New Guarantees for Cryptographic Circuits and Data Anonymization

by

Alon Jonathan Cohen

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

Doctor of Philosophy in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2019

Signature redacted

Author ....
Department of Electrical Engineering and Computer Science
May 23, 2019

Signature redacted

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

Signature redacted

Accepted by ....
Leslie A. Kolodziejski
Professor of Electrical Engineering and Computer Science
Chair, Department Committee on Graduate Students

New Guarantees for Cryptographic Circuits and Data Anonymization

by

Alon Jonathan Cohen

Submitted to the Department of Electrical Engineering and Computer Science
on May 23, 2019, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy in Electrical Engineering and Computer Science

Abstract

The first part of this thesis presents new definitions and constructions for three modern problems in cryptography: watermarking cryptographic circuits, updatable cryptographic circuits, and proxy reencryption. The second part is dedicate to advancing the understanding of data anonymization. We examine what it means for a data anonymization mechanism to prevent singling out in a data release, a necessary condition to be considered effectively anonymized under the European Union's General Data Protection Regulation. We also demonstrate that heretofore theoretical privacy attacks against ad-hoc privacy preserving technologies are in fact realistic and practical.

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

Acknowledgments

This thesis also reflects the direct and indirect contributions of innumerable people in my life. My advisor, Shafi Goldwasser, taught me most of the cryptography I know. She has been a wellspring of wisdom, advice, opportunity, and especially encouragement. Vinod Vaikuntanathan taught me the remainder. I often came to him with a question and left with the answer to a much better question. Kobbi Nissim has been an invaluable collaborator, helping me to understand that I might be able to contribute to legal thinking not despite my technical background, but because of it. Shafi, Vinod, and Kobbi do me the additional honor of constituting my thesis committee.

I am indebted to the many people from whom or with whom I have learned, including each of my many coauthors: Prabhanjan Ananth, Ran Canetti, Nishanth Dikkala, Abhishek Jain, Saleet Klein, Ryo Nishimaki, Sunoo Park, Govind Ramnarayan, Sarah Scheffler, Adam Smith, Daniel Wichs, and especially Justin Holmgren. More generally, the theory of cryptography community—warm and generous—has been a wonderful academic home for me.

Studying at MIT came with many advantages. Chief was the friendship of everybody else studying there with me, especially Daniel Grier and Gautam Kamath. Debbie Goodwin and Joanne Talbot Hanley were also indispensable during my six years there. I am also grateful for support as an undergraduate at UC Berkeley from Messrs. D. Burkett, Garcia, Hall, & Klein, and to Christos Papadimitriou, James Sethian, and Brian Harvey, but for whom I would not have studied computer science.

Finally, I cannot estimate the importance of the constant support of family, especially Rebecca Powell and my parents, Moshe and Leah Cohen.

Contents

Preface

This thesis reflects two intellectual trends that coincided with my time at MIT. The first was the theory of cryptography community's discovery and exploration of Obfustopia, which was jumpstarted by a pair papers made public a month before I moved to Cambridge. One was a candidate construction of a form of program obfuscation called indistinguishability obfuscation. The other demonstrated that this seemingly weak form of obfuscation was in fact a powerful cryptographic primitive, adding a new world to Impagliazzo's five. The community eagerly mapped out Obfustopia: the cryptographic constructions it enabled, its complexity-theoretic implications, and how obfusction itself might be realized. Indistinguishability obfuscation's conceptual significance extends beyond Obfustopia: many ideas and techniques first developed in the context of obfuscation have since been adapted to settings which do not require obfuscation.

Part 1 of this thesis is best understood with this context in mind. Chapter 1 presents a breakthrough in the theoretical study of program watermarking. Prior to our work, the study of watermarking consisted mostly of applied research. More theoretical works were either negative, purely definitional, or heavily restricted the adversary's space of watermark removal strategies. With obfuscation, we construct the first watermarking scheme for a large class of circuits against adversaries employing general removal strategies, and also demonstrate new families of circuits that cannot be watermarked. The core ideas of the construction presented in this thesis have evolved in a sequence of subsequent works, culminating in watermarking without obfuscation, albeit with weaker guarantees.

Is there some way to securely update an obfuscated program without reobfuscating the whole thing? What about updating a garbled circuit without regarbling? What about updating a randomized encoding? What about updating secret keys for functional and attribute-based encryption? Chapter 2 unifies these seemingly disparate problems into a single abstract framework, and shows that updatable randomized encodings suffice to solve them all. We construct updatable randomized encodings using obfuscation-style ideas, but using only primitives not known (nor believed, generally) to imply obfuscation. Chapter 2 also constructs updatable versions of multiparty computation and non-interactive proofs, primitives which do not fit into the general framework but for which updatable randomized encodings still suffice.

One tool used in that chapter is a new form of proxy reencryption. Proxy reencryption allows a person to securely outsource the task of atomically decrypting-and-reencrypting her ciphertexts, making them decryptable by a different person of her choice. Our use of proxy reencryption for updatable cryptography led to a startling discovery: the most well-established definition of security for proxy reencryption had a basic, crippling flaw. Chapter 3 describes this flaw and the resulting vulnerability in at least one implementation of proxy reencryption. We propose a new definition of security which has been embraced in subsequent work on proxy reencryption.

The second trend was a general chilling of the public's perception of Silicon Valley.[1] Until recently, technology companies were mostly unaffected by the skepticism people often have of large corporations. But then the public trust afforded to large technology companies seemed to give way to a feeling that they were failing to address serious negative externalities, whether due to ignorance, inability, or apathy. One driver of this shift was a growing appreciation of the enormity of the granular, individualized data continuously being collected about us, and its sensitivity. This shift has even affected computer scientists—both in industry and the academy—some of whom are trying to engage with social sciences and law in a serious way. While this was happening, I had been harboring an amateur's interest in law. In Cambridge, I found opportunities to work on problems that were at once legal and technical. Whether this was due to larger social trends or simply a happy accident, it is hard for me to say.

Part II is the fruit of one of these opportunities. Data privacy laws govern the use of sensitive personal information, delineating the boundaries of appropriate use. However, a significant conceptual gap between legal and mathematical thinking around data privacy leaves practitioners uncertain as to which technical offerings adequately match expectations expressed in legal standards. A core but ineffable concept in data privacy is data anonymization, the transformation of data to sever the connection between data and subject. This definition begs the question of what it means to sever this connection, and different answers underpin differing privacy concepts. While the question remains far from settled, this thesis contributes to its study using two very different approaches.

Chapter 4 examines what it means for a data anonymization mechanism to prevent singling out in a data release, a necessary condition to be considered effectively anonymized under the GDPR. With careful modelling—grounded in legal thinking on the topic—we formulate a rigorous privacy notion: security against predicate singling out attacks. We argue that any data anonymization system recognized as such by the GDPR must be secure against predicate singling out attacks. We examine the properties of this security notion and its relationships with existing privacy concepts. It is a case study in using mathematical formalism to model and analyze a legal concept, ultimately drawing conclusions that diverge from the legal consensus.

Chapter 5 demonstrates that heretofore theoretical privacy attacks against adhoc privacy preserving technologies are in fact realistic and practical, even when these technologies are deployed by experts. We carry out the first linear program reconstruction attack against a deployed real-world system designed specifically to prevent such attacks. This illustrates the practical effectiveness of the Fundamental Law of Information Recovery, a long-understood theoretical vulnerability of statistical query systems that answer many accurate queries. We also describe privacy attacks against a highly publicized k-anonymous dataset consisting of online educa-

[1]To be clear, this is an account of my understanding of the evolving zeitgeist. Whether my interpretation accurately reflects trends in popular opinion is for somebody with very different academic training than me to study, perhaps in a thesis of their own.

tional student records. While it was known that in principle k-anonymity does not compose, to the best of our knowledge ours is the first example of a real-world failure of composition.

This thesis is based on the following work:

- "Watermarking cryptographic capabilities" with Justin Holmgren, Ryo Nishimaki, Vinod Vaikuntanathan, and Daniel Wichs [CHN+18]
- "Cryptography with updates" with Prabhanjan Ananth and Abhishek Jain [ACJ17]
- "What about Bob? The inadequacy of CPA security for proxy re-encryption" [Coh19]
- "Towards Formalizing the GDPR's Notion of Singling Out" with Kobbi Nissim [CN19]
- "Linear Program Reconstruction in Practice" with Kobbi Nissim [CN18]

Part I

Cryptographic Circuits

Chapter 1

Watermarking Cryptographic Circuits

1.1 Introduction

Digital watermarking enables us to embed some special information called a *mark* into digital objects such as images, movies, music files, or programs. We often call such objects *marked.* There are two basic requirements for watermarking. The first is that a marked object should not be significantly different from the original object. The second is that malicious entities should not be able to remove embedded marks without somehow "destroying" the object (e.g., modify an image beyond recognition).

There are many works on watermarking perceptual objects such as images, movies, music files, etc. Most of them do not give a rigorous theoretical treatment and their constructions are heuristic and ad-hoc. (We briefly survey some of these works in Section 1.2.7). Barak, Goldreich, Impagliazzo, Rudich, Sahai, Vadhan and Yang, in their seminal work that laid the mathematical foundations of program obfuscation, also proposed definitions for program watermarking [BGI$^+$01a, BGI$^+$12]. Unfortunately, their results were all negative, showing that certain definitions of watermarking are impossible to achieve. The work of Hopper, Molnar and Wagner [HMW07] proposes

Based on "Watermarking cryptographic capabilities" with Justin Holmgren, Ryo Nishimaki, Vinod Vaikuntanathan, and Daniel Wichs [CHN$^+$18].

general and rigorous definitions for watermarking schemes, and explores in depth connections between the definitions, but does not provide any actual constructions.

Watermarking programs. Our first contribution is to define the notion of *public-key watermarking*, building on the work of Hopper, Molnar and Wagner [HMW07] who introduced a secret-key definition. We speak of a watermarking scheme for a circuit class $\mathcal{C} = \{\mathcal{C}_\lambda\}_{\lambda \in \mathbb{N}}$ where each $\mathcal{C}_\lambda$ is a set of circuits. A watermarking scheme for $\mathcal{C}$ consists of procedures $\mathsf{Mark}(\mathsf{mk}, \cdot)$ and $\mathsf{Extract}(\mathsf{xk}, \cdot)$ with a *secret marking key* mk and a *public extraction key* xk. Given a circuit C, the marking procedure $\widetilde{C} \leftarrow \mathsf{Mark}(\mathsf{mk}, C)$ creates a marked circuit $\widetilde{C}$ that evaluates C. Although we will see that we cannot achieve perfect correctness, in which $\widetilde{C}(x) = C(x)$ for all inputs x, we will be able to achieve *statistical correctness* where we allow a negligible error probability. The extraction procedure $\mathsf{Extract}(\mathsf{xk}, C^*)$ outputs either that the circuit is marked or $\mathsf{unmarked}$. Note that a watermarking scheme should satisfy a property called meaningfulness. This property means that for all circuits, we can not extract a valid mark from them under a randomly generated extraction key except with negligible probability. This property is for excluding trivial watermarking schemes.

For security, we consider a game where a challenger chooses a *random circuit* $C \leftarrow \mathcal{C}_\lambda$ and gives the adversary the marked circuit $\widetilde{C} \leftarrow \mathsf{Mark}(\mathsf{mk}, C)$. Intuitively, we require that the adversary cannot come up with *any* circuit that correctly evaluates C but does not have the mark embedded in it. This property is called *unremovability*. Following [HMW07] and adapting it to the public-key setting, we require that unremovability holds against *chosen circuit attackers*, namely adversaries that have oracle access to $\mathsf{Mark}(\mathsf{mk}, \cdot)$.

More precisely, the adversary produces a circuit C^* and we insist that either:

(a) $\mathsf{Extract}$ correctly detects that the circuit is marked by outputting $\mathsf{marked} \leftarrow \mathsf{Extract}(\mathsf{xk}, C^*)$; or

(b) The circuit C^* *does not even approximately* compute C, meaning that $C^*(x) = C(x)$ on at most an ε fraction of the inputs x.

The parameter ε is called the "approximation factor" and we can set it to some small constant or even to any $1/\operatorname{poly}$ fraction. (The smaller the ε, the better the security guarantee). During the attack, the adversary is also given the public extraction key xk and access to the marking oracle $\mathsf{Mark}(\mathsf{mk}, \cdot)$ that he can query on arbitrary circuits of his choice (even ones that are not in $\mathcal{C}_\lambda$). At this point, it is prudent to note that the very first idea that comes to mind, namely signing the circuit C using mk, is not a particularly good watermarking strategy as the adversary can simply strip off the signature leaving a perfectly functional circuit.

We call the above type of watermarking "messageless" to denote that it only distinguishes between marked and unmarked circuits. We also consider a stronger version called "message-embedding" watermarking where the marking procedure can be used to embed an arbitrary message into the circuit and the extraction procedure should recover the message. Similar to the above, the adversary's goal is to force the extraction procedure to recover a different message. (We refer the reader to Section 1.4 for formal definitions).

Why cryptographic programs? In this thesis, we focus on watermarking circuits that are cryptographic in nature, such as circuits evaluating a pseudorandom function (PRF) or implementing a signing or decryption procedure. One could reasonably ask: *why cryptographic programs*?

First, we observe that in the security definition for watermarking, the challenge circuit C has to be unknown to the adversary. For, if not, the adversary has a trivial watermark removing strategy: given the marked circuit $\widetilde{C}$, simply output C as the mark-removed circuit. Since C is an arbitrary program, it is very likely to be unmarked; on the other hand, C is (approximately) equivalent $\widetilde{C}$ in functionality.[1] Thus, it is natural for the challenger to pick C from a distribution with high min-entropy. (For simplicity, we consider picking circuits uniformly at random from $\mathcal{C}_\lambda$.)

Secondly, we observe that circuit families that are exactly learnable are not wa-

[1] One can attempt to get around this issue by requiring that the program output by the watermark remover should be distinct from C and $\widetilde{C}$. However, it is also easy to defeat these definitions by asking the watermarked remover to output an indistinguishability obfuscation of C.

termarkable. This is because the adversary can simply invoke $\widetilde{C}$ as a black box and recover a description of the original circuit C (or an equivalent version thereof) which is again very likely to be unmarked.

This naturally leads us to consider families of circuits where random circuits from the family are not exactly learnable, canonical examples of which are cryptographic programs: pseudo-random functions, signing algorithms and decryption algorithms. Jumping ahead, we remark that unlearnability is a necessary but not sufficient condition for being able to watermark a family of circuit. Indeed, we show families of pseudo-random functions that, despite being strongly unlearnable, cannot be watermarked even with approximate correctness.

That said, we regard the question of coming up with meaningful definitions and constructions of watermarking for general circuits (and even families of evasive circuits) as a challenging open question arising from this work.

Watermarking cryptographic programs: an application. To further highlight the usefulness of watermarking cryptographic functions, we describe an application of watermarking pseudorandom functions. However, we emphasize that the concept should have broader applicability beyond this example.

Consider an automobile manufacturer that wants to put electronic locks on its cars; the car contains a PRF F and can only be opened by running an identification protocol where it chooses a random input x and the user must respond with $F(x)$. When a car is sold to a new owner, the owner is given a software key (e.g., a smartphone application) consisting of marked program $\widetilde{C}$ that evaluates the PRF $F(\cdot)$ and is used to open the car. The mark can embed some identifying information such as the owner's name and address. Even if the software key is stolen, the thief cannot create a new piece of software that would still open the car while removing information about the original owner.

Impossibility of watermarking? The work of Barak et al. [BGI$^+$01a, BGI$^+$12] initiated the first theoretical study of program watermarking. They propose a game-based definition which appears significantly weaker than the definitions we consider in this thesis (it is in the symmetric-key setting with no marking/detection oracles given to the adversary), *but* requires perfect correctness. Unfortunately, they show that this definition is unachievable assuming the existence of indistinguishability obfuscation.

