Pseudorandom Functions with Structure: Extensions and Implications

by

Aloni Cohen

B.S., University of California, Berkeley (2012)

Submitted to the Department of Electrical Engineering and Computer Science in partial fulfillment of the requirements for the degree of

Masters of Science in Computer Science and Engineering

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2015

Author .... Signature redacted ....
Department of Electrical Engineering and Computer Science
May 20, 2015

Certified by .... Signature redacted ....
Shafi Goldwasser
RSA Professor of Electrical Engineering and Computer Science
Thesis Supervisor

Accepted by .... Signature redacted ....
Leslie A. Kolodziejski
Chair, Department Committee on Graduate Students

Pseudorandom Functions with Structure: Extensions and Implications

by

Aloni Cohen

Submitted to the Department of Electrical Engineering and Computer Science
on May 20, 2015, in partial fulfillment of the
requirements for the degree of
Masters of Science in Computer Science and Engineering

Abstract

In the first part of this work, we introduce a new type of pseudo-random function for which "aggregate queries" over exponential-sized sets can be efficiently answered. We show how to use algebraic properties of underlying classical pseudo random functions, to construct such "aggregate pseudo-random functions" for a number of classes of aggregation queries under cryptographic hardness assumptions. For example, one aggregate query we achieve is the product of all function values accepted by a polynomial-sized read-once boolean formula. On the flip side, we show that certain aggregate queries are impossible to support.

In the second part of this work, we show how various extensions of pseudo-random functions considered recently in the cryptographic literature, yield impossibility results for various extensions of machine learning models, continuing a line of investigation originated by Valiant and Kearns in the 1980s. The extended pseudo-random functions we address include constrained pseudo random functions, aggregatable pseudo random functions, and pseudo random functions secure under related-key attacks.

In the third part of this work, we demonstrate limitations of the recent notions of constrained pseudo-random functions and cryptographic watermarking schemes. Specifically, we construct pseudorandom function families that can be neither punctured nor watermarked. This is achieved by constructing new unobfuscatable pseudorandom function families for new ranges of parameters.

Thesis Supervisor: Shafi Goldwasser
Title: RSA Professor of Electrical Engineering and Computer Science

Acknowledgments

First and foremost, I would like to thank my advisor, Shafi Goldwasser, for her tireless mentorship. In our first meeting almost two years ago, Shafi enthusiastically and patiently introduced me to cryptography – to hybrids, pseudorandom functions, and indistinguishability. As I struggled to make sense of her scratched notes in the days after, I understood that I was at the threshold of a cavern filled with wonders I could not have imagined. Though I sometimes still struggle to make sense of Shafi's scratched notes, she continues to illuminate the cavern: making the wonders both more brilliant and more attainable.

I would also like to thank Vinod Vaikuntanathan, for his patience and diligence in all our collaborations. By now I should not be surprised when a marathon meeting leaves me exhausted, but his mind none the duller. I have also been fortunate enough to work, travel, and eat extensively with my fellow student Justin Holmgren who heeded Silvio's call to collaborate with his peers – catalyzing my (more probably both of our) progress.

Lastly, I could not forget the first two decades of my life, during which my family lay the foundation for any of my successes.

Contents

1 Introduction **1**

1.1 Aggregate Pseudorandom Functions 5

1.2 Augmented PRFs and Computational Learning 12

1.2.1 Constrained PRFs and Limits on Restriction Access Learnability . . . 13

1.2.2 New Learning Models Inspired by the Study of PRFs 14

1.3 New Negative Results from Unobfuscatable PRFs 18

2 Aggregate Pseudorandom Functions **23**

2.1 Definition and Generic Security 23

2.1.1 A General Security Theorem for Aggregate PRFs 26

2.1.2 Impossibility of Aggregate PRF for General Sets 28

2.2 Generic Construction for Interval Sets 29

2.3 Bit-Fixing Aggregate PRF from DDH 31

2.3.1 Construction 31

2.3.2 Security 33

2.4 Decision Trees 37

2.5 Read-once formulas 38

3 Connection to Learning **41**

3.1 Preliminaries 41

3.2 Membership Queries with Restriction Access 44

3.2.1 $\mathrm{MQ}_{\mathrm{RA}}$ Learning 45

3.2.2 Constrained PRFs 47

3.2.3 Hardness of Restriction Access Learning 48

3.3 Learning with Related Concepts 51

3.3.1 Related-Key Secure PRFs 52

3.3.2 Hardness of Related Concept Learning 53

3.4 Learning with Aggregate Queries 56

3.4.1 Hardness of Aggregate Query Learning 57

4 Unobfuscatable PRFs and Their Implications 61

4.1 Unpuncturable Pseudorandom Functions 61

4.2 Watermarking Schemes 62

4.2.1 Definition 63

4.3 Unwatermarkable Families 65

4.4 Construction of Unwatermarkable PRFs 69

4.4.1 Preliminaries 69

4.4.2 Construction 71

Chapter 1

Introduction

Pseudo-random functions (PRFs), introduced by Goldreich, Goldwasser and Micali [GGM86], are a family of indexed functions for which there exists a polynomial-time algorithm that, given an index (which can be viewed as a secret key) for a function, can evaluate it, but no probabilistic polynomial-time algorithm without the secret key can distinguish the function from a truly random function – even if allowed oracle query access to the function. Over the years, pseudo-random functions have been shown to be useful for numerous cryptographic applications. Interestingly, aside from their cryptographic applications, PRFs have also been used to show impossibility of computational learning in the membership queries model [Val84], and served as the underpinning of the proof of Razborov and Rudich [RR97] that natural proofs would not suffice for unrestricted circuit lower bounds.

Since their inception in the mid eighties, various *augmented* pseudo random functions with extra properties have been proposed, with both more structure and security against more sophisticated attacks. This was first done in the work of Goldreich, Goldwasser, and Nussboim [GGN10] on how to efficiently construct "huge objects" (e.g. a large graph implicitly described by access to its adjacency matrix) which maintain combinatorial properties expected of a *random* "huge object." Furthermore, they show several implementations of varying quality of such objects for which complex global properties can be computed, such as computing cliques in a random graph, computing random function inverses from a point in the range, and computing the parity of a random function's values over huge sets. More recently, further augmentations of PRFs have been proposed, including: the works on constrained PRFs[1] [KPTZ13, BGI14, BW13] which can release auxiliary secret keys whose knowledge enables computing the PRF in a restricted number of locations without compromising pseudo-randomness elsewhere; key-homomorphic PRFs [BLMR13] which are homomorphic with respect to the keys; and related-key secure PRFs [BC10, ABPP14]. These constructions yield fundamental objects with often surprising applications to cryptography and elsewhere. A case in point is the truly surprising use of constrained PRFs [SW14], to show that indistinguishability obfuscation can be used to resolve a long-standing problem of deniable encryption, among many others. Below we describe each of the three parts of this thesis, exploring different aspects of PRFs augmented to have extra properties. In the remainder of this chapter we provide an in-depth overview of our results and techniques.

Aggregate PRFs

In the first part of this thesis, we introduce a new type of augmented PRF which we call *aggregate pseudo random functions* (AGG-PRF). An AGG-PRF is a family of indexed functions each associated with a secret key, such that *given the secret key*, one can compute aggregates of the values of the function *over super-polynomially large sets* in *polynomial time*; and yet without the secret key, a polynomial time adversary (distinguisher) cannot distinguish the function from random, even when the adversary can make *aggregate queries*. The distinguisher can request and receive an aggregate of the function values over sets (of possibly super-polynomial size) that she can specify. Examples of aggregate queries can be the sum/product of all function values belonging to an exponential-sized interval, or more generally, the sum/product of all function values on points for which some polynomial time predicate holds. Since the sets over which our function values are aggregated are super-polynomial in size, they cannot be directly computed by simply querying the function on individual points. AGG-PRFs cast in the framework of [GGN10] are (truthful, pseudo) imple-

[1]Constrained PRFs are also known as Functional PRFs and as Delegatable PRFs.

mentations of random functions supporting aggregates as their "complex queries." Indeed, our first example of an AGG-PRF for computing parities over exponential-sized intervals comes from [GGN10] under the assumption that one-way functions exist.

We show AGG-PRFs under various cryptographic hardness assumptions (one-way functions and DDH) for a number of types of aggregation operators such as sums and products and for a number of set systems including intervals, hypercubes, and (the supports of) restricted computational models such as decision trees and read-once Boolean formulas. We also show negative results: there are no AGG-PRFs for more expressive set systems such as (the supports of) general CNF formulas.

Connections to Learning

In the second part of this thesis, we embark on a study of the connection between the augmented PRF constructions of recent years (constrained, related-key, aggregate) and the theory of computational learning. We recall at the outset that the fields of cryptography and machine learning share a curious historical relationship. The goals are in complete opposition and at the same time the aesthetics of the models, definitions, and techniques bear a striking similarity. For example, a cryptanalyst can attack a cryptosystem using a range of powers from only seeing ciphertext examples to requesting to see decryptions of ciphertexts of her choice. Analogously, machine learning allows different powers to the learner such as random examples versus membership queries, showing that certain powers allow learners to learn concepts in polynomial time whereas others do not. Even more directly, problems which pose challenges for machine learning, such as Learning Parity with Noise have been used as the underpinning for building secure cryptosystems. As mentioned above, [Val84] observes that the existence of PRFs in a complexity class $\mathcal{C}$ implies the existence of concept classes in $\mathcal{C}$ which can not be learned under membership queries, and [KV94] extends this direction to some public key constructions.

In the decades since the introduction of PAC learning, new computational learning models have been proposed, such as the recent "restriction access" model [DRWY12] which allows the learner to interact with the target concept by asking membership queries, but also to obtain an entire circuit that computes the concept on a random subset of the inputs. For example, in one shot, the learner can obtain a circuit that computes the concept class on all n-bit inputs that start with $n/2$ zeros. At the same time, the cryptographic research landscape has been swiftly moving in the direction of augmenting traditional PRFs and other cryptographic primitives to include higher functionalities. This brings to mind two natural questions:

- *Can one leverage augmented pseudo-random function constructions to establish limits on what can and cannot be learned in augmented machine learning models?*
- *Going even further afield, can augmented cryptographic constructs suggest interesting learning models?*

We address these questions in the second part of this thesis.

Unobfuscatable PRFs and Their Implications

In the third part of this thesis, we explore constructions of function families that are pseudorandom, but "learnable" in a non-black-box setting. This line of work was initiated in the context of program obfuscation by [BGI$^+$12], demonstrating the impossibility of achieving a natural notion of ideal obfuscation.

Their approach was to construct "unobfuscatable" function families: families of circuits which are not learnable by efficient black-box algorithms, but which become efficiently learnable with non-black-box access. In the latter setting, these functions are learnable in a very strong sense: given any circuit that equivalently computes the functionality, there is an efficient algorithm that reconstructs the original circuit. Intuitively, this demonstrates that the family is "unobfuscatable": from any exact implementation, the original circuit is recovered. That work also shows how to make such families that are pseudorandom.

More recently, Bitansky and Paneth [BP12] extend the techniques to work for any implementation agreeing on only a constant fraction, but weakening other properties of the earlier result. Specifically, these families are no longer pseudorandom, and are furthermore only very weakly non-black-box learnable. That work was motivated by applications to resettable zero knowledge protocols.

A natural goal is to interpolate between these two results: achieve a notion of obfuscation stronger than the former, while simultaneously maintaining its strong learnability properties. In this thesis, we construct new PRF families towards this goal, assuming one-way functions exist. As corollaries of our construction, we demonstrate that if PRFs exist at all, then:

- *There exist families of PRFs that cannot be constrained on all-but-one input (or equivalently, puncturable at one input).*
- *There exist families of "unwatermarkable"*[2] *pseudorandom functions, if a marked program must agree with the original on random inputs.*

The notion of cryptographic watermarking of circuits was recently considered in [CHV15], which constructs such a scheme for any PRF puncturable at one input. We explore the limitations of cryptographic watermarking for some notions of non-black-box learnability (in which the learner gets an implementation of the concept as input) and construct a new unobfuscatable family that is unwatermarkable.

1.1 Aggregate Pseudorandom Functions

Aggregate Pseudo Random Functions (AGG-PRF) are indexed families of pseudo-random functions for which a distinguisher (who runs in time polynomial in the security parameter) can request and receive the value of an aggregate (for example, the sum or the product) of the function values over certain large sets and yet cannot distinguish oracle access to the function from oracle access to a truly random function. At the same time, given the function index (in other words, the secret key), one can compute such aggregates over potentially super-polynomial size sets in polynomial time. Such an efficent aggregation algorithm cannot

[2]Cryptographic watermarking schemes are the subject of current, unpublished work of Justin Holmgren, Vinod Vaikuntanathan, and myself.

possibly exist for random functions. Thus, this is a PRF family that is very unlike random functions (in the sense of being able to efficiently aggregate over superpolynomial size sets), and yet is still computationally indistinguishable from random functions.

To make this notion precise, we need two ingredients. Let $\mathcal{F} = \{\mathcal{F}_\lambda\}_{\lambda>0}$ where each $\mathcal{F}_\lambda = \{f_k : \mathcal{D}_\lambda \to \mathcal{R}_\lambda\}_{k \in \mathcal{K}_\lambda}$ is a collection of functions on a domain $\mathcal{D}_\lambda$ to a range $\mathcal{R}_\lambda$, computable in time $\mathsf{poly}(\lambda)$.[3] The first ingredient is a collection of sets (also called a set system) $\mathcal{S} = \{S \subseteq \mathcal{D}\}$ over which the aggregates can be efficiently computed given the index k of the function. The second ingredient is an aggregation function $\Gamma : \mathcal{R}^* \to \{0,1\}^*$ which takes as input a tuple of function values $\{f(x) : x \in S\}$ for some set $S \in \mathcal{S}$ and outputs the aggregate $\Gamma(f(x_1), \dots, f(x_{|S|}))$.

The sets are typically super-polynomially large, but are efficiently recognizable. That is, for each set S, there is a corresponding $\mathsf{poly}(\lambda)$-size circuit C_S that takes as input an $x \in \mathcal{D}$ and outputs 1 if and only if $x \in S$. [4] Throughout this paper, we will consider relatively simple aggregate functions: we treat the range of the functions as an Abelian group, and will let Γ denote the group operation on its inputs. Note that the input to Γ is super-polynomially large (in the security parameter λ), making the aggregate computation non-trivial.

This family of functions, equipped with a set system $\mathcal{S}$ and an aggregation function Γ is called an aggregate PRF family (AGG-PRF) if the following two requirements hold:

1. *Aggregatability:* There exists a polynomial-time algorithm (in the security parameter λ) that given an index k to the PRF $f_k \in \mathcal{F}$ and a circuit C_S that recognizes a set $S \in \mathcal{S}$, can compute Γ over the PRF values $f_k(x)$ for all $x \in S$. That is, it can compute

$$AGG_{f_k,\Gamma}(S) := \Gamma_{x \in S}\ f_k(x)$$

2. *Pseudorandomness:* No polynomial-time distinguisher which can specify a set $S \in \mathcal{S}$ as a query, receiving as an answer either $AGG_{f_k,\Gamma}(S)$ for a random function $f_k \in \mathcal{F}$ or

[3] In this informal exposition, for the sake of brevity, we will sometimes omit the security parameter and refrain from referring to ensembles.

[4] All the sets we consider are efficiently recognizable, and we use the corresponding circuit as the representation of the set. We occasionally abuse notation and use S and C_S interchangeably.

$AGG_{h,\Gamma}(S)$ for a truly random functions h, can distinguish between the two cases.

We show a number of constructions of AGG-PRF for various set systems under different cryptographic assumptions. We describe our constructions below, starting from the least expressive set system.

Interval Sets

We first present AGG-PRFs over interval set systems with respect to aggregation functions that compute any group operation. The construction can be based on any (standard) PRF family.

Theorem 1.1.1 (Group summation over intervals, from one-way functions [GGN10]). [5] *Assume one-way functions exist. Then, there exists an AGG-PRF over $\mathbb{Z}_{2^\lambda}$, with respect to a collection of sets defined by intervals $[a, b] \subseteq \mathbb{Z}_{2^\lambda}$ and the aggregation function of addition over $\mathbb{Z}_{2^\lambda}$.*

The construction works as follows. Let $\{F_k : \mathbb{Z}_{2^\lambda} \to \mathbb{Z}_{2^\lambda}\}$ be a (standard) pseudo-random function family based on the existence of one-way functions [GGM86, HILL99]. Construct an AGG-PRF family G supporting efficient computation of group aggregation functions. Define

$$G_k(x) = F_k(x) - F_k(x-1)$$

To aggregate G, set

$$\sum_{x\in[a,b]} G_k(x) = F_k(b) - F_k(a-1)$$

Given k, this can be efficiently evaluated.

Another construction from [GGN10] achieves summation over the integers for PRFs whose range is $\{0, 1\}$. We omit the details of the construction, but state the theorem for completeness.

[5] Observed even earlier by Reingold and Naor and appeared in [GGI+02] in the context of small space streaming algorithms

Theorem 1.1.2 (Integer summation over intervals, from one-way functions [GGN10]). *Assume one-way functions exist. Then, there exists an AGG-PRF family that maps $\mathbb{Z}_{2^\lambda}$ to $\{0,1\}$, with respect to a collection of sets defined by intervals $[a,b] \subseteq \mathbb{Z}_{2^\lambda}$ and the aggregation function computing the summation over $\mathbb{Z}$.*

Hypercubes

We next construct AGG-PRFs over hypercube set systems. This partially addresses Open Problem 5.4 posed in [GGN10], whether one can efficiently implement a random function with range $\{0,1\}$ with complex queries that compute parities over the function values on hypercubes. Assuming DDH hardness, Theorem 1.1.3 answers the question for products rather than parities for a function whose range is a DDH group.

Throughout this section, we take $\mathcal{D}_\lambda = \{0,1\}^\ell$ for some polynomial $\ell = \ell(\lambda)$. A hypercube $S_\mathbf{y}$ is defined by a vector $\mathbf{y} \in \{0,1,\star\}^\ell$ as

$$S_\mathbf{y} = \{\mathbf{x} \in \{0,1\}^\ell : \ \forall i, y_i = \star \text{ or } x_i = y_i\}$$

We present a construction under the DDH assumption.

Theorem 1.1.3 (Hypercubes from DDH). *Let $\mathcal{HC} = \{\mathcal{HC}_{\ell(\lambda)}\}_{\lambda>0}$ where $\mathcal{HC}_\ell = \{0,1,\star\}^\ell$ be the set of hypercubes on $\{0,1\}^\ell$, for ℓ polynomially bounded in λ. Then, there exists an AGG-PRF family mapping $\{0,1\}^\ell$ to a group G, supporting the set system $\mathcal{HC}$ with the product aggregation function, assuming DDH hardness on G.*

We sketch the construction from DDH below, using the Naor-Reingold PRF [NR04]. Namely, the function is parametrized by an ℓ-tuple $\vec{k} = (k_1, \dots, k_\ell)$ and is defined as

$$F_{\vec{k}}(x) = g^{\prod_{i:x_i=1} k_i}$$

Let us illustrate aggregation over the hypercube $\mathbf{y} = (1, 0, \star, \star, \dots, \star)$. To aggregate the function F, observe that

$$\begin{aligned}\prod_{\{x:\ x_1=1,x_2=0\}} F_{\vec{k}}(x) &= \prod_{\{x:\ x_1=1,x_2=0\}} g^{\prod_{i:x_i=1} k_i} \\ &= g^{\sum_{\{x:x_1=1,x_2=0\}} \prod_{i:x_i=1} k_i} \\ &= g^{(k_1)(1)(k_2+1)(k_3+1)\cdots(k_\ell+1)}\end{aligned}$$

which can be efficiently computed given $\vec{k}$.

Decision Trees

A decision tree T on ℓ variables is a binary tree where each internal node is labeled by a variable x_i, the leaves are labeled by either 0 or 1, one of the two outgoing edges of an internal node is labeled 0, and the other is labeled 1. Computation of a decision tree on an input $(x_1, \ldots, x_\ell)$ starts from the root, and at each internal node n, proceeds by taking either the 0-outgoing edge or 1-outgoing edge depending on whether $x_n = 0$ or $x_n = 1$, respectively. Finally, the output of the computation is the label of the leaf reached through this process. The size of a decision tree is the number of nodes in the tree.

A decision tree T defines a set $S = S_T = \{x \in \{0,1\}^\ell : T(x) = 1\}$. We show how to compute product aggregates over sets defined by polynomial size decision trees, under the DDH assumption.

The construction is simply a result of the observation that the set $S = S_T$ can be written as a disjoint union of polynomially many hypercubes. Computing aggregates over each hypercube and multiplying the results together gives us the decision tree aggregate.

Theorem 1.1.4 (Decision Trees from DDH). *Assuming the hardness of the decisional Diffie-Hellman problem on group G, there is an AGG-PRF mapping $\{0,1\}^\ell$ to G that supports aggregation over sets recognized by polynomial-size decision trees, for ℓ polynomially bounded in λ.*

Read-Once Boolean Formulas

Finally, we show a construction of AGG-PRF over read-once Boolean formulas, the most expressive of our set systems, under the subexponential DDH assumption. A read-once Boolean formula is a Boolean circuit composed of AND, OR and NOT gates with fan-out 1, namely each input literal feeds into at most one gate, and each gate output feeds into at most one other gate. A read-once formula can be written as a binary tree where each internal node is labeled with an AND or OR gate, and each literal (a variable or its negation) appears in at most one leaf.

Theorem 1.1.5 (Read-Once Boolean Formulas from DDH). *Under the subexponential decisional Diffie-Hellman assumption on group G, there is an AGG-PRF mapping $\{0,1\}^{\ell}$ to G that supports aggregation over sets recognized by read-once Boolean formulas, for ℓ polynomially bounded in λ.*

We compute aggregates by recursion on the levels of the formula.

Limits of Aggregation

A natural question to ask is whether one can support aggregation over sets defined by general circuits. It is however easy to see that you cannot support any class of circuits for which deciding satisfiability is hard (for example, AC^0) as follows. Suppose C is a circuit which is either unsatisfiable or has a unique SAT assignment. Solving satisfiability for such circuits is known to be sufficient to solve SAT in general [VV86]. The algorithm for SAT simply runs the aggregator with a random PRF key k, and outputs YES if and only if the aggregator returns a non-zero value. It is natural to assume that if the formula is unsatisfiable, we will always get 0 from the aggregator. Otherwise, we get $f_k(x)$, where x is the (unique) satisfying assignment. Now, this might end up being 0 accidentally, but cannot be 0 always since otherwise, we will turn it into a PRF distinguisher. The distinguisher has the satisfying assignment hardcoded into it non-uniformly, and it simply checks if $PRF_k(x)$ is 0.

