmatrix}.$$

Again, the matrix $S_k$ in Definition 8.13 is a basis of the lattice $\Lambda^\perp(g)$ because $gS_k \equiv 0 \mod q$ and because $\det S_k = q$, i.e. $S_k$ is full-rank.

**Definition 8.14** (Gadget Trapdoor)**.** *Given $A \in R^{n \times m}$ and $G \in R^{n \times k}$, a matrix $T \in R^{m-k \times k}$ is a* trapdoor *for $A$ if $A \begin{pmatrix} T \\ \mathrm{id}_k \end{pmatrix} = G$.*

Assume we are given a trapdoor pair $A$ and $T$ like in Definition 8.14. We now want to sample a small solution $C$ of the matrix equation

$$AC = B \tag{8.6}$$

for some matrix $B$. For this we sample a small perturbation $P$ and compute the matrix $V = B - AP$. Then using the gadget matrix $G$ we can compute a small solution $Z$ to the equation $GZ = V$. Finally, we let $C = P + \begin{pmatrix} T \\ \mathrm{id}_k \end{pmatrix} Z$. To see that this solves Equation (8.6), we insert the definition of $C$ and get

$$AC = A \left[ P + \begin{pmatrix} T \\ \mathrm{id}_k \end{pmatrix} Z \right] = AP + GZ = AP + V = B.$$

See Micciancio and Peikert [MP12] for further details.

This construction is exactly what we require to generate the encoding $C$ in Equation (8.5). Given a graph $G(V, E)$, we are able to generate a trapdoor pair $(A_v, T_v)$ for each vertex $v \in V$. These trapdoor pairs then allow us to encode secret matrices in $R^{n \times n}$ along any edge $e \in \mathcal{E}$ in the graph.

# 9 Obfuscated Automata

In this chapter we consider a somewhat more general class of programs, namely *deterministic finite automata* (DFA). The theory of obfuscating a DFA has been considered before by Lynn et al. [LPS04]. They give an obfuscator in the random oracle model for a special class of regular expressions for which the symbols are given by point functions. As open problems they ask whether regular languages can be obfuscated and whether there is any non-trivial obfuscation result without using the random oracle model. Kuzurin et al. [Kuz+07] state that secure obfuscation of DFAs is one of the most challenging problems in the theory of program obfuscation. Note that we cannot simply apply existing circuit obfuscation solutions to the problem of obfuscating DFAs. Unlike a tree-like circuit which we can evaluate on an input by traversing it from the circuit root to one of the leaves, a DFA can cycle back to previous states. Furthermore, a DFA has a variable number of input symbols.

We give a VBB and perfect circuit-hiding obfuscator for *evasive* DFAs in the standard model. Note that evasive DFAs are more general than other known classes of evasive functions which can be practically obfuscated, such as point functions and conjunctions. We will now explain why we do not consider arbitrary DFAs. It is a classical result that certain types of finite automata can be learned from their input/accept/reject behaviour, cf. Balcázar et al. [Bal+97]. We will also give another possible learning strategy for finite automata in Section 9.4.1 which is applicable if certain information is given. Hence, we will only consider those automata which, without loss of generality, reject almost all inputs. We will call such an automaton *evasive*. Any security claims we make are only for an adversary who does not know any accepting input.

Our solution to the problem of DFA obfuscation uses tools that were developed for homomorphic encryption and multilinear maps. These tools are often used to construct indistinguishability obfuscation schemes. Some of the general purpose iO schemes have questionable hardness assumptions or have been broken altogether, see Ananth et al. [Ana+16a; Ana+16b, Appendix A]. To prove iO security, the underlying multilinear maps need to come with a hard generalised DDH problem. For our application, we instead require a different hard computational problem: Distinguishing two related encodings given certain public zero-testing information should be intractable. We will consider only encodings of evasive DFAs for this problem to make sense. Instead of considering iO we will consider virtual black-box obfuscation, and by extension perfect circuit-hiding obfuscation (recall Theorem 2.2).

## 9.1 Fully Homomorphic Evaluation of Finite Automata

Using the matrix *fully homomorphic encryption* (FHE) schemes by Hiromasa et al. [HAO15] and Genise et al. [Gen+19] Alice may generate a private key and publish an encrypted secret finite automaton to Bob. A finite automaton is represented by a set of transition matrices, one matrix for each possible input symbol. The transition matrices themselves are $n \times n$ square matrices where $n$ is the number of states of the automaton. The homomorphic properties allow Bob to evaluate the secret finite automaton on an arbitrary input. The result of this evaluation is an

encrypted vector which Alice may decrypt using her private key. This can for example be used to evaluate secret *regular expressions* by a remote user while a central server can decide about the result.

Desmoulins et al. [Des+18] consider pattern matching on encrypted streams. For this, they construct a searchable encryption scheme based on public key encryption and bilinear pairings. This approach is sensible when the original data needs to be protected by encryption. This is different to the situation we consider since we only wish to protect the substring pattern.

Genise et al. [Gen+19] state the matching of anti-virus signatures as a possible application of such regular expressions. Consider a security company that analyses computer viruses and distributes virus signatures to their clients. The company wants to protect its intellectual property (the virus signatures). It essentially requires a scheme which allows for the distribution of encrypted virus signatures that the clients can apply to their data. One problem with this setup is the need for interactivity. In their scheme, Alice uses a matrix FHE scheme to encrypt a virus signature represented by an automaton and sends it to Bob. Bob then applies his input to the hidden automaton which produces an encrypted state vector. If Bob wants to learn whether there indeed is a virus present, he needs to send back an encrypted state vector to Alice. She can then decrypt the encrypted state vector and notify Bob accordingly.

Additionally, the analysis of Genise et al. [Gen+19] does not consider an *adaptive attack* in the form of multiple queries with an oracle that reports accept/reject for arbitrary inputs. As mentioned, such an oracle can be used to leak parts or all of the finite automaton description, cf. Balcázar et al. [Bal+97]. In this adaptive setting, we argue that the number of allowed oracle queries needs to be small enough for arbitrary finite automata, or a specific class needs to be used: We propose the class of evasive finite automata.

We consider obfuscation for deterministic finite automata and in particular restrict to the class of evasive DFAs in light of Balcázar et al. [Bal+97]. DFAs can represent problems such as *regular expressions* and *conjunctions* (also known as *pattern matching with wildcards*).

We obtain an obfuscator for evasive regular expressions and consequently solve the open problem of *obfuscated substring matching*. Given a plaintext input $x \in \{0,1\}^n$, obfuscated substring matching is the problem of identifying whether $x$ contains a secret substring $s \in \{0,1\}^m$. We achieve something even more general, as the substring can be given by a regular expression. This gives a complete and non-interactive solution to the virus testing application suggested by Genise et al. [Gen+19]. As another special case example, our DFA obfuscator provides yet another solution for obfuscated conjunctions (recall Chapter 7).

## 9.2 Matrix (Graded) Encodings

Recall Definition 8.10. We want to consider a version of a graded encoding scheme where the plaintext elements are matrices and vectors. We need to be able to apply encoded matrices to encoded vectors which should be equivalent to applying the corresponding plaintext matrices to plaintext vectors.

In a graded encoding scheme, we attach a *grading* to each encoding. It is only possible to add encodings at the same *level* to produce another encoding of the same level. When multiplying elements of different levels, say $\ell_1$ and $\ell_2$, we produce an encoding of a higher level, for example $\ell_1 + \ell_2$. We should think of the public key $\mathbf{pk}$ as a collection of individual zero-testing keys $\mathbf{pk} = \{\mathbf{pk}_\ell\}_{\ell \in L}$. Concretely, for a fixed level $\ell$, if the public key contains $\mathbf{pk}_\ell \in \mathbf{pk}$ then we may zero-test encodings of level $\ell$. The downside is that in all instantiations such a grading implies much larger public keys. We will avoid this at the cost of an additional *circular security*

assumption.

### 9.2.1 GGH15

Let us briefly remind the reader of the matrix graded encoding scheme by Gentry et al. [GGH15] which we introduced in Section 8.4.4.

**Matrix Encoding.** Given a matrix $A \in R^{n\times m}$, encode a secret matrix $S \in R^{n\times n}$ with small entries as a matrix $C \in R^{m\times m}$ with small entries such that $AC = SA + E$ for some small error matrix $E \in R^{n\times m}$.

**Key Generation.** In Section 8.5 we saw how to sample an encoding $C$ using a lattice trapdoor. In practice, depending on a security parameter $\lambda \in \mathbb{N}$, we fix a modulus $q$, and matrix dimensions $n, m \in \mathbb{N}$. The small matrices are sampled from a $\beta$-bounded distribution $\chi$. Fix a maximal grading $\kappa \in \mathbb{N}$, then the modulus should satisfy $q > (4m\beta)^\kappa \lambda^{\omega(1)}$ which we require for security and additionally for a $\kappa$ grading.

**Vector Encoding.** Similarly, given a secret vector $s \in R^n$ with small entries, we can encode it by sampling a short vector $c \in R^m$ according to

$$Ac = sA + e$$

for some short error vector $e \in R^m$.

**Homomorphic Operations.** Recall that this construction is additively and multiplicatively homomorphic, see Equation (8.2) and Equation (8.3). Finally, applying an encoded matrix $C$ to a vector $c$ works via the identity

$$ACc = (SA + E)c = S(sA + e) + Ec = SsA + e'. \tag{9.1}$$

**Zero-testing.** Given an encoding $C$ of a secret $S$ at *multiplicative level* $\ell$ such that the error $E$ is bounded by $\|E\|_\infty \leq \beta(2m\beta)^{\ell-1}$, zero-testing is possible. Compute $AC$ and test whether $\|AC\|_\infty \leq \beta(2m\beta)^{\ell-1}$. If $S = 0$ then this test succeeds and if $S \neq 0$ then $\|AC\|_\infty > \beta(2m\beta)^{\ell-1}$ with high probability.

**Error Bounds and Correctness.** Assuming $\|C\|_\infty, \|S\|_\infty, \|E\|_\infty \leq \beta$ for some threshold $\beta$, it is immediately clear from Equation (8.2) that after adding two encodings, the resulting error is bounded by $\|E'\|_\infty \leq 2\beta$. Similarly, from Equation (8.3) we see that after multiplying two encodings, the resulting error is bounded by $\|E'\|_\infty \leq 2m\beta^2$ (and also for the secret $S_1S_2$ and encoding $C_1C_2$).

We said that the maximal grading of the encoding scheme with our choice of parameters is $\kappa$. By induction we find that after multiplying $\kappa$ encodings, the error is bounded by $\beta(2m\beta)^{\kappa-1}$. Now assume $\|c\|_\infty, \|s\|_\infty, \|e\|_\infty \leq \beta$ for an encoding $c$ of a vector $s$. Finally, by induction, from Equation (9.1) we find that after applying a sequence of $\kappa - 1$ matrices to an encoded vector, the resulting error is bounded by $\|e'\|_\infty \leq \beta(2m\beta)^{\kappa-1}$, see also [WZ17a].

**Security.** Chen et al. [CVW18] considered the GGH15 encoding scheme from the viewpoint of obfuscation for *matrix branching programs*. They give rules about the form of the secret matrices $S$ such that security can be reduced to the LWE assumption for lattices. To encode arbitrary matrices $M$, they give an embedding of $M$ into a larger matrix $S$ that is still compatible with matrix-multiplication. Chen et al. [CVW18] showed that their generalised GGH15 encodings for branching programs are secure under LWE. Specifically, in the general GGH15 scheme, they encode a secret $S$ along a *path* $(i, j)$ such that $A_j C = SA_i + E$ for different random matrices $A_i, A_j$. In the end we only publish the very first $A_1$ that is required for the final zero-test.