The main intuition behind the negative result is to consider an attacker that takes a marked program and applies indistinguishability obfuscation (iO) to it. If the marked program implements the original program with *perfect correctness* then the result of applying iO to it should be indistinguishable from that of applying iO to the original program. Since the latter is unlikely to be marked, the same should apply to the former. Therefore, this presents a valid attack against watermarking in general.

Barak et al. note that the above attack crucially relies on the *perfect* (rather than merely *statistical*) correctness of the marked program, meaning that it correctly evaluates the original program on every input. They mention that otherwise "it seems that obfuscators would be useful in constructing watermarking schemes, because a watermark could be embedded by changing the value of the function at a random input, after which an obfuscator is used to hide this change." This idea was not explored further in [BGI+01a, BGI+12] and it is far from clear if a restricted notion of obfuscation such as iO (or even extractability obfuscation or VGB) would be sufficient and what type of watermarking security can be achieved with this approach. Nevertheless, this idea serves as the starting point of our work.

1.1.1 Our Results

We start by giving new formal definitions of program watermarking, along the lines of what we described earlier. To avoid the [BGI+01a, BGI+12] impossibility result described above, our definition allows for statistical rather than perfect correctness. That is, for every circuit $C \in \mathcal{C}_\lambda$ and every input x,

$$\Pr[\widetilde{C}(x) \neq C(x) \mid \widetilde{C} \leftarrow \mathsf{Mark}(\mathsf{mk}, C)] \leq \mathsf{negl}(\lambda)$$

where the probability is over the choice of the keys and the coin tosses of the Mark algorithm. We call this *strong approximate correctness.*

This seemingly small relaxation allows us to circumvent the impossibility results and show algorithms to watermark large classes of pseudo-random functions (and signature algorithms, decryption algorithms [CHN+16]). Our main technical contribution is a method of watermarking any family of puncturable PRFs.[2] Our scheme has a public-key extraction procedure and achieves security in the presence of a marking oracle. We get a messageless scheme that allows for any $\varepsilon = 1/\operatorname{poly}(\lambda)$ approximation factor and a message-embedding scheme that allows for approximation factors $\varepsilon = 1/2 + 1/\operatorname{poly}(\lambda)$. In the case of message-embedding constructions, we show that there is an inherent lower bound of $\varepsilon = 1/2$. Both schemes rely on (polynomially secure) indistinguishability obfuscation (iO).

Theorem 1.1.1 (Informal). *Assuming indistinguishability obfuscation and injective one-way functions, there is a watermarking scheme that is secure against chosen circuit attacks for any family of puncturable PRFs.*

Theorem 1.1.1 shows that relaxing the correctness requirement to strong approximate correctness allows us to watermark any family of puncturable PRFs. A natural question is whether one can watermark arbitrary PRFs. We show impossibility results matching our constructions by demonstrating families of PRFs, signature and decryption algorithms that cannot be watermarked. We call such schemes *waterproof.*

We start by observing that learnable functions are waterproof, simply because an adversary can learn a canonical representation of the function given any program (even any oracle) that computes the function. Indeed, it is sufficient for the function family to be *non black-box learnable.* That is, the adversary should be able to use any program that computes the function to extract a canonical representation. Such function families were defined in the work of Barak et al. [BGI+01a] and are

[2] Puncturable pseudo-random functions (pPRFs) [BW13, BGI13, KPTZ13] are PRFs where the owner of the key K can produce a punctured key Kx that allows computation of the PRF on all inputs $y \neq x$. Moreover, given the punctured key, $\mathsf{PRF}_K(x)$ is pseudorandom. Puncturable PRFs can be constructed from one-way functions [BW13, BGI13, KPTZ13] or more efficiently, from several number-theoretic assumptions. [BLMR13b, BV15b, BFP+15].

called *unobfuscatable functions*. Indeed, [BGI$^+$01a, BGI$^+$12] show PRFs, signature and decryption algorithms that are strongly unobfuscatable—that is, an adversary can extract the canonical representation even given a program that only computes a function with strong approximate correctness. This immediately gives us waterproof PRFs, signature and decryption algorithms. (See Section 1.8 for more details.)

Theorem 1.1.2 (Informal). *Assuming the existence of one-way functions, there are waterproof PRFs and signature and decryption algorithms, even if: (a) we only require symmetric-key watermarking; and (b) we only require unremovability against stand-alone adversaries that do not have access to* Mark *or* Extract *oracles.*

We continue this line of thought and ask if we can further weaken the correctness requirement and overcome this impossibility result. Namely, we consider a weak approximate correctness requirement which states that the marked program $\widetilde{C}$ agrees with the original program C on most inputs. (In contrast to strong approximate correctness, here $\widetilde{C}$ can always make a mistake on some fixed set of inputs). We show that even this relaxation does not help. Our proof of this result involves constructing new types of *robust* unobfuscatable PRFs. (See Section 1.8.3 for more details).

Theorem 1.1.3 (Informal). *Assuming the existence of one-way functions, there are waterproof PRFs even under weak approximate correctness (and even with relaxations (a) and (b) as in Theorem 1.1.2).*

Organization

The next section summarizes our techniques. In Section 1.3, we provide preliminaries and basic definitions used throughout the chapter of watermarking. In Section 1.5, we define and construct a new cryptographic object called puncturable encryption. In Section 1.6, we describe our main result, namely our PRF watermarking and its security proof. In Section 1.7, we provide several extensions to our main construction. In Section 1.8, we provide negative results on watermarking. Section 1.9 concludes.

1.2 Overview of Our Techniques

1.2.1 Simplification: Token-Based Watermarking

Although our full watermarking scheme relies on indistinguishability obfuscation (iO), our main technical insights are largely unrelated to obfuscation. In order to elucidate our techniques clearly without getting entangled in details of iO, for the purposes of this introduction we consider a simplified model of watermarking that we call *token-based* watermarking. We treat watermarked programs $\widetilde{C} \leftarrow \mathsf{Mark}(\mathsf{mk}, \cdots)$ as tamper-proof hardware tokens which the adversary can only access as a black box.[3] The adversary can arbitrarily compose hardware tokens $\widetilde{C}_1, \ldots, \widetilde{C}_q$ and create a new token $C^* = C^*[\widetilde{C}_1, \ldots, \widetilde{C}_q]$ that has oracle access to the tokens $\widetilde{C}_i$ embedded inside of it. More formally, we can think of C^* as an oracle circuit with oracle-gates to $\widetilde{C}_1, \ldots, \widetilde{C}_q$. The extraction procedure $\mathsf{Extract}(\mathsf{xk}, C^*)$ will also treat any such token C^* as a black box. The goal of the adversary is to create a token C^* which functionally approximates the challenge watermarked program $\widetilde{C}$ but on which the extraction procedure fails to recover the correct embedded message. Most of the interesting aspects of constructing watermarking schemes already come up in the token-based setting.[4] However, the constructions in the token-based setting become simpler and do not rely on obfuscation. Therefore, we view it as a useful stepping stone to building intuition for our full results where the adversary gets the complete code of the watermarked programs.

1.2.2 A High Level Approach

At a high level, to watermark a PRF $F : \{0,1\}^n \to \{0,1\}^m$, we create a token $\widetilde{C}$ that evaluates F correctly on almost all inputs x, except for some special set of

[3] Alternately, one can think of this setting as assuming that $\widetilde{C}$ is obfuscated with an "ideal obfuscation" scheme. However, since software-only ideal obfuscation schemes don't exist, it's more accurate to think of $\widetilde{C}$ as a physical hardware token.

[4] For example, it's immediately clear that *exact* watermarking, where the marked program $\widetilde{C}$ is functionally equivalent to the original program C, is impossible in this setting since in that case the extraction procedure cannot distinguish between black-box access to the original unmarked program C and the marked program $\widetilde{C}$.

"marked-points" $\mathcal{X} \subseteq \{0,1\}^n$ which have negligible density in $\{0,1\}^n$. On the marked points, the watermarked program outputs specially constructed incorrect values that allow the extraction procedure to recover the embedded message. We will ensure that marked points are indistignuishable to the adversary from random inputs. Therefore, the adversary cannot create a new token C^* that agrees with $\widetilde{C}$ on a large fraction of random inputs (i.e., approximates F) but disagrees with $\widetilde{C}$ on sufficiently many marked points so as to cause the extraction procedure to fail.

1.2.3 A Simple Scheme with Weak Security

We start by considering a weak notion of token-based watermarking security, where both the marking key mk and the extraction key xk are secret and the adversary does not have access to either the marking oracle $\mathsf{Mark}(\mathsf{mk}, \cdot)$ or the extraction oracle $\mathsf{Extract}(\mathsf{xk}, \cdot)$. We also consider a messageless scheme where programs can only be marked or $\mathsf{unmarked}$. In particular, in the security game the adversary gets a single marked token $\widetilde{C} \leftarrow \mathsf{Mark}(\mathsf{mk}, F)$ as a challenge, where $F : \{0,1\}^n \to \{0,1\}^m$ is chosen at random from a PRF family $F \leftarrow \mathcal{F}$ (and n, m are super-logarithmic). The adversary's goal is to come up with some new token $C^* = C^*[\widetilde{C}]$ that approximately evaluates F but on which the extraction procedure fails to detect that the program is marked: $\mathsf{Extract}(\mathsf{xk}, C^*) = \mathsf{unmarked}$.

This can be easily achieved as follows. Choose a polynomial set of ℓ "marked-points" $\mathcal{X} = \{x_1, \ldots, x_\ell\} \subseteq \{0,1\}^n$ uniformly at random with corresponding random outputs $y_1, \ldots, y_\ell \leftarrow \{0,1\}^m$. Set $\mathsf{mk} = \mathsf{xk} = (x_1, \ldots, x_\ell, y_1, \ldots y_\ell)$. To mark a PRF F, the marking procedure $\widetilde{C} \leftarrow \mathsf{Mark}(\mathsf{mk}, F)$ outputs a token $\widetilde{C}$ that contains $x_1, \ldots, x_\ell, y_1, \ldots, y_\ell$ hard-coded and, on input x, if $x = x_i$ for some $i \in [\ell]$ it outputs y_i else it outputs $F(x)$. The extraction procedure $\mathsf{Extract}(\mathsf{xk}, C^*)$ tests if on *at least one* of the ℓ marked points $x_i \in \mathcal{X}$ the program evaluates to $C^*(x_i) = y_i$. If so, it outputs that the program is marked, and otherwise outputs $\mathsf{unmarked}$.

To prove that the above scheme is secure, we notice that an adversary that gets black-box access to a token $\widetilde{C} \leftarrow \mathsf{Mark}(\mathsf{mk}, F)$ for a random unknown $F \leftarrow \mathcal{F}$ cannot distinguish between the marked points $\mathcal{X} = \{x_1, \ldots, x_\ell\}$ and ℓ uniformly random and independent inputs without breaking PRF security. This implies that the adversary cannot come up with a new token $C^* = C^*[\widetilde{C}]$ such that $C^*(x) = \widetilde{C}(x)$ is "correct" on a large fraction of inputs $x \in \{0,1\}^n$, but $C^*(x_i) \neq \widetilde{C}(x_i) = y_i$ for all marked points $x_i \in \mathcal{X}$, as this would imply distinguishing between random points and marked points. More precisely, by setting $\ell = \Omega(\lambda/\varepsilon)$ where λ is the security parameter, we ensure that if the adversary creates any token $C^* = C^*[\widetilde{C}]$ that agrees with the marked token $\widetilde{C}$ on even an ε-fraction of inputs $x \in \{0,1\}^n$, then $C^*(x_i) = y_i$ for at least one marked point $x_i \in \mathcal{X}$ with overwhelming probability $1 - (1-\varepsilon)^\ell$ and therefore $\mathsf{Extract}(\mathsf{xk}, C^*) = \mathsf{marked}$ as desired.

1.2.4 Challenges in Allowing Mark/Extract Oracles

Unfortunately, the above scheme becomes completely insecure if the adversary has access to *either* a marking oracle $\mathsf{Mark}(\mathsf{mk}, \cdot)$ or the extraction oracle $\mathsf{Extract}(\mathsf{xk}, \cdot)$, let alone if the extraction key xk is made public. Let us describe the attacks.

Attack using the extraction oracle. If the adversary gets the challenge marked program $\widetilde{C} \leftarrow \mathsf{Mark}(\mathsf{mk}, F)$ as a token, he can create new tokens $C' = C'[\widetilde{C}]$ such that $C'(x) = \widetilde{C}(x)$ only for x satisfying $P(x) = 1$ where P is some predicate. By querying the extraction oracle $\mathsf{Extract}(\mathsf{xk}, C')$ to see if such tokens are deemed marked or $\mathsf{unmarked}$, the adversary will learn whether there exists some marked point x_i with $P(x_i) = 1$. By choosing such predicates carefully, these queries can completely reveal the marked points. [5]

[5]For example, a concrete instantiation of the above attack uses predicates of the form $P_w(x) = 1$ iff $x[1, \ldots, |w|] = w$ for some $w \in \{0,1\}^*$ (i.e., the first $|w|$ bits of x match w). By starting with w being the empty string, the adversary can iteratively add a bit to learn if there exists some marked point x_i with $P_{w||b}(x_i) = 1$ for $b \in \{0,1\}$. Whenever the above occurs for exactly one choice of $b \in \{0,1\}$, the adversary extends $w := w||b$ and continues to the next iteration. If this happens for both choices of $b \in \{0,1\}$ then the adversary branches the above process and continues down both paths for $w := w||0$ and $w := w||1$. Since there are ℓ marked points, this process will only branch ℓ times and the adversary will eventually recover all of the points $\mathcal{X} = \{x_1, \ldots, x_\ell\}$. Once the adversary learns $\mathcal{X}$, he can create a circuit $C^*[\widetilde{C}]$ such that $C^*(x) = \widetilde{C}(x)$ for any $x \notin \mathcal{X}$ and otherwise $C^*(x)$ outputs some incorrect value (e.g., an independent pseudorandom output). The circuit C^* closely approximates $\widetilde{C}$ (on all but a negligible fraction of inputs) yet the extraction procedure fails to detect C^* as marked.

Attack using the marking oracle. Assume the adversary makes just one call to the marking oracle with an arbitrary known PRF function $F' \in \mathcal{F}$ and gets back a token $\widetilde{C}' \leftarrow \mathsf{Mark}(\mathsf{mk}, F')$. In addition, the adversary gets a challenge token $\widetilde{C} \leftarrow \mathsf{Mark}(\mathsf{mk}, F)$ corresponding to a random unknown PRF $F \leftarrow \mathcal{F}$. The adversary can easily remove the mark by creating a new token $C^*[\widetilde{C}', \widetilde{C}]$ that gets oracle access to $\widetilde{C}'$ and $\widetilde{C}$ and does the following: on input x, if $\widetilde{C}'(x) = F'(x)$ then output $\widetilde{C}(x)$ else output some incorrect value (e.g., an independent pseudorandom output). The circuit C^* only differs from $\widetilde{C}$ on the marked points $x_i \in \mathcal{X}$ and therefore closely approximates $\widetilde{C}$ on all but a negligible fraction of inputs. However, the extraction procedure will fail to detect C^* as marked.

1.2.5 Toward a Fully Secure Token-Based Scheme

We now outline the main ideas for how to thwart the above attacks and get a token-based watermarking scheme with a public extraction key xk and with security in the presence of a marking oracle $\mathsf{Mark}(\mathsf{mk}, \cdot)$.