Theorem 1.1.6 (Impossibility for General Set Systems). *Suppose there is an efficient algo-*

rithm which on an index for $f \in \mathcal{F}$, a set system defined by $\{x : C(x) = 1\}$ for a polynomial size Boolean circuit C, and an aggregation function Γ, outputs the $\Gamma_{x:C(x)=1} f(x)$. Suppose further that if C is unsatisfiable this algorithm returns 0. Then, there is efficient algorithm that takes circuits C as input and w.h.p. over its coins, decides satisfiability for C.

Open Questions. As discussed in the introduction, augmented pseudo-random functions often have powerful and surprising applications, perhaps the most recent example being constrained PRFs [BW13, KPTZ13, BGI14]. Perhaps the most obvious open question that emerges from this work is to find applications for aggregate PRFs. We remark that a primitive similar to aggregate PRFs was used in [BGV11] to construct delegation protocols.

In this work we restricted our attention to particular types of aggregation functions and subsets over which the aggregation takes place, although our definition captures more general scenarios. We looked at aggregation functions that compute group operations over Abelian groups. Can we support more general aggregation functions that are not restricted to group operations, for example the majority function, or even non-symmetric functions? We show positive results for intervals, hypercubes, and sets recognized by read-once formulas and decision trees. On the other hand, we show that it is unlikely that we can support general sets, for example sets recognized by CNF formulas. This almost closes the gap between what is possible and what is hard. A concrete open question in this direction is to construct an aggregate PRF computing summation over an Abelian group for sets recognized by DNFs, or provide evidence that this cannot be done.

Related Work to Aggregate PRFs

As described above, the work of [GGN10] studies the general question of how one can efficiently construct random, "close-to" random, and "pseudo-random" large objects, such as functions or graphs, which truthfully obey global combinatorial properties rather simply appearing to do so to a polynomial time observer.

Formally, using the [GGN10] terminology, a PRF is a *pseudo-implementation* of a random function, and an AGG-PRF is a pseudo-implementation of a "random function that also answers aggregate queries" (as we defined them). Furthermore, the aggregatability property of AGG-PRF implies it is a *truthful* pseudo-implementation of such a function. Whereas in this work, we restrict our attention to aggregate queries, [GGN10] considers other complex-queries, such as in the case of a uniformly selected N node graph, providing a clique of size $\log_2 N$ that contains the queried vertex in addition to answering adjacency queries.

Our notion of aggregate PRFs bears resemblance to the notion of algebraic PRFs defined in the work of Benabbas, Gennaro and Vahlis [BGV11], used for verifiable computation. There are two main differences. First, algebraic PRFs support efficient aggregation over very specific subsets, whereas our constructions of aggregate PRFs support expressive subset classes, such as subsets recognized by hypercubes, decision trees and read-once Boolean formulas. Secondly, in the security notion for aggregate PRFs, the adversary obtains access to an oracle that computes the function as well as one that computes the aggregate values over super-polynomial size sets, whereas in algebraic PRFs, the adversary is restricted to accessing the function oracle alone. Our constructions from DDH use an algebraic property of the Naor-Reingold PRF in a similar manner as in [BGV11].

1.2 Augmented PRFs and Computational Learning

As discussed above, connections between PRFs and learning theory date back to the 80's in the pioneering work of [Val84] showing that PRF in a complexity class C implies the existence of concept classes in C which can not be learned with membership queries. In the second part of this work, we study the implications of the slew of augmented PRF constructions of recent years [BW13, BGI14, KPTZ13, BC10, ABPP14] and our new aggregate PRF to computational learning.

1.2.1 Constrained PRFs and Limits on Restriction Access Learnability

Recently, Dvir, Rao, Wigderson, and Yehudayoff [DRWY12] introduced a new learning model where the learner is allowed non-black-box information on the computational device (such as circuits, DNFs, or formulas) that decides the concept; their learner receives a simplified device resulting from partial assignments to input variables (i.e. restrictions). These partial restrictions lie somewhere in between function evaluation (full restrictions) which correspond to learning with membership queries and the full description of the original device (the empty restriction). The work of [DRWY12] studies a PAC version of restriction access, called PAC_{RA}, where the learner receives the circuit restricted with respect to random partial assignments. They show that both decision trees and DNF formulas can be learned efficiently in this model. Indeed, the PAC_{RA} model is quite a powerful generalization of the traditional PAC learning model, as it returns to the learner a computational description of the simplified concept.

Yet, in this section we will show limitations of this computational model under cryptographic assumptions. We show that the *constrained pseudo-random function families* introduced recently in [BW13, BGI14, KPTZ13] naturally define a concept class which is not learnable by an even stronger variant of the restriction access learning model which we define. In the stronger variant, which we name *membership queries with restriction access (MQ_{RA})* the learner can adaptively specify any restriction of the circuit from a specified class of restrictions $\mathcal{S}$ and receive the simplified device computing the concept on this restricted domain in return.

Definition 1.2.1 (Membership queries with restriction access ($\mathrm{MQ_{RA}}$)). *Let $\mathcal{C} = \{f : X \to \{0,1\}\}$ be a class of concepts f, and $\mathcal{S} = \{S \subseteq X\}$ be a collection of subsets of the domain. $\mathcal{S}$ is the set of allowable restrictions for concepts f. Let* Simp *be a simplification rule which, for a concept f and restriction S, outputs an implementation of f restricted to $\mathcal{S}$.*

An algorithm $\mathcal{A}$ is an (ϵ,δ,α)-$\mathrm{MQ_{RA}}$ learning algorithm *for concept class $\mathcal{C}$ with respect to a restrictions in $\mathcal{S}$ and simplification rule* Simp *if, for every $f \in \mathcal{C}$,* $\Pr[\mathcal{A}^{\mathsf{Simp}(f,\cdot)} = h] \geq 1-\delta$

where h is an ϵ-approximation to f – and furthermore, $\mathcal{A}$ only requests restrictions for an α-fraction of the whole domain X.

Informally, constrained PRFs are PRFs with two additional properties: 1) for any subset S of the domain in a specified collection $\mathcal{S}$, a *constrained key* K_S can be computed, knowledge of which enables efficient evaluation of the PRF on S; and 2) even with knowledge of constrained keys $K_{S_1}, \dots, K_{S_m}$ for the corresponding subsets, the function retains pseudo-randomness on all points not covered by any of these sets. Connecting this to restriction access, the constrained keys will allow for generation of restriction access examples (restricted implementations with fixed partial assignments) and the second property implies that those examples do not aid in the learning of the function.

Theorem 1.2.1 (Informal). *Suppose $\mathcal{F}$ is a family of constrained PRFs which can be constrained to sets in $\mathcal{S}$. If $\mathcal{F}$ is computable in circuit complexity class $\mathcal{C}$, then $\mathcal{C}$ is hard to MQ_{RA}-learn with restrictions in $\mathcal{S}$.*

Corollary 1.2.2 (Informal). *Existing constructions of constrained PRFs [BW13] yield the following corollaries:*

- *If one-way functions exist, then poly-sized circuits can not be learned with restrictions on sub-intervals of the input-domain; and*
- *Assuming the sub-exponential hardness of the multi-linear Diffie-Hellman problem, NC^1 cannot be learned with restriction on hypercubes.*

1.2.2 New Learning Models Inspired by the Study of PRFs

We proceed to define two new learning models inspired by recent directions in cryptography. The first model is the *related concept* model inspired by work into related-key attacks in cryptography. While we have cryptography and lower bounds in mind, we argue that this model is in some ways natural. The second model, learning with *aggregate queries*, is directly inspired by our development of aggregate pseudo-random functions in this work; rather than being a natural model in its own right, this model further illustrates how cryptography and learning are duals in many senses.

The Related Concept Learning Model

The idea that some functions or concepts are related to one another is quite natural. For a DNF formula, for instance, related concepts may include formulas where a clause has been added or formulas where the roles of two variables are swapped. For a decision tree, we could consider removing some accepting leaves and examining the resulting behavior. For a circuit, a related circuit might alter internal gates or fix the values on some wires. A similar phenomena occurs in cryptography, where secret keys corresponding to different instances of the same cryptographic primitive or even secret keys of different cryptographic primitives are related (if, for example, they were generated by a pseudo random process on the same seed).

We propose a new computational learning model where the learner is explicitly allowed to specify membership queries not only for the concept to be learned, but also for "related" concepts, given by a class of allowed transformations on the concept. We will show both a general negative result in the new model. Based on recent constructions of related-key secure PRFs by Bellare and Cash [BC10] and Abdalla et al [ABPP14], we demonstrate concept classes for which access to these related concepts is of no help.

To formalize the related concept learning model, we will consider *indexed concept classes* – wherein each concept is indexed by some key. This will enable the study of related concepts by considering concepts whose keys are related in some way. Most generally, we think of a key as a succinct representation of the computational device which decides the concept. This is a general framework; for example, we may consider the bit representation of a particular log-depth circuit as a key for a concept in the concept class NC^1. For a concept f_k in concept class $\mathcal{C}$, we allow the learner to query a membership oracle for f_k and also for related concepts $f_{\phi(k)}$ for any ϕ in a specified class of allowable functions Φ.

Definition 1.2.2 (Φ-Related-Concept Learning Model (Φ-RC)). *For a concept class* $\mathcal{C}_K$

indexed by K, let $\Phi = \{\phi : K \to K\}$ be a set of functions on K containing the identity function. A related-concept oracle RC_k, on query (ϕ, x), responds with $f_{\phi(k)}(x)$, for all $\phi \in \Phi$ and $x \in X$.

An algorithm A is an (ϵ, δ)-Φ-RK learning algorithm for a C_k if, for every $k \in K$, when given access to the oracle $RK_k(\cdot)$, the algorithm A outputs with probability at least $1 - \delta$ a function $h : \{0,1\}^n \to \{0,1\}$ that ϵ-approximates f_k.

Yet again, we are able to demonstrate the limitations of this model using the power of a strong type of pseudo-random function. We show that *related-key secure PRF families (RKA-PRF)* defined and instantiated in [BC10] and [ABPP14] give a natural concept class which is not learnable with related key queries. RKA-PRFs are defined with respect to a set Φ of functions on the set of PRF keys. Informally, the security notion guarantees that for a randomly selected key k, no efficient adversary can distinguish oracle access to f_k and related $f_{\phi(k)}$ from an random oracles. We leverage this strong pseudo-randomness property to show hard-to-learn concepts in the related concept model.

Theorem 1.2.3 (Informal). *Suppose $\mathcal{F}$ is a family of RKA-PRFs with respect to related-key functions Φ. If $\mathcal{F}$ is computable in circuit complexity class $\mathcal{C}$, then $\mathcal{C}$ is hard to learn in the Φ'-RC model for some Φ'.*

Existing constructions of RKA-PRFs [ABPP14] yield the following corollary:

Corollary 1.2.4 (Informal). *Assuming the hardness of the DDH problem, and collision-resistant hash functions, NC^1 is hard to Φ-RC-learn for an class of affine functions Φ.*

The Aggregate Learning Model

The other learning model we propose is inspired by our aggregate PRFs. Here, we consider a new extension to the power of the learning algorithm. Whereas membership queries are of the form "What is the label of an example x?", we grant the learner the power to request the evaluation of simple functions on tuples of examples $(x_1, ..., x_n)$ such as "How many of $x_1, \ldots, x_n$ are in C?" or "Compute the product of the labels of $x_1, ..., x_n$?". Clearly, if

n is polynomial then this will result only a polynomial gain in the query complexity of a learning algorithm in the best case. Instead, we propose to study cases when n may be super-polynomial, but the description of the tuples is succinct. For example, the learning algorithm might query the number of x's in a large interval that are positive examples in the concept.

As with the restriction access and related concept models – and the aggregate PRFs we define in this work – the Aggregate Queries (AQ) learning model will be considered with restrictions to both the types of aggregate functions Γ the learner can query, and the sets $\mathcal{S}$ over which the learner may request these functions to be evaluated on. We now present the AQ learning model:

Definition 1.2.3 (($\Gamma, \mathcal{S}$)-Aggregate Queries (AQ) Learning). *Let $\mathcal{C} : X \to \{0,1\}$ be a concept class, and let $\mathcal{S}$ be a collection of subsets of X. Let $\Gamma : \{0,1\}^* \to V$ be an aggregation function. For $f \in \mathcal{C}$, let AGG_f be an "aggregation" oracle, which for $S \in \mathcal{S}$, returns $\Gamma_{x \in S} f(x)$. Let MEM_f be the membership oracle, which for input x returns $f(x)$.*

An algorithm $\mathcal{A}$ is an (ϵ, δ)-$(\Gamma, \mathcal{S})$-AQ learning algorithm for $\mathcal{C}$ if for every $f \in \mathcal{C}$,

$$\Pr[\mathcal{A}^{MEM_f(\cdot), \mathsf{AGG}_f(\cdot)} = h] \geq 1 - \delta$$

where h is an ϵ-approximation to f.

Initially, AQ learning is reminiscent of learning with statistical queries (SQ). In fact, this apparent connection inspired this portion of our work. But the AQ setting is in fact incomparable to SQ learning, or even the weaker correlational SQ model as defined in [BF02]. On the one hand, AQ queries provide a sort of noiseless variant of SQ, giving more power to the AQ learner; on the other hand, the AQ learner is restricted to aggregating over sets in $\mathcal{S}$, whereas the SQ learner is not restricted in this way, thereby limiting the power of the AQ learner. The AQ setting where $\mathcal{S}$ contains every subset of the domain is indeed a noiseless version of correlational SQ. This does raise the natural question of a noiseless version of SQ and its variants; hardness results in such models would be interesting in that they would suggest that the hardness comes not from the noise but from an inherent loss of information in statistics/aggregates.

We will show under cryptographic assumptions, a general lower bound on the power of learning with aggregate queries. The negative examples will use the results in Section 1.1.

Theorem 1.2.5. *Let $\mathcal{F}$ be a boolean-valued aggregate PRF with respect to set system $\mathcal{S}$ and aggregation function Γ. If $\mathcal{F}$ is computable in complexity class $\mathcal{C}$, then $\mathcal{C}$ is hard to $(\Gamma, \mathcal{S})$-AQ learn.*

Corollary 1.2.6. *Using the results from Chapter 2, we get the following corollaries:*

- *The existence of one-way functions implies that P/poly is hard to $(\sum, \mathcal{S}_{[a,b]})$-AQ learn, with $\mathcal{S}_{[a,b]}$ the set of sub-intervals of the domain as defined in section 2.2 [GGM86].*
- *The DDH Assumption implies that NC^1 is hard to $(\sum, \mathcal{DT})$-AQ learn, with $\mathcal{S}_{[a,b]}$ the set of polynomial-sized decision trees as defined in section 2.4 [NR04].*
- *The subexponential DDH Assumption implies that NC^1 is hard to $(\prod, \mathcal{R})$-AQ learn, with $\mathcal{R}$ the set of read-once boolean formulas defined in section 2.5 [NR04].*

1.3 New Negative Results from Unobfuscatable PRFs

In the past few years the field of cryptographic program obfuscation has experienced rapid growth. The work focuses primarily on applications of indistinguishability obfuscation, a notion introduced in the foundational work of [BGI+12]. The main focus of that work, though, was to formulate rigorous definitions of obfuscation and to demonstrate that some families of circuits cannot be ideally obfuscated by constructing "unobfuscatable" function families as discussed below. The subsequent work of [BP12] refined the original techniques to demonstrate impossibility of a very strong notion of approximate VBB obfuscation, in which the obfuscated circuit must only agree with the original on a constant fraction of the domain. In the third part of this thesis, we consider further possible variants of these unobfuscatable families and establish a connection between such families and limitations for both constrained PRFs [BW13, BGI14, KPTZ13] and watermarking schemes [CHV15]. Below we describe the negative results, followed by an introduction of our techniques.

Unpuncturable PRFs

Informally, (point) puncturable PRFs are PRFs with two additional properties: (1) for any element x of the domain, a *punctured key* k_x can be computed, knowledge of which enables efficient evaluation of the PRF on all $x' \neq x$; (2) even with knowledge of the punctured key k_x, the function retains pseudo-randomness on x.[6] The classical GGM PRF is puncturable, implying that if PRFs exist at all, then puncturable PRFs exist [BW13, BGI14, KPTZ13, GGM86]. These objects have received much attention as they underly essentially all current applications of obfuscation. Many constructions exist for different settings, but no negative results are known.

Are all PRFs puncturable, or do there exist "unpuncturable" PRFs? For a family $\mathcal{F} = \{F_k\}$ to not be puncturable, it suffices to show that for any x, there is an efficient algorithm that can output k given any circuit that computes $F_k(x')$ for all $x' \neq x$. We construct such a family in Chapter 4.

Theorem 1.3.1 (Informal). *There exist families of PRFs that are not point puncturable.*

Unwatermarkable PRFs

The notion of watermarking of circuits has received only sporadic attention in the cryptography literature. Recently the notion was revisited in [CHV15], which put forth refined definitions for cryptographic watermarking of circuit families. At a high level, a δ-watermarking scheme for a class of circuits $\mathbb{C}$ is a triple of algorithms $(\mathsf{Setup}, \mathsf{Mark}, \mathsf{Verify})$ with the following properties, with respect to the keys $(\mathsf{mk}, \mathsf{vk}) \leftarrow \mathsf{Setup}(1^\lambda)$:[7]

[6]Puncturable PRFs are simply a reframing of the constrained PRFs previously discussed. A constrained key for a set S enables computation only on S, while a punctured key for a point x enables computation everywhere except for x. That is, the puncturable PRF key k_x is exactly the constrained PRF key $k_{X\setminus\{x\}}$. In this section and the corresponding chapter we refer to puncturable PRFs rather than constrained PRFs.

[7]In this work we consider a simplified definition than presented in [CHV15], thereby strengthening our impossibility result.

- *Completeness*: Verify on the marked version of C should return 1. That is, for every circuit C in the family $\mathbb{C}$, $\mathsf{Verify}(\mathsf{vk}, \mathsf{Mark}(\mathsf{mk}, C)) = 1$ with high probability.

- *Functionality preserving*: the marked circuit and the original circuit should agree almost everywhere. That is, for every circuit C in the family $\mathbb{C}$ and for random x in the domain of C, $\mathsf{Mark}(\mathsf{mk}, C)(x) = C(x)$ with high probability.

- *δ-Unremovability*: Given a random marked circuit, it should be infeasible to output a similar circuit on which Verify returns 0. That is, for all efficient algorithms A, and for random $C \leftarrow \mathbb{C}$, if $\hat{C} = A(\mathsf{Mark}(\mathsf{mk}, C))$ is a circuit that agrees with $\hat{C}$ on at least $1 - \delta$ fraction of the domain, then $\mathsf{Verify}(\mathsf{vk}, \hat{C}) = 1$ with high probability.

- *Unforgeability*: without access to any marked circuits, it should be infeasible to output a circuit on which Verify returns 1. That is, for all efficient algorithms A, $\mathsf{Verify}(\mathsf{vk}, A(1^\lambda)) = 0$ with high probability.

Given a few minutes of consideration, it should be immediately clear that some families $\mathbb{C}$ cannot be watermarked under this definition. As a simple example, consider the family of all constant functions C_y indexed by a constant y. By functionality preservation, $\mathsf{Mark}(C_y)(x) = y$ almost everywhere, so y and thus C_y can be trivially recovered. At this point, either $\mathsf{Verify}(C_y) = 0$ or $\mathsf{Verify}(C_y) = 1$, in which case either unremovability or unforgeability is violated, respectively. This example suggests that some learnability notion should suffice for unwatermarkability, and [CHV15] makes this intuition concrete. The exact learnability condition that suffices is quite similar to the condition needed for PRFs to be not puncturable: namely that given any approximate (agrees almost everywhere) implementation of a circuit C_K in the family, some "canonical version" of C_K can be efficiently extracted.

Pseudorandom families are far from learnable in the traditional sense, suggesting that the prior attack fails. Indeed, [CHV15] constructs a watermarking scheme for any family of puncturable PRFs, under some strong cryptographic assumptions. Perhaps all PRF families can be watermarked? In this thesis, we demonstrate that that there are PRF families that are unwatermarkable by constructing families that are learnable in the sense above.

Corollary 1.3.2. *Assuming one-way functions, for any non-negligible function $\delta(\lambda)$, there is a family of pseudorandom functions that is not δ-watermarkable.*

Techniques – Unobfuscatable families

The main technical hurdle in [BGI+12] was the construction of so called "totally unobfuscatable" pseudorandom families of circuits. These circuits have the property that given *any* implementation of a circuit C_k from the family, the index k is efficiently recoverable. The impossibility of the VBB obfuscation definition for this family follows directly: given an obfuscated circuit C_k, an adversary is able to output the index k, while a simulator with oracle access cannot.

A natural question is whether unobfuscatable families can be constructed to rule out approximate VBB. This would correspond to strengthening the above property to hold given any *approximate* implementation of the circuit. A weak notion of approximation is considered in [BGI+12], and [BP12] strengthens the notion of approximation considerably – handling approximations that agree on only a constant fraction of the domain.

For the goal of constructing PRF families that are not puncturable nor watermarkable, the result in [BP12] suffers from two shortcomings. First, the function families they construct are very far from pseudorandom. Second, they only require that given an approximate implementation of C_K, a single hard-core bit of C_K can be recovered, rather than C_K itself. Furthermore, these limitations are integral to their approach.

Our aim is to remedy these deficits, recovering the pseudorandomness and the ability to recover C_K.[8] We realize this gain by sacrificing on the notion of approximation: from any constant-fraction implementation to any almost-everywhere implementation.

[8]More precisely, we recover some canonical C'_K that agrees with C_K on $1-\delta$ fraction of the domain, which suffices.

Chapter 2

Aggregate Pseudorandom Functions

In this chapter, we introduce the notion of aggregate pseudorandom functions. We show several constructions of aggregate PRFs. In Section 2.2, we show as a warm-up a generic construction of aggregate PRFs for intervals (where the aggregation is any group operation). This construction is black-box: given any PRF with the appropriate domain and range, we construct a related family of aggregate PRFs and with no loss in security. In Section 2.3, we show a construction of aggregate PRFs for products over bit-fixing sets (hypercubes), from the decisional Diffie-Hellman assumption. We then generalize the DDH construction: in Section 2.4, to the class of sets recognized by polynomial-size decision trees; and in Section 2.5, to sets recognized by read-once Boolean formulas (from a subexponential DDH assumption). In the last of these constructions, we make use of Lemma 2.1.1 to argue security.