In our setting, we set all those matrices $A_i$ equal to a single matrix $A$, except for a special *final* matrix $M_f$ which we encode with respect to a different matrix $B$ such that $BC_f = M_f A + E$. We do this because unlike circuits, which can be translated into matrix branching programs of a fixed depth, DFAs usually have loops that connect states to themselves under input of certain symbols. We will keep the matrix $A$ secret and only publish $B$ such that we are forced to apply the final matrix $M_f$ before zero-testing. Hence, we need to assume circular security for the encodings. This also shrinks the size of the public parameters and allows for a much larger number of DFA inputs in our application.

### 9.2.2 HAO15

The matrix FHE scheme of Hiromasa et al. [HAO15] is somewhat related to the scheme by Gentry et al. [GGH15] we described in Section 9.2.1. The hardness of both schemes is connected to the hardness of finding *approximate eigenspaces*.

Depending on a security parameter $\lambda \in \mathbb{N}$, fix a modulus $q$, a lattice dimension $n$, and a distribution $\chi$ over $\mathbb{Z}$. We are working over the ring $R = \mathbb{Z}/q\mathbb{Z}$. Assume the matrices we want to encode are from $\{0,1\}^r$ for some $r \in \mathbb{N}$. Set $\ell = \lceil \log(q) \rceil$, $N = (n + r)\ell$.

Let $g = (2^i)_{i=0,\dots,\ell-1} \in R^\ell$ be the gadget vector. Fix $G = g^T \otimes \mathrm{id}_{n+r} \in R^{(n+r)\times N}$, the gadget matrix. We may further assume that there exists a randomised algorithm $G^{-1}(v)$ that for an input $v \in R^{n+r}$, samples a vector $v' \leftarrow G^{-1}(v) \in R^N$ such that $Gv' = v$. See also Section 8.5.

**Key Generation.** For key generation, we sample a secret matrix $S' \leftarrow \chi^{r\times n}$ and set

$$S = \left(\mathrm{id}_r \mid -S'\right) \in R^{r\times(n+r)}.$$

A priori, this matrix FHE scheme does not support zero-testing and since we are not interested in public encryption, we do not need any public parameters here. We will describe the public key when we discuss our solution for a zero-testing primitive.

**Matrix Encoding.** Given a matrix $M \in \{0,1\}^{r\times r}$, we sample $A' \leftarrow R^{n\times N}$ uniformly and $E \leftarrow \chi^{r\times N}$ and output the encoding

$$C = \left(\frac{S'A' + E}{A'}\right) + \left(\frac{MS}{0}\right) G \in R^{(n+r)\times N}.$$

It holds that $SC = MSG + E$.

**Vector Encoding.** Similarly, given a vector $v \in R^r$, we sample $a \leftarrow R^n$ uniformly and $e \leftarrow \chi^r$ and output the encoding

$$c = \left(\frac{S'a + e}{a}\right) + \left(\frac{v}{0}\right) \in R^{n+r}.$$

It holds that $Sc = v + e$.

**Vector Encryption.** The scheme also supports encryption and decryption of vectors. For this, fix an upper bound $b$ on the $\|\cdot\|_\infty$-norm of vectors that should be possible to encrypt and decrypt and set

$$\beta = \lfloor q/b \rfloor.$$

For example, to encrypt binary secrets we can set $b = 2$. To encrypt a vector $v$, we will scale it by $\beta$ such that the $\|\cdot\|_\infty$-norm of the error is bounded by $\beta$ with high probability. Formally, to encrypt $v \in \{0, \ldots, b-1\}^n$, output the encoding $c$ of $\beta v$ such that $Sc = \beta v + e$. To decrypt $c$, given the secret $S$, we compute

$$v = \left\lceil \frac{Sc \mod q}{\beta} \right\rfloor, \tag{9.2}$$

i.e. we round the entries of $(1/\beta)Sc$ to the closest integer.

**Homomorphic Operations.** Given two encodings $C_1, C_2$, addition is simply computing $C_1 + C_2$. The encodings can by multiplied by computing $C_1 G^{-1}(C_2)$, denote this by $C_1 \odot C_2$. Applying an encoded matrix $C$ to an encoded vector $c$ is computing $CG^{-1}(c)$.

**Zero-testing.** Testing whether a given encoding is an encoding of zero is slightly more complicated because we cannot publish the secret matrix $S$. For our application, the following construction is sufficient. Let $M_f$ be the $r \times r$ matrix which is zero everywhere except for a single 1 in its lower right corner, i.e. $(M_f)_{r,r} = 1$. Let $C_f$ be the encoding of $M_f$. Let $c$ be an encrypted vector. We need to test whether $C_f c$ is an encryption of $e_r$, the $r$-th canonical basis vector. To test for this, we publish the last row of the secret matrix $S$, call it $s_r \in R^{n+r}$. Assuming we only ever encrypt canonical basis vectors, the problem is then equivalent to checking whether $\lceil (1/\beta)(s_r \cdot c \mod q) \rfloor$ equals 1, see Equation (9.2). Equality holds if and only if $C_f c$ is an encryption of a vector that has a 1 in coordinate $r$, see the proof of Lemma 9.1 for details. This limited construction allows us to use the HAO15 matrix FHE scheme as a matrix encoding scheme.

**Error Bounds and Correctness.** In the plain HAO15 matrix FHE scheme, to decode an encoded matrix, the error needs to be bounded by $\|E\|_\infty \leq q/8$. In our application, we do not need to decode matrices, but decrypt vectors. To correctly decrypt encrypted vectors, we see from Equation (9.2) that the error needs to be bounded by $\|e\|_\infty \leq \beta/2 = q/4$. Hiromasa et al. [HAO15] showed that the noise growth is asymmetric and hence computing a polynomial length chain of homomorphic multiplications leads to a noise growth by a multiplicative polynomial factor. Denote with $|\chi|$ the standard deviation of the distribution $\chi$. Genise et al. [Gen+19] showed that error produced by the application of $\kappa$ matrices $(M_i)_{i=1,\ldots,\kappa}$ on a vector is bounded by

$$\|e_\kappa\|_\infty \leq |\chi| N \left( 1 + \kappa \max_{1 \leq i \leq \kappa} \left\| \prod_{j=\kappa}^{i} M_j \right\|_\infty \right).$$

For our application, we will consider matrices $(M_\sigma)_{\sigma \in \Sigma}$ that describe a DFA. We argue that for such matrices we obtain a large maximal grading $\kappa \sim q/\log(q)$. Genise et al. [Gen+19] introduced an *ambiguity measure* that better restricts the error bound for finite automata, depending on their *ambiguity type*. They considered more general NFAs whereas we shall restrict to DFAs only. They showed that DFAs are what they call *unambiguous* and that the error then can be bounded as $\|e_\kappa\|_\infty \leq |\chi|(N\kappa + 1)$. We find that for $\|e_\kappa\|_\infty$ to be bounded by $\beta/2 = q/4$, we require that

$$\kappa \leq \frac{q}{4\sqrt{n}(n+r)\lceil \log(q) \rceil}. \tag{9.3}$$

## 9.3 HAO15 Zero-Testing and Computational Assumptions

The original matrix FHE scheme by Hiromasa et al. [HAO15] enjoys CPA security and does not allow for zero-testing. In Section 9.2.2 we constructed a zero-testing primitive. This requires us to introduce an additional hardness assumption if we want to speak about security when using our extended HAO15 scheme as a matrix encoding scheme.

**Definition 9.1** (DFA Security)**.** *Consider the HAO15 matrix graded encoding scheme with security parameter $\lambda \in \mathbb{N}$ for a matrix dimension $r \in \mathbb{N}$. Let $D$ be a distribution over $\mathcal{M}_r \times \{0,1\}^{poly(\lambda)}$ where $\mathcal{M}_r$ is a family of sequences of matrices $(M_\sigma)_{\sigma\in\Sigma}$ over $\{0,1\}^{r\times r}$.*

*Let $((M_\sigma)_{\sigma\in\Sigma}, \alpha) \leftarrow D$ and $(M'_\sigma)_{\sigma\in\Sigma}$ be a sequence of matrices in $\{0,1\}^{r\times r}$ such that $M_\sigma - M'_\sigma$ is all zeroes apart from a single $\pm 1$ in some row but not the last row. Consider, for all $\sigma \in \Sigma$, the HAO15 encodings $C_\sigma$ of $M_\sigma$, $C'_\sigma$ of $M'_\sigma$, and $c$ and $c'$ of the canonical basis vector $e_1$ as in Section 9.2.2 under a secret key $S$ such that*

$$\begin{aligned} SC_\sigma &= M_\sigma SG + E,\ Sc = \beta e_1 + e, \\ SC'_\sigma &= M'_\sigma SG + E',\ Sc' = \beta e_1 + e', \end{aligned}$$

*where $E, E'$ and $e, e'$ are error matrices and error vectors, respectively.*

*We say that HAO15 satisfies* DFA security *for $D$ if the following two distributions are computationally indistinguishable:*

$$(s_r, (C_\sigma)_{\sigma\in\Sigma}, c, \alpha) \overset{c}{\approx} (s_r, (C'_\sigma)_{\sigma\in\Sigma}, c', \alpha),$$

*where $s_r$ is the last row of the secret key $S$.*

Note that Definition 9.1 is closely related to the definition of IND-CPA security for an asymmetric cipher: The adversary is given a number of encryptions of known messages and needs to distinguish them. In our case, we additionally require that the messages are related and there is some partial knowledge of the secret key revealed.

The sequences of matrices $(M_\sigma)_{\sigma\in\Sigma}$ that we will consider are matrices that encode DFAs such that the min-entropy of the shortest accepting input word conditioned on the auxiliary information $\alpha$ is at least $\lambda(r)$. See Definitions 9.2 and 9.3 for a formal definition of such a distribution.

Hiromasa et al. [HAO15, Theorem 4] states that the plain HAO15 scheme is semantically secure based on a circular security assumption and the hardness of the decisional learning with error problem (DLWE) for parameters $n, q, \chi$. If we did not publish $s_r$, then their assumption would imply that Definition 9.1 holds for appropriate parameters.

Given $s_r$ we may test whether the last coordinate of an encoded vector is 0 or 1. Hence we need to consider certain safeguards, which are described in detail in Section 9.4.1. We want that for every additional encoded *state vector*, the last coordinate is 0 with overwhelming probability. This is true for the distributions of evasive DFAs that we will consider. Using $s_r$ we can also learn the entries of the last row of the DFA matrices. Hence, we assume that the last row of the encoded matrices always follows a certain structure. This ensures indistinguishability as required by Definition 9.1.

Finally, we conjecture that the knowledge of the last row of the secret does not weaken the security of the HAO15 matrix encoding scheme. The hardness of (D)LWE with *leaky secrets* was studied by Goldwasser et al. [Gol+10].

## 9.4 Finite Automata and Transition Matrices

Fix a number of states $r \in \mathbb{N}$. Fix an alphabet $\Sigma$ and for each symbol $\sigma \in \Sigma$, let $M_\sigma \in \{0,1\}^{r \times r}$ be the *transition matrix* corresponding to $\sigma$.