Overview. Our first idea is to make the set of marked points $\mathcal{X} \subseteq \{0,1\}^n$ super-polynomial, yet still of negligible density inside of $\{0,1\}^n$. This will allow us to thwart the attack using an extraction oracle and even make the extraction key xk public. In particular, we ensure that even given the extraction key xk, which can be used to sample random marked points $x \leftarrow \mathcal{X}$, the adversary still cannot distinguish such points from uniformly random inputs. Thwarting the marking oracle attack is more difficult. We need to ensure that the set of marked points $\mathcal{X}_F$ is different for each PRF F that we will mark so that, even if the adversary can test if a point belongs to $\mathcal{X}_{F_i}$ for various PRFs F_i that were queried to the marking oracle, the marked points $\mathcal{X}_F$ for the challenge (unknown) PRF F will remain indistinguishable from uniform. However, this creates a difficulty since the extraction procedure $\mathsf{Extract}(\mathsf{xk}, \widetilde{C})$ must test the marked program $\widetilde{C}$ on the correct set of marked points $\mathcal{X}_F$ without knowing the function F from which $\widetilde{C}$ was created. We solve this by ensuring that one can find a marked point for the function F by querying F. In particular, the extraction procedure first queries $\widetilde{C}(z)$ on some special (pseudo-random) "find point" z and then, assuming $\widetilde{C}(z) = F(z)$, uses the output $\widetilde{C}(z)$ to sample a marked point $x \leftarrow \mathcal{X}_F$.

A concrete scheme. Let $\mathcal{F}$ be a PRF family consisting of functions $F : \{0,1\}^n \to \{0,1\}^\lambda$ where λ is the security parameter and n is sufficiently large. Let $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ be a CCA secure public-key encryption scheme with *pseudorandom ciphertexts* having message space $\{0,1\}^{3\lambda}$ and ciphertext space $\{0,1\}^n$.[6] Let $G : \{0,1\}^\lambda \to \{0,1\}^n$ be a PRG.

KEYS: We sample a key pair for the encryption scheme $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Gen}(1^\lambda)$ and define the marking/extraction key mk, xk to be the secret/public key respectively: $\mathsf{mk} = \mathsf{sk}, \mathsf{xk} = \mathsf{pk}$.

MARKING: For a PRF $F \in \mathcal{F}$, we define the set of "marked points" as:

$$\mathcal{X}_F = \{x \in \{0,1\}^n \ : \ \mathsf{Dec}_{\mathsf{sk}}(x) = (a||b||c) \in \{0,1\}^{3\lambda}, F(G(a)) = b\}.$$

To mark a PRF F the procedure $\widetilde{C} \leftarrow \mathsf{Mark}(\mathsf{mk}, F)$ creates a token $\widetilde{C}$ defined as follows:

Hard-Coded Constants: F, sk.

Input: $x \in \{0,1\}^n$

```
1. Try to decrypt $a||b||c \leftarrow \mathsf{Dec}_{\mathsf{sk}}(x)$ with $a, b, c \in \{0,1\}^\lambda$.
2. If decryption succeeds and $F(G(a)) = b$ output $c$.    // $x \in \mathcal{X}_F$ is a marked point
3. Otherwise output $F(x)$.
```

EXTRACTION: The extraction procedure $\mathsf{Extract}(\mathsf{xk}, C^*)$ repeats the following ℓ times:

[6] For simplicity, we assume ciphertexts are pseudorandom in $\{0,1\}^n$. For our full construction we will construct such schemes with additional puncturability properties using PRFs and iO. However, we can generalize this to other domains beside $\{0,1\}^n$ and, in the token-based setting, we could then rely on standard constructions of CCA secure encryption such as e.g., Cramer-Shoup [CS03].

- Choose random $a, c \leftarrow \{0,1\}^\lambda$ and let $z = G(a)$ and $b = C^*(z)$. // z is a find point
- Choose $x \leftarrow \mathsf{Enc}_{\mathsf{pk}}(a||b||c)$ and if $C^*(x) = c$ then output marked. // if $b = F(z)$ then $x \in \mathcal{X}_F$.

If all ℓ iterations fail, output $\mathsf{unmarked}$.

Intuitively, the construction relies on the fact that the marked program $\widetilde{C}$ can recognize marked points by using the decryption key. On the other hand the extraction procedure can find the marked points for a function F given a circuit C^* that approximates F by querying $C^*(z)$ where $z = G(a)$ is a "find point". If the circuit answers correctly on z so that $F(z) = C^*(z) = b$ then the extraction procedure will be able to correctly sample a marked point $x \leftarrow \mathsf{Enc}_{\mathsf{pk}}(a||b||c)$.

Security analysis overview. For the security analysis, consider an adversary that gets an extraction key $\mathsf{xk} = \mathsf{pk}$ and makes q queries to the marking oracle with arbitrary PRF functions $F_i \in \mathcal{F}$ and gets back marked tokens $\widetilde{C}_i \leftarrow \mathsf{Mark}(\mathsf{mk}, F_i)$. The adversary then gets a challenge marked token $\widetilde{C} \leftarrow \mathsf{Mark}(\mathsf{mk}, F)$ for a random unknown PRF $F \leftarrow \mathcal{F}$. The adversary can only query the tokens as a black box.

Firstly, we claim that even given the above view, the adversary cannot distinguish between getting random find/mark points z, x and completely random values z', x':

$$(\mathsf{view}, z, x) \approx (\mathsf{view}, z', x')\text{sampled according to}$$

$$a, c \leftarrow \{0,1\}^n, z = G(a), b = F(z), x \leftarrow \mathsf{Enc}_{\mathsf{pk}}(a||b||c), z', x' \leftarrow \{0,1\}^n.$$

To show this, we can first rely on CCA security to switch x to a uniformly random x'. This is because black-box access to the marked tokens $\widetilde{C}_i$ can be simulated by a CCA oracle that never decrypts x (it's unlikely that $F(z) = F_i(z)$ for some i, and therefore x is not a marked point for the queried functions F_i with overwhelming probability) while the challenge program $\widetilde{C}$ outputs $\widetilde{C}(x) = c$ but this is indistinguishable from $\widetilde{C}(x') = F(x')$ since both outcomes look random. We then rely on PRG security to switch z to uniform.

Secondly, we claim that the above "indistinguishability" property immediately implies "unremovability". In particular, if the adversary manages to produce a token C^* that ε-approximates the challenge program $\widetilde{C}$ then, for a random $z', x' \leftarrow \{0,1\}^n$ the probability that $C^*(z') = \widetilde{C}(z')$ and $C^*(x') = \widetilde{C}(x')$ is at least ε^2. Therefore, the same must hold (up to a negligible difference) when x, z are a random find/marked point. This means that each iteration of the extraction procedure outputs marked with probability at least ε^2 and therefore the probability that none of the iterations outputs marked is at most $(1-\varepsilon^2)^\ell$ which is negligible as long as $\ell = \Omega(\lambda/\varepsilon^2)$.

This analysis only provides *lunch-time security* where the adversary can query the marking oracle only prior to seeing the challenge program $\widetilde{C}$. This is because we relied on the fact that, with overwhelming probability, none of the queried functions F_i will satisfy $F_i(z) = F(z)$ where F is the challenge PRF. This may not hold in a stronger security model where the adversary can adaptively query the marking oracle with function F_i after seeing the watermarked version $\widetilde{C}$ of the challenge PRF F. However, we can salvage the same analysis and make it hold in the stronger model if we assume the PRF family satisfies an additional *injective* property, meaning that when $F \neq F'$ then $F(z) \neq F'(z)$ for all inputs z. We can construct such PRFs under natural assumptions such as DDH or LWE.

Embedding a message. We can extend the above construction to embed a message in the marked program. We do so by ensuring that the outputs of the marked circuit on the marked points x encode information about the message m, which can then be recovered by the extraction procedure. In particular, instead of simply having the marked circuit output the value c encrypted in the marked point x, we make it output $c \oplus \mathsf{m}$ where m is message we wish to embed. The extraction procedure can work as above but in each iteration $i = 1, \ldots, \ell$ it recovers a candidate message m_i. We simply test if there is a message which is recovered in a majority of the iterations. If so we output it, and otherwise we output $\mathsf{unmarked}$. A naive implementation of this approach would only work for an approximation factor $\varepsilon > 1/\sqrt{2}$ since only in that case could we expect that C^* answers correctly on both the find point and the marked point simultaneously with probability $> 1/2$ so as to get a correct majority. We show how to tweak the above approach to make it work for optimal approximation factor $\varepsilon > 1/2$ by testing C^* on many marked points for each find point and taking a majority-of-majorities.

On approximation factors. One might claim that it is limiting to consider adversaries that output a circuit satisfying $1/\operatorname{poly}(\lambda)$ for messageless watermarking schemes (resp. $1/2$ for message-embedding schemes). As we see in Section 1.7.2, approximation factor $1/2$ for message-embedding schemes is optimal. Approximation factor $1/\operatorname{poly}(\lambda)$ for the messageless scheme is essential in our proof as we saw in the technical overview in this section. However, we do not know such a lower bound for messageless schemes. There might be a possibility to achieve messageless watermarking scheme that satisfies $\mathsf{negl}(\lambda)$ approximation factor. Therefore, whether approximation factor $\mathsf{negl}(\lambda)$ is possible or not is an interesting research problem.

1.2.6 Using Indistinguishability Obfuscation

Lastly, we briefly mention our techniques for moving beyond token-based watermarking. On a high level, we can simply obfuscate the watermarked programs $\widetilde{C}$, instead of thinking of them as hardware tokens. However, the fact that we only have iO rather than ideal obfuscation makes this step non-trivial. Indeed, the token-based model can give false intuition since it allows us to watermark *any* PRF family but we show that in the standard model there are PRF families that cannot be watermarked. Nevertheless, it turns out that we can adapt the techniques from the token-based model to also work in the standard model using iO. The main differences are that: (1) we need the PRF family F that we are watermarking to be a *puncturable* PRF family,[7] (2) instead of a standard CCA secure encryption, we need a special type of puncturable encryption scheme where we can create a punctured secret key which doesn't decrypt a particular ciphertext. The latter primitive may be of independent interest and we

[7]See Footnote 2, Definition 1.3.4

show how to construct it using iO. We use a careful sequence on hybrids to show that the above changes are sufficient to get a provably secure watermarking scheme in the standard model.

1.2.7 Related Work

There has been a large body of work on watermarking in the applied research community. Notable contributions of this line of research include the discovery of *protocol attacks* such as the copy attack by Kutter, Voloshynovskiy and Herrigel [KVH00] and the ambiguity attack by Adelsback, Katzenbeisser and Veith [AKV03]. However, these works do not formally define security guarantees, and have resulted in a cat-and-mouse game of designing watermarking schemes that are broken fairly immediately.

We mention that there are several other works [NSS99, YF11, Nis13] that propose concrete schemes for watermarking cryptographic functions, under several different definitions and assumptions. For example, the work of Nishimaki [Nis13] gives formal definitions and provably secure constructions for watermarking cryptographic functions (such as trapdoor functions). The main aspect that sets our work apart from these works is that they only consider restricted attacks which attempt to remove a watermark by outputting a new program which has some specific format (rather than an arbitrary program). In particular, for all of these schemes, the mark can be removed via the attack described in [BGI$^+$01a, BGI$^+$12] where an adversary uses iO to obfuscate the marked program so as to preserve its functionality but completely change its structure.

Barak et al. [BGI$^+$01a, BGI$^+$12] proposed simulation-based and indistinguishability-based definitions of watermarking security; their main contribution is a negative result, described earlier in the introduction, which shows that indistinguishability obfuscation rules out any meaningful form of watermarking that exactly preserves functionality. Finally, Hopper, Molnar and Wagner [HMW07] formalized strong notions of watermarking security with approximate functionality; our definitions are inspired by their work. Their definition considers not just unremovability but also the dual notion of unforgeability which requires that the only marked programs that an adversary can produce are functionally similar to circuits already marked by a marking oracle. Cohen, Holmgren, and Vaikuntanathan [CHV15] also gave a definition of unforgeability for watermarking based on that of Hopper et al., and achieved a watermarking scheme that satisfies unforgeability and unremovability simultaneously under some parameter regime.

A subsequent sequence of works has sought to minimize the cryptographic assumptions needed to watermark PRF families, albeit in a weaker security model (see below). Starting from the general framework developed in this work, Boneh, Lewi, and Wu showed that so-called private programmable PRFs [BLW17] suffice to construct a family of watermarked PRFs, but they were unable to instantiate this seemingly weaker primitive from any non-obfuscation assumption. Kim and Wu then constructed a family of watermarked PRFs from private translucent PRFs [KW17] and showed how to base the latter on the subexponential hardness of learning with errors (LWE). Finally, Peikert and Shiehian [PS17] presented a construction of privately programmable PRFs, also from subexponential LWE. In addition to reducing security to lattice-based assumptions, these constructions are able to achieve both unremovability and unforgeability while also reducing the number of marked points to a polynomial.

A major drawback of the lattice-based constructions is that they achieve a weaker notion of security. Specifically, they use a different definition of security that handicaps the adversary by restricting the oracles available to it. Most significantly, the adversary gets no access to an Extract oracle (let alone a public extraction key). Extract-oracle attacks contribute substantially to the challenge of constructing watermarking schemes. Secondly, the adversary's Mark oracle cannot take circuits as input. In the work of Kim and Wu, the Mark oracle receives a PRF key and gives the adversary a marked PRF key. In the work of Boneh et al., the Mark oracle samples and marks a random circuit from the family, giving the adversary both the marked and unmarked versions. These restrictions to the class of attacks available to the adversary are central to the results of the two works. Finally, the approximation factors of their schemes are worse than ours, that is, their approximation factors are

$1 - \mathsf{negl}(\lambda)$.

Baldimtsi, Kiayias, and Samari [BKS17] presented a weaker model of watermarking cryptographic functionalities and a concrete watermarking scheme for public-key encryption in their model. Their watermarking scheme for public-key encryption is based on one-way functions. In their model, a marking algorithm and a extraction algorithm *can share a state information.* That is, their scheme is a *stateful* construction. This is a significant difference from ours.

1.3 Preliminaries

1.3.1 Notation

For any $n \in \mathbb{N}$, we write $[n]$ to denote the set $\{1, \ldots, n\}$. For two strings x_1 and x_2, $x_1 \| x_2$ denotes a concatenation of x_1 and x_2.

When D is a distribution, we write $y \leftarrow D$ to denote that y is randomly sampled from D. If S is a set, then we will also write S to denote the uniform distribution on that set.

We say that a function $f : \mathbb{N} \to \mathbb{R}$ is negligible if for all constants $c > 0$, there exists $N \in \mathbb{N}$ such that for all $n > N$, $f(n) < n^{-c}$.

We use abbreviation PPT to denote probabilistic polynomial time.

If $\mathcal{X} = \{X_\lambda\}_{\lambda \in \mathbb{N}}$ and $\mathcal{Y} = \{Y_\lambda\}_{\lambda \in \mathbb{N}}$ are two ensembles of random variables indexed by $\lambda \in \mathbb{N}$, we say that $\mathcal{X}$ and $\mathcal{Y}$ are computationally indistinguishable if for all PPT algorithms $\mathcal{D}$, there exists a negligible function ν such that for all λ,

$$\Pr\left[\mathcal{D}(x_b) = b \;\middle|\; \begin{array}{l} x_0 \leftarrow X_\lambda \\ x_1 \leftarrow Y_\lambda \\ b \leftarrow \{0,1\} \end{array}\right] \leq \frac{1}{2} + \nu(\lambda).$$

We write $\mathcal{X} \overset{\mathsf{c}}{\approx} \mathcal{Y}$ to denote that $\mathcal{X}$ and $\mathcal{Y}$ are computationally indistinguishable.

For two circuits C and D, we write $C \equiv D$ if C and D compute exactly the same function. If C and D agree on an ε fraction of their inputs, we write $C \cong_\varepsilon D$.

1.3.2 Definitions

In this section, we review basic notions and definitions used in this chapter.

Obfuscation. The notion of indistinguishability obfuscation (iO) was proposed by Barak et. al. [BGI+01a, BGI+12] and the first candidate construction was proposed by Garg, Gentry, Halevi, Raykova, Sahai, and Waters [GGH+13a].