2.1 Definition and Generic Security

We will let λ denote the security parameter throughout this paper.

Let $\mathcal{F} = \{\mathcal{F}_\lambda\}_{\lambda>0}$ be a function family where each function $f \in \mathcal{F}_\lambda$ maps a domain $\mathcal{D}_\lambda$ to a range $\mathcal{R}_\lambda$. An *aggregate function* family is associated with two objects:

1. an ensemble of sets $\mathcal{S} = \{\mathcal{S}_\lambda\}_{\lambda>0}$ where each $\mathcal{S}_\lambda$ is a collection of subsets of the domain $S \subseteq \mathcal{D}_\lambda$; and

2. an "aggregation function" $\Gamma_\lambda : (\mathcal{R}_\lambda)^* \to \mathcal{V}_\lambda$ that takes a tuple of values from the range $\mathcal{R}_\lambda$ of the function family and "aggregates" them to produce a value in an output set $\mathcal{V}_\lambda$.

Let us now make this notion formal. To do so, we will impose restrictions on the set ensembles and the aggregation function. First, we require set ensemble $\mathcal{S}_\lambda$ to be *efficiently recognizable.* That is, there is a polynomial-size Boolean circuit family $\mathcal{C} = \{\mathcal{C}_\lambda\}_{\lambda>0}$ such that for any set $S \in \mathcal{S}_\lambda$ there is a circuit $C = C_S \in \mathcal{C}_\lambda$ such that $x \in S$ if and only if $C(x) = 1$. Second, we require our aggregation functions Γ to be efficient in the length of its inputs, and symmetric; namely the output of the function does not depend on the order in which the inputs are fed into it. Summation over an Abelian group is an example of a possible aggregation function. Third and finally, elements in our sets $\mathcal{D}_\lambda$, $\mathcal{R}_\lambda$, and $\mathcal{V}_\lambda$ are all representable in $\mathsf{poly}(\lambda)$ bits, and the functions $f \in \mathcal{F}_\lambda$ are computable in $\mathsf{poly}(\lambda)$ time.

Define the aggregate function $\mathsf{AGG} = \mathsf{AGG}^\lambda_{f,\mathcal{S}_\lambda,\Gamma_\lambda}$ that is indexed by a function $f \in \mathcal{F}_\lambda$, takes as input a set $S \in \mathcal{S}_\lambda$ and "aggregates" the values of $f(x)$ for all $x \in \mathcal{S}_\lambda$. That is, $\mathsf{AGG}(S)$ outputs

$$\Gamma\big(f(x_1), f(x_2), \ldots, f(x_{|S|})\big)$$

where $S = \{x_1, \ldots, x_{|S|}\}$. More precisely, we have

$$\begin{aligned} \mathsf{AGG}^\lambda_{f,\mathcal{S}_\lambda,\Gamma_\lambda} :& \mathcal{S}_\lambda \to \mathcal{V}_\lambda \\ & S \ \mapsto \Gamma_{x_i \in S}\big(f(x_1), \ldots, f(x_{|S|})\big) \end{aligned}$$

We will furthermore require that the AGG can be computed in $\mathsf{poly}(\lambda)$ time. We require this in spite of the fact that the sets over which the aggregation is done can be exponentially large! Clearly, such a thing is impossible for a random function f but yet, we will show how to construct *pseudo-random* function families that support efficient aggregate evaluation. We will call such a pseudo-random function (PRF) family an *aggregate PRF* family. In other words, our objective is two fold:

1. Allow anyone who knows the (polynomial size) function description to efficiently com-

pute the aggregate function values over exponentially large sets; but at the same time,

2. Ensure that the function family is indistinguishable from a truly random function, even given an oracle that computes aggregate values.

A simple example of aggregates is that of computing the summation of function values over sub-intervals of the domain. That is, let domain and range be $\mathbb{Z}_p$ for some $p = p(\lambda)$, let the family of subsets be $\mathcal{S}_\lambda = \{[a, b] \subseteq \mathbb{Z}_p : a, b \in \mathbb{Z}_p; a \leq b\}$, and the aggregation function be $\Gamma_\lambda(y_1, \ldots, y_k) = \sum_{i=1}^{k} y_i \pmod{p}$. In this case, we are interested in computing

$$\mathsf{AGG}^{\lambda}_{f,\mathcal{S}_\lambda,\mathsf{sum}}([a, b]) = \sum_{a \leq x \leq b} f(x)$$

We will, in due course, show both constructions and impossibility results for aggregate PRFs, but first let us start with the formal definition.

Definition 2.1.1 (Aggregate PRF). *Let $\mathcal{F} = \{\mathcal{F}_\lambda\}_{\lambda>0}$ be a function family where each function $f \in \mathcal{F}_\lambda$ maps a domain $\mathcal{D}_\lambda$ to a range $\mathcal{R}_\lambda$, $\mathcal{S}$ be an efficiently recognizable ensemble of sets $\{\mathcal{S}_\lambda\}_{\lambda>0}$, and $\Gamma_\lambda : (\mathcal{R}_\lambda)^* \to \mathcal{V}_\lambda$ be an aggregation function. We say that $\mathcal{F}$ is an $(\mathcal{S}, \Gamma)$-* aggregate pseudorandom function family (also denoted $(\mathcal{S}, \Gamma)$-AGG-PRF) *if there exists an efficient algorithm* $\mathsf{Aggregate}_{k,\mathcal{S},\Gamma}(S)$*: On input a subset $S \in \mathcal{S}$ of the domain, outputs $v \in \mathcal{V}$, such that*

- **Efficient aggregation:** *For every $S \in \mathcal{S}$,* $\mathsf{Aggregate}_{k,\mathcal{S},\Gamma}(S) = \mathsf{AGG}_{k,\mathcal{S},\Gamma}(S)$ *where* $\mathsf{AGG}_{k,\mathcal{S},\Gamma}(S) := \Gamma_{x \in S}\ F_k(x)$.[1][2]

- **Pseudorandomness:** *For all probabilistic polynomial-time (in security parameter λ) algorithms A, and for randomly selected key $k \in K$:*

$$\left| \Pr_{f \leftarrow \mathcal{F}_\lambda}[A^{f_k, AGG_{f_k,\mathcal{S},\Gamma}}(1^\lambda)] - \Pr_{h \leftarrow \mathcal{H}_\lambda}[A^{h, AGG_{h,\mathcal{S},\Gamma}}(1^\lambda)] \right| \leq \mathsf{negl}(\lambda)$$

[1]We omit subscripts on AGG and Aggregate when clear from context.

[2]AGG is defined to be the correct aggregate value, while Aggregate is the algorithm by which we compute the value AGG. We make this distinction because while a random function cannot be efficiently aggregated, the aggregate value is still well-defined.

where $\mathcal{H}_\lambda$ is the set of all functions $D_\lambda \to R_\lambda$.

Remark. In this work, we restrict our attention to aggregation functions that treat the range $\mathcal{V}_\lambda = \mathcal{R}_\lambda$ as an Abelian group and compute the group sum (or product) of its inputs. We denote this setting by $\Gamma = \sum$ (or $\prod$, respectively). Supporting other types of aggregation functions (ex: max, a hash) is a direction for future work.

2.1.1 A General Security Theorem for Aggregate PRFs

How does the security of a function family in the AGG-PRF game relate to security in the normal PRF game (in which A uses only the oracle f and not AGG_f)? While we have tailored security reductions for some of our schemes, can we say something more general?

In this section, we show a general security theorem for aggregate pseudo-random functions. Namely, we show that any "sufficiently secure" PRF is also aggregation-secure (for any collection of efficiently recognizable sets and any group-aggregation operation), in the sense of Definition 2.1.1, by way of an *inefficient* reduction (with overhead polynomial in the size of the domain). In Section 2.5, we will use this to construct AGG-PRFs from a subexponential-time hardness assumption on the DDH problem. We also show that no such *general* reduction can be efficient, by demonstrating a PRF family that is not aggregation-secure. As a general security theorem cannot be shown without the use of complexity leveraging, this suggests a natural direction for future study: to devise constructions for similarly expressive aggregate PRFs from polynomial assumptions.

Lemma 2.1.1. *Let $\mathcal{F} = \{\mathcal{F}_\lambda\}_{\lambda>0}$ be a pseudo-random function family where each function $f \in \mathcal{F}_\lambda$ maps a domain $\mathcal{D}_\lambda$ to a range $\mathcal{R}_\lambda$. Suppose there is an adversary A that runs in time $t_A = t_A(\lambda)$ and achieves an advantage of $\epsilon_A = \epsilon_A(\lambda)$ in the aggregate PRF security game for the family $\mathcal{F}$ with an efficiently recognizable set system $\mathcal{S}_\lambda$ and an aggregation function Γ_λ that is computable in time polynomial in its input length. Then, there is an adversary B that runs in time $t_B = t_A + \mathsf{poly}(\lambda, |\mathcal{D}_\lambda|)$ and achieves an advantage of $\epsilon_B = \epsilon_A$ in the standard PRF game for the family $\mathcal{F}$.*

Proof. Let $f_K \leftarrow \mathcal{F}_\lambda$ be a random function from the family $\mathcal{F}_\lambda$. We construct the adversary B which is given access to an oracle $\mathcal{O}$ which is either f_K or a uniformly random function $h : \mathcal{D}_\lambda \to \mathcal{R}_\lambda$.

B works as follows: It queries the PRF on all inputs $x \in \mathcal{D}_\lambda$, builds the function table T_K of f_K and runs the adversary A, responding to its queries as follows:

1. Respond to its PRF query $x \in \mathcal{D}_\lambda$ by returning $T_K[x]$; and

2. Respond to its aggregate query (Γ, S) by (a) going through the table to look up all x such that $x \in S$; and (b) applying the aggregation function honestly to these values.

Finally, when A halts and returns a bit b, B outputs the bit b and halts.

B takes $O(|\mathcal{D}_\lambda|)$ time to build the truth table of the oracle. For each aggregate query (Γ, S), B first checks for each $x \in \mathcal{D}_\lambda$ whether $x \in S$. This takes $|\mathcal{D}_\lambda| \cdot \mathsf{poly}(\lambda)$ time, since S is efficiently recognizable. It then computes the aggregation function Γ over $f(x)$ such that $x \in S$, taking $\mathsf{poly}(|\mathcal{D}_\lambda|)$ time, since Γ is computable in time polynomial in its input length. The total time, therefore, is

$$t_B = t_A + \mathsf{poly}(\lambda, |\mathcal{D}_\lambda|)$$

Clearly, when $\mathcal{O}$ is the pseudo-random function f_K, B simulates an aggregatable PRF oracle to A, and when $\mathcal{O}$ is a random function, B simulates an aggregate random oracle to A. Thus, B has the same advantage in the PRF game as A does in the aggregate PRF game. $\square$

The above gives an inefficient reduction from the PRF security of a function family $\mathcal{F}$ to the AGG-PRF security of the same family running in time polynomial in the size of the domain. Can this reduction be made efficient; that is, can we replace $t_B = t_A + \mathsf{poly}(\lambda)$ into the Lemma 2.1.1?

This is not possible. Such a reduction would imply that every PRF family that supports efficient aggregate functionality AGG is AGG-PRF secure; this is clearly false. Take for example a pseudorandom function family $\mathcal{F}_0 = \{f : \mathbb{Z}_{2p} \to \mathbb{Z}_p\}$ such that for all f, there is no x with $f(x) = 0$. It is possible to construct such a pseudorandom function family

$\mathcal{F}_0$ (under the standard definition). While 0 is not in the image of any $f \in \mathcal{F}_0$, a random function with the same domain and range will, with high probability, have 0 in the image. For an aggregation oracle AGG_f computing *products* over $\mathbb{Z}_p$: $\mathsf{AGG}_f(\mathbb{Z}_{2p}) \neq 0$ if $f \in \mathcal{F}_0$, while $\mathsf{AGG}_f(\mathbb{Z}_{2p}) = 0$ with high probability for random f.

Thus, access to aggregates for products over $\mathbb{Z}_p$[3] would allow an adversary to trivially distinguish $f \in \mathcal{F}_0$ from a truly random map.

2.1.2 Impossibility of Aggregate PRF for General Sets

It is natural to ask whether whether an aggregate PRF might be constructed for more general sets than we present. There we constructed aggregate PRF for the sets of all satisfying assignments for read-once boolean formula and decision trees. As we show in the following, it is impossible to extend this to support the set of satisfying assignmnets for more general circuits.

Theorem 2.1.2. *Suppose there is an algorithm that has a PRF description K, a circuit C, and a fixed aggregation rule (sum over a finite field, say), and outputs the aggregate value*

$$\sum_{x:C(x)=1} f_K(x)$$

Then, there is an algorithm that takes circuits C as input and w.h.p. over it coins, decides the satisfiability of C.

Proof. The algorithm for SAT simply runs the aggregator with a randomly chosen K, and outputs YES if and only if the aggregator returns 1. The rationale is that if the formula is unsatisfiable, you will always get 0 from the aggregator.[4] Otherwise, you will get $f_K(x)$, where x is the satisfying assignment. (More generally, $\sum_{x:C(x)=1} f_K(x)$). Now, this might end

[3]Taken with respect to a set ensemble $\mathcal{S}$ containing, as an element, the whole domain $\mathbb{Z}_{2p}$. While this is not necessary (a sufficiently large subset would suffice), it is the case for the ensembles $\mathcal{S}$ we consider in this work.

[4]This proof may be extended to the case when the algorithm's output is not restricted to be 0 when the input circuit C is unsatisfiable, and even arbitrary outputs for sufficiently expressive classes of circuits.

up being 0 accidentally, but cannot be 0 always since otherwise, you will get a PRF distinguisher. The distinguisher has the satisfying assignment hardcoded into it non-uniformly,[5] and it simply checks if $f_K(x) = 0$. $\square$

This impossibility result can be generalized for efficient aggregation of functions that are not pseudo-random. For instance, if $f(x) \equiv 1$ was the constant function 1, the same computing the aggregate over f satisfying inputs to C would not only reveal the satisfiability of C, but even the number of satisfying assignments! In the PRF setting though, it seems that aggregates only reveal the (un)satisfiability of a circuit C, but not the number of satisfying assignments. Further studying the relationship between the (not necessarily pseudo-random) function f, the circuit representation of C, and the tractability of computing aggregates is an interesting direction. A negative result for a class for which satisfiability (or even counting assignments) is tractable would be very interesting.

2.2 Generic Construction for Interval Sets

Our first construction is from [GGN10][6]. The construction is entirely black-box: from any appropriate PRF family $\mathcal{G}$, we construct a related AGG-PRF family $\mathcal{F}$. Unlike the proofs in the sequel, this reduction exactly preserves the security of the starting PRF.

Let $\mathcal{G}_\lambda = \{g_K : \mathbb{Z}_{n(\lambda)} \to R_\lambda\}_{K \in \mathcal{K}_\lambda}$ be a PRF family, with $R = R_\lambda$ being a group where the group operation is denoted by $\oplus$[7]. We construct an aggregatable PRF $\mathcal{F}_\lambda = \{f_K\}_{K \in \mathcal{K}_\lambda}$ for which we can efficiently compute summation of $f_K(x)$ for all x in an interval $[a, b]$, for any $a \leq b \in \mathbb{Z}_n$. Let $\mathcal{S}_{[a,b]} = \{[a, b] \subseteq \mathbb{Z}_n : a, b \in \mathbb{Z}_n; a \leq b\}$ be the set of all interval subsets of $\mathbb{Z}_n$, $[a, b] = \{x \in \mathbb{Z}_n : a \leq x \leq b\}$. Define $\mathcal{F} = \{f_K : \mathbb{Z}_n \to R\}_{K \in \mathcal{K}}$ as follows:

[5]As pointed out by one reviewer, for sufficiently expressive classes of circuits C, this argument can be made uniform. Specifically, we use distinguish the challenge y from a pseudo-random generator from random by choosing $C := C_y$ that is satisfiable if and only if y is in the PRG image, and modify the remainder of the argument accordingly.

[6]See Example 3.1 and Footnote 18

[7]The only structure of $\mathbb{Z}_n$ we us is the *total order*. Our construction directly applies to any finite, totally-ordered domain D by first mapping D to $\mathbb{Z}_n$, preserving order.

$$f_K(x) = \begin{cases} g_K(0) & : x = 0 \\ g_K(x) \ominus g_K(x-1) & : x \neq 0 \end{cases}$$

Lemma 2.2.1. *Assuming that $\mathcal{G}$ is a pseudo-random function family, $\mathcal{F}$ is a $(\mathcal{S}_{[a,b]}, \oplus)$-aggregate pseudo-random function family.*

Proof. It follows immediately from the definition of f_K that one can compute the summation of $f_K(x)$ over any interval $[a, b]$. Indeed, rearranging the definition yields

$$\sum_{x \in [0,b]} f_K(x) = g_K(b) \text{ and } \sum_{x \in [a,b]} f_K(x) = g_K(b) \oplus -g_K(a-1)$$

We reduce the pseudo-randomness of $\mathcal{F}$ to that of $\mathcal{G}$. The key observation is that each query to the f_K oracle as well as the aggregation oracle for f_K can be answered using at most two black-box calls to the underlying function g_K. By assumption on $\mathcal{G}$, replacing the oracle for g_K with a uniformly random function $h : \mathbb{Z}_n \to R$ is computationally indistinguishable. Furthermore, the function f defined by replacing g by h, namely

$$f'(x) = \begin{cases} h(0) & : x = 0 \\ h(x) \ominus h(x-1) & : x \neq 0 \end{cases}$$

is a truly random function. Thus, the simulated oracle with g_K replaced by h implements a uniformly random function that supports aggregate queries. Security according to Definition 2.1.1 follows immediately. □

Another construction from the same work achieves summation over the integers for PRFs whose range is $\{0, 1\}$. We omit the details of the construction, but state the theorem for completeness.

Theorem 2.2.2 (Integer summation over intervals, from one-way functions [GGN10]). *Assume one-way functions exist. Then, there exists an $(\mathcal{S}_{[a,b]}, \sum)$-AGG-PRF family that maps $\mathbb{Z}_{2^\lambda}$ to $\{0, 1\}$, where $\sum$ denotes summation over $\mathbb{Z}$.*

2.3 Bit-Fixing Aggregate PRF from DDH

We now construct an aggregate PRF computing products for bit-fixing sets. Informally, our PRF will have domain $\{0,1\}^{poly(\lambda)}$, and support aggregation over sets like $\{x : x_1 = 0 \wedge x_2 = 1 \wedge x_7 = 0\}$. We will naturally represent such sets by a string in $\{0,1,\star\}^{\mathrm{poly}(\lambda)}$ with 0 and 1 indicating a fixed bit location, and $\star$ indicating a free bit location. We call each such set a 'hypercube.' The PRF will have a multiplicative group $\mathcal{G}$ as its range, and the aggregate functionality will compute group products.

Our PRF is exactly the Naor-Reingold PRF [NR04], for which we demonstrate efficient aggregation and security. We begin by stating the decisional Diffie-Hellman assumption.

Let $\mathcal{G} = \{\mathcal{G}_\lambda\}_{\lambda>0}$ be a family of groups of order $p = p(\lambda)$. The decisional Diffie-Hellman assumption for $\mathcal{G}$ says that the following two ensembles are computationally indistinguishable:

$$\begin{aligned}&\left\{(\mathcal{G}_\lambda, g, g^a, g^b, g^{ab}) : \ G \leftarrow \mathcal{G}_\lambda; \ g \leftarrow G; \ a, b \leftarrow \mathbb{Z}_p\right\}_{\lambda>0}\\ &\qquad\approx_c \left\{(G, g, g^a, g^b, g^c) : \ G \leftarrow \mathcal{G}_\lambda; \ g \leftarrow G; \ a, b, c \leftarrow \mathbb{Z}_p\right\}_{\lambda>0}\end{aligned}$$

We say that the $(t(\lambda), \epsilon(\lambda))$-DDH assumption holds if for every adversary running in time $t(\lambda)$, the advantage in distinguishing between the two distributions above is at most $\epsilon(\lambda)$.

2.3.1 Construction

Let $\mathcal{G} = \{\mathcal{G}_\lambda\}_{\lambda>0}$ be a family of groups of order $p = p(\lambda)$, each with a canonical generator g, for which the decisional Diffie Hellman (DDH) problem is hard. Let $\ell = \ell(\lambda)$ be a polynomial function. We will construct a PRF family $\mathcal{F}_\ell = \{\mathcal{F}_{\ell,\lambda}\}_{\lambda>0}$ where each function $f \in \mathcal{F}_{\ell,\lambda}$ maps $\{0,1\}^{\ell(\lambda)}$ to $\mathcal{G}_\lambda$. Our PRF family is exactly the Naor-Reingold PRF [NR04]. Namely, each function f is parametrized by $\ell + 1$ numbers $\vec{K} := (K_0, K_1, \ldots, K_\ell)$, where each $K_i \in \mathbb{Z}_p$.

$$f_{\vec{K}}(x_1, \ldots, x_\ell) = g^{K_0 \prod_{i=1}^{\ell} K_i^{x_i}} = g^{K_0 \prod_{i:x_i=1} K_i} \qquad \in \mathcal{G}_\lambda$$

The aggregation algorithm Aggregate for bit-fixing functions gets as input the PRF key $\vec{K}$ and a bit-fixing string $y \in \{0,1,\star\}^\ell$ and does the following:

- Define the strings K_i' as follows:

$$K_i' = \begin{cases} 1 & \text{if } y_i = 0 \\ K_i & \text{if } y_i = 1 \\ 1 + K_i & \text{otherwise} \end{cases}$$

- Output $g^{K_0 \prod_{i=1}^{\ell} K_i'}$ as the answer to the aggregate query.

Letting $\mathcal{HC} = \{\mathcal{HC}_{\ell(\lambda)}\}_{\lambda>0}$ where $\mathcal{HC}_\ell = \{0,1,\star\}^\ell$ is the set of hypercubes on $\{0,1\}^\ell$, we now prove the following:

Theorem 2.3.1. *The collection of functions $\mathcal{F}$ defined above is a secure aggregate PRF with respect to the subsets $\mathcal{HC}$ and the product aggregation function over $\mathcal{G}$ under the DDH assumption on the group $\mathcal{G}$.*

Correctness.