In case of a finite automata $M$, $\Sigma$ represents the different input symbols which induce transitions between the $r$ different states, i.e. $(M_\sigma)_{j,i} = 1$ if and only if there is a transition from state $i$ to state $j$ for an input $\sigma$. Hence, such an $M_\sigma$ acts on the $i$-th canonical basis vector $e_i$ such that $e_j = M_\sigma e_i$. Without loss of generality, let 1 be the initial state (represented by $e_1$) and let $r$ be the final state (represented by $e_r$). There is a distinction between *deterministic* and *non-deterministic* finite automata (DFA and NFA, respectively). On the one hand, a DFA has a unique state transition for each state and input. On the other hand, a NFA may transition into multiple states on each input or transition without any input at all. In general, a NFA will not have a unique accepting state.

### 9.4.1 General Safeguards

We will now introduce two general safeguards to avoid partial evaluation and leaking intermediate states and state transitions. These safeguards are important for our specific construction based on the HAO15 matrix encoding scheme of Section 9.2.2 as well as the general construction from arbitrary matrix (graded) encoding schemes we will introduce in Section 9.8.

**State Transitions.** Without loss of generality, let $\Sigma = \{\sigma_1, \ldots, \sigma_m\}$ be the set of symbols, for some $m \in \mathbb{N}$. Consider a DFA with $r \in \mathbb{N}$ states and let $r$ be the accepting state. To avoid leaking state transitions, we need to ensure that the matrices representing the DFA have the following structure:

$$M_{\sigma_1} = \begin{pmatrix} * & * \\ 0 & 0 \end{pmatrix}, \ldots, M_{\sigma_{m-1}} = \begin{pmatrix} * & * \\ 0 & 0 \end{pmatrix}, M_{\sigma_m} = \begin{pmatrix} * & \cdots & * & 0 & 0 \\ \vdots & \ddots & \vdots & \vdots & \vdots \\ * & \cdots & * & 0 & 0 \\ 0 & \cdots & 0 & 1 & 1 \end{pmatrix}.$$

We set the last row of the matrices $M_{\sigma_1}, \ldots, M_{\sigma_{m-1}}$ to zero. This means that none of the input symbols $\sigma_1, \ldots, \sigma_{m-1}$ can transition the DFA into the accepting state $r$. The structure of the matrix $M_{\sigma_m}$ is chosen such that $\sigma_m$ is the unique input symbol which can transition the DFA into the accepting state $r$. We also allow for an arbitrary number of additional inputs of the symbol $\sigma_m$ sending the state $r$ to itself.

This ensures that an attacker does not learn anything that is not already public knowledge in the system. As mentioned in Section 9.3, this is important for the validity of the security assumption in our application. We require that the last rows of all transition matrices follow the same structure.

**Partial Evaluation.** We need to make sure that no adversary can distinguish between states after merely partially evaluating the DFA. To see why, consider the following attack strategy.

We can evaluate the obfuscated DFA on progressively longer input words and each time record the encoded state vector. Although we do not learn the state vector itself, using a zero-testing primitive, we can decide when two states are the same for different inputs. Even if we force a fixed input word length (for example by restricting the zero-test to only be possible after evaluating a certain number of input symbols), we can simply prepend each different word by a fixed prefix. Using statistical analysis on the number of encountered states, we can then

try to either construct an accepted input directly or at least (partially) learn the *structure* of the underlying DFA.

To remedy this, we need to make sure that nothing can be learned about individual states after (partial) DFA evaluation, apart from whether or not they are accepting states. The key idea is to *erase* all non-accepting states before zero-testing is possible. For this, consider the following matrix

$$M_f = \begin{pmatrix} 0 & \cdots & 0 & 0 \\ \vdots & \ddots & \vdots & \vdots \\ 0 & \cdots & 0 & 0 \\ 0 & \cdots & 0 & 1 \end{pmatrix} \in \{0,1\}^{r \times r}.$$

It holds that for all canonical basis vectors $e_i$ for $i = 1, \ldots, r-1$ we have $M_f e_i = 0$, whereas $M_f e_r = e_r$. Another way to express this is that the matrix $M_f$ maps all state vectors to the zero vector if they are not equal to the final state vector but leaves the final state vector invariant.

## 9.5 Obfuscated Finite Automata

We would like to construct an obfuscator for finite automata. Every finite automaton $M$ induces a program $P_M : \Sigma^* \to \{0,1\}$ that outputs 1 for an accepted input sequence and 0 for a rejected one.

**Definition 9.2** (Evasive Finite Automata Collection)**.** *Let $\{\mathcal{M}_r\}_{r \in \mathbb{N}}$ be a collection of finite automata such that every automaton in $\mathcal{M}_r$ has $r$ states. The collection is called* evasive *if there exists a negligible function $\epsilon$ such that for every $r \in \mathbb{N}$ and for every polynomial size input $y \in \Sigma^*$:*

$$\Pr_{M \leftarrow \mathcal{M}_r}[P_M(y) = 1] \leq \epsilon(r).$$

It is important to limit to polynomial size inputs $y \in \Sigma^*$ in Definition 9.2 since otherwise we could let $y$ be the string that contains all possible substrings of a certain length. We need to consider evasive finite automata since the transition matrices of a non-evasive one can be learned from its input/accept/reject behaviour [Bal+97]. It is then natural to use Definition 2.7 – perfect circuit-hiding obfuscation – as the security notion for evasive automata. An adversary finds an accepted input with negligible probability and so cannot recover the description of the automata.

**Definition 9.3** (DFA Evasive Distribution with Auxiliary Information)**.** *Consider an ensemble $\mathcal{D} = \{D_\lambda\}_{\lambda \in \mathbb{N}}$ of distributions $D_\lambda$ over $\mathcal{M}_{r(\lambda)} \times \{0,1\}^{poly(\lambda)}$ where $\{\mathcal{M}_{r(\lambda)}\}_{r(\lambda) \in \mathbb{N}}$ is an evasive finite automata collection. We say that $\mathcal{D}$ is* DFA evasive with auxiliary information *if for every $(M, \alpha) \leftarrow \mathcal{D}_\lambda$ the min-entropy of the shortest accepted word $w \in \Sigma^*$ of $M$ conditioned on the auxiliary information $\alpha$ is at least $\lambda$.*

**Examples of DFA Evasive Distributions.** We will give two examples for DFA evasive distributions.

- *String Matching.* Given a plaintext input $s \in \{0,1\}^n$, obfuscated substring matching is the problem of identifying whether $s$ contains a secret substring $x \in \{0,1\}^k$, for $k \leq n \in \mathbb{N}$. We achieve something even more general, as the substring can be given by a regular expression.

Consider an alphabet of three symbols $\Sigma = \{0, 1, \perp\}$, where $\perp$ is the unique symbol that may transition the DFA into the accepting state as Section 9.4.1 demands. Consider $U_k$, the uniform distribution over $\{0,1\}^k$. Then any string sampled from $U_k$ has min-entropy at least $k$. Define now $\{\mathcal{M}_r\}_{r\in\mathbb{N}}$ to be the collection of evasive DFAs with $r = k+2$ states such that a DFA sampled from $\mathcal{M}_r$ matches some string sampled from $U_k$. Hence $\{\mathcal{M}_r\}_{r\in\mathbb{N}}$ is an evasive DFA collection which has min-entropy at least $\lambda(r) = r - 2 = k$. This collection is efficiently samplable by sampling a random string $x$ from $U_k$ and outputting the DFA matching the word $x \parallel \perp$ (i.e. $x$ concatenated with $\perp$).

- *Conjunctions.* Another example of an evasive DFA collection are conjunctions. Consider again the alphabet $\Sigma = \{0, 1, \perp\}$ as above. Given a conjunction evasive (see Definition 7.2) distribution $D_\mu$ for conjunctions of length $n(\mu)$, we can define an evasive DFA collection $\mathcal{M}_r$ with $r = n(\mu) + 2$ states which has min-entropy at least $\lambda(r) = \mu$: Every DFA from this collection accepts a string $y$ that satisfies the corresponding conjunction from $D_\mu$.

### 9.5.1 Obfuscator and Obfuscated Program

For every evasive DFA $M$ with maximal input length $\kappa$, there exists a program $P_M : \Sigma^* \to \{0,1\}$ that computes whether $M$ accepts an input word $w \in \Sigma^*$ (with $|w| \leq \kappa$) and evaluates to 1 in this case, otherwise to 0. Denote by $\mathcal{P}$ the family of all such programs $P_M$. The obfuscator $\mathcal{O} : \mathcal{P} \to \mathcal{P}'$ takes one such program $P_M \in \mathcal{P}$ and uses Algorithm 9.1 to output another program in a different family denoted by $\mathcal{P}'$.

Algorithm 9.1 uses the HAO15 matrix encoding scheme (assume the maximal grading is $\kappa$) to encode the required matrices and vectors. The output is given by the tuple $(s_r, (C_\sigma)_{\sigma\in\Sigma}, c)$. In this tuple, $s_r$ is the last row of the HAO15 secret $S$, $(C_\sigma)_{\sigma\in\Sigma}$ is the sequence of encodings of the state transition matrices $(M_\sigma)_{\sigma\in\Sigma}$, and $c$ is an encoding of the first canonical basis vector $e_1$.

We assume that the initial and accepting state of the finite automaton are given by the state 1 and state $r$, respectively. We further assume that the DFA matrices $(M_\sigma)_{\sigma\in\Sigma}$ satisfy the safeguard requirements described in Section 9.4.1. Erasing partial information from the final state using $M_f$ is equivalent to only being able to test whether the last coordinate of the state vector is 0 or 1. Recall, in Section 9.2.2, we assumed that our state vectors are always canonical basis vectors. This is certainly true for any DFA. Hence publishing only $s_r$ is equivalent to erasing partial state information using $M_f$.

As the decoding algorithm is a universal algorithm, we will simply denote the *obfuscated program* $\mathcal{O}(P_M)$ with the tuple $(s_r, (C_\sigma)_{\sigma\in\Sigma}, c)$. During the execution of the obfuscated program, Algorithm 9.2 is used to determine whether an input word $w \in \Sigma^*$ is accepted by the DFA or not.

**Algorithm 9.1** Encoding (Obfuscating the finite automaton)

**procedure** ENCODE($(M_\sigma)_{\sigma\in\Sigma}$)
  Run HAO15 matrix encoding scheme key generation and obtain secret key $S$.
  Compute $(C_\sigma)_{\sigma\in\Sigma}$ by encoding $(M_\sigma)_{\sigma\in\Sigma}$ such that $SC_\sigma = M_\sigma SG + E$ for all $\sigma \in \Sigma$.
  Compute state vector $c$ by encoding $e_1$ such that $Sc = \beta e_1 + e$.
  Let $s_r$ be the last row of $S$.
  **return** $(s_r, (C_\sigma)_{\sigma\in\Sigma}, c)$
**end procedure**

### 9.5.2 Obfuscated Program Evaluation

We may evaluate the obfuscated automaton on a word $w \in \Sigma^*$ with $|w| \leq \kappa$ as follows:

1. Compute the encoded vector $c_w$ corresponding to $(\prod_{i=|w|}^{1} M_{w_i})e_1$ using the sequence $(C_\sigma)_{\sigma\in\Sigma}$ and the encoded initial state $c$.
2. The input word $w$ is accepted if $c_w$ is an encryption of the $r$-th canonical basis vector and thus we simply output $\lceil (1/\beta)(s_r \cdot c_w \mod q) \rfloor$.