Definition 1.3.1 (Indistinguishability Obfuscation [BGI+12, GGH+13a]). *An indistinguishability obfuscator is a PPT algorithm $i\mathcal{O}$ satisfying the following two conditions.*

FUNCTIONALITY: *For every security parameter $\lambda \in \mathbb{N}$ and every circuit C, it holds with probability 1 that*

$$i\mathcal{O}(1^\lambda, C) \equiv C$$

INDISTINGUISHABILITY: *For all circuit families $C^0 = \{C^0_\lambda\}$ and $C^1 = \{C^1_\lambda\}$ such that $C^0_\lambda \equiv C^1_\lambda$ are functionally equivalent and $|C^0_\lambda| = |C^1_\lambda|$, it holds that*

$$\{i\mathcal{O}(1^\lambda, C^0_\lambda)\}_\lambda \overset{\mathsf{c}}{\approx} \{i\mathcal{O}(1^\lambda, C^1_\lambda)\}_\lambda$$

For simplicity, we write $i\mathcal{O}(C)$ instead of $i\mathcal{O}(1^\lambda, C)$ when the security parameter λ is clear from context.

Pseudorandom generators and functions. We review pseudorandom generators and several variants of pseudorandom functions (PRFs).

Definition 1.3.2 (Pseudorandom Generator). *A pseudorandom generator (PRG) $\mathsf{G} : \{0,1\}^\lambda \rightarrow \{0,1\}^{\lambda+\ell(\lambda)}$ with stretch $\ell(\lambda)$ (ℓ is some polynomial function) is a polynomial-time computable function that satisfies $\mathsf{G}(U_\lambda) \overset{\mathsf{c}}{\approx} U_{\lambda+\ell(\lambda)}$ where U_m denotes the uniform distribution over $\{0,1\}^m$.*

Definition 1.3.3 (Pseudorandom Functions). *A pseudorandom function family $\mathcal{F} = \{\mathcal{F}_\lambda\}_{\lambda\in\mathbb{N}}$ is a function family where each function $F \in \mathcal{F}_\lambda$ maps a domain D to a* *range* R *and satisfies the following condition. For all PPT adversary* $\mathcal{A}$ *and* $F \leftarrow \mathcal{F}_\lambda$, *it holds*

$$\left|\Pr[\mathcal{A}^{F(\cdot)} = 1] - \Pr[\mathcal{A}^{\mathcal{R}(\cdot)} = 1]\right| \leq \mathsf{negl}(\lambda)$$

where $F(\cdot) : \mathsf{D} \to \mathsf{R}$ *is a deterministic function and* $\mathcal{R}$ *is chosen uniformly at random from the set of all functions with the same domain/range.*

In this chapter, we basically set $\mathsf{D} := \{0,1\}^{n(\lambda)}$ and $\mathsf{R} := \{0,1\}^{m(\lambda)}$ for a pair of polynomial-time computable functions $n(\cdot)$ and $m(\cdot)$.

The notion of puncturable pseudorandom function (pPRF) was proposed by Sahai and Waters [SW14, BW13, KPTZ13, BGI13].

Definition 1.3.4 (Puncturable Pseudorandom Functions). *A puncturable pseudorandom function (pPRF) family* $\mathcal{F}$ *is a function family with a "puncturing" algorithm* $\mathsf{Puncture}$ *where each function* $F \in \mathcal{F}_\lambda$ *maps a domain* $\{0,1\}^{n(\cdot)}$ *to a range* $\{0,1\}^{m(\cdot)}$ *that satisfies the following two conditions.*

FUNCTIONALITY PRESERVING UNDER PUNCTURING: *For all polynomial size sets* $S \subseteq \{0,1\}^{n(\lambda)}$ *and for all* $x \in \{0,1\}^{n(\lambda)} \setminus S$, *it holds that*

$$\Pr[F(x) = F\{S\}(x) \mid F \leftarrow \mathcal{F}_\lambda,\ F\{S\} := \mathsf{Puncture}(F,S)] = 1.$$

PSEUDORANDOM AT PUNCTURED POINTS: *For poly-size set* $S = \{x_1, \ldots, x_{k(\lambda)}\} \subseteq \{0,1\}^{n(\lambda)}$ *it holds that for all PPT adversary* $\mathcal{A}$,

$$\mu(\lambda) := \left|\Pr[\mathcal{A}(F\{S\}, \{F(x_i)\}_{i\in[k]}) = 1] - \Pr[\mathcal{A}(F\{S\}, U_{m(\lambda)\cdot|S|}) = 1]\right| \leq \mathsf{negl}(\lambda)$$

where $F \leftarrow \mathcal{F}_\lambda$, $F\{S\} := \mathsf{Puncture}(F,S)$ *and* U_ℓ *denotes the uniform distribution over* ℓ *bits.*

Theorem 1.3.5 ([GGM86, BW13, BGI13, KPTZ13]). *If one-way functions exist, then for all efficiently computable* $n(\cdot)$ *and* $m(\cdot)$, *there exists a pPRF family whose input is an* $n(\cdot)$ *bit string and output is an* $m()$ *bit string.*

Definition 1.3.6 (Injective pPRF). *If a pPRF family $\mathcal{F} = \{\mathcal{F}_\lambda\}_\lambda$ satisfies the following, we call it an injective ~~prefix~~ pPRF family. For all $F \in \mathcal{F}_\lambda$ and $x, x' \in \mathsf{D}$, if $x \neq x'$, then $F(x) \neq F(x')$.*

Sahai and Waters showed that we can convert any pPRF into a statistically injective pPRF [SW14]. Here, "statistically" means with probability $1 - \mathsf{negl}(\lambda)$ over the random choice of $F \leftarrow \mathcal{F}_\lambda$, $F(\cdot)$ is injective.

Definition 1.3.7 (Injective Bit-Commitment). *An injective bit-commitment function is a PPT algorithm* Com *which takes as input a security parameter λ and a bit $b \in \{0, 1\}$, and outputs a commitment c, satisfying the following properties.*

COMPUTATIONALLY HIDING:

$$\left\{\mathsf{Com}(1^\lambda, 0)\right\}_\lambda \approx \left\{\mathsf{Com}(1^\lambda, 1)\right\}_\lambda$$

PERFECTLY BINDING: *For every λ, it holds that*

$$\Pr\left[c_0 = c_1 \;\middle|\; \begin{array}{l} c_0 \leftarrow \mathsf{Com}(1^\lambda, 0) \\ c_1 \leftarrow \mathsf{Com}(1^\lambda, 1) \end{array}\right] = 0$$

INJECTIVE: *For every security parameter λ, there is a bound ℓ_{rand} on the number of random bits used by* Com *such that $\mathsf{Com}(1^\lambda, \cdot\ ; \cdot)$ is an injective function on $\{0, 1\} \times \{0, 1\}^{\ell_{rand}}$.*

Definition 1.3.8 (Universal One-Way Hash Function). *A universal one-way hash function (UOWHF) family $\mathcal{H} = \{\mathcal{H}_\lambda\}_{\lambda \in \mathbb{N}}$ is a function family where each function $H \in \mathcal{H}_\lambda$ maps a domain D to a range R and satisfies the following condition. For all PPT adversary $\mathcal{A} := (\mathcal{A}_1, \mathcal{A}_2)$, it holds*

$$\Pr\left[x \neq x^* \ \wedge\ H(x) = H(x^*) \;\middle|\; \begin{array}{l} (x, s) \leftarrow \mathcal{A}_1(1^\lambda), \\ H \leftarrow \mathcal{H}_\lambda, \\ x^* \leftarrow \mathcal{A}_2(1^\lambda, H, x, s) \end{array}\right] \leq \mathsf{negl}(\lambda).$$

Theorem 1.3.9 ([Rom90]). *If one-way functions exist, then UOWHFs exists.*

Hoeffding's inequality. We will use the following well-known bound. If $X_1, \dots, X_N$ are independent Bernoulli variables with parameter p, then

$$\Pr\left[\sum_i X_i \geq (p+\varepsilon)\cdot N\right] \leq e^{-2\varepsilon^2 N}$$

In particular, if $N > \frac{\lambda}{\varepsilon^2}$, then this probability is exponentially small in λ.

1.4 Definition of Watermarking

We begin by defining the notion of program watermarking. Our definition is similar to the game-based definition of Barak et al. [BGI+12, Definition 8.4] (It is called occasional watermarking) with the main difference that: (1) we allow "statistical" rather than perfect correctness, (2) the challenge circuit to be marked is chosen uniformly at random from the circuit family (for example, in the case of PRFs, this corresponds to marking a random PRF key), (3) we strengthen the definition to the public-key extraction setting and give the attacker access to the marking oracle.

Definition 1.4.1 (Watermarking Syntax). *A message-embedding watermarking scheme for a circuit class* $\{\mathcal{C}_\lambda\}_{\lambda\in\mathbb{N}}$ *and a message space* $\mathcal{M} = \{\mathcal{M}_\lambda\}$ *consists of three probabilistic polynomial-time algorithms* $(\mathsf{Gen}, \mathsf{Mark}, \mathsf{Extract})$.

KEY GENERATION: $(\mathsf{xk}, \mathsf{mk}) \leftarrow (\mathsf{Gen}(1^\lambda)$ *takes as input the security parameterin unary and outputs a pair of keys an extraction key* xk *and mark key* mk.

MARK: $\widetilde{C} \leftarrow \mathsf{Mark}(\mathsf{mk}, C, \mathsf{m})$ *takes as input a mark key, an arbitrary circuit* C *(not necessarily in* $\mathcal{C}_\lambda$*) and a message* $\mathsf{m} \in \mathcal{M}_\lambda$ *and outputs a marked circuit* $\widetilde{C}$.

EXTRACT: $\mathsf{m}' \leftarrow \mathsf{Extract}(\mathsf{xk}, C')$ *takes as input an extraction key and an arbitrary circuit* C'*, and outputs a message* $\mathsf{m}' \in \mathcal{M} \cup \{\mathsf{unmarked}\}$.

Definition 1.4.2 (Watermarking Security). *A watermarking scheme* $(\mathsf{Gen}, \mathsf{Mark}, \mathsf{Extract})$ *for circuit family* $\{\mathcal{C}_\lambda\}_{\lambda\in\mathbb{N}}$ *and with message space* $\mathcal{M} = \{\mathcal{M}_\lambda\}$ *is required to satisfy the following properties.*

STATISTICAL CORRECTNESS: *There is a negligible function $\nu(\lambda)$ such that for any circuit $C \in \mathcal{C}_\lambda$, any message $\mathsf{m} \in \mathcal{M}_\lambda$ and any input x in the domain of C, it holds that*

$$\Pr\left[\widetilde{C}(x) = C(x) \;\middle|\; \begin{array}{c} (\mathsf{xk}, \mathsf{mk}) \leftarrow \mathsf{Gen}(1^\lambda) \\ \widetilde{C} \leftarrow \mathsf{Mark}(\mathsf{mk}, C, \mathsf{m}) \end{array}\right] \geq 1 - \nu(\lambda).$$

EXTRACTION CORRECTNESS: *For every $C \in \mathcal{C}_\lambda$, $\mathsf{m} \in \mathcal{M}_\lambda$ and $(\mathsf{xk}, \mathsf{mk}) \leftarrow \mathsf{Gen}(1^\lambda)$:*

$$\Pr[\mathsf{m}' \neq \mathsf{m} \mid \mathsf{m}' \leftarrow \mathsf{Extract}(\mathsf{xk}, \mathsf{Mark}(\mathsf{mk}, C, \mathsf{m}))] \leq \mathsf{negl}(\lambda).$$

MEANINGFULNESS: *For every circuit C (not necessarily in $\mathcal{C}_\lambda$), it holds that*

$$\Pr_{(\mathsf{xk},\mathsf{mk}) \leftarrow \mathsf{Gen}(1^\lambda)}[\mathsf{Extract}(\mathsf{xk}, C) \neq \mathsf{unmarked}] \leq \mathsf{negl}(\lambda).$$

ε-UNREMOVABILITY: *For every PPT $\mathcal{A}$ we have*

$$\Pr[\mathsf{Exp}^{\mathsf{nrmv}}_{\mathcal{A}}(\lambda, \varepsilon) = 1] \leq \mathsf{negl}(\lambda)$$

where ε is a parameter of the scheme called the approximation factor *and $\mathsf{Exp}^{\mathsf{nrmv}}_{\mathcal{A}}(\lambda, \varepsilon)$ is the game defined next.*

We say a watermarking scheme is ε-secure if it satsifies these properties.

Definition 1.4.3 (ε-Unremovability Security Game). *The game $\mathsf{Exp}^{\mathsf{nrmv}}_{\mathcal{A}}(\lambda, \varepsilon)$ is defined as follows.*

1. *The challenger generates $(\mathsf{xk}, \mathsf{mk}) \leftarrow \mathsf{Gen}(1^\lambda)$ and gives xk to the adversary $\mathcal{A}$.*

2. *The adversary has oracle access to the mark oracle $\mathcal{MO}$. If $\mathcal{MO}$ is queried a circuit C_i (not necessarily in $\mathcal{C}_\lambda$) and message m_i, then it answers with $\mathsf{Mark}(\mathsf{mk}, C_i, \mathsf{m}_i)$.*

3. *At some point, the adversary makes a query to the challenge oracle $\mathcal{CO}$. If $\mathcal{CO}$*

is queried with a message $\mathsf{m} \in \mathcal{M}_\lambda$*, it samples a circuit* $C \leftarrow \mathcal{C}_\lambda$ *uniformly at random and answers* $\widetilde{C} \leftarrow \mathsf{Mark}(\mathsf{mk}, C, \mathsf{m})$.

4. *Again,* $\mathcal{A}$ *queries many pairs of a circuit and a message to* $\mathcal{MO}$.

5. *Finally, the adversary outputs a circuit* C^**. If it holds that* $C^* \cong_\varepsilon C$ *and* $\mathsf{Extract}(\mathsf{xk}, C^*) \neq \mathsf{m}$ *then the experiment outputs* 1*, otherwise* 0. [8]

Our main construction achieves what we call "lunch-time security", in which step 4 of the above game is omitted. This and other variations are discussed in Section 1.7.

1.5 Puncturable Encryption

One of our main abstractions is a *puncturable encryption* system. This is a public-key encryption system in which the decryption key can be punctured on a set of ciphertexts. We will rely on a strong ciphertext pseudorandomness property which holds even given access to a punctured decryption key. We will additionally require that valid ciphertexts are *sparse*, and that a decryption key punctured at two ciphertexts $\{c_0, c_1\}$ is functionally equivalent to the non-punctured decryption key, except possibly on $\{c_0, c_1\}$.

In this section we define the puncturable encryption abstraction that we use in Section 1.6. We instantiate this definition in Section 1.5.1 and prove its security Section 1.5.2.

Definition 1.5.1 (Puncturable Encryption Syntax). *Syntactically, a puncturable encryption scheme* PE *for a message space* $\mathcal{M} = \{0,1\}^\ell$ *is a triple of probabilistic algorithms* $(\mathsf{Gen}, \mathsf{Puncture}, \mathsf{Enc})$ *and a deterministic algorithm* Dec*. The space of ciphertexts will be* $\{0,1\}^n$ *where* $n = \mathrm{poly}(\ell, \lambda)$*. For clarity and simplicity, we will restrict our exposition to the case when* $\lambda = \ell$.

KEY GENERATION: $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Gen}(1^\lambda)$ *takes the security parameter in unary, and outputs an encryption key* pk *and a decryption key* sk.

[8]The definition would be equivalent if we had required $C^* \cong_\varepsilon \widetilde{C}$ instead of $C^* \cong_\varepsilon C$, up to a negligible difference in ε, since by statistical correctness we have $C \cong_\delta \widetilde{C}$ for some $\delta = 1 - \mathsf{negl}(\lambda)$.

PUNCTURING: $\mathsf{sk}\{c_0, c_1\} \leftarrow \mathsf{Puncture}(\mathsf{sk}, c_0, c_1)$ *takes a decryption key* sk, *and a set* $\{c_0, c_1\} \subset \{0,1\}^n$.[9] $\mathsf{Puncture}$ *outputs a "punctured" decryption key* $\mathsf{sk}\{c_0, c_1\}$.