We show that the answer we computed for an aggregate query $y \in \{0,1,\star\}^\ell$ is correct. Define the sets

$$\mathsf{Match}(y) := \{x \in \{0,1\}^\ell : \ \forall i, y_i = \star \text{ or } x_i = y_i\} \text{ and } \mathsf{Fixed}(y) := \{i \in [\ell] : y_i \in \{0,1\}\}$$

Thus, Match(y) is the set of all 0-1 strings x that match all the fixed locations of y, but can take any value on the wildcard locations of y. Fixed(y) is the set of all locations i where the bit y_i is fixed. Note that:

$$\begin{aligned}
\mathsf{AGG}(\vec{K}, y) &= \textstyle\prod_{x \in \mathsf{Match}(y)} f_{\vec{K}}(x) && \text{(by definition of AGG)} \\
&= \textstyle\prod_{x \in \mathsf{Match}(y)} g^{K_0 \prod_{i=1}^{\ell} K_i^{x_i}} && \text{(by definition of } f_{\vec{K}}\text{)} \\
&= g^{K_0 \sum_{x \in \mathsf{Match}(y)} \prod_{i=1}^{\ell} K_i^{x_i}} && \\
&= g^{K_0 \left(\prod_{i \in \mathsf{Fixed}(y)} K_i^{y_i}\right) \cdot \left(\prod_{i \in [\ell] \setminus \mathsf{Fixed}(y)} (1+K_i)\right)} && \text{(inverting sums and products)} \\
&= g^{K_0 \prod_{i=1}^{\ell} K_i'} && \text{(by definition of } K_i'\text{)} \\
&= \mathsf{Aggregate}(\vec{K}, y) && \text{(by definition of Aggregate)}
\end{aligned}$$

2.3.2 Security

The security proof is adapted from [CH15], which proves a stronger generalization of this theorem. We let a string $y \in \{0, 1, \star\}^{\ell}$ denote all of: a string, the hypercube that it corresponds to, and the characteristic vector for that hypercube. We use one additional representation: the characteristic product of an ℓ-dimensional hypercube $y_1 \dots y_\ell \in \{0, 1, \star\}$ can be written as $\vec{y}^1 \otimes \cdots \otimes \vec{y}^\ell$ where

$$\vec{y}^j = \begin{cases} (1\ 0) & \text{if } y_j = 0 \\ (0\ 1) & \text{if } y_j = 1 \\ (1\ 1) & \text{if } y_j = \star \end{cases} \tag{2.1}$$

Definition 2.1.1 requires that for all p.p.t algorithms A and random $\vec{K}$:

$$|\Pr_{\vec{K},A}[A^{f_{\vec{K}}, AGG_{f_{\vec{K}}, \mathcal{HC}, \Pi}}(1^\lambda)] - \Pr_{h,A}[A^{h, AGG_{h, \mathcal{HC}, \Pi}}(1^\lambda)]| \leq \mathsf{negl}(\lambda)$$

To prove the theorem, we must prove the adversary's views in the two cases (ie: the pseudorandom $f_{\vec{K}}$ and the random h) are indistinguishable. We already know what the view in the pseudorandom case is: A's queries are answered according to $f_{\vec{K}}(\cdot)$ and $\mathsf{Aggregate}(\vec{K}, \cdot)$. But how can we simulate the view in the random case?

In the setting of classical pseudorandom functions, a challenger simulates access to a random function by answering new query with a fresh random value and each repeated query consistently. To do so, the challenger must keep state – all the previous queries –

and check for dependencies – whether a query is a repeat. We can use the same approach: keep track of all of A's previous queries $y_i \in \{0,1,\star\}^\ell$ and answer queries consistently with the history. But what does "consistently with the history" mean exactly in the setting of aggregates, and can consistent responses be computed efficiently?

Let χ_S be the characteristic vector of a set S. For a set S and any function h: $\mathsf{AGG}(h,S) = \prod_{x\in S} h(x)$. If $S = S_1 \sqcup S_2$ is the disjoint union of two sets, then $\mathsf{AGG}(h,S) = \mathsf{AGG}(h,S_1)\cdot\mathsf{AGG}(h,S_2)$. Moreover, if $\chi(S) = \sum_i \alpha_i \chi_{S_i}$, then

$$\mathsf{AGG}(h,S) = \prod_i \mathsf{AGG}(h,S_i)^{\alpha_i}$$

In this case, we say that S is a linear combination of the sets $\{S_i\}$. On the other hand, for a random function h, if S is linearly *independent* of a collection of sets $\{S_i\}$, the aggregate value $\mathsf{AGG}(h,S)$ over S is independent of the aggregate values $\mathsf{AGG}(h,S_i)$ over the sets S_i.

Thus, to answer "consistently with the history", we must be able to compute linear-dependencies between hypercube sets represented by strings $y \in \{0,1,\star\}$. Note that Gaussian elimination does not suffice, as the vectors in question are characteristic vectors of length 2^ℓ. This problem can efficiently solved, as in [CH15] and [BW04]. We therefore take for granted an efficient algorithm Span that takes as input a collection of hypercubes of the form $y_1,\ldots,y_q \in \{0,1,\star\}^\ell$ such that:

$$\mathsf{Span}(y_1,\ldots,y_q) = \begin{cases} \alpha_1,\ldots,\alpha_{q-1} & \text{if } y_q = \sum_{i=i}^{q-1} \alpha_i y_i \\ \perp & \text{otherwise} \end{cases}$$

We also define Span_J to be Span computed on the first J indices of the inputs y. That is, Span_J computes linear combinations over $y_1^J,\ldots,y_q^J$ where $y_i^J \in \{0,1,\star\}^J$ is y_i restricted to the first J locations. More intuitively, Span_J computes linear dependencies over the first J dimensions of the corresponding hypercubes.

To simulate the random hybrid, we answer the qth query y_q as:

$$\begin{cases} g_q \leftarrow G & \text{if } \mathsf{Span}(y_1, \dots, y_q) = \perp \\ g_q = \prod_{i=1}^{q-1} g_i^{\alpha_i} & \text{if } \mathsf{Span}(y_1, \dots, y_q) = \alpha_1, \dots, \alpha_{q-1} \end{cases}$$

This exactly simulates the oracle $\mathsf{AGG}(h, \cdot)$ for a truly random function h.

Hybrids

Now we must show that the views of the adversary in the pseudorandom and random cases are indistinguishable. We do so by considering a series of $\ell + 1$ hybrid views H_J for which each pair of hybrids will be indistinguishable. In the Jth hybrid we sample a partial key $(K_{J+1}, \dots, K_\ell) \leftarrow \mathbb{Z}_p^{\ell - J}$. We answer the qth query y_q as:

$$g_q^{\prod_{j=J+1}^{\ell} K'_j} \text{ for}$$

$$g_q = \begin{cases} g_q \leftarrow G & \text{if } \mathsf{Span}_J(y_1, \dots, y_q) = \perp \\ g_q = \prod_{i=1}^{q-1} g_i^{\alpha_i} & \text{if } \mathsf{Span}_J(y_1, \dots, y_q) = \alpha_1, \dots, \alpha_{q-1} \end{cases}$$

It is easy to see that hybrid H_J is exactly the random hybrid defined above, as $\mathsf{Span}_\ell = \mathsf{Span}$. If we interpret Span_0 to always return 1 (all empty strings are equal), then it is easy to see that H_0 is exactly the aggregate pseudorandom function oracle. The base g_q of the exponentiation will always be the same $g_1 = g^{K_0}$ for some uniform K_0.

Reduction to Matrix DDH

It remains to show that adjacent hybrids are indistinguishable. We reduce the problem of distinguishing adjacent hybrids to the Matrix DDH problem, which is as hard as the DDH problem [BHHO08]. The below is adapted from [CH15]; a more general and complete exposition is included in that work.

Let G be a group of order p with generator g, and $C \in \mathbb{Z}_p^{2 \times q}$, and let g^C denote element-wise exponentiation. An instance of the Matrix DDH problem is a tuple (g, g^C), where C is either a uniform matrix in $\mathbb{Z}_p^{2\times q}$ or a uniformly random rank 1 matrix $C = (a_1, a_2)^T(b_1, \dots, b_q)$ for uniform a_i, $b_i \in \mathbb{Z}_p$. The Matrix DDH assumption states that the uniform and rank 1 cases are computationally indistinguishable. Our goal, then, is to devise a method for answering the AGG-PRF adversary's queries such that the answers come from hybrids H_J and H_{J+1} depending on the type of Matrix DDH challenge received.

Given a Matrix DDH challenge (g, g^C), our reduction samples a partial key $(K_{J+2}, \dots, K_\ell) \leftarrow \mathbb{Z}_p^{\ell-J+1}$, and answer the qth query y_q as:

$$g_q^{\prod_{j=J+2}^{\ell} K'_j} \text{ where}$$

$$\begin{cases} g_q = g^{\sum_{i=1}^{q-1} \alpha_i \langle (c_{1,i}\ c_{2,i}), \vec{y}^{J+1}\rangle} & \text{if } \mathsf{Span}_J(y_1, \dots, y_q) = \alpha_1, \dots, \alpha_{q-1} \\ g_q = g^{\langle (c_{1,q}\ c_{2,q}), \vec{y}^{J+1}\rangle} & \text{otherwise} \end{cases}$$

where $\vec{y}^{J+1}$ is defined in Equation 2.1. In the first case

In the case when $C = (a_1\ a_2)^T(b_1\ \dots\ b_q)$, this exponent factors into:

$$\langle (c_{1,i}\ c_{2,i}), \vec{y}^{J+1}\rangle = b_i \langle (a_1\ a_2), \vec{y}^{J+1}\rangle \tag{2.2}$$
$$= (a_1 b_i) K'_J \text{ where } K_J := \frac{a_2}{a_1} \tag{2.3}$$

Since each b_i is uniform and independent of the other b's, the queries are answered precisely as in hybrid H_J; the base g_q is chosen either independent of the previous g_i or as a linear combination of them according to $\mathsf{Span}_J(y_1, \dots, y_q)$ and exponentiated with partial key $K_{J+1} = \frac{a_2}{a_1}, K_{J+2}, \dots, K_\ell$.

In the case when C is a uniform matrix, the exponenet does not factor as above. The way in which the exponent defining g_q is computed essentially implements a truly random *bilinear* map on inputs $(\vec{y}^1 \otimes \dots \otimes \vec{y}^J \times \vec{y}^{J+1})$. This can be equivalently viewed as a truly random *linear* map on inputs $(\vec{y}^1 \otimes \dots \otimes \vec{y}^J \otimes \vec{y}^{J+1})$. This is exactly equal to hybrid H_{J+1}.

2.4 Decision Trees

We generalize the previous construction from DDH to support sets specified by polynomial-sized decision trees by observing that such decision trees can be written as disjoint unions of hypercubes.

A decision tree family $\mathcal{T}_\lambda$ of size $p(\lambda)$ over $\ell(\lambda)$ variables consists of binary trees with at most $p(\lambda)$ nodes, where each internal node is labeled with a variable x_i for $i \in [\ell]$, the two outgoing edges of an internal node are labeled 0 and 1, and the leaves are labeled with 0 or 1. On input an $x \in \{0,1\}^\ell$, the computation of the decision tree starts from the root, and upon reaching an internal node n labeled by a variable x_i, takes either the 0-outgoing edge or the 1-outgoing edge out of the node n, depending on whether x_i is 0 or 1, respectively.

We now show how to construct a PRF family $\mathcal{F}_\ell = \{\mathcal{F}_{\ell,\lambda}\}_{\lambda>0}$ where each $\mathcal{F}_{\ell,\lambda}$ consists of functions that map $\mathcal{D}_\lambda := \{0,1\}^\ell$ to a group $\mathcal{G}_\lambda$, that supports aggregation over sets recognized by decision trees. That is, let $\mathcal{S}_\lambda = \{S \subseteq \{0,1\}^\ell : \exists$ a decision tree $T_S \in \mathcal{T}_\lambda$ that recognizes $S\}$.

Our construction uses a hypercube-aggregate PRF family $\mathcal{F}'_\ell$ as a sub-routine. First, we need the following simple lemma.

Lemma 2.4.1 (Decision Trees as Disjoint Unions of Hypercubes). *Let $S \subseteq \{0,1\}^\ell$ be recognized by a decision tree T_S of size $p = p(\lambda)$. Then, S is a disjoint union of at most p hybercubes $H_{y_1}, \ldots, H_{y_p}$, where each $y_i \in \{0,1,\star\}^\ell$ and $H_{y_i} = \mathsf{Match}(y_i)$. Furthermore, given T_S, one can in polynomial time compute these hypercubes.*

Given the lemma, Aggregate is simple: on input a set S represented by a decision tree T_S, compute the disjoint hypercubes $H_{y_1}, \ldots, H_{y_p}$. Run the hypercube aggregation algorithm to compute

$$g_i \leftarrow \mathsf{Aggregate}_{\mathcal{F}}(K, y_i)$$

and outputs $g := \prod_{i=1}^{p} g_i$.

Basing the construction on the hypercube-aggregate PRF scheme from Section 2.3, we get a decision tree-aggregate PRF based on the sub-exponential DDH assumption. The security of this PRF follows from Lemma 2.1.1 by an argument identical to the one in Section 2.3.

2.5 Read-once formulas

Read-once boolean formula provide a different generalization of hypercubes and they too admit an efficient aggregation algorithm for the Naor-Reingold PRF, with a similar security guarantee.

A boolean formula on ℓ variables is a circuit on $x = (x_1, \ldots, x_\ell) \in \{0,1\}^\ell$ composed of only AND, OR, and NOT gates. A *read-once boolean formula* is a boolean formula with fan-out 1, namely each input literal feeds into at most one gate, and each gate output feeds into at most one other gate.[8] Let R_λ be the family of all read-once boolean formulas over $\ell(\lambda)$ variables. Without loss of generality, we restrict these circuits to be in a standard form: namely, composed of fan-in 2 and fan-out 1 AND and OR gates, and any NOT gates occurring at the inputs.

In this form, the circuit for any read-once boolean formula can be identified with a labelled binary tree; we identify a formula by the label of its root C_ϕ. Nodes with zero children are variables or their negation, labelled by x_i or $\bar{x}_i$, while all other nodes have 2 children and represent gates with fan-in 2. For such a node with label C, its children have labels C_L and C_R. Note that each child is itself a read-once boolean formula on fewer inputs, and their inputs are disjoint Let the gate type of a node C be $\mathsf{type}(C) \in \{AND, OR\}$.

We describe a recursive aggregation algorithm for computing products of PRF values over all accepting inputs for a given read-once boolean formula C_ϕ. Looking forward, we require the formula to be read-once in order for the recursion to be correct. The algorithm described reduces to that of Section 2.3 in the case where ϕ describes a hypercube.

Construction

The aggregation algorithm for read-once Boolean formulas takes as input the PRF key $\vec{K} = (K_0, \ldots, K_\ell)$ and a formula $C_\phi \in R_\lambda$ where C_ϕ only reads the variables $x_1, \ldots, x_m$

[8]We allow a formula to ignore some inputs variables; this enables the model to express hypercubes directly.

for some $m \leq \ell$. We abuse notation and interpret C_ϕ to be a formula on both $\{0,1\}^\ell$ and $\{0,1\}^m$ in the natural way.

$$\mathsf{AGG}_{k,\prod}(C_\phi) = \prod_{x:C_\phi(x)=1} g^{K_0 \prod_{i\in[\ell]} K_i^{x_i}} \tag{2.4}$$

$$= g^{K_0 \sum_{x:C_\phi(x)=1} \prod_{i\in[\ell]} K_i^{x_i}} \tag{2.5}$$

$$= g^{K_0 \cdot A(C_\phi,1) \cdot \prod_{m<j\leq\ell}(1+K_i)} \tag{2.6}$$

where we define $A(C,1) := \sum_{\{x\in\{0,1\}^m : C(x)=1\}} \prod_{i\in[m]} K_i^{x_i}$. If $A(C,1)$ is efficiently computable, then **Aggregate** will simply compute it and return (3). To this end, we provide a recursive procedure for computing $A(C,1)$.

Generalizing the definition for any sub-formula C with variables named x_1 to x_m, define the values $A(C,0)$ and $A(C,1)$:

$$A(C,b) := \sum_{\{x\in\{0,1\}^m:\ C(x)=b\}} \prod_{i\in[m]} K_i^{x_i}.$$

Recursively compute $A(C,b)$ as follows:

- If C is a literal for variable x_i, then by definition:

$$A(C,0) = \begin{cases} 1 & \text{if } C = x_i \\ K_i & \text{if } C = \bar{x}_i \end{cases} \qquad A(C,1) = \begin{cases} K_i & \text{if } C = x_i \\ 1 & \text{if } C = \bar{x}_i \end{cases}$$

- Else, if $\mathsf{type}(C) = AND$: Let C_L and C_R be the children of C. By hypothesis, we can recursively compute $A(C_L,b)$ and $A(C_R,b)$ for $b \in \{0,1\}$. Compute $A(C,b)$ as:

$$A(C,1) = A(C_L,1) \cdot A(C_R,1)$$
$$A(C,0) = A(C_L,0) \cdot A(C_R,0) + A(C_L,1) \cdot A(C_R,0) + A(C_L,0) \cdot A(C_R,1)$$

- Else, $\mathsf{type}(C) = OR$: Let C_L and C_R be the children of C. By hypothesis, we can recursively compute $A(C_L, b)$ and $A(C_R, b)$ for $b \in \{0, 1\}$. Compute $A(C, b)$ as:

$$A(C,1) = A(C_L,1)\cdot A(C_R,1) + A(C_L,1)\cdot A(C_R,0) + A(C_L,0)\cdot A(C_R,1)$$
$$A(C,0) = A(C_L,0)\cdot A(C_R,0))$$

Lemma 2.5.1. *$A(C, b)$ as computed above is equal to $\sum_{\{x\in\{0,1\}^m:\ C(x)=b\}} \prod_{i\in[m]} K_i^{x_i}$*

Proof. For C a literal, the correctness is immediate. We must check the recursion for each $\mathsf{type}(C) \in \{AND, OR\}$ and $b \in \{0, 1\}$. We only show the case for $b = 1$ when C is an OR gate; the other three cases can be shown similarly.

Let $S_{b_L,b_R} = \{x = (x_L, x_R) : (C_L(x_L), C_R(x_R) = (b_L, b_R)\}$ be the set of inputs (x_L, x_R) to C such that $C_L(x_L) = b_L$ and $C_R(x_R) = b_R$. The set $\{x : C(x) = 1\}$ can be decomposed into the disjoint union $S_{0,1} \sqcup S_{1,0} \sqcup S_{1,1}$. Furthermore,

$$A(C,1) = \sum_{x\in S_{0,1}} \prod_{i\in[m]} K_i^{x_i} + \sum_{x\in S_{1,0}} \prod_{i\in[m]} K_i^{x_i} + \sum_{x\in S_{1,1}} \prod_{i\in[m]} K_i^{x_i}$$

Because C is read-once, the sets of inputs on which C_L and C_R depend are disjoint; this implies that $A(C_L, b_L)\cdot A(C_R, b_R) = \sum_{x\in S_{b_L,b_R}} \prod_{i\in[m]} K_i^{x_i}$, yielding the desired recursion. □

Theorem 2.5.2. *Let $\epsilon > 0$ be a constant, choose the security parameter $\lambda = \Omega(\ell^{1/\epsilon})$, and assume $(2^{\lambda^\epsilon}, 2^{-\lambda^\epsilon})$-hardness of the DDH assumption. Then, the collection of functions $\mathcal{F}_\lambda$ defined above is a secure aggregate PRF with respect to the subsets R_λ and the product aggregation function over the group $\mathcal{G}$.*

Proof. Correctness is immediate from Lemma 2.5.1, and Equation (3). Security follows from the decisional Diffie-Hellman assumption in much the same way it did in the case of bit-fixing functions.

□

Chapter 3

Connection to Learning

3.1 Preliminaries

Notation: For a probability distribution D over a set X, we denote by $x \leftarrow D$ to mean that x is sampled according to D, and $x \leftarrow X$ to denote uniform sampling form X. For an algorithm A and a function $\mathcal{O}$, we denote that A has oracle access to $\mathcal{O}$ by $A^{\mathcal{O}(\cdot)}$.

We recall the definition of a "concept class". In this section, we will often need to explicitly reason about the representations of the concept classes discussed. Therefore we make use of the notion of a "representation class" as defined by [KV94] alongside that of concept classes. This formalization enables us to discuss both these traditional learning models (namely, PAC and learning with membership queries) as well as the new models we present below. Our definitions are parametrized by $\lambda \in \mathbb{N}$.[1]

Definition 3.1.1 (Representation class [KV94]). *Let $K = \{K_\lambda\}_{\lambda \in \mathbb{N}}$ be a family of sets, where each $k \in K_\lambda$ has description in $\{0,1\}^{s_k(\lambda)}$ for some polynomial $s_k(\cdot)$. Let $X = \{X_\lambda\}_{\lambda \in \mathbb{N}}$ be a set, where each X_λ is called a* domain *and each $x \in X_\lambda$ has description in $\{0,1\}^{s_x(\lambda)}$ for some polynomial $s_x(\cdot)$. With each λ and each $k \in K_\lambda$, we associate a Boolean function $f_k : X_\lambda \to \{0,1\}$.*[2] *We call each such function f_k a* concept, *and k its* index *or its* description.

[1]When clear from the context, we will omit the subscript λ.

[2]This association is an efficient procedure for evaluating f_k. Concretely, we might consider that there is a universal circuit F_λ such that for each λ, $f_k(\cdot) = F_\lambda(k, \cdot)$.

For each λ, we define the concept class $C_\lambda = \{f_k : k \in K_\lambda\}$ *to be the set of all concepts with index in K_λ. We define the* representation class $C = \{C_\lambda\}$ *to be the union of all concept classes C_λ.*

This formalization allows us to easily associate complexity classes with concepts in learning theory. For example, to capture the set of all DNF formulas on λ inputs with size at most $p(\lambda)$ for a polynomial p, we will let $X_\lambda = \{0,1\}^\lambda$, and $K_\lambda^{p(\lambda)}$ be the set of descriptions of all DNF formulas on λ variables with size at most $p(\lambda)$ under some reasonable representation. Then a concept $f_k(x)$ evaluates the formula k on input x. Finally, $\mathsf{DNF}_\lambda^{p(\lambda)} = \{f_k : k \in K_\lambda^{p(\lambda)}\}$ is the concept class, and $\mathsf{DNF}^{p(\lambda)} = \{DNF_\lambda^{p(\lambda)}\}_{\lambda \in \mathbb{N}}$. $DNF^{p(\lambda)}$ is the representation class that computes all DNF formulas on λ variables with description of size at most $p(\lambda)$ in the given representation.