Again, Algorithm 9.2 presents an algorithmic description.

**Algorithm 9.2** Evaluation (Executing the obfuscated program)

**procedure** EVALUATE$(s_r, (C_\sigma)_{\sigma\in\Sigma}, c; w \in \Sigma^*)$
  **for all** $i = 1, \ldots, |w|$ **do**
    Update the state vector $c = C_{w_i} G^{-1}(c)$.
  **end for**
  **return** $\lceil (1/\beta)(s_r \cdot c \mod q) \rfloor$
**end procedure**

**Lemma 9.1** (Correctness)**.** *Consider the algorithms* ENCODE *(Algorithm 9.1) and* EVALUATE *(Algorithm 9.2) (based on the modified HAO15 matrix FHE scheme with maximal grading $\kappa$ determined by Equation (9.3)). For every DFA $\mathcal{M}$ represented by $(M_\sigma)_{\sigma\in\Sigma}$, for every*

$$(s_r, (C_\sigma)_{\sigma\in\Sigma}, c) \leftarrow \textsc{Encode}((M_\sigma)_{\sigma\in\Sigma})$$

*and for every input $w \in \Sigma^*$ with $|w| < \kappa$ it holds that*

$$\textsc{Evaluate}(s_r, (C_\sigma)_{\sigma\in\Sigma}, c; w) = P_{\mathcal{M}}(w).$$

*Proof.* Recall the modified HAO15 matrix FHE scheme from Section 9.2.2. Given a sequence of transition matrices $(M_\sigma)_{\sigma\in\Sigma}$, the obfuscator produces the tuple $(s_r, (C_\sigma)_{\sigma\in\Sigma}, c)$ such that $C_\sigma$ is an encoding of $M_\sigma$ for all $\sigma \in \Sigma$. This means that $SC_\sigma = M_\sigma SG + E$, where $S$ is the HAO15 secret. Further, $c$ is an encoding such that $Sc = \beta e_1 + e$, where $e_1$ is the first canonical basis vector. Finally, $s_r$ is the last row of the secret $S$.

The evaluation algorithm computes the final state vector

$$c_w = \left( \bigodot_{i=|w|}^{1} C_{w_i} \right) G^{-1}(c).$$

This corresponds to the following calculation with plaintext information

$$t = \left( \prod_{i=|w|}^{1} M_{w_i} \right) e_1.$$

The automaton accepts the input if $t = e_r$. We see that $c_w$ is an encoding of $t$ such that $Sc_w = \beta t + e$ for some error $e$. Given only $s_r$, we have the following equation

$$\left( \frac{0_{(r-1)\times(n+r)}}{s_r} \right) c_w = \beta \left( \frac{0_{r-1}}{t_r} \right) + \left( \frac{0_{r-1}}{e'} \right),$$

where $e'$ is the last coordinate of $e$. By Equation (9.3) the error is bounded by $\|e_\kappa\|_\infty \leq \beta/2$ if we choose the maximal grading $\kappa$ such that

$$\kappa = \frac{q}{4\sqrt{n}(n+r)\lceil \log(q) \rceil}.$$

If $|w| < \kappa$, then $|e'| \leq \|e\|_\infty < \|e_\kappa\|_\infty \leq \beta/2$. Hence, computing $\lceil (1/\beta)(s_r \cdot c_w \mod q) \rfloor$ correctly determines whether $c_w$ is an encryption of the accepting state $e_r$ or not. Correctness follows as required. □

### 9.5.3 Security

In this section we analyse the security of our DFA obfuscator using the HAO15 matrix encoding scheme which we introduced in Section 9.2.2.

Note that we make no claim of security once an accepting input is known to an adversary. First and foremost, there is a classical result by Balcázar et al. [Bal+97] that shows that the description of a finite automaton can be learned from oracle access when given accepted and rejected inputs. Second, we need to keep in mind that an actual matrix graded encoding scheme could exhibit non-modelled (and thus unwanted) behaviour, cf. Ananth et al. [Ana+16a; Ana+16b, Appendix A].

**Theorem 9.1.** *Let $\mathcal{D} = \{D_\lambda\}_{\lambda \in \mathbb{N}}$ be an efficiently samplable DFA evasive distribution with auxiliary information (Definition 9.3). Assume that for every $\lambda \in \mathbb{N}$ it holds that HAO15 with security parameter $\lambda$ is DFA secure for $D_\lambda$ (Definition 9.1). Then the obfuscator $\mathcal{O}$ is a VBB obfuscator for $\mathcal{D}$.*

*Proof.* The obfuscator is functionality preserving by Lemma 9.1. It is also clear that the obfuscator causes only a polynomial slowdown when compared to an unobfuscated DFA since the evaluation Algorithm 9.2 runs in time polynomial in all the involved parameters.

By Theorem 2.1 it suffices to show that there exists a (non-uniform) PPT simulator $\mathcal{S}$ such that, for the distribution ensemble $\mathcal{D} = \{D_\lambda\}_{\lambda \in \mathbb{N}}$, it holds that

$$(\mathcal{O}(P), \alpha) \overset{c}{\approx} (\mathcal{S}(|P|), \alpha),$$

where $(P, \alpha) \leftarrow D_\lambda$. Recall that $D_\lambda$ is a distribution over $\{\mathcal{M}_{r(\lambda)}\}_{r(\lambda) \in \mathbb{N}} \times \{0,1\}^{\mathrm{poly}(\lambda)}$.

We will construct the simulator $\mathcal{S}$: It takes as input $|P|$ and determines the parameter $r \in \mathbb{N}$ and runs Algorithm 9.3.

**Algorithm 9.3** Encoding Simulator

**procedure** SimulateEncode($r \in \mathbb{N}$)
  Sample random DFA $(M'_\sigma)_{\sigma \in \Sigma}$ from $\mathcal{M}_r$, this DFA has min-entropy $\lambda$.
  Run HAO15 matrix encoding scheme key generation and obtain secret key $S'$.
  Compute $(C'_\sigma)_{\sigma \in \Sigma}$ by encoding $(M'_\sigma)_{\sigma \in \Sigma}$ such that $S'C'_\sigma = M'_\sigma S'G + E$ for all $\sigma \in \Sigma$.
  Compute state vector $c'$ by encoding $e_1$ such that $S'c' = \beta e_1 + e$.
  Let $s'_r$ be the last row of $S'$.
  **return** $(s'_r, (C'_\sigma)_{\sigma \in \Sigma}, c')$
**end procedure**

Denote now with $(s_r, (C_\sigma)_{\sigma \in \Sigma}, c)$ a real instance obtained from obfuscating an evasive DFA given by the transition matrices $(M_\sigma)_{\sigma \in \Sigma}$ sampled from the distribution $\mathcal{M}_{r(\lambda)}$ such that the DFA has min-entropy $\lambda$. Similarly, let $(s'_r, (C'_\sigma)_{\sigma \in \Sigma}, c')$ be the output from the simulator $\mathcal{S}$ called on

$r$. This is essentially an obfuscation of a random evasive DFA given by the transition matrices $(M'_\sigma)_{\sigma\in\Sigma}$, again with min-entropy $\lambda$. The last rows of $M_\sigma$ and $M'_\sigma$ are the same for all $\sigma \in \Sigma$. This follows from our assumption of Section 9.5.1: The input $(M_\sigma)_{\sigma\in\Sigma}$ to Algorithm 9.1 satisfies the safeguards of Section 9.4.1.

We will now show, using a sequence of distributions, that both tuples $(s_r, (C_\sigma)_{\sigma\in\Sigma}, c, \alpha)$ and $(s'_r, (C'_\sigma)_{\sigma\in\Sigma}, c', \alpha)$ are computationally indistinguishable. The strategy is to start from the real and simulated distributions and remove state transitions one by one from both until we meet in the middle where both encoded DFAs are the same. Hence, we need to consider the matrices $M^\Delta_\sigma = M_\sigma - M'_\sigma$. If an entry of $M^\Delta_\sigma$ is 1, we remove a state transition from $M_\sigma$; if an entry is -1, we remove a state transition from $M'_\sigma$. This ensures that the min-entropy of the intermediate DFAs can only stay the same or grow, but never shrink. Note that removing state transitions may result in a system that does not accept any inputs, or may not even be an encoding of a DFA.

- Game $(0, 0, 0)$: Here we consider $(s_r, (C_\sigma)_{\sigma\in\Sigma}, c)$, a real instance obtained from the DFA obfuscator $\mathcal{O}$.
- Game $(0, 0, 1)$: Here we consider $(s'_r, (C'_\sigma)_{\sigma\in\Sigma}, c')$, the output of the simulator $\mathcal{S}$.
- Game $(i, j, 0)$ (for $1 \le i < r, 1 \le j \le r$): Start from the real DFA matrices $(M_\sigma)_{\sigma\in\Sigma}$. For all $\sigma \in \Sigma$, do the following:

  Step 1: Replace full columns.

  **for** $1 \le t < j$ **do**
  **for** $1 \le s < r$ **do**
  **if** $(M^\Delta_\sigma)_{s,t} = 1$ **then**
  Replace $(M_\sigma)_{s,t}$ with 0.
  **end if**
  **end for**
  **end for**

  Step 2: Replace partial columns.

  **for** $1 \le s \le i$ **do**
  **if** $(M^\Delta_\sigma)_{s,j} = 1$ **then**
  Replace $(M_\sigma)_{s,j}$ with 0.
  **end if**
  **end for**

  This yields the distribution $(s_r, (C^{(i,j,0)}_\sigma)_{\sigma\in\Sigma}, c)$, where $(C^{(i,j,0)}_\sigma)_{\sigma\in\Sigma}$ is a randomly chosen encoding of the resulting transition matrices with respect to the fixed secret key $S$.
- Game $(i, j, 1)$ (for $1 \le i < r, 1 \le j \le r$): Start from the simulated DFA matrices $(M'_\sigma)_{\sigma\in\Sigma}$. For all $\sigma \in \Sigma$, do the following:

  Step 1: Replace full columns.

  **for** $1 \le t < j$ **do**
  **for** $1 \le s < r$ **do**
  **if** $(M^\Delta_\sigma)_{s,t} = -1$ **then**
  Replace $(M'_\sigma)_{s,t}$ with 0.
  **end if**
  **end for**
  **end for**

  Step 2: Replace partial columns.

  **for** $1 \le s \le i$ **do**
  **if** $(M^\Delta_\sigma)_{s,j} = -1$ **then**
  Replace $(M'_\sigma)_{s,j}$ with 0.
  **end if**
  **end for**

  This yields the distribution $(s'_r, (C^{(i,j,1)}_\sigma)_{\sigma\in\Sigma}, c')$, where $(C^{(i,j,1)}_\sigma)_{\sigma\in\Sigma}$ is a randomly chosen encoding of the resulting transition matrices with respect to the fixed secret key $S'$.