ENCRYPTION: $c \leftarrow \mathsf{Enc}(\mathsf{pk}, m)$ *takes an encryption key* pk *and a message* $m \in \{0,1\}^\ell$, *and outputs a ciphertext* c *in* $\{0,1\}^n$.

DECRYPTION: m *or* $\perp \leftarrow \mathsf{Dec}(\mathsf{sk}, c)$ *takes a possibly punctured decryption key* sk *and a string* $c \in \{0,1\}^n$. *It outputs a message* m *or the special symbol* $\perp$.

Definition 1.5.2 (Puncturable Encryption Security). *A puncturable encryption scheme* $\mathsf{PE} = (\mathsf{Gen}, \mathsf{Puncture}, \mathsf{Enc}, \mathsf{Dec})$ *with message space* $\mathcal{M}$ *is required to satisfy the following properties.*

CORRECTNESS: *We require that for all messages* m,

$$\Pr\left[\mathsf{Dec}(\mathsf{sk}, c) = m \;\middle|\; \begin{array}{l} (\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Gen}(1^\lambda), \\ c \leftarrow \mathsf{Enc}(\mathsf{pk}, m) \end{array}\right] = 1.$$

PUNCTURED CORRECTNESS: *We also require the same to hold for keys which are punctured. For all possible keys* $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Gen}(1^\lambda)$, *all strings* $c_0, c_1 \in \{0,1\}^n$, *all punctured keys* $\mathsf{sk}' \leftarrow \mathsf{Puncture}(\mathsf{sk}, c_0, c_1)$, *and all potential ciphertexts* $c \in \{0,1\}^n \setminus \{c_0, c_1\}$:

$$\mathsf{Dec}(\mathsf{sk}, c) = \mathsf{Dec}(\mathsf{sk}', c).$$

CIPHERTEXT PSEUDORANDOMNESS: *We require that in the following game, all PPT adversaries* $\mathcal{A}$ *have negligible advantage.*

Game 1.5.3 (Ciphertext Pseudorandomness).

1. $\mathcal{A}$ *sends a message* m^* *to the challenger.*
2. *The challenger does the following:*
   - *Samples* $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Gen}(1^\lambda)$
   - *Computes encryption* $c^* \leftarrow \mathsf{Enc}(\mathsf{pk}, m^*)$.

[9] We can assume that the set $\{c_0, c_1\}$ is represented as a list in sorted order.

- *Samples* $r^* \leftarrow \{0,1\}^n$.
- *Generates the punctured key* $\mathsf{sk}' \leftarrow \mathsf{Puncture}(\mathsf{sk}, \{c^*, r^*\})$
- *Samples* $b \leftarrow \{0,1\}$ *and sends the following to* $\mathcal{A}$*:*

$$\begin{array}{ll} (c^*, r^*, \mathsf{pk}, \mathsf{sk}') & \textit{if } b = 0 \\ (r^*, c^*, \mathsf{pk}, \mathsf{sk}') & \textit{if } b = 1 \end{array}$$

3. *The adversary outputs* b' *and wins if* $b = b'$*.*

SPARSENESS: *We also require that most strings are not valid ciphertexts:*

$$\Pr\left[\mathsf{Dec}(\mathsf{sk}, c) \neq \bot \;\middle|\; (\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Gen}(1^\lambda), c \leftarrow \{0,1\}^n\right] \leq \mathsf{negl}(\lambda).$$

Remark 1.5.4. The notion of puncturable encryption is similar to that of puncturable deterministic encryption (PDE) introduced by Waters [Wat15a]. However, there are differences between them: (1) PDE is *symmetric* key encryption, that is, an encryption key is equal to a decryption key. (2) A key is punctured at *plaintexts* in PDE. (3) Ciphertexts are *not required to be pseudorandom* in PDE. Therefore, puncturable encryption is a stronger tool than PDE.

One of our contributions is the following theorem.

Theorem 1.5.5. *Assuming the existence of injective one-way functions, and an indistinguishability obfuscator for all circuits, there exists a puncturable encryption system.*

We provide a construction of the puncturable encryption in the next section.

1.5.1 Construction

We construct a puncturable encryption scheme in which the length n of ciphertexts is 12 times the length ℓ of plaintexts. Our construction utilizes the following ingredients:

- A length-doubling $\mathsf{PRG} : \{0,1\}^\ell \rightarrow \{0,1\}^{2\ell}$

- A family of injective pPRFs (See Definition 1.3.6) $\{\mathcal{F}_\lambda : \{0,1\}^{3\ell} \to \{0,1\}^{9\ell}\}$. [10]
- A family of pPRFs $\{G_\lambda : \{0,1\}^{9\ell} \to \{0,1\}^{\ell}\}$.
- An injective bit-commitment Com using randomness in $\{0,1\}^{9\ell}$, which can in fact be constructed by an injective one-way function. We only use this in our security proof.

Construction 1.5.6 (Punctruable Encryption Scheme PE).

$\mathsf{Gen}(1^\lambda)$: *Sample functions $F \leftarrow \mathcal{F}_\lambda$ and $G \leftarrow \mathcal{G}_\lambda$, generates* pk *as the $i\mathcal{O}$-obfuscation of the program E in Figure 1-1, and returns* $(\mathsf{pk}, \mathsf{sk}) := (i\mathcal{O}(E), D)$*, where* sk *is the (un-obfuscated) program D in Figure 1-2.*

$\mathsf{Puncture}(\mathsf{sk}, c_0, c_1)$: *Output* sk'*, where* sk' *is the $i\mathcal{O}$-obfuscation of the program D' described in Figure 1-3, that is,* $\mathsf{sk}' := i\mathcal{O}(D')$.

$\mathsf{Enc}(\mathsf{pk}, m)$: *Take $m \in \{0,1\}^\ell$, sample $r \leftarrow \{0,1\}^\ell$, and outputs* $c \leftarrow \mathsf{pk}(m, r)$.

$\mathsf{Dec}(\mathsf{sk}, c)$: *Take $c \in \{0,1\}^{12\ell}$ and returns* $m := \mathsf{sk}(c)$.

The size of the programs is appropriately padded to be the maximum size of all modified programs, which will appear in the security proof.

Remark 1.5.7. We note that in all of our obfuscated programs (including the hybrids), whenever α_i or β_i or γ_i for $i \in \{0,1\}$ are treated symmetrically, then we can and do store them in lexicographical order. A random ordering would also suffice for security.

Correctness and punctured correctness. Correctness follows from the fact that indistinguishability obfuscation exactly preserves functionality, and observing in the punctured case that sk' is defined to be functionally equivalent to sk except on inputs in $\{c_0, c_1\}$.

[10] As in [SW14], any puncturable PRF family from $\{0,1\}^k \to \{0,1\}^{2k+\omega(\log\lambda)}$ can be made statistically injective (with no additional assumptions) by utilizing a family of pairwise-independent hash functions.

```
Constants: Injective pPRF $F : \{0,1\}^{3\ell} \to \{0,1\}^{9\ell}$, pPRF $G : \{0,1\}^{9\ell} \to \{0,1\}^{\ell}$
Inputs: $m \in \{0,1\}^{\ell}, r \in \{0,1\}^{\ell}$

  1. Compute $\alpha = \mathsf{PRG}(r)$.
  2. Compute $\beta = F(\alpha \| m)$.
  3. Compute $\gamma = G(\beta) \oplus m$.
  4. Output $(\alpha, \beta, \gamma)$.
```

Figure 1-1: Encryption Program E (pre-obfuscation)

```
Constants: Injective pPRF $F : \{0,1\}^{3\ell} \to \{0,1\}^{9\ell}$, pPRF $G : \{0,1\}^{9\ell} \to \{0,1\}^{\ell}$
Inputs: $c = (\alpha \| \beta \| \gamma)$, where $\alpha \in \{0,1\}^{2\ell}$, $\beta \in \{0,1\}^{9\ell}$, and $\gamma \in \{0,1\}^{\ell}$.

  1. Compute $m = G(\beta) \oplus \gamma$.
  2. If $\beta = F(\alpha \| m)$, output $m$.
  3. Else output $\perp$.
```

Figure 1-2: Decryption Program D

```
Constants: Set $\{c_0, c_1\} \subset \{0,1\}^n$, injective pPRF $F : \{0,1\}^{3\ell} \to \{0,1\}^{9\ell}$, and
pPRF $G : \{0,1\}^{9\ell} \to \{0,1\}^{\ell}$
Inputs: $c = (\alpha \| \beta \| \gamma)$, where $\alpha \in \{0,1\}^{2\ell}$, $\beta \in \{0,1\}^{9\ell}$, and $\gamma \in \{0,1\}^{\ell}$.

  1. If $c \in \{c_0, c_1\}$, output $\perp$.
  2. Compute $m = G(\beta) \oplus \gamma$.
  3. If $\beta = F(\alpha \| m)$, output $m$.
  4. Else output $\perp$.
```

Figure 1-3: Punctured Decryption Program D' at $\{c_0, c_1\}$ (pre-obfuscation)

Sparseness. Sparseness follows from, for example, the length-doubling PRG; most values of α are not in the image of PRG.

Therefore, what remains is proving ciphertext pseudorandomness. We provide the proof in the next section.

Table 1.1: An overview of hybrid distributions. The table is split in two to fit within the margins. The differences between each hybrid distribution and the previous one are underlined.

Hybrid	α_0	β_0	γ_0	pk $:= i\mathcal{O}$ of below	sk$' := i\mathcal{O}$ of below
REAL_0	$\mathsf{PRG}(t)$	$F(\alpha_0 \| m^*)$	$G(\beta_0) \oplus m^*$	E	D'
Hyb_1	$\underline{\text{random}}$	$F(\alpha_0 \| m^*)$	$G(\beta_0) \oplus m^*$	E	D'
Hyb_2	random	$F(\alpha_0 \| m^*)$	$G(\beta_0) \oplus m^*$	$\underline{E\{\alpha_0 \| m^*, \alpha_1 \| m^*\}}$	$\underline{D'_2\{\alpha_0 \| m^*, \alpha_1 \| m^*\}}$
Hyb_3	random	$\underline{\text{random}}$	$G(\beta_0) \oplus m^*$	$E\{\alpha_0 \| m^*, \alpha_1 \| m^*\}$	$\underline{D'_3\{\alpha_0 \| m^*, \alpha_1 \| m^*\}}$
Hyb_4	random	random	$G(\beta_0) \oplus m^*$	$\underline{E\{\alpha_0 \| m^*, \alpha_1 \| m^*, \beta_0, \beta_1\}}$	$\underline{D'_4\{\alpha_0 \| m^*, \alpha_1 \| m^*, \beta_0, \beta_1\}}$
Hyb_5	random	random	$\underline{\text{random}}$	$E\{\alpha_0 \| m^*, \alpha_1 \| m^*, \beta_0, \beta_1\}$	$D'_4\{\alpha_0 \| m^*, \alpha_1 \| m^*, \beta_0, \beta_1\}$
RAND	random	random	random	$\underline{E}$	$\underline{D'}$

1.5.2 Ciphertext Pseudorandomness

Theorem 1.5.8. *If $\mathcal{F}$ is an injective pPRF family, $\mathcal{G}$ is a pPRF family,* PRG *is a pseudorandom generator,* Com *is a injective bit-commitment function, and $i\mathcal{O}$ is a secure iO, then the* PE *scheme above satisfies the ciphertext pseudorandomness.*

Proof. We give a sequence of main hybrid distributions Hyb_1 through Hyb_5. The goal of the hybrids to reach a game in which the challenge encryption c_0 and the random ciphertext c_1 are treated symmetrically in pk and sk$'$, and in which both are sampled uniformly at random by the challenger. We proceed by iteratively replacing pieces of c_0 by uniformly random values, puncturing F and G as necessary. We give an overview of the hybrids in Table 1.1.

REAL_b: The real distribution is defined by the real security game:

1. $\mathcal{A}$ sends a message $m^* \in \mathcal{M}$ to the challenger.
2. The challenger does the following:

(a) Samples an injective pPRF $F : \{0,1\}^{3\ell} \to \{0,1\}^{9\ell}$ and pPRF $G : \{0,1\}^{9\ell} \to \{0,1\}^{\ell}$.

Samples $t \leftarrow \{0,1\}^{\ell}$,

$\alpha_0 = \mathsf{PRG}(t) \in \{0,1\}^{2\ell}$,

$\beta_0 = F(\alpha_0 \| m^*)$,

$\gamma_0 = G(\beta_0) \oplus m^*$.

Let $c_0 = \alpha_0 \| \beta_0 \| \gamma_0$.

(b) Samples $c_1 \leftarrow \{0,1\}^{12\ell}$.

Parse $c_1 = \alpha_1 \| \beta_1 \| \gamma_1$.

(c) Generates pk as the $i\mathcal{O}$-obfuscation of Figure 1-1 and sk' as the $i\mathcal{O}$-obfuscation of Figure 1-3.

(d) Samples $b \leftarrow \{0,1\}$ and sends the following to $\mathcal{A}$:

$$\begin{aligned} &(c_0, c_1, \mathsf{pk}, \mathsf{sk}') \quad \text{if } b = 0 \\ &(c_1, c_0, \mathsf{pk}, \mathsf{sk}') \quad \text{if } b = 1 \end{aligned}$$

3. The adversary outputs b' and wins if $b = b'$.

That is, REAL_0 is $(c_0, c_1, \mathsf{pk}, \mathsf{sk}')$ and REAL_1 is $(c_1, c_0, \mathsf{pk}, \mathsf{sk}')$.

RAND: Before we define several hybrid distributions, we define an intermediate hybrid between REAL_0 and REAL_1. We define RAND as $(r', c_1, \mathsf{pk}, \mathsf{sk}')$ where r' is a uniformly random element in $\{0,1\}^{12\ell}$.

Hyb_1: We sample uniformly random $\alpha_0 \leftarrow \{0,1\}^{2\ell}$ for c_0.

Hyb_2: We puncture programs E and D' at $\{\alpha_0 \| m^*, \alpha_1 \| m^*\}$ by puncturing F at $\{\alpha_0 \| m^*, \alpha_1 \| m^*\}$. The new programs $E\{\alpha_0 \| m^*, \alpha_1 \| m^*\}$ and $D'_2\{\alpha_0 \| m^*, \alpha_1 \| m^*\}$ are described in Figure 1-4 and 1-5, respectively where $\hat{\beta} = F'(\alpha_1 \| m^*)$ and $\hat{\gamma} = G(\hat{\beta}) \oplus m^*$. The modifications are underlined.