As a final observation, note that a Boolean-valued PRF family $\mathcal{F} = \{\mathcal{F}_\lambda\}$ where $\mathcal{F}_\lambda = \{f_k : X_\lambda \to \{0,1\}\}$ with keyspace $K = \{K_\lambda\}$ and domain $X = \{X_\lambda\}$ satisfies the syntax of a representation class as defined above. This formalization is useful precisely because it captures both PRF families and complexity classes, enabling lower bounds in various learning models.

In proving lower bounds for learning representation classes, it will be convenient to have a notion of containment for two representation classes.

Definition 3.1.2 ($\subseteq$). *For two representation classes $\mathcal{F} = \{\mathcal{F}_\lambda\}$ and $\mathcal{G} = \{\mathcal{G}_\lambda\}$ on the same domain $X = \{X_\lambda\}$, and with indexing sets $I = \{I_\lambda\}$ and $K = \{K_\lambda\}$ respectively, we say $\mathcal{F} \subseteq \mathcal{G}$ if for all sufficiently large λ, for all $i \in I_\lambda$, there exists $k \in K_\lambda$ such that $g_k \equiv f_i$.*

Informally, if a representation class contains a PRF family, then this class is hard to MQ-learn (as in [Val84]). We apply similar reasoning to more powerful learning models. For example, if $\mathcal{G}$ is the representation class $DNF^{p(\lambda)}$ as defined above, then $\mathcal{F} \subseteq DNF^{p(\lambda)}$ is equivalent to saying that for all sufficiently large λ, the concept class $\mathcal{F}_\lambda$ can be decided by a DNF on λ inputs of $p(\lambda)$ size.

We now recall some standard definitions.

Definition 3.1.3 (ϵ-approximation). *Let $f, h : X \to \{0,1\}$ be arbitrary functions. We say h* ϵ-approximates *f if $\Pr_{x \leftarrow X}[h(x) \neq f(x)] \leq \epsilon$.*

In general, ϵ-approximation may be considered under a general distribution on X, but we will consider only the uniform distribution in this work.

Definition 3.1.4 (*PAC* learning). *For a concept $f : X_\lambda \to \{0,1\}$, and a probability distribution D_λ over X_λ, the example oracle $EX(f, D_\lambda)$ takes no input and returns $(x, f(x))$ for $x \leftarrow D_\lambda$. An algorithm $\mathcal{A}$ is an (ϵ, δ)-PAC learning algorithm for representation class $\mathcal{C}$ if for all sufficiently large λ, $\epsilon = \epsilon(\lambda) > 0$, $\delta = \delta(\lambda) > 0$ and $f \in \mathcal{C}_\lambda$,*

$$\Pr[\mathcal{A}^{EX(f,D_\lambda)} = h : \ h \textit{ is an } \epsilon\textit{-approximation to } f] \geq 1 - \delta$$

Definition 3.1.5 (*MQ* learning). *For a concept $f : X_\lambda \to \{0,1\}$, the membership oracle $MEM(f)$ takes as input a point $x \in X_\lambda$ and returns $f(x)$. An algorithm $\mathcal{A}$ is an (ϵ, δ)-MQ learning algorithm for representation class $\mathcal{C}$ if for all sufficiently large λ, $\epsilon = \epsilon(\lambda) > 0, \delta = \delta(\lambda) > 0$, and $f \in \mathcal{C}_\lambda$,*

$$\Pr[\mathcal{A}^{MEM(f)} = h : \ h \textit{ is an } \epsilon\textit{-approximation to } f] \geq 1 - \delta$$

We consider only PAC learning *with uniform examples*, where D_λ is the uniform distribution over X_λ. In this case, MQ is strictly stronger than PAC: everything that is PAC learnable is MQ learnable.

Observe that for any $f : X_\lambda \to \{0,1\}$, either $h(x) = 0$ or $h(x) = 1$ will $\frac{1}{2}$-approximate f. Furthermore, if $\mathcal{A}$ is inefficient, f may be learned exactly. For a learning algorithm to be non-trivial, we require that it is *efficient* in λ, and that it at least *weakly* learns $\mathcal{C}$.

Definition 3.1.6 (Efficient- and weak- learning).

- *$\mathcal{A}$ is said to be* efficient *if the time complexity of $\mathcal{A}$ and h are polynomial in $1/\epsilon, 1/\delta$, and λ.*

- *$\mathcal{A}$ is said to* weakly *learn $\mathcal{C}$ if there exist some polynomials $p_\epsilon(\lambda), p_\delta(\lambda)$ for which $\epsilon \leq \frac{1}{2} - \frac{1}{p_\epsilon(\lambda)}$ and $\delta \leq 1 - \frac{1}{p_\delta(\lambda)}$.*

- *We say a representation class is* learnable *if it is both efficiently and weakly learnable. Otherwise, it is* hard to learn.

Lastly, we recall the efficiently recognizable ensembles of sets as defined in Chapter 2. We occasionally call such ensembles indexed, or succinct. Throughout this section, we require this property of our set ensembles $\mathcal{S}$. Both the MQ_{RA} and AQ learning models that we present are defined with respect to $\mathcal{S} = \{\mathcal{S}_\lambda\}$, an efficiently recognizable ensemble of subsets of the domain X_λ.

3.2 Membership Queries with Restriction Access

In the PAC-with-Restriction Access model of learning of Dvir, et al [DRWY12], a powerful generalization of PAC learning is studied: rather than receiving random examples of the form $(x, f(x))$ for the concept f, the learning algorithm receives a random "restriction" of f - an implementation of the concept for a subset of the domain. Given this implementation of the restricted concept, the learning algorithm can both evaluate f on many related inputs, and study the properties of the restricted implementation itself. We consider an even stronger setting: instead of receiving random restrictions, the learner can adaptively request any restriction from a specified class $\mathcal{S}$. We call this model *membership queries with restriction access (MQ_{RA})*.

As a concrete example to help motivate and understand the definitions, we consider DNF formulas. For a DNF formula ϕ, a natural restriction might set the values of some of the variables. Consequently, some literals and clauses may have their values determined, yielding a simpler DNF formula ϕ' which agrees with ϕ on this restricted domain. This is the 'restricted concept' that the learner receives.

This model is quite powerful; indeed, decision trees and DNFs are efficiently learnable in the PAC-with-restriction-access learning model whereas neither is known to be learnable in plain PAC model [DRWY12]. Might this access model be too powerful or are there concepts that cannot be learned?

Looking forward, we will show that constrained PRFs correspond to hard-to-learn concepts in the $\mathrm{MQ}_{\mathrm{RA}}$ learning model. In the remainder, we will formally define the learning model, define constrained PRFs, and prove the main lower bound of this section.

3.2.1 $\mathrm{MQ}_{\mathrm{RA}}$ Learning

While the original restriction access model only discusses restrictions fixing individual input bits for a circuit, we consider more general notions of restrictions.

Definition 3.2.1 (Restriction). *For a concept $f : X_\lambda \to \{0,1\}$, a* restriction *$S \subseteq X_\lambda$ is a subset of the domain. The* restricted concept *$f|_S : S \to \{0,1\}$ is equal to f on S.*

While general restrictions can be studied, we consider the setting in which all restrictions S are in a specified set of restrictions $\mathcal{S}$. For a DNF formula ϕ, a restriction might be $S = \{x : x_1 = 1 \wedge x_4 = 0\}$. This restriction is contained in the set of 'bit-fixing' restrictions in which individual input bits are fixed. In fact, this class of restrictions is all that is considered in [DRWY12]; we generalize their model by allowing more general classes of restrictions.

In the previous example, a restricted DNF can be naturally represented as another DNF. More generally, we allow a learning algorithm to receive representations of restricted concepts. These representations are computed according to a Simplification Rule.[3]

Definition 3.2.2 (Simplification Rule). *For each λ, let $\mathcal{C}_\lambda = \{f_k : X_\lambda \to \{0,1\}\}_{k \in K_\lambda}$ be a concept class, $\mathcal{S}_\lambda$ an efficiently recognizable ensemble of subsets of X_λ, and $S \in \mathcal{S}_\lambda$ be a restriction. A* simplification *of $f_k \in C_\lambda$ according to S is the description $k_S \in K_\lambda$ of a concept f_{K_S} such that $f_{k_S} = f_k|_S$. A* simplification rule *for $\mathcal{C} = \{\mathcal{C}_\lambda\}$ and $\mathcal{S} = \{\mathcal{S}_\lambda\}$ is a mapping $\mathsf{Simp}_\lambda : (k, S) \mapsto k_S$ for all $k \in K_\lambda, S \in \mathcal{S}_\lambda$.*

[3] Whereas a DNF with some fixed input bits is naturally represented by a smaller DNF, when considering general representation classes and general restrictions, this is not always the case. Indeed, the simplification of f according to S may be in fact more complex. We use the term "Simplification Rule" for compatibility with [DRWY12].

In the PAC-learning with restriction access (PAC_{RA}) learning model considered in [DRWY12], the learner only receives random restrictions. Instead, we consider the setting where the learner can adaptively request any restriction from a specified class $\mathcal{S}$. This model – which we call *membership queries learning with restriction access (MQ_{RA})* – is a strict generalization of PAC_{RA} for efficiently samplable distributions over restrictions (including all the positive results in [DRWY12]). Further observe that this strictly generalizes the membership oracle of MQ learning if $\mathcal{S}$ is such that for each x, it is easy to find a restriction S covering x.

In traditional learning models (PAC, MQ) it is trivial to output a hypothesis that $\frac{1}{2}$-approximates any concept f; a successful learning algorithm is required to learn substantially more than half of the concept. With restriction queries, the learning algorithm is explicitly given the power to compute on some fraction α of the domain. Consequently, outputting an $\epsilon \geq (\frac{1-\alpha}{2})$-approximation to f is trivial; we require a successful learning algorithm to do substantially better. This reasoning is reflected in the definition of *weak* $\mathrm{MQ_{RA}}$learning below.

Definition 3.2.3 (Membership queries with restriction access ($\mathrm{MQ_{RA}}$)). *In a given execution of an oracle algorithm $\mathcal{A}$ with access to a restriction oracle* Simp, *let $X_S \subseteq X_\lambda$ be the union of all restrictions $S \in \mathcal{S}_\lambda$ queried by $\mathcal{A}$. $\mathcal{S}$ is an efficiently recognizable ensemble of subsets of the domain X_λ.*

An algorithm $\mathcal{A}$ is an $(\epsilon, \delta, \alpha)$-$\mathrm{MQ_{RA}}$learning algorithm *for representation class $\mathcal{C}$ with respect to a restrictions in $\mathcal{S}$ and simplification rule* Simp *if, for all sufficiently large λ, for every $f_k \in \mathcal{C}_\lambda$, $\Pr[\mathcal{A}^{\mathsf{Simp}(k,\cdot)} = h] \geq 1 - \delta$ where h is an ϵ-approximation to f, – and furthermore – $|X_S| \leq \alpha |X_\lambda|$.*

$\mathcal{A}$ is said to weakly $\mathrm{MQ_{RA}}$-learn *if $\alpha \leq 1 - \frac{1}{p_\alpha(\lambda)}$, $\epsilon \leq (1-\alpha)(\frac{1}{2} - \frac{1}{p_\epsilon(\lambda)})$, $\delta \leq 1 - \frac{1}{p_\delta(\lambda)}$ for some polynomials $p_\alpha, p_\epsilon, p_\delta$.*

3.2.2 Constrained PRFs

We look to constrained pseudorandom functions for hard-to-learn concepts in the restriction access model. To support the extra power of the restriction access model, our PRFs will need to allow efficient evaluation on restrictions of the domain while maintaining some hardness on the remainder. Constrained PRFs [KPTZ13, BGI14, BW13] provide just this power. For showing hardness of restriction access learning, the constrained keys will correspond to restricted concepts; the strong pseudorandomness property will give the hardness result.

Syntax A family of functions $\mathcal{F} = \{F_\lambda : K_\lambda \times X_\lambda \to Y_\lambda\}$ is said to be *constrained* with respect to a set system $\mathcal{S}$, if it supports the additional efficient algorithms:

- $\mathit{Constrain}_\lambda(k, S)$: A randomized algorithm, on input $(k, S) \in K_\lambda \times \mathcal{S}_\lambda$, outputs a *constrained key* k_S. We $\tilde{K}_\lambda \triangleq \mathit{Support}(\mathit{Constrain}(k, S))$ the set of all constrained keys.

- $\mathit{Eval}_\lambda(k_S, x)$: A deterministic algorithms taking input $(k_S, x) \in \tilde{K}_\lambda \times X_\lambda$, and satisfying the following correctness guarantee:

$$\mathit{Eval}(\mathit{Constrain}(k, S), x) = \begin{cases} F(k, x) & \text{if } x \in S \\ \perp \notin Y & \text{otherwise.} \end{cases}$$

Security Game

- C picks a random key $k \in K_\lambda$ and initializes two empty subsets of the domain: $C, V = \emptyset$. C and V are subsets of X_λ which must satisfy the invariant that $C \cap V = \emptyset$. C will keep track the inputs $x \in X_\lambda$ to the Challenge oracle, and V will be the union of all sets S queries to Constrain plus all points $x \in X_\lambda$ to the Eval oracle.

- C picks $b \in \{0, 1\}$ to run EXP(b), and exposes the following three oracles to A :

  Eval(x): On input $x \in X_\lambda$, outputs $F(k, x)$. $V \leftarrow V \cup \{x\}$.

  Constrain(S): On input $S \in \mathcal{S}_\lambda$, outputs k_S. $V \leftarrow W \cup S$.

Challenge(x): On input $x \in X_\lambda$, outputs:

$$\begin{array}{ll} F(k,x) & \text{in EXP}(0) \\ y \leftarrow Y_\lambda & \text{in EXP}(1) \end{array}.$$

In EXP(1), the responses to Challenge are selected uniformly at random from the range, with the requirement that the responses be consistent for identical inputs x.

- The adversary queries the oracles with the requirement that $C \cap V = \emptyset$, and outputs a bit $b' \in \{0, 1\}$.

Definition 3.2.4. *The advantage is defined as* $ADV_\lambda^{cPRF}(\mathsf{A}) := \Pr[b' = b]$ *in the above security game.*

Definition 3.2.5 (Constrained PRF (cPRF)). *A family of functions* $\mathcal{F} = \{F_\lambda : K_\lambda \times X_\lambda \to Y_\lambda\}$ *constrained with respect to* $\mathcal{S}$ *is a* constrained PRF *if for all probabilistic polynomial-time adversaries* A *and for all sufficiently large* λ *and all polynomials* $p(n)$*:*

$$ADV_\lambda^{cPRF}(\mathsf{A}) < \frac{1}{2} + \frac{1}{p(n)},$$

over the randomness of C *and* A.

3.2.3 Hardness of Restriction Access Learning

We will now prove that if a constrained PRF $\mathcal{F}$ with respect to set system $\mathcal{S}$ is computable in representation class $\mathcal{C}$, then $\mathcal{C}$ hard to MQ_{RA}-learn with respect to $\mathcal{S}$ and some simplification rule.

Theorem 3.2.1. *Let* $\mathcal{F} = \{\mathcal{F}_\lambda\}$ *be a Boolean-valued constrained PRF (also interpreted as a representation class) with respect to sets* $\mathcal{S}$ *and key-space* K*. Let* $EVAL = \{EVAL_\lambda\}$ *be a representation class where each* $EVAL_\lambda$ *is defined as:*

$$EVAL_\lambda = \{g_{k_S}(\cdot) : \; g_{k_S}(x) = \mathsf{PRF.Eval}(k_S, x)\}.$$

Namely, each concept in the class $EVAL_\lambda$ is indexed by $k_S \in \tilde{K}_\lambda$ and has X_λ as its domain. For any representation class $\mathcal{C} = \{\mathcal{C}_\lambda\}$ such that $\mathcal{F} \subseteq \mathcal{C}$ and $EVAL \subseteq \mathcal{C}$, there exists a simplification rule Simp *such that $\mathcal{C}$ is hard to MQ_{RA}-learn with respect to the set of restrictions $\mathcal{S}$ and the simplification rule* Simp.

Existing constructions of constrained PRFs [BW13] yield the following corollaries:

Corollary 3.2.2. *Let $n = n(\lambda)$ be a polynomial, and assume that for the $n+1$-MDDH problem, every adversary time* $\text{poly}(\lambda)$ *the advantage is at most $\epsilon(\lambda)/2^n$. Then there is a simplification rule such that NC^1 is hard to MQ_{RA}-learn with respect to restrictions in $\mathcal{HC}$*[4].

Corollary 3.2.3. *Assuming the existence of one-way functions, there is a simplification rule such that P/poly is hard to MQ_{RA}-learn with respect to restrictions in $\mathcal{S}_{[a,b]}$* [5].

Remarks: The Simplification Rule here is really the crux of the issue. In our theorem, *there exists* a simplification rule under which we get a hardness result. This may seem somewhat artificial. On the other hand, this implies that the restriction-access learnability (whether PAC- or MQ-RA) of a concept class crucially depends on the simplification rule, as the trivial simplification rule of $Simp(k, S) = k$ admits a trivial learning-algorithm in either setting. This work reinforces that the choice simplification rule can affect the learnability of a given representation class. Positive results for restriction access learning that were independent of the representation would be interesting.

Proof of Theorem 3.2.1. We interpret $\mathcal{F} = \{\mathcal{F}_\lambda\}$ as a representation class. For each λ, the concepts $f_k \in \mathcal{F}_\lambda$ are indexed by K_λ and have domain X_λ. Let $EVAL = \{EVAL_\lambda\}$ be a representation class defined as in the theorem statement. The indexing set for $EVAL_\lambda$ is $\tilde{K}_\lambda$, the set of constrained keys k_S for $k \in K_\lambda, S \in \mathcal{S}_\lambda$.

Let $\mathcal{C} = \{\mathcal{C}_\lambda\}$ be a representation class, with domain X_λ and indexing set I_λ. For $i \in I_\lambda$, c_i is a concept in $\mathcal{C}_\lambda$.

[4]as defined in Chapter 2.
[5]as defined in Chapter 2.

By hypothesis, $\mathcal{F} \subseteq \mathcal{C}$: for sufficiently large λ, for all $k \in K_\lambda$ there exists $i \in I_\lambda$ such that $c_i \equiv f_k$. Similarly, for all $k_S \in \tilde{K}_\lambda$ there exists $i \in I_\lambda$ such that $c_i \equiv Eval_\lambda(k_S, \cdot)$. For concreteness, let M_λ be this map from $K_\lambda \cup \tilde{K}_\lambda$ to I_λ.[6]

We can now specify the simplification rule $\mathsf{Simp}_\lambda : I_\lambda \times \mathcal{S}_\lambda \to I_\lambda$. Letting $M_\lambda(K_\lambda) \subseteq I_\lambda$ be the image of K_λ under M_λ:

$$\mathsf{Simp}_\lambda(i, S) = \begin{cases} M_\lambda(Constrain_\lambda(M_\lambda^{-1}(i), S)) & \text{if } i \in M_\lambda(K_\lambda) \\ i & \text{otherwise.} \end{cases}$$

For example, i may be a circuit computing the PRF f_k for some $k = M^{-1}(i)$. The simplification computes the circuit corresponding to a constrained PRF key, if the starting circuit already computes a member of the PRF family $\mathcal{F}_\lambda$.[7]

Reduction: Suppose, for contradiction, that there exists an such an efficient learning algorithm $\mathcal{A}$ for $\mathcal{C}$ as in the statement of the theorem. We construct algorithm $\mathcal{B}$ breaking the constrained PRF security. In the PRF security game, $\mathcal{B}$ is presented with the oracles $f_k(\cdot)$, $Constrain_\lambda(k, \cdot)$, and $Challenge_\lambda(\cdot)$, for some $k \leftarrow K_\lambda$. Run $\mathcal{A}$, and answer queries $S \in \mathcal{S}_\lambda$ to the restriction oracle by querying $Constrain_\lambda(k, S)$, receiving k_S, and returning $M_\lambda(k_S)$. Once $\mathcal{A}$ terminates, it outputs hypothesis h. By assumption on $\mathcal{A}$, with probability at least $1 - \delta > \frac{1}{p_\delta(\lambda)}$, the hypothesis h is an ϵ-approximation of $c_{M(k)} \equiv f_k$ with $\epsilon \leq \frac{1-\alpha}{2}$ and $\alpha < 1 - \frac{1}{p_\alpha(\lambda)}$.

After receiving hypothesis h, $\mathcal{B}$ estimates the probability $\Pr_{x \leftarrow X \setminus X_S}[h(x) = Challenge_\lambda(x)]$. In EXP(0), this probability is at least $1 - \epsilon$ with probability at least $1 - \delta$; in EXP(1), it is exactly $1/2$. To sample uniform $x \in X \setminus X_S$, we simply take a uniform $x \in X$: with probability $1 - \alpha \geq 1/p_\alpha(n)$, $x \in X \setminus X_S$. Thus, $\mathcal{B}$ runs in expected polynomial time. If the estimate is close to ϵ, guess EXP(0); otherwise, flip an fair coin $b' \in \{0,1\}$ and guess EXP(b'). The advantage ADV_λ^{cPRF} of $\mathcal{B}$ in the PRF security game is at least $\frac{1}{3p_\delta(\lambda)}$ for all

[6] This is a non-uniform reduction.

[7] Note that while the inverse map M_λ^{-1} may be inefficient, in our reduction, the concept in question is represented by a PRF key k. Thus $\mathcal{B}$ must only compute the forward map M_λ.

sufficiently large λ (see Analysis for details), directly violating the security of $\mathcal{F}$.

Analysis: Let $p_b \triangleq \Pr_{x \in X \setminus X_S}[h(x) \neq \mathit{Challenge}_\lambda(x) | EXP(b)]$ be the probability taken with respect to experiment EXP(b). In EXP(1), $\mathit{Challenge}_\lambda$ is a uniformly random function. Thus, $p_1 = \frac{1}{2}$. With high probability, $\mathcal{B}$ will output a random bit $b' \in \{0,1\}$, guessing correctly with probability 1/2.

In EXP(0), h is an ϵ-approximation to f_k, and thus to $\mathit{Challenge}_\lambda$, with probability at least $1-\delta$. In this case, $p_0 \geq 1 - \epsilon \geq \frac{1}{2} + \frac{1}{p_\epsilon(\lambda)}$. By a Hoeffding bound, $\mathcal{B}$ will guess $b' = 0$ with high probability by estimating p using only polynomial in $\lambda, p_\epsilon(\lambda)$ samples. On the other hand, if h is not an ϵ-approximation, $\mathcal{B}$ will $b' = 0$ with probability at least 1/2.