For $1 \le i < r$, $1 \le j \le r$, in Game $(i, j, \{0, 1\})$, the min-entropy of the encoded DFA is at least $\lambda(r)$ since we only ever remove state transitions. We have that Game $(i, j, \{0, 1\})$ and Game $(i + 1, j, \{0, 1\})$ for $0 \le i < r - 1$, $0 \le j \le r$ are indistinguishable by the DFA security assumption. We also have that Game $(r - 1, j, \{0, 1\})$ and Game $(1, j + 1, \{0, 1\})$ for $1 \le j < r$ are indistinguishable by the DFA security assumption. Finally, the two Games $(r - 1, r, 0)$

and $(r-1, r, 1)$ encode the same DFA under different secret keys $S$ and $S'$ and again are indistinguishable. Hence, by a hybrid argument, it follows that

$$(s_r, (C_\sigma)_{\sigma\in\Sigma}, c, \alpha) \overset{c}{\approx} (s'_r, (C'_\sigma)_{\sigma\in\Sigma}, c', \alpha).$$

We showed that a real obfuscation is computationally indistinguishable from a simulated instance. This completes the proof. □

There is an equivalence of VBB obfuscation and perfect circuit-hiding obfuscation for evasive programs. This allows us to state the following theorem.

**Theorem 9.2.** *Let $\mathcal{D} = \{D_\lambda\}_{\lambda\in\mathbb{N}}$ be an efficiently samplable DFA evasive distribution with auxiliary information (Definition 9.3). Assume that for every $\lambda \in \mathbb{N}$ it holds that HAO15 with security parameter $\lambda$ is DFA secure for $D_\lambda$ (Definition 9.1). Then the obfuscator $\mathcal{O}$ is a perfect circuit-hiding obfuscator for $\mathcal{D}$.*

*Proof.* This follows from Theorem 9.1 and Theorem 2.2. □

## 9.6 Parameters

Genise et al. [Gen+19] gave example parameters and runtime analysis for both the matrix FHE schemes of Hiromasa et al. [HAO15] (see Section 9.2.2) and Genise et al. [Gen+19]. For a finite automaton with 1024 states, they chose a 42-bit modulus $q$. Note that such an overstretched modulus is potentially dangerous in the GGHLM19 setting and does not satisfy the claimed security level as was shown by Lee and Wallet [LW20]. Nevertheless, the HAO15 scheme is assumed to be secure for these parameters and allows for input words of length up to roughly 140000 symbols.

In our case, we achieve obfuscated evaluation of any evasive DFA with sufficient min-entropy. Since we require zero-testing, we obtain a slightly smaller maximal grading. For HAO15, recall Equation (9.3) which we can use to compute the maximal grading if we encode DFA matrices. With a zero-testing primitive, the same parameters as above ($n = 1024, r = 1024, q \approx 2^{42}$) yield a maximal input word length of roughly $10^5$ symbols. This is already more than enough for the applications that we described in the introduction, such as substring matching or virus testing. Alternatively, we can search for substrings in an input which is longer than the bound $\kappa$ by running the obfuscated program on overlapping substrings of the original input.

## 9.7 Alternatives

One possible alternative is to use the matrix FHE scheme by Genise et al. [Gen+19]. Again, we are working over the ring $R = \mathbb{Z}/q\mathbb{Z}$ for some modulus $q$. Fix lattice parameters $n$ and $\ell = \lceil \log(q) \rceil$. Let $\chi^{i\times j}$ be a distribution on $R^{i\times j}$. Let $g = (2^i)_{i=0,\dots,\ell-1} \in R^\ell$ be the gadget vector and fix $G = \mathrm{id}_n \otimes g \in R^{n\times n\ell}$, the gadget matrix. We may assume that there exists a randomised algorithm $G^{-1}(v)$ that for an input $v \in R^n$, samples a vector $v' \leftarrow G^{-1}(v) \in R^{n\ell}$ such that $Gv' = v$. The main algorithms of the GGHLM19 scheme are as follows.

- *Key Generation.* Sample private secrets $E \leftarrow \chi^{n\times n}$ and $S \leftarrow \chi^{n\times n\ell}$ such that $S$ is invertible.
- *Noise Sampling.* For an input vector $v \in R^n$, the algorithm NoiseSamp$(v)$ samples $r \leftarrow G^{-1}(v)$ and outputs $e = Er$.

- *Vector Encoding.* To encode a vector $v \in R^n$, sample $e \leftarrow \text{NoiseSamp}(v)$ and then output $S^{-1}(v + e) \in R^n$.

- *Vector Encryption and Decryption.* Fix an upper bound $b$ on the $\|\cdot\|_\infty$-norm of vectors that should be possible to encrypt and decrypt and set

$$\beta = \lfloor q/b \rfloor. \tag{9.4}$$

  For example, to encrypt binary secrets we can set $b = 2$.

  To encrypt a vector $v$, we will scale it by $\beta$ such that the $\|\cdot\|_\infty$-norm of the error is bounded by $\beta$ with high probability. Formally, to encrypt $v \in R^n$, output the encoding of $\beta v$. To decrypt a ciphertext vector $c \in R^n$, compute $u = Sc = \beta v + e$. Finally, decode $u$ by rounding it to the nearest multiple of $\beta$:

$$v = \left\lceil \frac{Sc \mod q}{\beta} \right\rfloor.$$

- *Matrix Encryption.* To encrypt a matrix $M \in R^{n\times n}$, we first compute $M' = MSG \in R^{n\times n\ell}$ and partition $M'$ into its column vectors $m_i$ for $i = 1, \ldots, n\ell$. We then encrypt each vector $m_i$ to obtain ciphertext vectors $c_i$ from which we form the ciphertext matrix $C \in R^{n\times n\ell}$.

  Hence, $C = S^{-1}(MSG + E)$ for some low-norm matrix $E \in R^{n\times n\ell}$.

An encrypted matrix $C$ can be evaluated on an encrypted vector $c$ by computing $CG^{-1}(c)$. Two encrypted matrices $C_1$ and $C_2$ can be multiplied simply by computing $C' = C_1 G^{-1}(C_2)$ which yields $C' = S^{-1}(M_1 M_2 SG + E')$ for a slightly larger error $E'$.

The main computational assumption of Genise et al. [Gen+19] is based on the following distribution, which can be considered an inhomogeneous matrix version of the NTRU distribution over a base ring $R$, see Section 4.3.

**Definition 9.4** (Matrix Inhomogeneous NTRU (MiNTRU))**.** *Fix a prime $q$ and set $R = \mathbb{Z}/q\mathbb{Z}$. Fix lattice parameters $n$ and $\ell = \lceil \log(q) \rceil$, and set $m = n\ell$. Let $\chi$ be a distribution over $R$. The* matrix inhomogeneous NTRU distribution *is defined as follows: Sample $S \leftarrow R^{n\times n}$, $E \leftarrow \chi^{n\times m}$, and output $A = S^{-1}(G - E) \in R^{n\times m}$.*

The hardness assumption is that the MiNTRU distribution is *pseudorandom*, i.e. that it is indistinguishable from the uniform distribution over $R^{n\times m}$. This is called the MiNTRU-problem.

It can be shown that the secret may be sampled from the error distribution, leading to the definition of the MiNTRU distribution with small secrets. This distribution is abbreviated as MiNTRU$^s$; Genise et al. [Gen+19] showed that it is pseudorandom if MiNTRU is pseudorandom. Finally, the GGHLM19 scheme is semantically secure (with a certain error distribution) if MiNTRU$^s$ is pseudorandom:

**Lemma 9.2.** *Consider the error distribution $\psi[E, Y] = \{ER \mid R \leftarrow G^{-1}(Y)\}$. If MiNTRU$^s$ is pseudorandom, then the GGHLM19 encryption scheme using the error distribution $\psi[E, MG]$ is semantically secure.*

*Proof.* See [Gen+19, Proposition 4.2]. □

We can extend the GGHLM19 matrix FHE scheme to a matrix encoding scheme analogously to what we did with the HAO15 matrix FHE scheme in Section 9.2.2.

**Definition 9.5** (DFA Security for GGHLM19)**.** *Consider the GGHLM19 matrix graded encoding scheme with security parameter $\lambda \in \mathbb{N}$ for a matrix dimension $r \in \mathbb{N}$. Let $D$ be a distribution over $\mathcal{M}_r \times \{0,1\}^{poly(\lambda)}$ where $\mathcal{M}_r$ is a family of sequences of matrices $(M_\sigma)_{\sigma\in\Sigma}$ over $\{0,1\}^{r\times r}$.*

*Let $((M_\sigma)_{\sigma\in\Sigma}, \alpha) \leftarrow D$ and $(M'_\sigma)_{\sigma\in\Sigma}$ be a sequence of matrices in $\{0,1\}^{r\times r}$ such that $M_\sigma - M'_\sigma$ is all zeroes apart from a single $\pm 1$ in some row but not the last row. Consider, for all $\sigma \in \Sigma$, the GGHLM19 encodings $C_\sigma$ of $M_\sigma$, $C'_\sigma$ of $M'_\sigma$, and $c$ and $c'$ of the canonical basis vector $e_1$ under a secret key $S$ such that*

$$C_\sigma = S^{-1}(M_\sigma SG + E),\ c = S^{-1}(\beta e_1 + e),$$
$$C'_\sigma = S^{-1}(M'_\sigma SG + E'),\ c' = S^{-1}(\beta e_1 + e'),$$

*where $E, E'$ and $e, e'$ are error matrices and error vectors, respectively. c We say that GGHLM19 satisfies* DFA security *for $D$ if the following two distributions are computationally indistinguishable:*

$$(s_r, (C_\sigma)_{\sigma\in\Sigma}, c, \alpha) \overset{c}{\approx} (s_r, (C'_\sigma)_{\sigma\in\Sigma}, c', \alpha),$$

*where $s_r$ is the last row of the secret key $S$.*

## 9.8 General Encoding Schemes

In Section 9.5 we gave a specialised construction based on our extension of the HAO15 matrix FHE scheme (recall Section 9.2.2). In doing so, we were able to give a security reduction from VBB and perfect circuit-hiding to the decisional assumption of Section 9.3. In this section we want to sketch a generic construction for obfuscated evasive DFAs from arbitrary matrix graded encoding schemes. We will refrain from giving a security reduction to a generic assumption. This should rather be investigated on a case-by-case basis.

We will assume that we are given a matrix graded encoding scheme (with maximal grading $\kappa$). The obfuscator runs the key generation algorithm such that the secret key **sk** allows to encode matrices at and between two subsequent levels. We require that the public key **pk** allows for zero-testing only at the second level.

The obfuscator takes as an input an evasive DFA represented by the transition matrices $(M_\sigma)_{\sigma\in\Sigma}$ and outputs the tuple $(s_r, (C_\sigma)_{\sigma\in\Sigma}, c)$. In the output tuple, $c$ is an encoding of the first canonical basis vector $e_1$ at the first level and $z$ is an encoding of the $r$-th canonical basis vector $e_r$ at the second level, $(C_\sigma)_{\sigma\in\Sigma}$ is the sequence of encodings of the state transition matrices $(M_\sigma)_{\sigma\in\Sigma}$ at the first level and $C_f$ is an encoding of the final matrix $M_f$ between the first and second level. We assume that the initial and accepting state of the finite automaton are given by the state 1 and state $r$, respectively. If necessary, transform the DFA matrices $(M_\sigma)_{\sigma\in\Sigma}$ to satisfy the safeguard requirements described in Section 9.4.1. See Algorithm 9.4 for an algorithmic description.

We may evaluate the obfuscated automaton on a word $w \in \Sigma^*$ with $|w| \le \kappa$ as follows:

1. Compute the encoded vector $c_w$ corresponding to $(\prod_{i=1}^{|w|} M_{w_i})e_1$ using the sequence $(C_\sigma)_{\sigma\in\Sigma}$ and the encoded initial state $c$.