Hyb_3: We sample uniformly random $\beta_0, \hat{\beta} \leftarrow \{0,1\}^{9\ell}$ for c_0 and slightly modify program $D'_{23}\{\alpha_0 \| m^*, \alpha_1 \| m^*\}$ defined in in Figure 1-6. The modifications are underlined.

```
Encryption Program $E\{\alpha_0\|m^*, \alpha_1\|m^*\}$
Constants: $\underline{\text{Punctured } F' = F\{\alpha_0\|m^*, \alpha_1\|m^*\}}$ and (not-punctured) $G$.
Inputs: $m \in \{0,1\}^\ell, r \in \{0,1\}^\ell$
  1. Compute $\alpha = \mathsf{PRG}(r)$.
  2. Compute $\beta = F'(\alpha\|m)$.
  3. Compute $\gamma = G(\beta) \oplus m$.
  4. Output $(\alpha, \beta, \gamma)$.
```

Figure 1-4: Program $E\{\alpha_0\|m^*, \alpha_1\|m^*\}$ (pre-obfuscation)

```
Constants: Set $\{c_0, c_1\} \subset \{0,1\}^n$, punctured $F' = F\{\alpha_0\|m^*, \alpha_1\|m^*\}$, $G$, and
the values $\alpha_0$, $\alpha_1$, $\hat{\beta}$, $\hat{\gamma}$, $m^*$.
Inputs: $c = (\alpha\|\beta\|\gamma)$, where $\alpha \in \{0,1\}^{2\ell}$, $\beta \in \{0,1\}^{9\ell}$, and $\gamma \in \{0,1\}^\ell$.
  1. $\underline{\text{If } \alpha = \alpha_1 \text{ and } \beta = \hat{\beta} \text{ and } \gamma = \hat{\gamma}, \text{ output } m^*.}$
  2. If $c \in \{c_0, c_1\}$, output $\perp$.
  3. Compute $m = G(\beta) \oplus \gamma$.
  4. $\underline{\text{If } (\alpha, m) \in \{(\alpha_0, m^*), (\alpha_1, m^*)\}, \text{ output } \perp.}$
  5. If $\beta = F'(\alpha\|m)$, output $m$.
  6. Else output $\perp$.
```

Figure 1-5: Punctured Program $D'_2\{\alpha_0\|m^*, \alpha_1\|m^*\}$ in Hyb_2 (pre-obfuscation)

```
Constants: Set $\{c_0, c_1\} \subset \{0,1\}^n$, punctured $F' = F\{\alpha_0\|m^*, \alpha_1\|m^*\}$, $G$, and
the values $\alpha_0$, $\alpha_1$.
Inputs: $c = (\alpha\|\beta\|\gamma)$, where $\alpha \in \{0,1\}^{2\ell}$, $\beta \in \{0,1\}^{9\ell}$, and $\gamma \in \{0,1\}^\ell$.
  1. $\underline{\text{Removed branch.}}$
  2. If $c \in \{c_0, c_1\}$, output $\perp$.
  3. Compute $m = G(\beta) \oplus \gamma$.
  4. If $(\alpha, m) \in \{(\alpha_0, m^*), (\alpha_1, m^*)\}$, output $\perp$.
  5. If $\beta = F'(\alpha\|m)$, output $m$.
  6. Else output $\perp$.
```

Figure 1-6: Punctured Program $D'_3\{\alpha_0\|m^*, \alpha_1\|m^*\}$ in Hyb_3 (pre-obfuscation)

Hyb_4: We puncture programs E and D' at $\{\alpha_0\|m^*, \alpha_1\|m^*, \beta_0, \beta_1\}$ by puncturing G at $\{\beta_0, \beta_1\}$. These modified programs are described in Figure 1-7 and 1-8.

```
Constants: Punctured $F' = F\{\alpha_0\|m^*, \alpha_1\|m^*\}$ and punctured $G' = G\{\beta_0, \beta_1\}$.
Inputs: $m \in \{0,1\}^\ell, r \in \{0,1\}^\ell$
  1. Compute $\alpha = \mathsf{PRG}(r)$.
  2. Compute $\beta = F'(\alpha\|m)$.
  3. Compute $\gamma = G'(\beta) \oplus m$.
  4. Output $(\alpha, \beta, \gamma)$.
```

Figure 1-7: Encryption Program $E\{\alpha_0\|m^*, \alpha_1\|m^*, \beta_0, \beta_1\}$ (pre-obfuscation)

```
Constants: Set $\{c_0, c_1\} \subset \{0,1\}^n$, punctured $F' = F\{\alpha_0\|m^*, \alpha_1\|m^*\}$, and punctured $G' = G\{\beta_0, \beta_1\}$, and the values $\alpha_0$, $\alpha_1$, $\beta_0, \beta_1$, $m^*$.
Inputs: $c = (\alpha\|\beta\|\gamma)$, where $\alpha \in \{0,1\}^{2\ell}$ and $\beta \in \{0,1\}^{9\ell}$.
  1. Removed branch.
  2. If $\beta \in \{\beta_0, \beta_1\}$, output $\perp$.
  3. Compute $m = G'(\beta) \oplus \gamma$.
  4. If $(\alpha, m) \in \{(\alpha_0, m^*), (\alpha_1, m^*)\}$, output $\perp$.
  5. If $\beta = F'(\alpha\|m)$, output $m$.
  6. Else output $\perp$.
```

Figure 1-8: Punctured Program $D'_4\{\alpha_0\|m^*, \alpha_1\|m^*\}$ in Hyb_4 (pre-obfuscation)

Hyb_5: We sample uniformly random $\gamma_0 \leftarrow \{0,1\}^\ell$ for c_0.

Our goal is to prove $\mathsf{REAL}_0 \overset{c}{\approx} \mathsf{Hyb}_1 \overset{c}{\approx} \mathsf{Hyb}_2 \overset{c}{\approx} \mathsf{Hyb}_3 \overset{c}{\approx} \mathsf{Hyb}_4 \overset{c}{\approx} \mathsf{Hyb}_5 \overset{c}{\approx} \mathsf{RAND}$ since we can prove $\mathsf{RAND} \overset{c}{\approx} \mathsf{REAL}_1$ in the reverse manner and it means $\mathsf{REAL}_0 \overset{c}{\approx} \mathsf{REAL}_1$.

Lemma 1.5.9. *If* PRG *is a pseudorandom generator, then* $\mathsf{Hyb}_0 \overset{c}{\approx} \mathsf{Hyb}_1$.

Proof of Lemma 1.5.9. These distributions are indistinguishable due to the pseudorandomness of PRG. □

Lemma 1.5.10. *If $\mathcal{F}$ is an injective pPRF family and iO is a secure iO, then* $\mathsf{Hyb}_1 \overset{c}{\approx} \mathsf{Hyb}_2$.

Proof of Lemma 1.5.10. To prove this lemma, we define auxiliary hybrids.

Hyb_1^1: We alter the generation of pk. We puncture F at $\alpha_0\|m^*$ and $\alpha_1\|m^*$ and use it for pk. That is, we use $F' = F\{\alpha_0\|m^*, \alpha_1\|m^*\}$ to generate the encryption program E.

Hyb_1^2: We modify the generation of sk'. The constants $\hat{\beta} = F(\alpha_1\|m^*)$ and $\hat{\gamma} = G(\hat{\beta}) \oplus m^*$ are hard-coded. We add the following line in the beginning of sk': "If $c \in \alpha_1\|\hat{\beta}\|\hat{\gamma}$, output m^*." . For reference, we describe the modified decryption program from Hybrid Hyb_1^2 in Figure 1-9.

```
Constants: Set $\{c_0, c_1\} \subset \{0,1\}^n$, punctured $F'$, and $G$, and the values $\alpha_0$, $\alpha_1$, $\hat{\beta}$, $\hat{\gamma}$, $m^*$.
Inputs: $c = (\alpha\|\beta\|\gamma)$, where $\alpha \in \{0,1\}^{2\ell}$, $\beta \in \{0,1\}^{9\ell}$, and $\gamma \in \{0,1\}^{\ell}$.
  1. $\underline{\text{If } \alpha = \alpha_1 \text{ and } \beta = \hat{\beta} \text{ and } \gamma = \hat{\gamma}, \text{ output } m^*.}$
  2. If $c \in \{c_0, c_1\}$, output $\perp$.
  3. Compute $m = G(\beta) \oplus \gamma$.
  4. If $\beta = F'(\alpha\|m)$, output $m$.
  5. Else output $\perp$.
```

Figure 1-9: Modified Program of D' in Hyb_1^2 (pre-obfuscation)

Hyb_1^3: We again modify the generation of sk'. We add the following check: "If $(\alpha, m) \in \{(\alpha_0, m^*), (\alpha_1, m^*)\}$, output $\perp$." For reference, we describe the modified decryption for sk' from Hybrid Hyb_1^3 in Figure 1-10.

Claim: If $\mathcal{F}$ is an injective pPRF family and iO is a secure iO, then $\mathsf{Hyb}_1 \overset{c}{\approx} \mathsf{Hyb}_1^1$.

Proof. A modified program that uses F' is functionally equivalent to E because F' is never evaluated on strings of these forms due to the uniform randomness of α_0, α_1. Values α_0 and α_1 are with high probability not in the image of PRG. Thus, the claim holds due to the functional equivalence explained above and the security of $i\mathcal{O}$. □

Claim: If iO is a secure iO, then $\mathsf{Hyb}_1^1 \overset{c}{\approx} \mathsf{Hyb}_1^2$.

```
Constants: Set $\{c_0, c_1\} \subset \{0,1\}^n$, punctured $F'$, and $G$, and the values $\alpha_0$, $\alpha_1$, $\hat{\beta}$, $\hat{\gamma}$, $m^*$.
Inputs: $c = (\alpha \| \beta \| \gamma)$, where $\alpha \in \{0,1\}^{2\ell}$, $\beta \in \{0,1\}^{9\ell}$, and $\gamma \in \{0,1\}^{\ell}$.
  1. If $\alpha = \alpha_1$ and $\beta = \hat{\beta}$ and $\gamma = \hat{\gamma}$, output $m^*$.
  2. If $c \in \{c_0, c_1\}$, output $\perp$.
  3. Compute $m = G(\beta) \oplus \gamma$.
  4. $\underline{\text{If } (\alpha, m) \in \{(\alpha_0, m^*), (\alpha_1, m^*)\}, \text{ output } \perp.}$
  5. If $\beta = F'(\alpha \| m)$, output $m$.
  6. Else output $\perp$.
```

Figure 1-10: Modified Program of D' in Hyb_1^3 (pre-obfuscation)

Proof. The decryption programs in these hybrids are functionally equivalent, as $\alpha_1 \| \hat{\beta} \| \hat{\gamma}$ is already a valid encryption of m^*. Notice, that these $\hat{\beta}$ do not correspond to either the β_0 or β_1 (and similarly for $\hat{\gamma}$). The claim holds due to the functional equivalence explained above and the security of $i\mathcal{O}$. □

Claim: If iO is a secure iO, then $\mathsf{Hyb}_1^2 \overset{c}{\approx} \mathsf{Hyb}_1^3$.

Proof. The decryption programs in these hybrids are functionally equivalent by two cases:

1. When $(\alpha, m) = (\alpha_0, m^*)$, then either $c = c_0$, in which case sk' already would output $\perp$, or $c \neq c_0$, in which case sk' rejects c as an invalid ciphertext (because every pair (α, m) together define a unique valid ciphertext due to the injective property of F).
2. When $(\alpha, m) = (\alpha_1, m^*)$, we only reach this line if $c \neq \alpha_1 \| \hat{\beta} \| \hat{\gamma}$ (by the check introduced in Hybrid Hyb_1^2). In this case, sk' already rejects c as an invalid ciphertext.

Thus, the claim holds due to the functional equivalence explained above and the security of $i\mathcal{O}$. □

Claim: If iO is a secure iO, then $\mathsf{Hyb}_1^3 \overset{c}{\approx} \mathsf{Hyb}_2$.

Proof. In Hyb_2, instead of using the un-punctured key for F in sk', we puncture F at the points $\alpha_0 \| m^*$ and $\alpha_1 \| m^*$. For sk', the modified program is functionally equivalent to that in the previous hybrid because—by the checks added in the previous hybrid—F will never be evaluated on such inputs. $\square$

Thus, the lemma holds. $\square$

Lemma 1.5.11. *If $\mathcal{F}$ is an injective pPRF family,* Com *is secure injective commitment, and iO is a secure iO, then* $\mathsf{Hyb}_2 \overset{c}{\approx} \mathsf{Hyb}_3$

Proof of Lemma 1.5.11. To prove the lemma, we define auxiliary hybrids.

Hyb_2^1: We alter the generation of the the key sk' in the security game. Instead of using $\hat{\beta} = F(\alpha_1 \| m^*)$, we sample $\hat{\beta}$ uniformly at random from $\{0,1\}^{9\ell}$.

Hyb_2^2: We change Line 1 of Figure 1-5. Value $\hat{z} := \mathsf{Com}(0;\, \hat{\beta})$ is hard-coded, and we replace the check "$\beta = \hat{\beta}$" with the check "$\mathsf{Com}(0;\, \beta) = \hat{z}$".

Hyb_2^3: We change the hard-coded value $\hat{z}$ into "$\mathsf{Com}(1;\, \hat{\beta})$".

Hyb_2^4: We replace the expression "$\mathsf{Com}(0;\, \beta) = \hat{z}$" with FALSE.

For reference, we describe sk' from Hybrid Hyb_2^4 in Figure 1-11.

Constants: Set $\{c_0, c_1\} \subset \{0,1\}^n$, punctured F', G, and the values α_0, α_1, m^*, $\hat{\gamma}$.
Inputs: $c = (\alpha \| \beta \| \gamma)$, where $\alpha \in \{0,1\}^{2\ell}$, $\beta \in \{0,1\}^{9\ell}$, and $\gamma \in \{0,1\}^{\ell}$.

```
1. For some $i$, if $\alpha = \alpha_1$ and FALSE and $\gamma = \hat{\gamma}$, output $m^*$. (i.e., this never
   happens)
2. If $c \in C$, output $\perp$.
3. Compute $m = G(\beta) \oplus \gamma$.
4. If $(\alpha, m) \in \{(\alpha_0, m^*), (\alpha_1, m^*)\}$, output $\perp$.
5. If $\beta = F'(\alpha \| m)$, output $m$.
6. Else output $\perp$.
```

Figure 1-11: Modified Program of D_2' in Hyb_2^4 (pre-obfuscation)

Claim: If $\mathcal{F}$ is an injective pPRF family, then $\mathsf{Hyb}_2 \overset{c}{\approx} \mathsf{Hyb}_2^1$

Proof. This holds due to the pseudorandomness of F at punctured points. □

Claim: If Com is a secure injective commitment and $i\mathcal{O}$ is a secure iO, then $\mathsf{Hyb}_2^1 \overset{c}{\approx} \mathsf{Hyb}_2^2$.

Proof. The modified decryption programs are functionally equivalent by the injective property of Com. Thus, the holds due to the injective property of Com and the security of $i\mathcal{O}$. □

Claim: If Com is a secure injective commitment, then $\mathsf{Hyb}_2^2 \overset{c}{\approx} \mathsf{Hyb}_2^3$.

Proof. This holds due to the computational hiding property of Com. □

Claim: If Com is a secure injective commitment and $i\mathcal{O}$ is a secure iO, then $\mathsf{Hyb}_2^3 \overset{c}{\approx} \mathsf{Hyb}_2^4$.

Proof. The modified decryption programs are functionally equivalent with high probability because of the perfect binding property of Com (which follows from injectivity). In fact, we remove the entire line 1 as in Hyb_3, which also preserves functionality. Thus, the claim holds due to the functional equivalence explained above and the security of $i\mathcal{O}$, □

Claim: If $\mathcal{F}$ is an injective pPRF family, then $\mathsf{Hyb}_2^4 \overset{c}{\approx} \mathsf{Hyb}_3$.

Proof. This holds due the pseudorandomness of F at the punctured points. □

Thus, the lemma holds. □

Lemma 1.5.12. *If $\mathcal{G}$ is a pPRF family and $i\mathcal{O}$ is a secure iO, then* $\mathsf{Hyb}_3 \overset{c}{\approx} \mathsf{Hyb}_4$.

Proof of Lemma 1.5.12. To prove this lemma, we define auxiliary hybrids.

Hyb_3^1: We alter the generation of pk (see Line 2(d) 1-4). We puncture G in pk at β_0 and β_1.

Hyb_3^2: We alter the generation of sk', changing Line 2 of Figure 1-6. Instead of "If $c \in \{c_0, c_1\}$: output $\perp$", we replace it with "If $\beta \in \{\beta_0, \beta_1\}$: output $\perp$".

Claim: If $\mathcal{G}$ is a pPRF family and $i\mathcal{O}$ is a secure iO, then $\mathsf{Hyb}_3 \overset{\mathsf{c}}{\approx} \mathsf{Hyb}_3^1$.