Let $negl(\lambda)$ be the error probability from the Hoeffding bound, which can be made exponentially small in λ. The success probability is: $\Pr[b = b' | b = 0] \geq (1-\delta)(1-negl(\lambda)) + \frac{\delta}{2}$ which, for $1 - \delta \geq \frac{1}{p_\delta(\lambda)}$ is at least $\frac{1}{3p_{\delta(\lambda)}} + \frac{1}{2}$ for sufficiently large λ. Thus $\mathcal{B}$ a non-negligible advantage of $1/3p_{\delta(\lambda)}$ in the constrained PRF security game. □

3.3 Learning with Related Concepts

The idea that some functions or concepts are related to one another is very natural. For a DNF formula, for instance, related concepts may include formulas where a clause has been added or formulas where the roles of two variables are swapped. For a decision tree, we could consider removing some accepting leaves and examining the resulting behavior. We might consider a circuit; related circuits might alter internal gates or fix the values of specific input or internal wires.

Formally, we consider indexed representation classes. As discussed in the preliminaries, general classes of functions are easily represented as a indexed family. For example, we may consider the bit representation of a function (say, a log-depth circuit) as an index into a whole class (NC^1). This formalism enables the study of related concepts by instead considering concepts whose keys are related in some way. The related concept setting shares an important property with the restriction access setting: different representations of the same functions might have very different properties. Exploring the properties of different representations – and perhaps their RC learnability as defined below – is a direction for future work.

In our model of learning with related concepts, we allow the learner to query a membership oracle for the concept $f_k \in \mathcal{C}_\lambda$ and also for some 'related' concepts $f_{\phi(k)} \in \mathcal{C}_\lambda$ for some functions ϕ. The *related-concept deriving (RCD)* function ϕ is restricted to be from a specified class, Φ_λ. For each $\phi \in \Phi_\lambda$, a learner can access the membership oracle for $f_{\phi(k)}$. For example: let $K_\lambda = \{0,1\}^\lambda$ and let

$$\Phi_\lambda^\oplus = \{\phi_\Delta : k \mapsto k \oplus \Delta\}_{\Delta \in \{0,1\}^\lambda} \tag{3.1}$$

Definition 3.3.1 (Φ-Related-Concept Learning Model). *For $\mathcal{C}$ a representation class indexed by $\{K_\lambda\}$, let $\Phi = \{\Phi_\lambda\}$, with each $\Phi_\lambda = \{\phi : K_\lambda \to K_\lambda\}$ a set of functions on K_λ containing the identity function* id_λ. *The related-concept oracle RC_k, on query (ϕ, x), responds with $f_{\phi(k)}(x)$, for all $\phi \in \Phi_\lambda$ and $x \in X_\lambda$.*

An algorithm A is an (ϵ,δ)-Φ-RC learning algorithm for a $\mathcal{C}$ if, for all sufficiently large λ, for every $k \in K_\lambda$, $\Pr[\mathcal{A}^{RK_k(\cdot,\cdot)} = h] \geq 1 - \delta$ *where h is an ϵ-approximation f_k.*

Studying the related-concept learnability of standard representation classes (ex: DNFs and decision trees) under different RCD classes Φ is an interesting direction for future study.

3.3.1 Related-Key Secure PRFs

Again we look to pseudorandom functions for hard-to-learn concepts. To support the extra power of the related concept model, our PRFs will need to maintain their pseudorandomness even when the PRF adversary has access to the function computed with related keys. Related-key secure PRFs [BC10, ABPP14] provide just this guarantee. As in the definition of RC learning, the security of related-key PRFs is given with respect to a class Φ of related-key deriving functions. As we describe in the remainder of the section, related-key secure PRFs prove hard to weakly Φ-RC learn.

Security Game Let $\Phi_\lambda \subseteq Fun(K_\lambda, K_\lambda)$ be a subset of functions on K_λ. The set $\Phi = \{\Phi_\lambda\}$ is called the *Related-Key Deriving (RKD)* class and each function $\phi \in \Phi_\lambda$ is an RKD function.

- C picks a random key $k \in K_\lambda$, a bit $b \in \{0,1\}$, and exposes the oracle according to EXP(b):

  $\text{RKFn}_\lambda(\phi, x)$: On input $(\phi, x) \in \Phi_\lambda \times X_\lambda$, outputs:

  $$\begin{array}{cl} F(\phi(k), x) & \text{in EXP}(0) \\ y \leftarrow Y_\lambda & \text{in EXP}(1) \end{array}.$$

  In EXP(1), the responses to RKFn_λ are selected uniformly at random from the range, with the requirement that the responses be consistent for identical inputs (ϕ, x).

- The adversary interacts with the oracle, and outputs a bit $b' \in \{0,1\}$.

  Definition 3.3.2. *The advantage is defined as* $ADV_\lambda^{\Phi\text{-}RKA}(\mathsf{A}) := \Pr[b' = b]$ *in the above security game.*

Definition 3.3.3 (Φ Related-key attack PRF (Φ-RKA-PRF)). *Let* $\mathcal{F} = \{F_\lambda : K_\lambda \times X_\lambda \to Y_\lambda\}$ *be family of functions and let* $\Phi = \{\Phi_\lambda\}$ *with each* $\Phi_\lambda \subseteq Fun(K_\lambda, K_\lambda)$ *be a set of functions on* K_λ. $\mathcal{F}$ *is a* Φ related-key attack PRF family *if for all probabilistic polynomial-time adversaries* A *and for all sufficiently large* λ *and all polynomials* $p(n)$*:*

$$ADV_\lambda^{\Phi\text{-}RKA}(\mathsf{A}) < \frac{1}{2} + \frac{1}{p(n)},$$

over the randomness of C *and* A.

3.3.2 Hardness of Related Concept Learning

In the Appendix C, we present a concept that can be RC-learned under $\Phi^{\oplus}$ (Equation 3.1), but is hard to weakly learn with access to membership queries. We construct the concept $\mathcal{F}$ from a PRF $\mathcal{G}$ and a PRP P. Informally, the construction works by hardcoding the the PRF key in the function values *on a related PRF*. With the appropriate related-concept access, a learner can learn the PRF key.

We now present a general theorem relating RKA-PRFs to hardness of RC learning. This connection yields hardness for a class $\mathcal{C}$ with respect to restricted classes of relation functions Φ. More general hardness results will require new techniques.

Theorem 3.3.1. *Let $\mathcal{F}$ be a boolean-valued Φ-RKA-PRF with respect to related-key deriving class Φ and keyspace K. For a representation class $\mathcal{C}$, if $\mathcal{F} \subseteq \mathcal{C}$, then there exists an related-concept deriving class Ψ such that $\mathcal{C}$ is hard to Ψ-RC.*

As a corollary, we get a lower bound coming from the RKA-PRF literature. For a group $(G, +)$, and $K = G^m$, define the the element-wise addition RKD functions as

$$\Phi_+^m = \{\phi_\Delta : k[1], \ldots, k[m] \mapsto k[1] + \Delta[1], \ldots, k[m] + \Delta[m]\}_{\Delta \in G^m} \tag{3.2}$$

Notice that Φ_+^m directly generalizes $\Phi^\oplus$ with $G = \mathbb{Z}_2$. For this natural RKD function family, we are able to provide a strong lower bound based on the hardness of DDH and the existence of collision-resistant hash functions using the RKA-PRF constructions from [ABPP14].

Corollary 3.3.2 (Negative Result from RKA-PRF). *If the DDH assumption holds and collision-resistant hash functions exist, NC^1 is hard to Φ_+^m-RKA-learn.*

Proof of Theorem 3.3.1. We interpret $\mathcal{F} = \{\mathcal{F}_\lambda\}$ as a representation class. For each λ, the concepts $f_k \in \mathcal{F}_\lambda$ are indexed by K_λ and have domain X_λ. Let $\mathcal{C} = \{\mathcal{C}_\lambda\}$ be a representation class, with domain X_λ and indexing set I_λ. For $i \in I_\lambda$, c_i is a concept in $\mathcal{C}_\lambda$.

By hypothesis, $\mathcal{F} \subseteq \mathcal{C}$: for sufficiently large λ, for all $k \in K_\lambda$ there exists $i \in I_\lambda$ such that $c_i \equiv f_k$. For concreteness, let M_λ be this map from K_λ to I_λ.[8]

We can now specify the RCD class $\Psi_\lambda : I_\lambda \to I_\lambda$. Let $M_\lambda(K_\lambda) \subseteq I_\lambda$ be the image of K_λ

[8] This is a non-uniform reduction in general, but in most cases, the map M is known. That is, M_λ is the map that takes a key and outputs a circuit computing the function.

under M_λ. We define $\Psi_\lambda = \{\psi_\phi : \phi \in \Phi_\lambda\}$:

$$\psi_\phi(i) = \begin{cases} M_\lambda \circ \phi \circ M_\lambda^{-1}(i) & \text{if } i \in M_\lambda(K_\lambda) \\ i & \text{otherwise.} \end{cases}$$

Reduction Suppose, for contradiction, that there exists an efficient Ψ-RC learning algorithm $\mathcal{A}$ for $\mathcal{C}$ as in the statement of the theorem. We construct algorithm $\mathcal{B}$ breaking the Φ-RKA-PRF security of $\mathcal{F}$. In the PRF security game, $\mathcal{B}$ is presented with the oracle $RKFn(\cdot,\cdot)$; $\mathcal{A}$ is presented with the oracle $RC(\cdot,\cdot)$. Run $\mathcal{A}$, and answer queries $(\psi_\phi, x) \in \Psi_\lambda \times X_\lambda$ to RC by querying $RKFn$ on (ϕ, x) and passing the response along to $\mathcal{A}$. Let $X_{\mathcal{A}} = \{x \in X_\lambda : \mathcal{A} \text{ queried } (\psi, x) \text{ for some } \psi\}$. Once $\mathcal{A}$ terminates, it outputs hypothesis h. In EXP(0), $RKFn()$ responds according to f_k for some $k \in K\lambda$; in this case, $\mathcal{B}$ simulates the RC oracle for the concept $c_{M(k)}$.

After receiving hypothesis h, $\mathcal{B}$ estimates the probability $\Pr_{x \leftarrow X \setminus X_{\mathcal{A}}}[h(x) = RKFn_\lambda(x)]$. In EXP(0), this probability is at least $1-\epsilon$ with probability at least $1-\delta$; in EXP(1), it is exactly 1/2. To sample uniform $x \in X \setminus X_{\mathcal{A}}$, we simply take a uniform $x \in X$: with high probability $x \in X \setminus X_{\mathcal{A}}$. If the estimate is close to ϵ, guess EXP(0); otherwise, flip an fair coin $b' \in \{0,1\}$ and guess EXP(b'). The advantage $ADV_\lambda^{\Phi\text{-}RKA}$ of $\mathcal{B}$ in the PRF security game is at least $\frac{1}{3p_\delta(n)}$ (see Analysis for details) for all sufficiently large λ, directly violating the security of $\mathcal{F}$.

Analysis Let $p_b \triangleq \Pr_{x \in X \setminus X_{\mathcal{A}}}[h(x) \neq RKFn(\mathrm{id}_\lambda, x) | EXP(b)]$ be the probability taken with respect to experiment EXP(b). In EXP(1), $RKFn$ is a uniformly random function. Thus, $p_1 = \frac{1}{2}$. With high probability, $\mathcal{B}$ will output a random bit $b' \in \{0,1\}$, guessing correctly with probability 1/2.

In EXP(0), h is an ϵ-approximation to $RKFn(\mathrm{id}, \cdot)$ with probability at least $1-\delta$. In this case, $p_0 \geq 1-\epsilon \geq \frac{1}{2} + \frac{1}{p_\epsilon(\lambda)}$. By a Hoeffding bound, $\mathcal{B}$ will guess $b' = 0$ with high probability by estimating p using only polynomial in $\lambda, p_\epsilon(\lambda)$ samples. On the other hand, if h is not an ϵ-approximation, $\mathcal{B}$ will $b' = 0$ with probability at least 1/2.

Let $negl(\lambda)$ be the error probability from the Hoeffding bound, which can be made exponentially small in λ. The success probability is: $\Pr[b = b'|b = 0] \geq (1-\delta)(1-negl(\lambda))+\frac{\delta}{2}$ which, for $1 - \delta \geq \frac{1}{p_\delta(\lambda)}$ is at least $\frac{1}{3p_{\delta(\lambda)}} + \frac{1}{2}$ for sufficiently large λ. Thus $\mathcal{B}$ a non-negligible advantage of $1/3p_{\delta(\lambda)}$ in the Φ-RKA-PRF security game. □

Proof. For $n \in \mathbb{N}$ let $\mathbb{G} = \langle g \rangle$ be a group of prime order $p = p(n)$, $X_n = \{0,1\}^{m(n)} \setminus \{0^n\}$, $K_n = \mathbb{Z}_p^m(n)$, and define $F_k(x)$ as in Theorem 4.5 of [Abdalla] (). Let ϕ_+^m be as above over K. □

3.4 Learning with Aggregate Queries

This computational learning model is inspired by our aggregate PRFs. Rather than being a natural model in its own right, this model further illustrates how cryptography and learning are in some senses duals. Here, we consider a new extension to the power of the learning algorithm. Whereas membership queries are of the form "What is the label of an example x?", we grant the learner the power to request the evaluation of simple functions on tuples of examples $(x_1, ..., x_k)$ such as "How many of $(x_1 ... x_k)$ are in C?" or "Compute the product of the labels of $(x_1, ..., x_k)$?". Clearly, if k is polynomial then this will result only a polynomial gain in the query complexity of a learning algorithm in the best case. Instead, we propose to study cases when k may be super polynomial, but the description of the tuples is succinct. For example, the learning algorithm might query the number of x's in a large interval that are positive examples in the concept.

As with the restriction access and related concept models – and the aggregate PRFs we define in this work – the Aggregate Queries (AQ) learning model will be considered with restrictions to both the types of aggregate functions Γ the learner can query, and the sets $\mathcal{S}$ over which the learner may request these functions to be evaluated on. We now present the AQ learning model informally:

Definition 3.4.1 (($\Gamma, \mathcal{S}$)-Aggregate Queries (AQ) Learning). *Let $\mathcal{C}$ be a representation class with domains $X = \{X_\lambda\}$, and $\mathcal{S} = \{\mathcal{S}_\lambda\}$ where each $\mathcal{S}_\lambda$ is a collection of efficiently recog-*

nizeable subsets of the X_λ. $\Gamma : \{0,1\}^* \to V_\lambda$ *be an aggregation function [as in def:]. Let* $\mathsf{AGG}_k^\lambda \triangleq \mathsf{AGG}_{f_k,\mathcal{S}_\lambda,\Gamma_\lambda}^\lambda$ *be the aggregation oracle for* $f_k \in \mathcal{C}_\lambda$*, for* $S \in \mathcal{S}_\lambda$ *and* Γ_λ.

An algorithm $\mathcal{A}$ *is an* (ϵ,δ)*-*$(\Gamma,\mathcal{S})$*-AQ learning algorithm for* $\mathcal{C}$ *if, for all sufficiently large* λ*, for every* $f_k \in \mathcal{C}_\lambda$, $\Pr[\mathcal{A}^{MEM_{f_k}(\cdot),\mathsf{AGG}_{f_k}^\lambda(\cdot)} = h] \geq 1-\delta$ *where* h *is an* ϵ*-approximation to* f_k.

3.4.1 Hardness of Aggregate Query Learning

Theorem 3.4.1. *Let* $\mathcal{F}$ *be a boolean-valued aggregate PRF with respect to set system* $\mathcal{S} = \{\mathcal{S}_\lambda\}$ *and accumulation function* $\Gamma = \{\Gamma_\lambda\}$*. For a representation class* $\mathcal{C}$*, if* $\mathcal{F} \subseteq \mathcal{C}$*, then* $\mathcal{C}$ *is hard to* $(\Gamma,\mathcal{S})$*-AQ learn.*

Looking back to our constructions of aggregate pseudorandom function families from the prequel, we have the following corollaries.

Corollary 3.4.2. *The existence of one-way functions implies that P/poly is hard to* $(\sum, \mathcal{S}_{[a,b]})$*-AQ learn, with* $\mathcal{S}_{[a,b]}$ *the set of sub-intervals of the domain as defined in section 2.2 [GGM86].*

Corollary 3.4.3. *The DDH Assumption implies that* NC^1 *is hard to* $(\sum, \mathcal{DT})$*-AQ learn, with* $\mathcal{S}_{[a,b]}$ *the set of polynomial-sized decision trees as defined in section 2.4 [NR04].*

Corollary 3.4.4. *The subexponential DDH Assumption implies that* NC^1 *is hard to* $(\prod, \mathcal{R})$*-AQ learn, with* $\mathcal{R}$ *the set of read-once boolean formulas defined in section 2.5 [NR04].*

Proof of Theorem 3.4.1. Interpreting $\mathcal{F}$ itself as a concept class, we will show an efficient reduction from violating the pseudorandomness property of $\mathcal{F}$ to weakly $(\Gamma,\mathcal{S})$-AQ learning $\mathcal{F}$. By assumption, $\mathcal{F} \subseteq \mathcal{C}$, implying that $\mathcal{C}$ is hard to learn as well.

Reduction Suppose for contradiction that there exists an efficient weak learning algorithm $\mathcal{A}$ for $\mathcal{F}$. We define algorithm $\mathcal{B}$ violating the aggregate PRF security of $\mathcal{F}$. In the PRF security game, $\mathcal{B}$ is presented with two oracles: $F(\cdot)$ and AGG_F^λ for a function F chosen according to the secret bit $b \in \{0,1\}$. In EXP(0), $F = f_k$ for random $k \in K_\lambda$; by assumption $f_k \in \mathcal{C}_\lambda$. In EXP(1), F is a uniformly random function from X to $\{0,1\}$. The learning algorithm $\mathcal{A}$

is presented with precisely the same oracles. $\mathcal{B}$ runs $\mathcal{A}$, simulating its oracles by passing queries and responses to its own oracles. $X_{\mathcal{A}} = \{x \in X_\lambda : \mathcal{A} \text{ queried } (\psi, x) \text{ for some } \psi\}$. Once $\mathcal{A}$ terminates, it outputs hypothesis h.

Analysis After receiving hypothesis h, $\mathcal{B}$ estimates the probability

$$p = \Pr_{x \leftarrow X \setminus X_{\mathcal{A}}} [h(x) = F(x)]$$

(using polynomial in $\lambda, p_\epsilon(\lambda)$ samples). In EXP(0), this probability is at least $1 - \epsilon$ with probability at least $1 - \delta$; in EXP(1), it is exactly 1/2. To sample uniform $x \in X \setminus X_{\mathcal{A}}$, we simply take a uniform $x \in X$: with high probability $x \in X \setminus X_{\mathcal{A}}$. If the estimate is close to ϵ, guess EXP(0); otherwise, flip an fair coin $b' \in \{0, 1\}$ and guess EXP(b'). The advantage ADV_λ^{APRF} of $\mathcal{B}$ in the PRF security game is at least $\frac{1}{3p_\delta(n)}$ for all sufficiently large λ (as shown below), directly violating the security of $\mathcal{F}$.

Let

$$p_b \triangleq \Pr_{x \in X \setminus X_{\mathcal{A}}} [h(x) \neq F(x) | EXP(b)]$$

be the probability taken with respect to experiment EXP(b). In EXP(1), F is a uniformly random function. Thus, $p_1 = \frac{1}{2}$. With high probability, $\mathcal{B}$ will output a random bit $b' \in \{0, 1\}$, guessing correctly with probability 1/2.

In EXP(0), h is an ϵ-approximation to F with probability at least $1 - \delta$. In this case, $p_0 \geq 1 - \epsilon \geq \frac{1}{2} + \frac{1}{p_\epsilon(\lambda)}$. By a Hoeffding bound, $\mathcal{B}$ will guess $b' = 0$ with high probability by estimating p using only polynomial in $\lambda, p_\epsilon(\lambda)$ samples. On the other hand, if h is not an ϵ-approximation, $\mathcal{B}$ will $b' = 0$ with probability at least 1/2.

Let negl(λ) be the error probability from the Hoeffding bound, which can be made exponentially small in λ. The success probability is:

$$\Pr[b = b' | b = 0] \geq (1 - \delta)(1 - \text{negl}(\lambda)) + \frac{\delta}{2}$$

which, for $1 - \delta \geq \frac{1}{p_\delta(\lambda)}$ is at least $\frac{1}{3p_{\delta(\lambda)}} + \frac{1}{2}$ for sufficiently large λ. Thus $\mathcal{B}$ a non-negligible advantage of $1/3p_{\delta(\lambda)}$ in the $(\Gamma, \mathcal{S})$-aggregate-PRF security game. $\square$

Chapter 4

Unobfuscatable PRFs and Their Implications

In the third part of this thesis, we explore constructions of function families that are pseudorandom, but "learnable" in a non-black-box setting, and as a result show negative results for puncturing and watermarking of PRFs. We begin by exploring the simpler setting of puncturing, then define watermarking schemes and finally demonstrate the construction of unwatermarkable PRFs.

4.1 Unpuncturable Pseudorandom Functions

In this section, we construct a family of PRFs that is not point-puncturable by proving the following theorem.

Theorem 4.1.1. *Assuming the existence of one-way functions, there exists a family of pseudorandom functions $\mathcal{F} = \{\mathcal{F}_\lambda\}_{\lambda \in \mathbb{N}}$ with $\mathcal{F}_\lambda = \{f_k : \{0,1\}^{n(\lambda)} \to \{0,1\}^{\ell(\lambda)}\}_{k \in \{0,1\}^\lambda}$ for some polynomials n and ℓ, and an efficient algorithm L such that for any $k \in K_\lambda$ and any circuit g_k that agrees with f_k everywhere except on at most a single point:*

$$L(g_k) = k.$$

Corollary 4.1.2 (Unpuncturable PRF). *If pseudorandom function families exist, then there exist pseudorandom function families that are not point-puncturable.*

Since the proof of Theorem 4.1.1 is only a slight modification of the construction in [BGI+12], and the construction of unwatermarkable PRFs in the sequel is significantly more complex, we here present only a the main idea of the construction.

Proof idea. The constructions presented in [BGI+12] guarantee a PRF family for which access to any exact implementation enables recovery of the PRF key k. They extend the result to cover a very restricted form of approximate implementations: the implementation agrees with the original on any specific input x with high probability over some distribution of implementations. The circuit families they construct are formed from a number of sub-functionalities: C_k, D_k, and C'_k.