2. Evaluate the zero-test using **pk** on $C_f c_w - z$. The word $w$ is accepted by the automaton represented by $(M_\sigma)_{\sigma\in\Sigma}$ if the zero-test succeeds and we output 1 in this case, 0 otherwise.

See Algorithm 9.5 for an algorithmic description.

We argue that one should consider VBB or perfect circuit-hiding obfuscation instead of iO for evasive finite automata. One important reason is the possibility of *zeroising attacks*. This class

**Algorithm 9.4** Encoding (Obfuscating the finite automaton)

**procedure** ENCODE($(M_\sigma)_{\sigma\in\Sigma}$)
Run matrix graded encoding scheme key generation and obtain **sk**, **pk**.
Compute $(C_\sigma)_{\sigma\in\Sigma}$ by encoding $(M_\sigma)_{\sigma\in\Sigma}$ at the first level using **sk** (no zero-testing possible).
Compute $C_f$ by encoding $M_f$ at the second level using **sk** (zero-testing possible).
Compute state vectors $c$ and $z$ by encoding $e_1$ at the first and $e_r$ at the second level using **sk**, respectively.
**return** (**pk**, $(C_\sigma)_{\sigma\in\Sigma}$, $c$, $C_f$, $z$)
**end procedure**

**Algorithm 9.5** Evaluation (Executing the obfuscated program)

**procedure** EVALUATE($s_r$, $(C_\sigma)_{\sigma\in\Sigma}$, $c$; $w \in \Sigma^*$)
Initialise the state vector $s = c$.
**for all** $i = 1, \ldots, |w|$ **do**
Update the state vector $s = C_{w_i} s$.
**end for**
Evaluate the zero-test using **pk** on $C_f s - z$.
**return** 1 **if** the zero-test failed **else** 0
**end procedure**

of attacks affects several obfuscation constructions based on graded encoding schemes. The idea is that given an encoding of zero, we get a system of equations over $\mathbb{Z}$ instead of $\mathbb{Z}/q\mathbb{Z}$ that depend not only on small error terms but also on the secret matrices themselves. This seems to be especially problematic for iO schemes which are the prevalent constructions using graded encoding schemes.

# 10 Conclusion and Future Directions

And thus comes to a close our exciting voyage through the world of (special purpose) program obfuscation. We salute the reader for having stuck with us this far. During this (hopefully enjoyable) journey, we have seen that there is no singular, true notion of security for a program obfuscator. Instead, depending on the context, we should choose between VBB obfuscation, input-hiding obfuscation, perfect circuit-hiding obfuscation, or indistinguishability obfuscation. We have also seen that special purpose program obfuscation is somewhat easier to realise, especially for the class of evasive functions and programs. For evasive programs, we saw that some of the security notions imply each other.

Problems from number theory gave us inspiration for obfuscation schemes for Hamming distance testing and conjunctions. For a fixed dimension $n \in \mathbb{N}$, the problem of testing whether a vector $y \in \{0,1\}^n$ is contained in a Hamming ball $B_{H,r}(x)$ of radius $r \in \mathbb{N}$ around a target vector $x \in \{0,1\}^n$ can be reformulated to factoring $XY^{-1} = \prod_{i=1}^{n} p_i^{x_i - y_i} \mod q$ for appropriately chosen primes $q$ and $(p_i)_{i=1,\dots,n}$. This gave us an encoding of the target vector in the form of $X$, hidden by the hardness of the modular subset product problem.

We have constructed an obfuscator for evasive DFAs by using ideas from lattice cryptography. Given a representation of a DFA in the form of a set of transition matrices $(M_\sigma)_{\sigma\in\Sigma}$, we use a matrix encoding scheme to obtain encodings $(C_\sigma)_{\sigma\in\Sigma}$, which hide the transition matrices, such that $SC_\sigma = M_\sigma SG + E$. Using the multiplicative homomorphism of the encoding scheme and a zero-testing functionality, we can then evaluate the hidden DFA on an input word $w \in \Sigma^*$ (as long as the input length $|w|$ is smaller than some predetermined threshold $\kappa$) and obtain a correct accept/reject answer.

But there are still more open questions and problems left. It would be valuable to attempt a rigorous search-to-decision reduction of the modular subset product problem in the low-density case. We would also be interested in how exactly non-uniform high entropy distributions affect the hardness of MSP. This is an important question related to biometric matching. We would also like to see other cryptography schemes based on MSP, even though it seems strongly connected to the Hamming distance testing problem.

One last hard problem which begs further analysis, and which has already had countless hours of research dedicated towards, is to find a true algebraic multilinear map with a hard MDDH problem. Such a primitive would not only revolutionise the work on cryptographic program obfuscation, but we expect would greatly impact the whole field of cryptography itself.

# Bibliography


[14] *April 2014 Web Server Survey*. https://news.netcraft.com/archives/2014/04/02/april-2014-web-server-survey.html. Accessed: 2020-05-08. 2014.

[ABD16] Martin Albrecht, Shi Bai and Léo Ducas. 'A Subfield Lattice Attack on Overstretched NTRU Assumptions'. In: *CRYPTO 2016*. Springer, 2016, pp. 153–178.

[Ajt99] Miklós Ajtai. 'Generating Hard Instances of the Short Basis Problem'. In: *ICALP 1999*. Springer, 1999, pp. 1–9.

[Ana+16a] Prabhanjan Ananth, Aayush Jain, Moni Naor, Amit Sahai and Eylon Yogev. 'Universal Constructions and Robust Combiners for Indistinguishability Obfuscation and Witness Encryption'. In: *CRYPTO 2016*. Springer, 2016, pp. 491–520.

[Ana+16b] Prabhanjan Ananth, Aayush Jain, Moni Naor, Amit Sahai and Eylon Yogev. *Universal Obfuscation and Witness Encryption: Boosting Correctness and Combining Security*. Cryptology ePrint Archive, Report 2016/281. https://eprint.iacr.org/2016/281. 2016.

[Arb02] Genevieve Arboit. 'A method for watermarking java programs via opaque predicates'. In: *ICECR-5*. 2002, pp. 102–110.

[AS92] Noga Alon and Joel H Spencer. *The Probabilistic Method*. Wiley, 1992.

[Bal+97] José L. Balcázar, Josep Díaz, Ricard Gavaldà and Osamu Watanabe. 'Algorithms for Learning Finite Automata from Queries: A Unified View'. In: *Advances in Algorithms, Languages, and Complexity*. Springer, 1997, pp. 53–72.

[Ban+16] Sebastian Banescu, Christian Collberg, Vijay Ganesh, Zack Newsham and Alexander Pretschner. 'Code Obfuscation against Symbolic Execution Attacks'. In: *ACSAC 2016*. ACM, 2016, pp. 189–200.

[Bar+01] Boaz Barak, Oded Goldreich, Rusell Impagliazzo, Steven Rudich, Amit Sahai, Salil Vadhan and Ke Yang. 'On the (Im)possibility of Obfuscating Programs'. In: *CRYPTO 2001*. Springer. 2001, pp. 1–18.

[Bar+12] Boaz Barak, Oded Goldreich, Russell Impagliazzo, Steven Rudich, Amit Sahai, Salil Vadhan and Ke Yang. 'On the (Im)Possibility of Obfuscating Programs'. In: *J. ACM* 59.2 (2012).

[Bar+14a] Boaz Barak, Nir Bitansky, Ran Canetti, Yael Tauman Kalai, Omer Paneth and Amit Sahai. 'Obfuscation for Evasive Functions'. In: *TCC 2014*. Springer. 2014, pp. 26–51.

[Bar+14b] Boaz Barak, Sanjam Garg, Yael Tauman Kalai, Omer Paneth and Amit Sahai. 'Protecting Obfuscation against Algebraic Attacks'. In: *EUROCRYPT 2014*. Springer, 2014, pp. 221–238.

[Bar+19] James Bartusek, Tancrède Lepoint, Fermi Ma and Mark Zhandry. 'New Techniques for Obfuscating Conjunctions'. In: *EUROCRYPT 2019*. Springer, 2019, pp. 636–666.

[Bar89] David A. Barrington. 'Bounded-width polynomial-size branching programs recognize exactly those languages in $NC^1$.' In: *J. Comput. Syst. Sci.* 38.1 (1989), pp. 150–164.

[BCJ11] Anja Becker, Jean-Sébastien Coron and Antoine Joux. 'Improved Generic Algorithms for Hard Knapsacks'. In: *EUROCRYPT 2011*. Springer, 2011, pp. 364–385.

[Bec+] Anja Becker, Léo Ducas, Nicolas Gama and Thijs Laarhoven. 'New directions in nearest neighbor searching with applications to lattice sieving'. In: *Proceedings of the 2016 Annual ACM-SIAM Symposium on Discrete Algorithms*, pp. 10–24.

[Bio+17] Fabrizio Biondi, Sébastien Josse, Axel Legay and Thomas Sirvent. 'Effectiveness of Synthesis in Concolic Deobfuscation'. In: *Computers & Security* 70 (2017), pp. 500–515.

[Bis+18] Allison Bishop, Lucas Kowalczyk, Tal Malkin, Valerio Pastro, Mariana Raykova and Kevin Shi. 'A Simple Obfuscation Scheme for Pattern-Matching with Wildcards'. In: *CRYPTO 2018*. Springer, 2018, pp. 731–752.

[Bit+14] Nir Bitansky, Ran Canetti, Henry Cohn, Shafi Goldwasser, Yael Tauman Kalai, Omer Paneth and Alon Rosen. 'The Impossibility of Obfuscation with Auxiliary Input or a Universal Simulator'. In: *CRYPTO 2014*. Springer, 2014, pp. 71–89.

[Boy04] Xavier Boyen. 'Reusable cryptographic fuzzy extractors'. In: *CCS 2004*. ACM. 2004, pp. 82–91.

[BP12] Nir Bitansky and Omer Paneth. 'Point Obfuscation and 3-Round Zero-Knowledge'. In: *TCC 2012*. Springer, 2012, pp. 190–208.

[BR14] Zvika Brakerski and Guy N. Rothblum. 'Virtual Black-Box Obfuscation for All Circuits via Generic Graded Encoding'. In: *TCC 2014*. Springer, 2014, pp. 1–25.

[BR17] Zvika Brakerski and Guy N Rothblum. 'Obfuscating Conjunctions'. In: *Journal of Cryptology* 30.1 (2017), pp. 289–320.

[Bra+16] Zvika Brakerski, Vinod Vaikuntanathan, Hoeteck Wee and Daniel Wichs. 'Obfuscating Conjunctions under Entropic Ring LWE'. In: *ITCS 2016*. ACM. 2016, pp. 147–156.

[Bri+08] Julien Bringer, Hervé Chabanne, Gerard Cohen, Bruno Kindarji and Gilles Zemor. 'Theoretical and Practical Boundaries of Binary Secure Sketches'. In: *IEEE Transactions on Information Forensics and Security* 3.4 (2008), pp. 673–683.

[BS03] Dan Boneh and Alice Silverberg. 'Applications of multilinear forms to cryptography'. In: *Contemporary Mathematics* 324.1 (2003), pp. 71–90.

[BS16] Mihir Bellare and Igors Stepanovs. 'Point-Function Obfuscation: A Framework and Generic Constructions'. In: *TCC 2016*. 2016, pp. 565–594.

[Buc88] Johannes Buchmann. 'A Subexponential algorithm for the determination of class groups and regulators of algebraic number fields'. In: *Séminaire de théorie des nombres, Paris* 1989.1990 (1988), pp. 27–41.