Proof. The encryption programs in these hybrids are functionally equivalent by the sparsity of F since β_0 and β_1 are now chosen at random, with high probability they are not in the image of F. Thus, the claim holds due the functional equivalence explained above and the security of $i\mathcal{O}$. □

Claim: If $i\mathcal{O}$ is a secure iO, then $\mathsf{Hyb}_3^1 \overset{\mathsf{c}}{\approx} \mathsf{Hyb}_3^2$.

Proof. To see that the modified decryption programs in these hybrids are functionally equivalent, we observe that with high probability, neither of these lines has any effect.

Since with high probability, none of the β_0 and β_1 are in the image of F, if $\beta \in \{\beta_0, \beta_1\}$—which is the case when $c \in \{c_0, c_1\}$—then $\mathsf{sk}'(c) = \perp$ with high probability, even without the extra check.

We do not remove the check because checking if $\beta \in \{\beta_0, \beta_1\}$ will allow us to puncture G on this set in the following hybrid. This holds due the functional equivalence explained above and the security of $i\mathcal{O}$. □

Claim: If $\mathcal{G}$ is a pPRF family and $i\mathcal{O}$ is a secure iO, then $\mathsf{Hyb}_3^2 \overset{\mathsf{c}}{\approx} \mathsf{Hyb}_4$.

Proof. In Hyb_4, we alter the generation of sk'. We puncture G at $\{\beta_0, \beta_1\}$ in sk'. This change is functionally equivalent because of the ostensibly useless checks in the previous hybrid. Thus, the claim holds due the functional equivalence explained above and the security of $i\mathcal{O}$. □

Thus, the lemma holds. □

Lemma 1.5.13. *If $\mathcal{G}$ is a pPRF family, then* $\mathsf{Hyb}_4 \overset{c}{\approx} \mathsf{Hyb}_5$

Proof of Lemma 1.5.13. In Hyb_5, we sample γ_0 uniformly at random from $\{0,1\}^\ell$. This change is indistinguishable by the pseudorandomness of G at the punctured set. □

Lemma 1.5.14. *Under the same assumptions as in Theorem 1.5.8,* $\mathsf{Hyb}_5 \overset{c}{\approx} \mathsf{RAND}$

Proof of Lemma 1.5.14. This is proved in the same way as Lemma 1.5.9, 1.5.10, 1.5.11, 1.5.12, and 1.5.13. □

Therefore, the construction satisfies the ciphertext pseudorandomness. □

Therefore, we complete the proof of Theorem 1.5.5.

1.6 Watermarking PRFs

In this section, we construct schemes for watermarking any puncturable PRF family. One is secure against lunch-time attacks and the other is fully secure. Both of them are in the public-key extraction setting. As we explain in Section 1.2.3, the simple scheme is not secure in these settings (the attacker has access to the marking or extraction oracles).

For all of the schemes, let $\mathcal{C}$ be some puncturable PRF (pPRF) family where, for $C \leftarrow \mathcal{C}_\lambda$ we have $C(\cdot) : \mathsf{D}_\lambda \to \mathsf{R}_\lambda$ with $\mathsf{D}_\lambda = \{0,1\}^{n(\lambda)}$, and $\mathsf{R}_\lambda = \{0,1\}^{m(\lambda)}$ for some $n(\lambda), m(\lambda) = \Omega(\lambda)$. We often drop λ from D_λ and R_λ. We construct a watermarking scheme for *PRF evaluation* of $\mathcal{C}$. We identify the PRF evaluation circuits computing the function $C(\cdot)$ and assume (without loss of generality) that the marking procedure just takes C as an input.

Theorem 1.6.1. *Assuming the existence of injective one-way functions, and an indistinguishability obfuscator for all circuits, for all $\varepsilon(\lambda) = \frac{1}{2} + 1/\operatorname{poly}(\lambda)$, all message spaces $\mathcal{M} = \{0,1\}^w$ (for $w = \operatorname{poly}(\lambda)$), all integer functions $n(\lambda) = \Omega(\lambda)$ and*

$m(\lambda) = \Omega(\lambda)$ *there exists a watermarking scheme with message space* $\mathcal{M}$ *which is* ε*-secure against lunch-time attacks for every pPRF ensemble* $\{\mathcal{C}_\lambda\}_{\lambda\in\mathbb{N}}$ *such that functions* C *in* $\mathcal{C}_\lambda$ *map* $\{0,1\}^{n(\lambda)} \to \{0,1\}^{m(\lambda)}$.

If we assume pPRF family $\mathcal{C} = \{C_\lambda\}_{\lambda\in\mathbb{N}}$ satisfies a "nice" property, that is, the injective property in Definition 1.7.1, and the mark oracle accepts only pPRF keys as input, then we can show the full security in Definition 1.4.3 where the adversary has access to the mark oracle even after the challenge program is given. See Section 1.7.1 for the details.

1.6.1 Scheme Outline

Assume we want to mark a PRF family $\mathcal{C}$ with domain $\mathsf{D} = \{0,1\}^n$ and range $\mathsf{R} = \{0,1\}^m$, where both n and m are sufficiently large. In this overview, suppose for simplicity that the space of marks is $\{0,1\}^m$. Our construction relies on a puncturable encryption scheme PE with ciphertext space $\mathcal{C} = \{0,1\}^n$ and message space $\mathcal{M} = \{0,1\}^\ell$ for sufficiently large ℓ. We follow the watermarking framework described in the introduction, in which a marked program is changed on a small set of "marked points", determined by a set of "find points" which are *not* changed.

Roughly speaking, a marked point in our scheme is a valid ciphertext of PE. A valid ciphertext when marking a program C is defined as any encryption of any plaintext $a\|b\|c$ such that $b = H(C(\mathsf{PRG}(a)))$, where H is a UOWHF. On such inputs, the marked program's output is changed to $\mathsf{G}'(c) \oplus \mathsf{m}$ where G' is a publicly known pseudorandom generator and m is the desired mark. Note that there are super-polynomially many marked-points, but yet they are only a negligible fraction of the total domain.

Given the above marking scheme, there is a natural procedure to extract the mark m. We first pick random values $a, c \leftarrow \{0,1\}^{\ell/3}$ and compute the corresponding find-point $\alpha := \mathsf{PRG}(a)$. Then we compute $b := H(C'(\alpha))$ and use this to find the corresponding marked-point $x \leftarrow \mathsf{PE.Enc}(\mathsf{pk}, a\|b\|c)$. Finally, we compute $y = C'(x)$ and record $\mathsf{m}' := y \oplus \mathsf{G}'(c)$ as a candidate for the embedded message. If $C' = \mathsf{Mark}(C)$, correctness is obvious. The bulk of our work is making extraction work for arbitrary efficiently computable $C' \approx_\varepsilon \mathsf{Mark}(C)$.

In order to guarantee that the correct message is extracted with high probability, we amplify our procedure in two steps. First, we fix $a\|b$ and sample multiple independent c's, extract as above, and take the majority result. We then repeat this process with independently sampled a's, and again taking the majority result. Compared to earlier versions of this work [CHV15, NW15], this "majority of majorities" approach allows us to attain optimal thresholds for unremovability (any $\frac{1}{2} + \frac{1}{\mathrm{poly}(\lambda)}$).

1.6.2 A Message-Embedding Construction

In this section, we formally construct our main message-embedding watermarking scheme. We show it satisfies unremovability in the public-key extraction setting and in the presence of a marking oracle. We obtain a scheme in which unremovability holds for any approximation factor $\varepsilon(\lambda) = \frac{1}{2} + 1/\mathrm{poly}(\lambda)$.

Setup. Our goal is to construct a watermarking scheme for a pPRF family $\mathcal{C}$ with domain $\{0,1\}^n$ and range $\{0,1\}^m$. For any positive integer w, let $\mathcal{M} = \{0,1\}^{w \cdot m}$ denote the message space. We will think of messages $\mathsf{m} \in \mathcal{M}$ as consisting of w/m chunks in $\{0,1\}^m$, so we will write $\mathsf{m} = \mathsf{m}_1\|\cdots\|\mathsf{m}_w$. Let PE be a puncturable encryption scheme with ciphertext length n and plaintext length $\ell + \log w$. Let $\mathsf{G} : \{0,1\}^{\ell/3} \to \{0,1\}^n$ and $\mathsf{G}' : \{0,1\}^{\ell/3} \to \{0,1\}^m$ be PRGs, and let $H : \{0,1\}^m \to \{0,1\}^{\ell/3}$ be a UOWHF.

Construction. For any approximation factor $\varepsilon(\lambda) = \frac{1}{2} + \rho(\lambda)$ where $\rho(\lambda)$ is some inverse polynomial, we set $Q = Q(\lambda) = \lambda/\rho(\lambda)^2$ and $R = R(\lambda) = \lambda/\rho(\lambda)^2$ and define our construction as follows.

$\mathsf{Gen}(1^\lambda)$: Sample a key pair $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{PE.Gen}(1^\lambda)$. Output $(\mathsf{xk}, \mathsf{mk})$ where $\mathsf{xk} = \mathsf{pk}$ and $\mathsf{mk} = \mathsf{sk}$.

$\mathsf{Mark}(\mathsf{mk}, C, \mathsf{m})$: Outputs the iO-obfuscation of circuit M constructed from C in Fig. 1-12, i.e., $i\mathcal{O}(M)$.

```
Constants: PE decryption key $\mathsf{sk}$, pPRF $F$, circuit $C$, and message $\mathsf{m} = \mathsf{m}_1 \| \dots \| \mathsf{m}_w$
Inputs: $x \in \{0,1\}^n$
  1. Try to parse $a\|b\|c\|i \leftarrow \mathsf{PE.Dec}(\mathsf{sk}, x)$, where $|a| = |b| = |c| = \ell/3$ and $i \in [w]$.
  2. If $a\|b\|c\|i \neq \perp$ and $H(C(\mathsf{G}(a))) = b$, output $\mathsf{G}'(c) \oplus \mathsf{m}_i$.
  3. Otherwise, output $C(x)$.
```

Figure 1-12: The program M, which is a modification of C (pre-obfuscated program)

$\mathsf{Extract}(\mathsf{xk}, C')$: For each $i \in [w]$, let $\mathsf{m}_i = \mathsf{Extract}_i(\mathsf{xk}, C')$, where $\mathsf{Extract}_i$ is defined in Fig. 1-13. $\mathsf{Extract}_i$ makes use of a subroutine $\mathsf{WeakExtract}_i$, which is defined in Fig. 1-14. Output $\mathsf{m}_1 \| \dots \| \mathsf{m}_w$.

```
$\mathsf{Extract}_i(\mathsf{xk}, C')$:
  1. For $j = 1, \dots, Q$,
     (a) Sample uniformly random $a_j \leftarrow \{0,1\}^{\ell/3}$.
     (b) Compute $b_j = H(C'(\mathsf{G}(a_j)))$
     (c) Run $\mathsf{m}_i^{(j)} \leftarrow \mathsf{WeakExtract}_i(\mathsf{xk}, C', a_j, b_j)$
  2. If there exists a "majority-of-majorities message" $\mathsf{m}_i \neq \perp$ such that $|\{j : \mathsf{m}_i^{(j)} = \mathsf{m}_i\}| > Q/2$, then output $\mathsf{m}_i$; else output unmarked.
```

Figure 1-13: The sub-routine algorithm $\mathsf{Extract}_i(\mathsf{xk}, C')$

```
$\mathsf{WeakExtract}_i(\mathsf{xk}, C', a, b)$:
  1. For $k = 1, \dots, R$,
     (a) Sample $c_k \leftarrow \{0,1\}^{\ell/3}$ and $x_k \leftarrow \mathsf{PE.Enc}(\mathsf{pk}, a\|b\|c_k\|i)$.
     (b) Compute $\mathsf{m}_i^{(k)} = \mathsf{G}'(c_k) \oplus C'(x_k)$.
  2. Define the "majority message" $\mathsf{m}_i$ such that $|\{k : \mathsf{m}_i^{(k)} = \mathsf{m}_i\}| > R/2$ if such a $\mathsf{m}_i$ exists; otherwise, define $\mathsf{m}_i = \perp$.
```

Figure 1-14: The sub-routine algorithm $\mathsf{WeakExtract}_i(\mathsf{xk}, C', a, b)$

It is easy to check that this construction satisfies statistical and extraction correctness, and meaningfulness.

Proposition 1.6.2. *The above construction satisfies Theorem 1.6.1.*

1.6.3 Security Proofs

To prove the proposition, we must prove ε-unremovability against lunch-time attacks.

Overview. Recall that in our scheme, there are two sparse sets of points: "find points", which are unchanged between a marked and unmarked program, and "mark points", which are changed. To extract from a circuit C', one repeatedly performs the following 4 steps, which we will refer to as *weak extraction*:

1. Sample a find point x, and queries $C(x)$
2. Use the resulting value to sample many mark points $x_1, \ldots, x_k$, where $k = \lambda/\rho^2$
3. For each x_i, query $C(x_i)$ to compute a guess m_i
4. If some m_i occurs more than $k/2$ times, return it. Otherwise, return $\perp$.

If this procedure returns some message m many times (more than half), then m is the extracted value.

Weak extraction can fail if $C(x)$ has been changed by the remover, or if most of $C(x_1), \ldots, C(x_k)$ have been changed. The first happens with probability at most $1 - \varepsilon$, by the pseudorandomness of find points. The second happens with negligible probability by a Chernoff bound. By repeating this process with many find points, the error probability is reduced to negligible.

Proof of ε-unremovability. First, we define two security experiments to state a useful lemma that is used to prove Proposition 1.6.2. These two experiments are similar to the unremovability security game, but the goal of the adversary is now to distinguish a mark-point of a marked program from a uniformly random string of the same length, while first given access to a marking oracle and also given the corresponding find point.

For any PPT adversary $\mathcal{D}$, we define the following two experiments, $\mathsf{Exp}^{\mathcal{D}}_{\mathsf{REAL}}(\lambda, i)$ and $\mathsf{Exp}^{\mathcal{D}}_{\mathsf{RAND}}(\lambda)$.

$\mathsf{Exp}^{\mathcal{D}}_{\mathsf{REAL}}(\lambda, i)$:

1. $(\mathsf{xk}, \mathsf{mk}) \leftarrow \mathsf{Gen}(1^\lambda)$
2. $(s, \mathsf{m}) \leftarrow \mathcal{D}^{\mathsf{Mark}(\mathsf{mk},\cdot,\cdot)}(\mathsf{xk})$
3. $C \leftarrow \mathcal{C}$ and $\widetilde{C} \leftarrow \mathsf{Mark}(\mathsf{mk}, C, \mathsf{m})$
4. $a \leftarrow \{0,1\}^{\ell/3}$, $b = H(\widetilde{C}(\mathsf{G}(a)))$
5. $c \leftarrow \{0,1\}^{\ell/3}$
6. $x_{\mathsf{REAL}} \leftarrow \mathsf{PE.Enc}(\mathsf{pk}, a\|b\|c\|i)$
7. Finally, output $\mathcal{D}(s, \widetilde{C}, a, x_{\mathsf{REAL}})$

$\mathsf{Exp}^{\mathcal{D}}_{\mathsf{RAND}}(\lambda)$:

1. $(\mathsf{xk}, \mathsf{mk}) \leftarrow \mathsf{Gen}(1^\lambda)$
2. $(s, \mathsf{m}) \leftarrow \mathcal{D}^{\mathsf{Mark}(\mathsf{mk},\cdot,\cdot)}(\mathsf{xk})$
3. $C \leftarrow \mathcal{C}$ and $\widetilde{C} \leftarrow \mathsf{Mark}(\mathsf{mk}, C, \mathsf{m})$
4. $a \leftarrow \{0,1\}^{\ell/3}$
5. $x_{\mathsf{RAND}} \leftarrow \{0,1\}^n$
6. Finally, output $\mathcal{D}(s, \widetilde{C}, a, x_{\mathsf{RAND}})$

Lemma 1.6.3. *Under the same conditions as in Theorem 1.6.1, for all PPT distinguishers $\mathcal{D}$ and for all $i \in [w]$, it holds that*

$$\left|\Pr[\mathsf{Exp}^{\mathcal{D}}_{\mathsf{REAL}}(\lambda, i) = 1] - \Pr[\mathsf{Exp}^{\mathcal{D}}_{\mathsf{RAND}}(\lambda) = 1]\right| < \mathsf{negl}(\lambda)$$

We also define a "many-message" version of these two experiments:

$\mathsf{Exp}^{\mathcal{D}}_{\mathsf{REAL}^R}(\lambda, i)$:

1. $(\mathsf{xk}, \mathsf{mk}) \leftarrow \mathsf{Gen}(1^\lambda)$
2. $(s, \mathsf{m}) \leftarrow \mathcal{D}^{\mathsf{Mark}(\mathsf{mk},\cdot,\cdot)}(\mathsf{xk})$
3. $C \leftarrow \mathcal{C}$ and $\widetilde{C} \leftarrow \mathsf{Mark}(\mathsf{mk}, C, \mathsf{m})$

4. $a \leftarrow \{0,1\}^{\ell/3}$, $b = H(\widetilde{C}(\mathsf{G}(a)))$.
5. $c \leftarrow \{0,1\}^{\ell/3}$
6. For $j = 1, \ldots, R$:
   sample $x_{\mathsf{REAL},j} \leftarrow \mathsf{PE.Enc}(\mathsf{pk}, a\|b\|c\|i)$
7. Finally, output $\mathcal{D}(s, \widetilde{C}, a, \boldsymbol{x}_{\mathsf{REAL}})$, where $\boldsymbol{x}_{\mathsf{REAL}} = (x_{\mathsf{REAL},1}, \ldots, x_{\mathsf{REAL},R})$.