In their approximate-functionality setting, the sub-functionality D_k is robust to point-errors, but the sub-functionalities C and C' are not. The function C_k (respectively C'_k) returns some key-recovery information when evaluated at specific single hidden input and behaves pseudorandomly everywhere else; if the point-error occurs at this point, the key k cannot be recovered. The simple fix is to duplicate this trapdoor information. Replace C_k (resp. C'_k) with $C_{k,1}$, $C_{k,2}$, and $C_{k,3}$; on the hidden input these new functionalities will agree, but they will behave pseudorandomly on all other inputs. Then run the recovery algorithm using each pair of $C_{k,i}$ and $C'_{k,i}$, and output the majority.

Because errors are restricted to a single input, the majority will be the true PRF key k. □

4.2 Watermarking Schemes

Preliminaries. We will let $\{\mathbb{C}_\lambda\}_{\lambda\in\mathbb{N}}$ be the family of all circuits with domain $\mathcal{D}_\lambda$ and range $\mathcal{R}_\lambda$. We will let $\{\mathcal{C}_\lambda\}_{\lambda\in\mathbb{N}}$ be a particular family of circuits; that is $\mathcal{C}_\lambda \subseteq \mathbb{C}_\lambda$ for all $\lambda \in \mathbb{N}$.

Let $\rho : \mathbb{N} \to [0,1]$ be a function. For circuits $C, C' \in \mathbb{C}_\lambda$, we say that $C \sim_\rho C'$ if

$$\Pr_{x \leftarrow \mathcal{D}_\lambda}[C(x) \neq C'(x)] \leq \rho(\lambda)$$

We call such circuits "ρ-close". When ρ is not made explicit (ex: "a function 'approximates' or 'is close to' another"), we mean that there is some negligible function ρ for which the functions are ρ-close.

We refer to probabilistic polynomial time algorithms simply as "p.p.t. algorithms".

The definitions presented in this work are significantly weaker versions of those in [CHV15].[1]

4.2.1 Definition

Software watermarking is the process of embedding a "mark" in a program so that the marked program preserves functionality, and furthermore, it is impossible to remove the mark without destroying functionality.

Roughly speaking, in a watermarking scheme, there is a marking algorithm Mark that takes as input a circuit C and uses the mk to produce a marked program $\#C$[2] while the verification algorithm Verify uses the verification key vk to recognize legally marked programs. A watermarking scheme should satisfy three properties:

- *Approximately Functionality Preserving:* The program $\#C$ should agree with C on random inputs.

- *Unremovability:* We say that a watermark remover $\mathcal{R}$ succeeds given $\#C$ if she produces as program $\hat{C}$ that is approximately (functionally) equivalent to $\#C$ and yet, Verify fails to recognize $\hat{C}$ as a marked program. Unremovability says that this should be hard for polynomial-time removers $\mathcal{R}$.

[1]Specifically, we restrict our attention to negligible ρ and $\gamma = 1$. Furthermore, we consider a setting with no public verification, verification oracles, or mark oracle. The adversary can receive only a single challenge, nothing more.

[2]Marked programs are denoted as $\#C$ (in the case of circuits) in this paper.

- *Unforgeability:* The other side of the coin is unforgeability which requires that a forger $\mathcal{F}$ cannot succeed in producing a new marked programs.

A watermarking scheme $\mathcal{W}$ for a family of circuits $\mathcal{C} = \{\mathcal{C}_\lambda\}_{\lambda\in\mathbb{N}}$ is a tuple of p.p.t. algorithms $(\mathsf{Setup}, \mathsf{Mark}, \mathsf{Extract})$, where:

- $(\mathsf{mk}, \mathsf{vk}) \leftarrow \mathsf{Setup}(1^\lambda)$ is a p.p.t. key generation algorithm takes as input a security parameter $\lambda \in \mathbb{N}$ and outputs a *marking key* mk and a *verification key* vk.

- $C_\# \leftarrow \mathsf{Mark}(\mathsf{mk}, C)$ is a p.p.t. marking algorithm that takes as input the marking key mk and a circuit $C \in \mathbb{C}_\lambda$ and outputs a circuit $C_\#$.

- $b \leftarrow \mathsf{Verify}(\mathsf{vk}, C)$ is a p.p.t. algorithm which takes as input the (public) verification key vk and a (possibly marked) circuit $C \in \mathbb{C}$, and outputs either `accept` (1) or `reject` (0).

Note that while Mark and Verify take any circuit as input, we only require the correctness and security properties that follow to hold when the input to Mark is a circuit from the class $\mathcal{C} \subseteq \mathbb{C}$.

Correctness Properties. Having defined the syntax of a watermarking scheme, we now define the desired correctness properties. First, it is functionality preserving, namely marking a circuit C does not change its functionality too much. We formalize this by requiring that the marked and unmarked circuits agree on a $1 - \mathrm{negl}(\lambda)$ fraction of the domain. Secondly, we require that the verification algorithm almost always accepts a marked circuit.

Definition 4.2.1 (Functionality Preserving). *We say that a watermarking scheme* $(\mathsf{Setup}, \mathsf{Mark}, \mathsf{Verify})$ *is functionality preserving if there exists some negligible function* ρ *such that for all* $C \in \mathcal{C}$*:*

$$\mathsf{Mark}(\mathsf{mk}, C) \sim_\rho C$$

Definition 4.2.2 (Completeness). *A watermarking scheme* $(\mathsf{Setup}, \mathsf{Mark}, \mathsf{Verify})$ *is said to* *be* complete *if*

$$\Pr\left[\mathsf{Verify}(\mathsf{vk}, \mathsf{Mark}(\mathsf{mk}, C)) = 1 \;\middle|\; \begin{array}{l} \mathsf{mk}, \mathsf{vk} \leftarrow \mathsf{Setup}(1^\lambda) \\ C \in \mathcal{C} \end{array}\right] \geq 1 - \mathrm{negl}(\lambda)$$

Security Properties. We turn to the desired security properties of the watermarking scheme. We define a scheme's "uremovability" with respect to the following security game.

Game 4.2.1 (Unremovability). *First, the challenger generates* $(\mathsf{mk}, \mathsf{vk}) \leftarrow \mathsf{Setup}(1^\lambda)$*. The challenger then samples a circuit* C^* *uniformly from* $\mathcal{C}$ *and gives* $\#C^* = \mathsf{Mark}(\mathsf{mk}, C^*)$ *to the adversary,* $\mathcal{A}$*. Finally,* $\mathcal{A}$ *outputs a circuit* $\hat{C}$*.*

Our notion of unremovability is that no adversary can – with better than negligible probability – output a program that is δ-close to the challenge program but on which Verify returns zero with any noticeable probability.

Definition 4.2.3 (δ-Unremovable). *In the security game, we say that the adversary* δ-removes *if* $\Pr[(\hat{C} \sim_\delta \#C^*) \wedge (\mathsf{Verify}(\mathsf{vk}, \hat{C}) = 0)]$ *is non-negligible.* $\mathcal{A}$*'s* δ-removing advantage *is the probability that* $\mathcal{A}$ *δ-removes. The scheme is* δ-unremovable *if all p.p.t.* $\mathcal{A}$ *have negligible δ-removing advantage.*

We say a scheme is unforgeable if no adversary can – with better than negligible probability – output a circuit on which Verify returns 1 with non-negligible probability. Note that in this setting, the adversary gets only the security parameter as input.

Definition 4.2.4 (Unforgeable). *The scheme is* unforgeable *if all p.p.t.* $\mathcal{A}$*:*

$$\Pr\left[\mathsf{Verify}(\mathsf{vk}, \hat{C}) = 1 \;\middle|\; \begin{array}{l} \mathsf{mk}, \mathsf{vk} \leftarrow \mathsf{Setup}(1^\lambda), \\ \hat{C} \leftarrow \mathcal{A}(1^\lambda) \end{array}\right] < \mathrm{negl}(\lambda)$$

4.3 Unwatermarkable Families

A natural question is whether there are families of functions that for which there does not exist any watermarking scheme (waterproof). Barak et al [BGI+12] observed that general-purpose indistinguishability obfuscation rules out a notion of watermarking that *exactly* preserves functionality, but watermarking schemes that change functionality on even a negligible fraction of the domain are not.

In this section, we discuss a number of conditions sufficient to prove that a family of circuits cannot be watermarked. In this context, if a family is non-black-box learnable given access to an approximation of a circuit in the family, then the family is not watermarkable. Because it suffices to learn the family with an approximate implementation, we focus on non-black-box learnability. For such a family, from a challenge marked program the learning algorithm is able to recover the original (unmarked) program. This violates unremovability of a watermarking scheme.

We construct waterproof PRFs using techniques closely related to the unobfuscatable function families of [BGI+12] and [BP12].

Consider an indexed family of functions $\mathcal{F} = \{f_K\}$ where each function is indexed by the key K. In our setting, the learning algorithm will be given any circuit g agrees almost everywhere with f_K, a uniformly sampled function from the family. The (randomized) learner will then output some hypothesis function h. If h is sufficiently close to f_K, then we can conclude that the family $\mathcal{F}$ cannot be watermarked.

As a warm up, we begin with a very strong notion of learnability, in which the learning algorithm – here called an *extractor* – can not only output a hypothesis h which agrees with f_K on all inputs, but output the circuit f_K itself.

Definition 4.3.1 (Robustly Extractable Families). *Let $\mathcal{F} = \{\mathcal{F}_\lambda\}_{\lambda \in \mathbb{N}}$ be a circuit ensemble where $\mathcal{F}_\lambda$ is a collection $\{f_k\}_{k \in \{0,1\}^\lambda}$. We say that $\mathcal{F}$ is* robustly extractable *if there exists an efficient extractor E such that for all large enough $\lambda \in \mathbb{N}$ and random $k \leftarrow \{0,1\}^\lambda$, E extracts k from any circuit g such that $g \sim_{\mathrm{negl}(\lambda)} f_K$:*

$$\Pr[k \leftarrow E(g, 1^\lambda)] \textit{ is non-negligible.}$$

Theorem 4.3.1. *If $\mathcal{F}$ is robustly extractable, then there does not exist a δ-watermarking* *scheme* $(\mathsf{Setup}, \mathsf{Mark}, \mathsf{Verify})$.

Proof. For a functionality-preserving watermarking scheme for the family $\mathcal{F}$, for all circuits $f_k \in \mathcal{F}$, the marked program $\#f_k = \mathsf{Mark}(\mathsf{mk}, f_k)$ is negl-close to the original. If $\mathcal{F}$ is negl-robustly extractable, then given a challenge marked program, the extractor E outputs f_k with noticeable probability. Unless $\mathsf{Verify}(\mathsf{vk}, f_k) = 1$ with high probability, the extractor E violates even 0-unremovability. Otherwise we may trivially violate 1-unforgeability by simply outputting a random $f_k \leftarrow \mathcal{F}$ (without ever receiving a marked program). $\square$

Towards proving the main theorems of this section, we weaken Definition 4.3.1 in two ways. Combined together in Definition 4.3.2, these weaker notions of learnability will capture richer functiontionalities and allow us to construct a PRF family that cannot be watermarked (Theorem 4.3.3). The following discussion motivates the stronger definition and outlines the proof of the corresponding Theorem 4.3.2

Learnable versus extractable. What if the family is only "learnable," but not "extractable:" instead of outputting f_k itself, the learning algorithm $L(C)$ can only output a circuit h that was functionally equivalent to f_k? One might think that this is indeed sufficient to prove Theorem 4.3.1, but the proof encounters a difficulty.

As before, we run the learner on the challenge program to get $h = L(\#f_k)$; if $\mathsf{Verify}(\mathsf{vk}, h) = 0$ with noticeable probability, then unremovability is violated. On the other hand, if $\mathsf{Verify}(\mathsf{vk}, h) = 1$ with high probability, is unforgeability violated? In the extractable setting, it was possible to sample a program which verifies *without ever seeing a marked version*, simply by picking $f_k \leftarrow \mathcal{F}$. In the weaker learnable setting, we only know how to sample from this verifying distribution by evaulating $L(\#f_k)$ marked circuit. But the forger is not given access to marked circuits.

To get around this issue, we consider families that are learnable with *implementation independence*; that is, for any g and g' which are both approximations of $f_K \in \mathcal{F}$, the distributions $L(g)$ and $L(g')$ are statistically indistinguishable.[3] To complete the above

[3]Computational indistinguishability would also suffice. If indistinguishability obfuscation exists, then

proof, a forger will simply evaluate $h \leftarrow L(f_k)$ for random (unmarked) f_k (rather than on the marked $\# f_k$). Implementation independence of L guarantees that $\mathsf{Verify}(\mathsf{vk}, h) = 1$ with high probability.

Approximate versus exact learning. In Definition 4.3.1 (and the discussion above), we required that an algorithm learning a family $\mathcal{F}$ is able to exactly recover the functionality f_K, when given g that approximates f_k. What can we prove if $h = L(g)$ is only required to δ-approximate the original function f_k?

Though we cannot violate 0-unremovability, we might hope to violate δ-unremovability. For a functionality preserving watermarking scheme, when given a marked program $\# f_k$, the learning algorithm returns $h = L(\# f_k)$ which is a δ-approximation (with noticeable probability). By similar reasoning to Theorem 4.3.1, it must be that either $\mathsf{Verify}(\mathsf{vk}, h) = 1$ with high probability or δ-unremovability is violated. If L is implementation independent as above, we contradict unforgeability.

Definition 4.3.2 (Robustly, δ-Approximately Implementation Independent Learnable Families). *Let $\mathcal{F} = \{\mathcal{F}_\lambda\}_{\lambda \in \mathbb{N}}$ be a circuit ensemble where each family $\mathcal{F}_n = \{f_k\}_{k \in \{0,1\}^\lambda}$. We say that $\mathcal{F}$ is* robustly, δ-approximately learnable *if there exists an efficient learner L such that for all large enough $\lambda \in \mathbb{N}$, random $k \leftarrow \{0,1\}^\lambda$, and any circuit g such that $g \sim_{\mathrm{negl}(\lambda)} f_K$:*

$$\Pr[h \sim_\delta f_k : h \leftarrow L(g, 1^\lambda)] \textit{ is non-negligible.}$$

We say that L is implementation independent *if for all g_1 and g_2 that are both negligibly close to f_k, the distributions $L(g_1, 1^\lambda)$ and $L(g_2, 1^\lambda)$ are statistically close.*

Theorem 4.3.2. *If $\mathcal{F}$ is robustly, δ-approximately learnable with implementation independence, then there does not exist a δ-watermarking scheme for $\mathcal{F}$.*

Already, this rules out watermarking a large array of families. For instance, any family that is improperly PAC learnable cannot be watermarked. The main result of this section is that

an learning exact improper learning algorithm can be transformed into an computationally implementation independent one.

there exists a PRF family is learnable according to Definition 4.3.2.

Theorem 4.3.3. *Assuming one-way functions, there exists a pseudorandom function family $\mathcal{F}_\delta$ that is robustly, δ-approximately learnable with implementation independence, for any $\delta = \frac{1}{\mathrm{poly}(\lambda)}$.*

Corollary 4.3.4. *Assuming one-way functions, for any non-negligible function $\delta(\lambda)$, there is a family of pseudorandom functions that is not δ-watermarkable.*

4.4 Construction of Unwatermarkable PRFs

Our starting point is the constructions of unobfuscatable function families in [BGI$^+$12] and [BP12], and an understanding of those constructions will prove helpful towards understanding ours. The former work presents a construction of 0-robustly extractable PRF families; from any exact implementation, the key can be recovered. They extend this notion to a very weak form of approximate functionality. The latter work handles a very strong form of approximation: the approximate implementation must only agree on some constant fraction of the domain. They achieve this, they sacrifice the total learnability of the earlier construction, learning only a predicate of the PRF key instead of the key itself. We require a notion of approximation stronger than [BGI$^+$12] but weaker than [BP12], and a notion of learnability weaker than [BGI$^+$12] but stronger than [BP12], and achieve this by adapting techniques from both works.

4.4.1 Preliminaries

The construction requires an invoker randomizable pseudorandom function [BGI$^+$12] and a decomposable encryption scheme [BP12]. The following definitions and discussion are taken almost verbatim from the respective works.

Definition 4.4.1 (Invoker-Randomizable Pseudorandom Functions [BGI$^+$12]). *A function ensemble $\{f_k\}_{k\in\{0,1\}^*}$ such that $f_k : \{0,1\}^{n+m} \to \{0,1\}^m$, where n and m are polynomially* *related to $|k|$, is called an* invoker-randomizable pseudorandom function ensemble *if the following holds:*

1. *$\{f_k\}_{k\in\{0,1\}^*}$ is a PRF family.*

2. *For every k and $x \in \{0,1\}^n$, the mapping $r \mapsto f_k(x,r)$ is a permutation over $\{0,1\}^m$.*

Property 2 implies that, for every fixed k and $x \in \{0,1\}^n$, if r is chosen uniformly in $\{0,1\}^m$, then the value $f_k(x,r)$ is distributed uniformly (and independently of x) in $\{0,1\}^m$.

Lemma 4.4.1 ([BGI+12]). *If pseudorandom functions exist, then there exist invoker-randomizable pseudorandom functions.*

Definition 4.4.2 (Decomposable Encryption [BP12]). *An encryption scheme* $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ *is* decomposable *if there exists an efficient algorithm* pub *that operates on ciphertexts and satisfies the following conditions:*

1. *For a ciphertext c, $\mathsf{pub}(c)$ is independent of the plaintext and samplable; that is, there exists an efficient sampler* $\mathsf{PubSamp}$ *such that, for any secret key $sk \in \{0,1\}^n$:*

$$\mathsf{PubSamp}(1^n) \equiv \mathsf{pub}(\mathsf{Enc}_{sk}(0))) \equiv \mathsf{pub}(\mathsf{Enc}_{sk}(1))$$

2. *A ciphertext c is deterministically defined by $\mathsf{pub}(c)$ and the plaintext; that is, for every secret key sk and two distinct ciphertexts c and c', if $\mathsf{pub}(c) = \mathsf{pub}(c')$, then $\mathsf{Dec}_{sk}(c) \neq \mathsf{Dec}_{sk}(c')$.*

We use as our decomposable encryption scheme a specific symmetric-key encryption scheme which enjoys a number of other necessary properties. Given a PRF $\{f_k\}_{k\in\{0,1\}^*}$ with one-bit output, for security parameter n, the secret key is a random $sk \in \{0,1\}^n$, and the encryption of a bit b is computed by sampling a random $r \leftarrow \{0,1\}^n$ and outputting $(r, F_{sk}(r) \oplus b)$. We encrypt multi-bit strings by encrypting each bit in turn. Besides being decomposable, this encryption scheme satisfies a number of necessary properties [BP12]:

- It is CCA-1 secure.
- It is decomposable.
- The support of $(\mathsf{Enc}_{sk}(0))$ and $(\mathsf{Enc}_{sk}(1))$ are each a non-negligible fraction (in reality, at least $\frac{1}{2}$ − negl) of the cipher-text space.
- For a fixed secret key sk, random samples from $(b, \mathsf{Enc}_{sk}(b))_{b \leftarrow \{0,1\}}$ are indistinguishable from uniformly random strings.

4.4.2 Construction

The key k for the PRF is given by a tuple $k = (\alpha, \beta, sk, s_1, s_2, s_e, s_h, s_b, s^*)$. For security parameter λ, α and β are uniformly random λ-bit strings, sk is a secret key for the decomposable encryption scheme described above, s_h is a key for an invoker-randomizable pseudorandom function, and s_1, s_2, s_e, s_b, and s^* are independent keys for a family of PRFs. We denote by F_s a PRF with key s.

The domain of the PRF will be of the form (i, q) for $i \in \{1, \dots, 9\}$, and $q \in \{0,1\}^{\ell(n)}$, for some polynomial ℓ. The range is similarly bit strings of length polynomial in ℓ. The function will be defined in terms of 9 auxiliary functions, and the index i will select among them. We use a combination of ideas from [BGI+12] and [BP12] to construct a PRF family for which s^* can be recovered from any (negligibly-close) approximation to f_k, which will enable us to compute f_k restricted to $i = 9$. This allows us to recover a 1/9-close approximation of f_k that is implementation independent (simply by returning 0 whenever $i \neq 9$). To achieve a δ-close approximation for any $\delta = \frac{1}{\mathrm{poly}(\lambda)}$, we simply augment the index i with an additional $\log(\delta)$ bits: if all these bits are 0, then we index as before; otherwise, use index $i = 9$. Instead of recovering 1/9th of the function, we now recover $1 - \delta$ of the function. This establishes the theorem.[4]

[4]Note that the result is a PRF family that depends on the choice of δ. The argument would fail if δ was a negligible function, because an approximation for could "erase" all the structure of the PRF family, thwarting learnability. Removing this dependence (ie: constructing a family that works for all inverse polynomial δ simultaneously) would be interesting.

We now define the auxiliary functionalities we will use in the construction.

- $\mathbb{R}_s$: The function $\mathbb{R}_s$ is parameterized by a PRF key s. It takes as input q and returns $\mathbb{R}_s(q) = F_s(q)$, the PRF evaluated at q. That is, $\mathbb{R}_s$ simply evaluates a PRF.

- $\mathbb{C}_{a,b,s}$: The function $\mathbb{C}_{a,b,s}$ is parameterized by two bit strings a and b, and a PRF key s. It takes as input q and returns $\mathbb{C}_{a,b,s}(q) = b \oplus F_s(q \oplus a)$, where F_s is the PRF given by key s. That is, $\mathbb{C}$ evaluates a PRF on a point related to the queried point, then uses the value to mask the bitstring b.

- $\mathbb{E}_{sk,\alpha,s_e}$: The function $\mathbb{E}_{sk,\alpha,s_e}$ is parameterized by a secret key sk for the encryption scheme, a bitstring α, and a PRF key s_E. It takes as input q and returns $\mathbb{E}_{sk,\alpha,s_e}(q) = \mathsf{Enc}_{sk}(\alpha; r)$ with randomness $r = F_{s_e}(q)$. That is, $\mathbb{E}$ returns an encryption of α using randomness derived by evaluating the PRF on the query.