[Can97] Ran Canetti. 'Towards Realizing Random Oracles: Hash Functions That Hide All Partial Information'. In: *CRYPTO 1997*. Springer, 1997, pp. 455–469.

[Cas59] John William Scott Cassels. *An introduction to the geometry of numbers*. Springer, 1959.

[CDE08] Cristian Cadar, Daniel Dunbar and Dawson Engler. 'KLEE: Unassisted and Automatic Generation of High-Coverage Tests for Complex Systems Programs'. In: *OSDI 2008*. USENIX Association, 2008, pp. 209–224.

[Che+15] Jung Hee Cheon, Kyoohyung Han, Changmin Lee, Hansol Ryu and Damien Stehlé. 'Cryptanalysis of the Multilinear Map over the Integers'. In: *EUROCRYPT 2015*. Springer, 2015, pp. 3–12.

[Che+16] Jung Hee Cheon, Pierre-Alain Fouque, Changmin Lee, Brice Minaud and Hansol Ryu. 'Cryptanalysis of the New CLT Multilinear Map over the Integers'. In: *EUROCRYPT 2016*. Springer, 2016, pp. 509–536.

[Che+18] Cheng Chen, Nicholas Genise, Daniele Micciancio, Yuriy Polyakov and Kurt Rohloff. *Implementing Token-Based Obfuscation under (Ring) LWE*. Cryptology ePrint Archive, Report 2018/1222. https://eprint.iacr.org/2018/1222. 2018.

[Che+52] Herman Chernoff et al. 'A Measure of Asymptotic Efficiency for Tests of a Hypothesis Based on the Sum of Observations'. In: *The Annals of Mathematical Statistics* 23.4 (1952), pp. 493–507.

[Cho+01] Stanley Chow, Yuan Gu, Harold Johnson and Vladimir A. Zakharov. 'An Approach to the Obfuscation of Control-Flow of Sequential Computer Programs'. In: *ISC 2001*. Springer, 2001, pp. 144–155.

[CJL16] Jung Hee Cheon, Jinhyuck Jeong and Changmin Lee. 'An algorithm for NTRU problems and cryptanalysis of the GGH multilinear map without a low-level encoding of zero'. In: *LMS J. Comput. Math.* 19.A (2016), pp. 255–266.

[CLS06] Scott Contini, Arjen K. Lenstra and Ron Steinfeld. 'VSH, an Efficient and Provable Collision-Resistant Hash Function'. In: *EUROCRYPT 2006*. Springer, 2006, pp. 165–182.

[CLT13] Jean-Sébastien Coron, Tancrède Lepoint and Mehdi Tibouchi. 'Practical Multilinear Maps over the Integers'. In: *CRYPTO 2013*. Springer, 2013, pp. 476–493.

[CLT15] Jean-Sébastien Coron, Tancrède Lepoint and Mehdi Tibouchi. 'New Multilinear Maps Over the Integers'. In: *CRYPTO 2015*. Springer, 2015, pp. 267–286.

[CN09] Christian Collberg and Jasvir Nagra. *Surreptitious Software: Obfuscation, Watermarking, and Tamperproofing for Software Protection*. Addison-Wesley, 2009.

[Cor+16] Jean-Sébastien Coron, Moon Sung Lee, Tancrède Lepoint and Mehdi Tibouchi. 'Cryptanalysis of GGH15 Multilinear Maps'. In: *CRYPTO 2016*. Springer, 2016, pp. 607–628.

[Cos+92] Matthijs J Coster, Antoine Joux, Brian A LaMacchia, Andrew M Odlyzko, Claus-Peter Schnorr and Jacques Stern. 'Improved low-density subset sum algorithms'. In: *Computational Complexity* 2.2 (1992), pp. 111–128.

[Cre+16] Giovanni Di Crescenzo, Lisa Bahler, Brian A. Coan, Yuriy Polyakov, Kurt Rohloff and David Bruce Cousins. 'Practical implementations of program obfuscators for point functions'. In: *HPCS 2016*. IEEE, 2016, pp. 460–467.

[CRV10] Ran Canetti, Guy N Rothblum and Mayank Varia. 'Obfuscation of Hyperplane Membership'. In: *TCC 2010*. Springer. 2010, pp. 72–89.

[CTL] Christian Collberg, Clark Thomborson and Douglas Low. 'Manufacturing Cheap, Resilient, and Stealthy Opaque Constructs'. In: *POPL 1998*. ACM, pp. 184–196.

[CTL97] Christian Collberg, Clark Thomborson and Douglas Low. *A taxonomy of Obfuscating Transformations*. Tech. rep. Department of Computer Science, The University of Auckland, New Zealand, 1997.

[CVW18] Yilei Chen, Vinod Vaikuntanathan and Hoeteck Wee. 'GGH15 Beyond Permutation Branching Programs: Proofs, Attacks, and Candidates'. In: *CRYPTO 2018*. Springer, 2018, pp. 577–607.

[Dal+06] Mila Dalla Preda, Matias Madou, Koen De Bosschere and Roberto Giacobazzi. 'Opaque Predicates Detection by Abstract Interpretation'. In: *AMAST 2006*. Springer, 2006, pp. 81–95.

[Des+18] Nicolas Desmoulins, Pierre-Alain Fouque, Cristina Onete and Olivier Sanders. 'Pattern Matching on Encrypted Streams'. In: *ASIACRYPT 2018*. Springer, 2018, pp. 121–148.

[Dix70] John D Dixon. 'The number of steps in the Euclidean algorithm'. In: *Journal of number theory* 2.4 (1970), pp. 414–422.

[Dod+08] Yevgeniy Dodis, Rafail Ostrovsky, Leonid Reyzin and Adam Smith. 'Fuzzy Extractors: How to Generate Strong Keys from Biometrics and Other Noisy Data'. In: *SIAM Journal On Computing* 38.1 (2008), pp. 97–139.

[DP19] Léo Ducas and Cécile Pierrot. 'Polynomial time bounded distance decoding near Minkowski's bound in discrete logarithm lattices'. In: *Designs, Codes and Cryptography* 87.8 (2019), pp. 1737–1748.

[DRS04] Yevgeniy Dodis, Leonid Reyzin and Adam D. Smith. 'Fuzzy Extractors: How to Generate Strong Keys from Biometrics and Other Noisy Data'. In: *EUROCRYPT 2004*. Springer, 2004, pp. 523–540.

[DS05] Yevgeniy Dodis and Adam Smith. 'Correcting Errors Without Leaking Partial Information'. In: *STOC 2005*. ACM. 2005, pp. 654–663.

[FA91] Daniel C. Fielder and Cecil O. Alford. 'Pascal's Triangle: Top Gun or Just One of the Gang?' In: *Applications of Fibonacci Numbers*. Springer, 1991, pp. 77–90.

[FMR13] Benjamin Fuller, Xianrui Meng and Leonid Reyzin. 'Computational Fuzzy Extractors'. In: *ASIACRYPT 2013*. Springer, 2013, pp. 174–193.

[FRS16] Benjamin Fuller, Leonid Reyzin and Adam Smith. 'When are Fuzzy Extractors Possible?' In: *ASIACRYPT 2016*. Springer. 2016, pp. 277–306.

[Gal12] Steven D Galbraith. *Mathematics of Public Key Cryptography*. Cambridge University Press, 2012.

[Gar+13] Sanjam Garg, Craig Gentry, Shai Halevi, Mariana Raykova, Amit Sahai and Brent Waters. 'Candidate Indistinguishability Obfuscation and Functional Encryption for all Circuits'. In: *FOCS 2013*. IEEE. 2013, pp. 40–49.

[GCT05] Jun Ge, Soma Chaudhuri and Akhilesh Tyagi. 'Control Flow Based Obfuscation'. In: *DRM 2005*. ACM, 2005, pp. 83–92.

[Gen+19] Nicholas Genise, Craig Gentry, Shai Halevi, Baiyu Li and Daniele Micciancio. 'Homomorphic Encryption for Finite Automata'. In: *ASIACRYPT 2019*. Springer, 2019, pp. 473–502.

[GG10] Yoann Guillot and Alexandre Gazet. 'Automatic binary deobfuscation'. In: *Journal in computer virology* 6.3 (2010), pp. 261–276.

[GGH13] Sanjam Garg, Craig Gentry and Shai Halevi. 'Candidate multilinear maps from ideal lattices'. In: *EUROCRYPT 2013*. Springer. 2013, pp. 1–17.

[GGH15] Craig Gentry, Sergey Gorbunov and Shai Halevi. 'Graph-induced multilinear maps from lattices'. In: *Theory of Cryptography Conference*. Springer. 2015, pp. 498–527.

[Gie01] Oliver Giel. 'Branching Program Size Is Almost Linear in Formula Size'. In: *Journal of Computer and System Sciences* 63.2 (2001), pp. 222–235.

[GKW17] Rishab Goyal, Venkata Koppula and Brent Waters. 'Lockable Obfuscation'. In: *FOCS 2017*. IEEE, 2017, pp. 612–621.

[Gol+10] Shafi Goldwasser, Yael Kalai, Chris Peikert and Vinod Vaikuntanathan. 'Robustness of the Learning with Errors Assumption'. In: *Innovations in Computer Science*. 2010, pp. 230–240.

[Gol+13] Shafi Goldwasser, Yael Kalai, Raluca Ada Popa, Vinod Vaikuntanathan and Nickolai Zeldovich. 'Reusable garbled circuits and succinct functional encryption'. In: *STOC 2013*. ACM. 2013, pp. 555–564.

[GSW13] Craig Gentry, Amit Sahai and Brent Waters. 'Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based'. In: *CRYPTO 2013*. Springer, 2013, pp. 75–92.

[GZ19] Steven D. Galbraith and Lukas Zobernig. 'Obfuscated Fuzzy Hamming Distance and Conjunctions from Subset Product Problems'. In: *TCC 2019*. Springer, 2019, pp. 81–110.

[GZ20] Steven D. Galbraith and Lukas Zobernig. 'Obfuscating Finite Automata'. In: *SAC 2020*. Springer, 2020, forthcoming.

[Hal+17] Shai Halevi, Tzipora Halevi, Victor Shoup and Noah Stephens-Davidowitz. 'Implementing BP-Obfuscation Using Graph-Induced Encoding.' In: *IACR Cryptology ePrint Archive* 2017 (2017), p. 104.

[HAO15] Ryo Hiromasa, Masayuki Abe and Tatsuaki Okamoto. 'Packing Messages and Optimizing Bootstrapping in GSW-FHE'. In: *PKC 2015*. Springer, 2015, pp. 699–715.

[Hen94] Doug Hensley. 'The Number of Steps in the Euclidean Algorithm'. In: *Journal of Number Theory* 49.2 (1994), pp. 142–182.

[HJ16] Yupu Hu and Huiwen Jia. 'Cryptanalysis of GGH Map'. In: *EUROCRYPT 2016*. Springer, 2016, pp. 537–565.

[HM18] Alexander Helm and Alexander May. 'Subset Sum Quantumly in $1.17^n$'. In: *TQC 2018*. Vol. 111. Leibniz International Proceedings in Informatics (LIPIcs). Leibniz-Zentrum fuer Informatik, 2018, 5:1–5:15.

[HPS14] J. Hoffstein, J. Pipher and J.H. Silverman. *An Introduction to Mathematical Cryptography*. second. Undergraduate Texts in Mathematics. Springer, 2014.