- $\mathbb{H}_{sk,s_h}$: The function $\mathbb{H}_{sk,s_h}$ is parameterized by a secret key sk for the encryption scheme, and a invoker-randomizable PRF key s_h. It takes as input two ciphertexts of bits c and d, the description of a two-bit gate $\odot$, and some additional input $\bar{q}$, and returns $\mathbb{H}_{sk,s_h}(c, d, \odot, \bar{q}) = \mathsf{Enc}_{sk}(\mathsf{Dec}_{sk}(c) \odot \mathsf{Dec}_{sk}(d); r)$ with randomness $r = F_{s_h}(c, d, \odot, \bar{q})$. That is, $\mathbb{H}$ implements a homomorphic evaluation of $\odot$ on the ciphertexts c and d by decrypting and re-encrypting, with randomness derived by applying a PRF to the whole input.

- $\mathbb{B}_{sk,\alpha,\beta,s_b}$: The function $\mathbb{B}_{sk,\alpha,\beta,s_b}$ is parameterized by a secret key sk for the symmetric-key encryption scheme, bitstrings α and β, and a PRF key s_b. It takes as input n ciphertexts $c_1, \ldots, c_\lambda$ and additional input $\bar{q}$, and returns

$$\mathbb{B}_{sk,\alpha,\beta,s_b}(c_1, \ldots, c_\lambda, \bar{q}) = \alpha \oplus F_{s_b}(m_1 \oplus \beta_1, \ldots, m_\lambda \oplus \beta_\lambda, \mathsf{pub}(c_1), \ldots, \mathsf{pub}(c_\lambda), \bar{q})$$

  where $m_i = \mathsf{Dec}_{sk}(c_i)$.

Having defined the auxiliary functions, our pseudorandom function f_k for $k = (\alpha, \beta, sk, s_1, s_2, s_e, s_h, s_b,$ is a combination of these functions. The argument (i, q) selects which function is evaluated, and q is parsed appropriately by each of the functionalities. For example, $\mathbb{B}$ parses q as λ ciphertexts $c_1, \ldots, c_\lambda$, and all remaining bits as $\bar{q}$.

$$f_k(i,q) = \begin{cases} \mathbb{C}_1(q) := \mathbb{C}_{\alpha,\beta,s_1}(q) & \text{if } i = 1 \\ \mathbb{C}_2(q) := \mathbb{C}_{\alpha,s^*,s_2}(q) & \text{if } i = 2 \\ \mathbb{E}(q) := \mathbb{E}_{sk,\alpha,s_e}(q) & \text{if } i = 3 \\ \mathbb{H}(q) := \mathbb{H}_{sk,s_h}(q) & \text{if } i = 4 \\ \mathbb{B}(q) := \mathbb{B}_{sk,\alpha,\beta,s_b}(q) & \text{if } i = 5 \\ \mathbb{R}_1 := \mathbb{R}_{s_1}(q) & \text{if } i = 6 \\ \mathbb{R}_2 := \mathbb{R}_{s_2}(q) & \text{if } i = 7 \\ \mathbb{R}_b := \mathbb{R}_{s_b}(q) & \text{if } i = 8 \\ \mathbb{R}^* := \mathbb{R}_{s^*}(q) & \text{if } i = 9 \end{cases}$$

While this construction may appear daunting, each subfunction serves a very concrete purpose in the argument; understanding the proof ideas will help clarify the construction. We must now argue two properties of this family: learnability as in Definition 4.3.2, and pseudorandomness.

Learnability

We must show that $F_\lambda = \{f_k\}$ is robustly, $\frac{1}{9}$-approximately learnable by an implementation-independent algorithm, L.[5] It suffices to show that, given any close implementation g of f_k for random key k, s^* can be recovered, because $\mathbb{R}^* = \mathbb{R}_{s^*}$ comprises 1/9th of the functionality.

To begin, consider the case the when the implementation is perfect: $g \equiv f_k$. In this case, recovery of s^* is straightforward. Given α, $\mathbb{C}_1$, and $\mathbb{R}_1$ it is easy to find β: for any q, $\beta = \mathbb{C}_1(q) \oplus \mathbb{R}_1(q \oplus \alpha)$. That is, it is easy to construct a circuit that, on input α, outputs β

[5]As discussed earlier, it suffices to prove learnability for $\delta = 1/9$. We may then change the how the subfunctions are indexed to achieve any inverse polynomial.

(by fixing some uniformly random q in the above). [6] But we don't know α, only encryptions of α (coming from $\mathbb{E}$), so how might we recover β?

Using $\mathbb{H}$, it is easy to homomorphically evaluate the circuit on such an encryption, yielding an encryption $c = (c_1, \ldots, c_n)$ of $\beta = (\beta_1, \ldots, \beta_n)$. For any $\bar{q}$, evaluating $\mathbb{B}(c, \bar{q})$ will yield $\alpha \oplus F_{s_b}(\vec{0}, c, \bar{q})$. Evaluating $\mathbb{R}_b(\vec{0}, \mathsf{pub}(c_1), \ldots, \mathsf{pub}(c_n), \bar{q})$ immediately yields α in the clear. Now we can directly recover $s^* = \mathbb{C}(q) \oplus \mathbb{R}_2(q \oplus \alpha)$, for any q.

How does this argument change when g and f_k may disagree on an (arbitrary) negligible fraction of the domain? The first observation is that in the above algorithm, each of $\mathbb{C}_1$, $\mathbb{C}_2$, $\mathbb{E}$, $\mathbb{R}_1$, and $\mathbb{R}_2$, can each evaluated (homomorphically in the case of $\mathbb{C}_1$) at a single point that is distributed uniformly at random. With high probability, g will agree with f_k on these random inputs.

It remains to consider robustness to error in $\mathbb{H}$, $\mathbb{B}$, and $\mathbb{R}_b$. The same idea does not immediately work, because the queries to these circuits are not uniformly distributed.

For $\mathbb{H}$, we leverage the invoker-randomizability of the PRF F_{s_h}, using the argument presented in [BGI+12][7]. In every query to $\mathbb{H}(c, d, \odot, \bar{q})$, the input $\bar{q}$ only effects the randomness used in the final encrypted output. For each such query, pick $\bar{q}$ uniformly and independently at random. Now $\mathbb{H}$ returns a uniformly random encryption of $\mathsf{Dec}_{sk}(c) \odot \mathsf{Dec}_{sk}(d)$. This is because the randomness used for the encryption is now uniformly sampled by F_{s_h}. The distribution over the output induced by the random choice of $\bar{q}$ depends only on $(\mathsf{Dec}_{sk}(c), \mathsf{Dec}_{sk}(d), \odot) \in \{0,1\}^2 \times \{0,1\}^2 \times \{0,1\}^4$. As in [BGI+12], the probability of returning an incorrect answer on such a query is at most 64-times that of a random query, which is still negligible.

For $\mathbb{B}$ and $\mathbb{R}_b$, we leverage the properties of the decomposable symmetric-key encryption scheme, using the argument presented in [BP12].[8] We modify the procedure of using $\mathbb{B}$ and $\mathbb{R}_b$ to recover α given an encryption c of β. Instead of querying $\mathbb{B}$ on $(c, \bar{q})$, sample a fresh random m, and using $\mathbb{H}$, compute an encryption c' of $\beta \oplus m$. Note that c' is a uniformly

[6] This is ability is what enables the learnability; the black-box learner cannot construct such a circuit and thus cannot continue with the homomorphic evaluation in the next step.

[7] Proof of Theorem 4.3

[8] Proof of Claim 3.8

random encryption (by invoker-pseudorandomness) of the uniformly random string $\beta \oplus m$, and is thus a uniformly-distributed string of the appropriate length. Independently sample a random $\bar{q}$ and query $\alpha' := \mathbb{B}(c', \bar{q})$. This query to $\mathbb{B}$ is now distributed uniformly, and will therefore be answered correctly with high probability.

To recover α, we evaluate $\alpha = \alpha' \oplus \mathbb{R}_b(m, \mathsf{pub}(c_1), \ldots, \mathsf{pub}(c_\lambda), \bar{q})$. This query to $\mathbb{R}_b$ is also distributed uniformly at random (for random $\bar{q}$), and will therefore be answered correctly with high probability.

Pseudorandomness

Our proof that the family $\{f_k\}$ is pseudorandom follows that of [BP12]; the main technical change comes from the fact that $\mathbb{B}$ depends on α. We consider a polynomial-time adversary $\mathcal{A}$ with oracle access to f_k. For simplicity, we ignore the indexing of the subfunctions of f_k and assume that $\mathcal{A}$ has direct oracle access to each of the constituent functions, showing that they are simultaneously pseudorandom.

Let E_1 be the the event that $\mathcal{A}$ produces *distinct* queries $q = (c, \bar{q})$, $q' = (c', \bar{q}')$ such that:

$$(m \oplus \beta, \mathsf{pub}(c_1), \ldots, \mathsf{pub}(c_\lambda), \bar{q}) = (m' \oplus \beta, \mathsf{pub}(c_1'), \ldots, \mathsf{pub}(c_\lambda'), \bar{q}')$$

where $m, m' \in \{0,1\}^\lambda$ are the decryptions under sk of c and c' respectively.

Claim 4.4.1.1. $\Pr_{k,\mathcal{A}}[E_1] = 0$

Proof. Recall that for any ciphertext c, $\mathsf{pub}(c)$ and the plaintext m uniquely determine the ciphertext. If $m \oplus \beta = m' \oplus \beta$, and $\mathsf{pub}(c_i) = \mathsf{pub}(c_i)'$ for all i, then $c = c'$. Therefore $q = q'$. □

We consider two "bad" events, and argue that if $\mathcal{A}$ is to distinguish f_k from a random function, (at least) one of the events must occur.

- Let E_α be the event that $\mathcal{A}$ produces queries q and q' such that $q \oplus \alpha = q'$.

- Let E_β be the event that $\mathcal{A}$ produces queries $q = (c, \bar{q})$ and q' such that $q' = (m \oplus \beta, \mathsf{pub}(c_1), \ldots, \mathsf{pub}(c_\lambda), \bar{q})$, where $m \in \{0,1\}^\lambda$ is the decryption under sk of c.

Claim 4.4.1.2. *If* $\Pr_{k,\mathcal{A}}[E_\alpha] \leq \text{negl}(\lambda)$ *and* $\Pr_{k,\mathcal{A}}[E_\beta] \leq \text{negl}(n)$*, then* $\mathcal{A}$ *cannot distinguish between* f_k *and a random oracle.*

Proof. Because f_k depends on the PRF keys s_1, s_2, s_e, s_h, and s_b (but not s^*) only by black-box application of the respective PRFs, we can indistinguishably replace all applications of these PRFs by (independent) truly random functions. If E_α never occurs, than the responses from $\mathbb{C}_1$ and $\mathbb{R}_1$ (respectively $\mathbb{C}_2$ and $\mathbb{R}_2$) are uncorrelated; thus we can indistinguishably replace $\mathbb{C}_1$ (respectively, $\mathbb{C}_2$) by a independent random function. At this point, $\mathcal{A}$'s oracle only depends on s^* through calls to the PRF F_s^*; we can now replace $\mathbb{R}^*$ with a independent random function. By similar reasoning, if E_β never occurs, then the responses from $\mathbb{B}$ and $\mathbb{R}_b$ are uncorrelated; thus we can indistinguishably replace $\mathbb{B}$ with another independent random oracle. The above holds with high probability, conditioning on $\neg E_\alpha$ and $\neg E_\beta$.

Now $\mathcal{A}$ is left with oracles of $\mathbb{E}$ and $\mathbb{H}$ in which the PRFs F_{s_e} and F_{s_h} have been replaced by random (along with 7 additional independent random oracles). The ciphertexts of the encyption scheme we use are pseudorandom. Thus, access to these two oracles may be replaced with random without noticeably affecting the output distribution of $\mathcal{A}$. □

All that remains is to bound the probabilities of E_α and E_β. We consider two cases separately: when E_α occurs before E_β and vice-versa, arguing that the probability of either event occurring first is negligible. Let $E_{\alpha,i}$ (respectively, $E_{\beta,i}$) be the event that E_α (respectively E_β) occurs in the first i queries.

Claim 4.4.1.3. *For all* i*,* $\Pr_{k,\mathcal{A}}[E_{\beta,i} | \neg E_{\alpha,i-1}] \leq \text{negl}(\lambda)$

Proof. It suffices to show that for all i:

$$\Pr_{k,\mathcal{A}}[E_{\beta,i} | \neg E_{\alpha,i-1}, \neg E_{\beta,i-1}] \leq \text{negl}(\lambda).$$

Furthermore, because the events are efficiently testable given only α, β, and sk, it is enough to prove the claim when all the underlying PRFs (corresponding to s_1, s_2, s_e, s_h, s_b, and s^* are replaced by (independent) truly random functions.

As in Claim 4.4.1.2, if E_α doesn't occur in the first $i-1$ queries, than the responses from $\mathbb{C}_1$ and $\mathbb{R}_1$ (respectively $\mathbb{C}_2$ and $\mathbb{R}_2$) are uncorrelated on these queries; thus we can indistinguishably replace $\mathbb{C}_1$ (respectively, $\mathbb{C}_2$) by a independent random function. By similar reasoning, if E_β doesn't occur in the first $i-1$ queries, then the responses from $\mathbb{B}$ and $\mathbb{R}_b$ are uncorrelated on these queries; thus we can indistinguishably replace $\mathbb{B}$ with another independent random oracle. The above holds with high probability, conditioning on $\neg E_{\alpha,i-1}$ and $\neg E_{\beta,i-1}$.

The view of $\mathcal{A}$ after the first $i-1$ queries is now independent of β. Now E_β amounts to outputting a ciphertext c and string q such that $\mathsf{Dec}_{sk}(c) \oplus q = \beta$, for $\beta \leftarrow \{0,1\}^\lambda$ drawn independently of the view of the adversary. This occurs with vanishingly small probability.

□

Claim 4.4.1.4. $\Pr_{k,\mathcal{A}}[E_{\alpha,i} | \neg E_{\beta,i-1}] \leq \mathrm{negl}(\lambda)$

Proof. It suffices to show that for all i:

$$\Pr_{k,\mathcal{A}}[E_{\alpha,i} | \neg E_{\beta,i-1}, \neg E_{\alpha,i-1}] \leq \mathrm{negl}(\lambda).$$

Again, because the events are efficiently testable given only α, β, and sk, it is enough to prove the claim when all the underlying PRFs (corresponding to s_1, s_2, s_e, s_h, s_b, and s^* are replaced by (independent) truly random functions. As in the previous claim, we may indistinguishably replace the first $i-$ responses of $\mathbb{C}_1$, $\mathbb{C}_2$, $\mathbb{B}$, $\mathbb{R}_b$, $\mathbb{R}_1$, and $\mathbb{R}_2$ by independent random functions. The above holds with high probability, conditioning on $\neg E_{\alpha,i-1}$ and $\neg E_{\beta,i-1}$.

The view of the adversary is depends on α only by way of $\mathbb{E}$, the circuit that outputs random encryptions of α. Furthermore, besides the oracles $\mathbb{E}$ and $\mathbb{H}$, all of the oracle responses $\mathcal{A}$ receives are uniformly random (and independent of α). But just as in [BGI+12][9] and

[9]Claim 3.6.1

[BP12][10], with only these two oracles, any CCA-1 encryption scheme is semantically secure. Thus we can indistinguishably replace $\mathbb{E}_{sk,\alpha,s_e}$ with $\mathbb{E}_{sk,\alpha,s_e}$ – returning only encryptions of 0. Finally, the view of $\mathcal{A}$ is information theoretically independent of α; as before, we conclude that $E_{\alpha,i}$ occurs with vanishingly small probability. $\square$

[10]Claim 3.3

Bibliography

[ABPP14] Michel Abdalla, Fabrice Benhamouda, Alain Passelègue, and Kenneth G. Paterson. Related-key security for pseudorandom functions beyond the linear barrier. In Juan A. Garay and Rosario Gennaro, editors, *Advances in Cryptology - CRYPTO 2014 - 34th Annual Cryptology Conference, Santa Barbara, CA, USA, August 17-21, 2014, Proceedings, Part I*, volume 8616 of *Lecture Notes in Computer Science*, pages 77–94. Springer, 2014.

[BC10] Mihir Bellare and David Cash. Pseudorandom functions and permutations provably secure against related-key attacks. In Tal Rabin, editor, *Advances in Cryptology - CRYPTO 2010, 30th Annual Cryptology Conference, Santa Barbara, CA, USA, August 15-19, 2010. Proceedings*, volume 6223 of *Lecture Notes in Computer Science*, pages 666–684. Springer, 2010.

[BF02] Nader H Bshouty and Vitaly Feldman. On using extended statistical queries to avoid membership queries. *The Journal of Machine Learning Research*, 2:359–395, 2002.

[BGI$^+$12] Boaz Barak, Oded Goldreich, Russell Impagliazzo, Steven Rudich, Amit Sahai, Salil P. Vadhan, and Ke Yang. On the (im)possibility of obfuscating programs. *J. ACM*, 59(2):6, 2012.

[BGI14] Elette Boyle, Shafi Goldwasser, and Ioana Ivan. Functional signatures and pseudorandom functions. In Krawczyk [Kra14], pages 501–519.

[BGV11] Siavosh Benabbas, Rosario Gennaro, and Yevgeniy Vahlis. Verifiable delegation of computation over large datasets. In Phillip Rogaway, editor, *Advances in Cryptology - CRYPTO 2011 - 31st Annual Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2011. Proceedings*, volume 6841 of *Lecture Notes in Computer Science*, pages 111–131. Springer, 2011.

[BHHO08] Dan Boneh, Shai Halevi, Mike Hamburg, and Rafail Ostrovsky. Circular-secure encryption from decision diffie-hellman. In *Advances in Cryptology–CRYPTO 2008*, pages 108–125. Springer, 2008.

[BLMR13] Dan Boneh, Kevin Lewi, Hart William Montgomery, and Ananth Raghunathan. Key homomorphic prfs and their applications. In Ran Canetti and Juan A. Garay, editors, *Advances in Cryptology - CRYPTO 2013 - 33rd Annual Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2013. Proceedings, Part I*, volume 8042 of *Lecture Notes in Computer Science*, pages 410–428. Springer, 2013.

[BP12] Nir Bitansky and Omer Paneth. From the impossibility of obfuscation to a new non-black-box simulation technique. In *53rd Annual IEEE Symposium on Foundations of Computer Science, FOCS 2012, New Brunswick, NJ, USA, October 20-23, 2012*, pages 223–232. IEEE Computer Society, 2012.

[BW04] Andrej Bogdanov and Hoeteck Wee. A stateful implementation of a random function supporting parity queries over hypercubes. In *8th International Workshop on Randomization and Computation, RANDOM 2004, Cambridge, MA, USA, August 22-24, 2004, Proceedings*, pages 298–309, 2004.

[BW13] Dan Boneh and Brent Waters. Constrained pseudorandom functions and their applications. In Sako and Sarkar [SS13], pages 280–300.

[CH15] Aloni Cohen and Justin Holmgren. Multilinear pseudorandom functions. In *Automata, Languages, and Programming*. Springer, 2015.

[CHV15] Aloni Cohen, Justin Holmgren, and Vinod Vaikuntanathan. Publicly verifiable software watermarking. Cryptology ePrint Archive, Report 2015/373, 2015.

[DRWY12] Zeev Dvir, Anup Rao, Avi Wigderson, and Amir Yehudayoff. Restriction access. In Shafi Goldwasser, editor, *Innovations in Theoretical Computer Science 2012, Cambridge, MA, USA, January 8-10, 2012*, pages 19–33. ACM, 2012.

[GGI$^+$02] Anna C. Gilbert, Sudipto Guha, Piotr Indyk, Yannis Kotidis, S. Muthukrishnan, and Martin Strauss. Fast, small-space algorithms for approximate histogram maintenance. In *Proceedings on 34th Annual ACM Symposium on Theory of Computing, May 19-21, 2002, Montréal, Québec, Canada*, pages 389–398, 2002.

[GGM86] Oded Goldreich, Shafi Goldwasser, and Silvio Micali. How to construct random functions. *J. ACM*, 33(4):792–807, 1986. Extended abstract in FOCS 84.

[GGN10] Oded Goldreich, Shafi Goldwasser, and Asaf Nussboim. On the implementation of huge random objects. *SIAM J. Comput.*, 39(7):2761–2822, 2010.

[HILL99] Johan Håstad, Russell Impagliazzo, Leonid A. Levin, and Michael Luby. A pseudorandom generator from any one-way function. *SIAM J. Comput.*, 28(4):1364–1396, 1999.

[KPTZ13] Aggelos Kiayias, Stavros Papadopoulos, Nikos Triandopoulos, and Thomas Zacharias. Delegatable pseudorandom functions and applications. In Sadeghi et al. [SGY13], pages 669–684.

[Kra14] Hugo Krawczyk, editor. *Public-Key Cryptography - PKC 2014 - 17th International Conference on Practice and Theory in Public-Key Cryptography, Buenos Aires, Argentina, March 26-28, 2014. Proceedings*, volume 8383 of *Lecture Notes in Computer Science*. Springer, 2014.

[KV94] Michael J. Kearns and Leslie G. Valiant. Cryptographic limitations on learning boolean formulae and finite automata. *J. ACM*, 41(1):67–95, 1994.

[NR04] Moni Naor and Omer Reingold. Number-theoretic constructions of efficient pseudo-random functions. *J. ACM*, 51(2):231–262, 2004.

[RR97] Alexander A. Razborov and Steven Rudich. Natural proofs. *J. Comput. Syst. Sci.*, 55(1):24–35, 1997.

[SGY13] Ahmad-Reza Sadeghi, Virgil D. Gligor, and Moti Yung, editors. *2013 ACM SIGSAC Conference on Computer and Communications Security, CCS'13, Berlin, Germany, November 4-8, 2013.* ACM, 2013.

[SS13] Kazue Sako and Palash Sarkar, editors. *Advances in Cryptology - ASIACRYPT 2013 - 19th International Conference on the Theory and Application of Cryptology and Information Security, Bengaluru, India, December 1-5, 2013, Proceedings, Part II*, volume 8270 of *Lecture Notes in Computer Science.* Springer, 2013.

[SW14] Amit Sahai and Brent Waters. How to use indistinguishability obfuscation: deniable encryption, and more. In David B. Shmoys, editor, *Symposium on Theory of Computing, STOC 2014, New York, NY, USA, May 31 - June 03, 2014*, pages 475–484. ACM, 2014.

[Val84] Leslie G Valiant. A theory of the learnable. *Communications of the ACM*, 27(11):1134–1142, 1984.

[VV86] Leslie G. Valiant and Vijay V. Vazirani. NP is as easy as detecting unique solutions. *Theor. Comput. Sci.*, 47(3):85–93, 1986.