[Hur91] Adolf Hurwitz. 'Über die angenäherte Darstellung der Irrationalzahlen durch rationale Brüche'. In: *Mathematische Annalen* 39.2 (1891), pp. 279–284.

[Hut+17] C. Huth, D. Becker, J. G. Merchan, P. Duplys and T. Güneysu. 'Securing Systems With Indispensable Entropy: LWE-Based Lossless Computational Fuzzy Extractor for the Internet of Things'. In: *IEEE Access* 5 (2017), pp. 11909–11926.

[HW75] G. H. Hardy and E. M. Wright. *An Introduction to the Theory of Numbers*. Fourth. Oxford, 1975.

[IN96] Russell Impagliazzo and Moni Naor. 'Efficient cryptographic schemes provably as secure as subset sum'. In: *Journal of cryptology* 9.4 (1996), pp. 199–216.

[Jun+15] P. Junod, J. Rinaldini, J. Wehrli and J. Michielin. 'Obfuscator-LLVM – Software Protection for the Masses'. In: *2015 IEEE/ACM 1st International Workshop on Software Protection*. 2015, pp. 3–9.

[KC16] Koray Karabina and Onur Canpolat. 'A new cryptographic primitive for noise tolerant template security'. In: *Pattern Recognition Letters* 80 (2016), pp. 70–75.

[Kil88] Joe Kilian. 'Founding Crytpography on Oblivious Transfer'. In: *STOC 1988*. ACM, 1988, pp. 20–31.

[KRS94a] Jens Knoop, Oliver Rüthing and Bernhard Steffen. 'Optimal code motion: Theory and practice'. In: *TOPLAS 1194* 16.4 (1994), pp. 1117–1155.

[KRS94b] Jens Knoop, Oliver Rüthing and Bernhard Steffen. 'Partial Dead Code Elimination'. In: vol. 29. 6. ACM, 1994, pp. 147–158.

[Kuz+07] Nikolay Kuzurin, Alexander Shokurov, Nikolay Varnovsky and Vladimir Zakharov. 'On the Concept of Software Obfuscation in Computer Security'. In: *ISC 2007*. Springer, 2007, pp. 281–298.

[Lan02] Serge Lang. *Algebra*. Vol. 211. Springer, 2002.

[Las00] A Lasjaunias. 'A Survey of Diophantine Approximation in Fields of Power Series'. In: *Monatshefte für Mathematik* 130.3 (2000), pp. 211–229.

[Leg98] A. M. Legendre. *Essai sur la théorie des nombres*. Paris, 1798.

[LLL82] Arjen Klaas Lenstra, Hendrik Willem Lenstra and László Lovász. 'Factoring polynomials with rational coefficients'. In: *Mathematische Annalen* 261.4 (1982), pp. 515–534.

[LO85] Jeffrey C Lagarias and Andrew M Odlyzko. 'Solving Low-Density Subset Sum Problems'. In: *Journal of the ACM* 32.1 (1985), pp. 229–246.

[LPS04] Ben Lynn, Manoj Prabhakaran and Amit Sahai. 'Positive Results and Techniques for Obfuscation'. In: *EUROCRYPT 2004*. Springer, 2004, pp. 20–39.

[LSM06] Qiming Li, Yagiz Sutcu and Nasir Memon. 'Secure Sketch for Biometric Templates'. In: *ASIACRYPT 2016*. Springer. 2006, pp. 99–113.

[LW20] Changmin Lee and Alexandre Wallet. *Lattice analysis on MiNTRU problem*. Cryptology ePrint Archive, Report 2020/230. https://eprint.iacr.org/2020/230. 2020.

[Mal17] Francesca Malagoli. 'Continued fractions in function fields: polynomial analogues of McMullen's and Zaremba's conjectures'. PhD Thesis. 2017.

[MC06] Ginger Myles and Christian Collberg. 'Software watermarking via opaque predicates: Implementation, analysis, and attacks'. In: *Electronic Commerce Research* 6.2 (2006), pp. 155–171.

[Min+15] Jiang Ming, Dongpeng Xu, Li Wang and Dinghao Wu. 'LOOP: Logic-Oriented Opaque Predicate Detection in Obfuscated Binary Code'. In: *CCS 2015*. ACM, 2015, pp. 757–768.

[MM11] Daniele Micciancio and Petros Mol. 'Pseudorandom Knapsacks and the Sample Complexity of LWE Search-to-Decision Reductions'. In: *CRYPTO 2011*. Springer. 2011, pp. 465–484.

[MP12] Daniele Micciancio and Chris Peikert. 'Trapdoors for Lattices: Simpler, Tighter, Faster, Smaller.' In: *EUROCRYPT 2012*. Vol. 7237. Springer. 2012, pp. 700–718.

[MR09] Jean-Yves Marion and Daniel Reynaud. 'Dynamic binary instrumentation for deobfuscation and unpacking'. In: *In-Depth Security Conference*. 2009.

[MT06] Anirban Majumdar and Clark Thomborson. 'Manufacturing Opaque Predicates in Distributed Systems for Code Obfuscation'. In: *ACSC 2996*. Australian Computer Society, Inc., 2006, pp. 187–196.

[MVD06] M. Madou, L. Van Put and K. De Bosschere. 'Understanding Obfuscated Code'. In: *ICPC 2006*. IEEE, 2006, pp. 268–274.

[MZ18] Fermi Ma and Mark Zhandry. 'The MMap Strikes Back: Obfuscation and New Multilinear Maps Immune to CLT13 Zeroizing Attacks'. In: *TCC 2018*. Springer, 2018, pp. 513–543.

[Nac16] David Naccache. 'A Number-Theoretic Error-Correcting Code'. In: *SECITC 2015*. Springer. 2016, p. 25.

[Pal+00] Jens Palsberg, S. Krishnaswamy, Minseok Kwon, Di Ma, Qiuyun Shao and Y. Zhang. 'Experience with Software Watermarking'. In: *ACSAC 2000*. IEEE, 2000, pp. 308–316.

[PAS17] R. Krishna Ram Prakash, P. P. Amritha and M. Sethumadhavan. 'Opaque Predicate Detection by Static Analysis of Binary Executables'. In: *SSCC 2017*. Vol. 746. Communications in Computer and Information Science. Springer, 2017, pp. 250–258.

[PCH16] Andre Pawlowski, Moritz Contag and Thorsten Holz. 'Probfuscation: An Obfuscation Approach Using Probabilistic Control Flows'. In: *DIMVA 2016*. Springer, 2016, pp. 165–185.

[Pei09] Chris Peikert. 'Public-Key Cryptosystems from the Worst-Case Shortest Vector Problem: Extended Abstract'. In: *STOC 2009*. ACM, 2009, pp. 333–342.

[Pei16] Chris Peikert. 'A Decade of Lattice Cryptography'. In: *Found. Trends Theor. Comput. Sci.* 10.4 (2016), pp. 283–424.

[Pel19] Alice Pellet–Mary. 'On ideal lattices and the GGH13 multilinear map'. PhD Thesis. 2019.

[Reg05] Oded Regev. 'On Lattices, Learning with Errors, Random Linear Codes, and Cryptography'. In: *STOC 2005*. ACM, 2005, pp. 84–93.

[Rot13] Ron D Rothblum. 'On the Circular Security of Bit-Encryption'. In: *Theory of Cryptography*. Springer, 2013, pp. 579–598.

[SAB10] E. J. Schwartz, T. Avgerinos and D. Brumley. 'All You Ever Wanted to Know about Dynamic Taint Analysis and Forward Symbolic Execution (but Might Have Been Afraid to Ask)'. In: *S&P 2010*. IEEE, 2010, pp. 317–331.

[Sha+09] Monirul Sharif, Andrea Lanzi, Jonathon Giffin and Wenke Lee. 'Automatic Reverse Engineering of Malware Emulators'. In: *S&P 2009*. IEEE, 2009, pp. 94–109.

[Sho09] Victor Shoup. *A Computational Introduction to Number Theory and Algebra*. Cambridge university press, 2009.

[SLM07] Yagiz Sutcu, Qiming Li and Nasir Memon. 'Protecting Biometric Templates With Sketch: Theory and Practice'. In: *IEEE Transactions on Information Forensics and Security* 2.3 (2007), pp. 503–512.

[SS15] Florent Saudel and Jonathan Salwan. 'Triton: A dynamic symbolic execution framework'. In: *Symposium sur la sécurité des technologies de l'information et des communications, SSTIC*. 2015, pp. 31–54.

[SS16] Brendan Sheridan and Micah Sherr. 'On Manufacturing Resilient Opaque Constructs Against Static Analysis'. In: *ESORICS 2016*. Springer, 2016, pp. 39–58.

[Ste92] Andreas Stein. 'Baby Step-Giant Step-Verfahren in reell-quadratischen Kongruenzfunktionenkörpern mit Charakteristik ungleich 2'. Diploma Thesis, Universität des Saarlandes, Saarbrücken. 1992.

[Sti09] Henning Stichtenoth. *Algebraic Function Fields and Codes*. Vol. 254. Springer, 2009.

[SW14] Amit Sahai and Brent Waters. 'How to Use Indistinguishability Obfuscation: Deniable Encryption, and More'. In: *STOC 2014*. ACM, 2014, pp. 475–484.

[Tuy+05] Pim Tuyls, Anton HM Akkermans, Tom AM Kevenaar, Geert-Jan Schrijen, Asker M Bazen and Raimond NJ Veldhuis. 'Practical Biometric Authentication with Template Protection'. In: *AVBPA 2005*. Springer. 2005, pp. 436–446.

[UDM05] Sharath K. Udupa, Saumya K. Debray and Matias Madou. 'Deobfuscation: Reverse Engineering Obfuscated Code'. In: *WCRE 2005*. IEEE, 2005, pp. 45–54.

[Wan+08] X. Wang, Y. Jhi, S. Zhu and P. Liu. 'STILL: Exploit Code Detection via Static Taint and Initialization Analyses'. In: *ACSAC 2008*. IEEE, 2008, pp. 289–298.

[Wee05] Hoeteck Wee. 'On Obfuscating Point Functions'. In: *STOC 2005*. ACM, 2005, pp. 523–532.

[WZ17a] Daniel Wichs and Giorgos Zirdelis. 'Obfuscating Compute-and-Compare Programs under LWE'. In: *FOCS 2017*. IEEE. 2017, pp. 600–611.

[WZ17b] Daniel Wichs and Giorgos Zirdelis. *Obfuscating Compute-and-Compare Programs under LWE*. Cryptology ePrint Archive, Report 2017/276. https://eprint.iacr.org/2017/276. 2017.

[XMW16] Dongpeng Xu, Jiang Ming and Dinghao Wu. 'Generalized Dynamic Opaque Predicates: A New Control Flow Obfuscation Method'. In: *ISC 2016*. Springer, 2016, pp. 323–342.

[Yad+15] Babak Yadegari, Brian Johannesmeyer, Ben Whitely and Saumya Debray. 'A Generic Approach to Automatic Deobfuscation of Executable Code'. In: *S&P 2015*. IEEE, 2015, pp. 674–691.

[YD15] Babak Yadegari and Saumya Debray. 'Symbolic Execution of Obfuscated Code'. In: *CCS 2015*. ACM, 2015, pp. 732–744.

[ZGR19] Lukas Zobernig, Steven D Galbraith and Giovanni Russello. 'When Are Opaque Predicates Useful?' In: *TrustCom/BigDataSE 2019*. IEEE. 2019, pp. 168–175.