$\mathsf{Exp}^{\mathcal{D}}_{\mathsf{RAND}^R}(\lambda)$:

1. $(\mathsf{xk}, \mathsf{mk}) \leftarrow \mathsf{Gen}(1^\lambda)$
2. $(s, \mathsf{m}) \leftarrow \mathcal{D}^{\mathsf{Mark}(\mathsf{mk},\cdot,\cdot)}(\mathsf{xk})$
3. $C \leftarrow \mathcal{C}$ and $\widetilde{C} \leftarrow \mathsf{Mark}(\mathsf{mk}, C, \mathsf{m})$
4. $a \leftarrow \{0,1\}^{\ell/3}$
5. For $j = 1, \ldots, R$:
   sample $x_{\mathsf{RAND},j} \leftarrow \{0,1\}^n$
6. Finally, output $\mathcal{D}(s, \widetilde{C}, a, \boldsymbol{x}_{\mathsf{RAND}})$, where $\boldsymbol{x}_{\mathsf{RAND}} = (x_{\mathsf{RAND},1}, \ldots, x_{\mathsf{RAND},R})$.

Corollary 1.6.4. *For all PPT $\mathcal{D}$ and for all $i \in [w]$, it holds that*

$$\left|\Pr[\mathsf{Exp}^{\mathcal{D}}_{\mathsf{REAL}^R}(\lambda, i) = 1] - \Pr[\mathsf{Exp}^{\mathcal{D}}_{\mathsf{RAND}^R}(\lambda) = 1]\right| < \mathsf{negl}(\lambda)$$

Proof. This follows from a simple hybrid argument. □

Before proving Lemma 1.6.3, we first show that it would imply Proposition 1.6.2.

Proof of Proposition 1.6.2. We show that for every i and every PPT adversary $(\mathcal{A}_1, \mathcal{A}_2)$,

$$\Pr\left[\mathsf{Extract}_i(\mathsf{xk}, C^*) \neq \mathsf{m}^{(i)} \;\wedge\; C^* \cong_\varepsilon \widetilde{C} \;\middle|\; \begin{array}{l} (\mathsf{xk}, \mathsf{mk}) \leftarrow \mathsf{Gen}(1^\lambda) \\ (\mathsf{m}, s) \leftarrow \mathcal{A}_1^{\mathsf{Mark}(\mathsf{mk},\cdot,\cdot)}(1^\lambda, \mathsf{xk}, \mathsf{mk}) \\ C \leftarrow \mathcal{C} \\ \widetilde{C} \leftarrow \mathsf{Mark}(\mathsf{mk}, C, \mathsf{m}) \\ C^* \leftarrow \mathcal{A}_2(s, \widetilde{C}) \end{array}\right] \leq \mathsf{negl}(\lambda)$$

Suppose for the sake of contradiction that a PPT adversary $(\mathcal{A}_1, \mathcal{A}_2)$ wins this game with non-negligible probability. That is, with non-negligible probability, $\mathcal{A}_2$ outputs a program $C^* \cong_\varepsilon \widetilde{C}$ such that $\mathsf{Extract}_i(C^*) \neq \mathsf{m}^{(i)}$ with non-negligible probability. For convenience of notation, let Δ denote the point-wise xor of $\widetilde{C}$ and C^*. That is, let $\Delta(x) = C^*(x) \oplus \widetilde{C}(x)$. Recall that $\varepsilon(\lambda) = \frac{1}{2} + \rho(\lambda)$. Because $\mathsf{Extract}_i$ takes the majority answer after running $\mathsf{WeakExtract}_i$ many $(\lambda/\rho(\lambda)^2)$ times, it must be (by a Chernoff bound) that for any such C^*,

$$p_{C^*} := \Pr\left[\mathsf{WeakExtract}_i(C^*) \neq \mathsf{m}^{(i)}\right] \geq \frac{1}{2} - \rho(\lambda) + \frac{1}{\mathrm{poly}(\lambda)}$$

for some polynomial poly. Since $\mathsf{WeakExtract}_i$ only accesses C^* in a black-box way, and since we know that $\mathsf{WeakExtract}_i(\widetilde{C}) = \mathsf{m}^{(i)}$ with high probability, it must be the case that C^* differs from $\widetilde{C}$ at some of the points queried by $\mathsf{WeakExtract}_i$. Furthermore $\mathsf{WeakExtract}_i$ is robust against differences at mark points (since it suffices for C^* to agree with $\widetilde{C}$ at a majority of the queried mark points). Thus we have (by a union bound) that

$$p_{C^*} \leq \Pr_a[\Delta(\mathsf{G}(a)) \neq 0] + \Pr_{\substack{a \leftarrow \{0,1\}^{\ell/2} \\ x_k \leftarrow \mathsf{PE.Enc}(a\|b\|i)}}\left[|\{k : \Delta(x_k) \neq 0 \ \wedge\ k \in [R]\}| > \frac{R}{2}\right] + \mathsf{negl}(\lambda)$$

The first term corresponds to the probability of $\mathcal{A}$ changing the find point queried by $\mathsf{WeakExtract}_i$, and the second corresponds to the probability of $\mathcal{A}$ changing many mark points. The third term is the probability that $\mathsf{WeakExtract}_i(\widetilde{C}) \neq \mathsf{m}_i$.

For the first term, we note that by the pseudorandomness of $\mathsf{G}(a)$, it must hold that for all polynomials poly, there is a negligible negl such that

$$\Pr\left[C^* \cong_\varepsilon \widetilde{C} \ \wedge\ \Pr_a[\Delta(\mathsf{G}(a)) \neq 0] \geq 1 - \varepsilon(\lambda) + \frac{1}{\mathrm{poly}(\lambda)}\right] \leq \mathsf{negl}(\lambda).$$

Indeed, otherwise we can break the security of G by running $\mathcal{A}$, and empirically testing whether the Δ output by $\mathcal{A}_2$ exhibits a $\frac{1}{\mathrm{poly}(\lambda)}$ advantage in distinguishing $\mathsf{G}(a)$ points from random points. If it does, we evaluate Δ on our challenge to try to distinguish;

otherwise we guess randomly.

For the other term, Corollary 1.6.4 states that the x_i's are jointly indistinguishable from i.i.d. random x_i's sampled from $\{0,1\}^m$, even though $\mathcal{A}_1$ has oracle access to $\mathsf{Mark}(\mathsf{mk},\cdot,\cdot)$. Combined with a Chernoff bound, which states that

$$\Pr\left[C^* \cong_\varepsilon \widetilde{C} \ \wedge \Pr_{x_1,\ldots,x_R \leftarrow \{0,1\}^n}\left[|\{x_k : \Delta(x_k) \neq 0\}| > \frac{R}{2}\right] \geq \frac{1}{\mathrm{poly}(\lambda)}\right] = 0,$$

this implies that for every polynomial poly,

$$\Pr\left[C^* \cong_\varepsilon \widetilde{C} \ \wedge \Pr_{\substack{a \leftarrow \{0,1\}^{\ell/2} \\ x_1,\ldots,x_R \leftarrow \mathsf{PE.Enc}(a\|b\|i)}}\left[|\{x_k : \Delta(x_k) \neq 0\}| > \frac{R}{2}\right] \geq \frac{1}{\mathrm{poly}(\lambda)}\right] \leq \mathsf{negl}(\lambda).$$

Combining these four inequalities yields a contradiction.

□

Now we turn to proving Lemma 1.6.3.

Proof of Lemma 1.6.3. We define a sequence of hybrid experiments to prove this lemma. We call all variables that $\mathcal{D}$ sees in the experiment Exp a view of $\mathcal{D}$ and denote it by $\mathsf{view}(\mathsf{Exp})$.

Hyb_0: This experiment is exactly the same as $\mathsf{Exp}^{\mathcal{D}}_{\mathsf{REAL}}(\lambda, i)$.

Hyb_1: In this hybrid experiment, we change the marking oracle. For the adversary's queries $(C^{(1)}, \mathsf{m}^{(1)}), \ldots, (C^{(q)}, \mathsf{m}^{(q)})$, instead of generating marked program $\widetilde{C}^{(\iota)} \leftarrow i\mathcal{O}(M^{(\iota)})$, we set $\widetilde{C}^{(\iota)} \leftarrow i\mathcal{O}(M^{(\iota)}\{x_0, x_1\})$ where $M^{(\iota)}\{x_0, x_1\}$ is defined in Figure 1-15, having hard-coded $C^{(\iota)}$, $\mathsf{sk}' \leftarrow \mathsf{PE.Puncture}(\mathsf{sk}, x_0, x_1)$, $x_0 := x_{\mathsf{REAL}}$, $x_1 \leftarrow \{0,1\}^n$, and $\mathsf{m}^{(\iota)}$.

Hyb_2: In this hybrid experiment, we change the marked challenge program $\widetilde{C}$. We use the punctured decryption key sk' and hard-code the output values corresponding to x_0 and x_1 as $y_0 = \mathsf{G}'(c)$ and $y_1 \leftarrow \{0,1\}^m$ respectively. That is, we set $\widetilde{C} \leftarrow i\mathcal{O}(M\{x_0, x_1\})$ where $M\{x_0, x_1\}$ is defined in Figure 1-16.

```
Constants: $\underline{\text{punctured PE decryption key } \mathsf{sk}' := \mathsf{sk}\{x_0, x_1\}}$, pPRF key $C^{(\iota)}$,
    $\underline{\text{values } x_0, x_1}$, message $\mathsf{m}^{(\iota)} = \mathsf{m}_1^{(\iota)} \| \cdots \| \mathsf{m}_w^{(\iota)}$
Inputs: $x \in \{0,1\}^n$
    1. $\underline{\text{If } x \in \{x_0, x_1\}, \text{ then output } C^{(\iota)}(x).}$
    2. Compute $\underline{a\|b\|c\|i \leftarrow \mathsf{PE.Dec}(\mathsf{sk}', x)}$, where $|a| = |b| = |c| = \ell/3$, and $i \in [w]$.
    3. If $a\|b\|c\|i \neq \perp$ and $H(C^{(\iota)}(\mathsf{G}(a))) = b$, output $\mathsf{G}'(c) \oplus \mathsf{m}_i^{\iota}$.
    4. Otherwise, output $C^{(\iota)}(x)$.
```

Figure 1-15: Program $M^{(\iota)}\{x_0, x_1\}$ in Hyb_1

```
Constants: $\underline{\text{punctured PE decryption key } \mathsf{sk}' := \mathsf{sk}\{x_0, x_1\}}$, pPRF key $F$, pPRF key
    $C$, $\underline{\text{values } x_0, x_1, y_0, y_1}$, message $\mathsf{m} = \mathsf{m}_1 \| \cdots \| \mathsf{m}_w$
Inputs: $x \in \{0,1\}^n$
    1. $\underline{\text{If } x = x_\sigma \text{ for } \sigma \in \{0,1\}, \text{ then output } y_\sigma.}$
    2. Compute $\underline{a\|b\|c\|i \leftarrow \mathsf{PE.Dec}(\mathsf{sk}', x)}$, where $|a| = |b| = |c| = \ell/3$ and $i \in [w]$.
    3. If $a\|b\|c\|i \neq \perp$ and $H(C(\mathsf{G}(a))) = b$, output $\mathsf{G}'(c) \oplus \mathsf{m}_i$.
    4. Otherwise, output $C(x)$.
```

Figure 1-16: Program $M\{x_0, x_1\}$ in Hyb_2

Hyb_3 In this experiment, x_0 is changed to be uniformly sampled from $\{0,1\}^n$.

Hyb_4 In this experiment, y_0 is changed to be uniformly sampled from $\{0,1\}^m$

$\mathsf{Exp}^{\mathcal{D}}_{\mathsf{RAND}}$: The only changes from Hyb_3 are that the challenge program $\widetilde{C}$ and marked keys $\widetilde{C}^{(\iota)}$ for all $\iota \in [q]$ are changed back to the original programs but the values x_0 remain random.

We describe an overview of the main hybrid experiments in Table 1.2.

Lemma 1.6.5. *If $\mathcal{F}$ is a pPRF family, H is a UOWHF,* PE *satisfies the punctured correctness and sparseness, and $i\mathcal{O}$ is a secure indistinguishability obfuscator, then* $\mathsf{view}(\mathsf{Hyb}_0) \overset{\mathsf{c}}{\approx} \mathsf{view}(\mathsf{Hyb}_1)$.

Proof of Lemma 1.6.5. To prove the lemma, we define auxiliary hybrid experiments Hyb_0^{ι} for $\iota \in [q]$ where the mark oracle gives $i\mathcal{O}(M^{(\iota)}\{x_0, x_1\})$ for the first ι queries $C^{(1)}, \ldots, C^{(\iota)}$ of $\mathcal{D}$.

Table 1.2: An overview of hybrid experiment

Hybrid	Challenge: $i\mathcal{O}(\cdot)$	Answers of $\mathcal{MO}$: $i\mathcal{O}(\cdot)$	x_0	x_1
$\mathsf{Exp}^{\mathcal{D}}_{\mathsf{REAL}}$	M	$M^{(\iota)}$	x_{REAL}	none
Hyb_1	M	$\underline{M^{(\iota)}\{x_0, x_1\}}$	x_{REAL}	random
Hyb_2	$\underline{M\{x_0, x_1\}}$	$M^{(\iota)}\{x_0, x_1\}$	x_{REAL}	random
Hyb_3	$M\{x_0, x_1\}$	$M^{(\iota)}\{x_0, x_1\}$	$\underline{x_{\mathsf{RAND}}}$	random
$\mathsf{Exp}^{\mathcal{D}}_{\mathsf{RAND}}$	$\underline{M}$	$\underline{M^{(\iota)}}$	x_{RAND}	none

Claim: In Hyb_0^{ι}, the probability that $H(C^{(\iota+1)}(\mathsf{PRG}(a))) = b$ is negligible, where $b := H(\widetilde{C}(\mathsf{PRG}(a)))$.

Proof. If for some PPT $\mathcal{D}$, this event happens with non-negligible probability, we show how to invert H at a random input with nearly the same non-negligible probability, thus contradicting the one-wayness of H.

We use the fact that $C(\mathsf{PRG}(a))$, and therefore $\widetilde{C}(\mathsf{PRG}(a))$, is pseudorandom, because up until this point in the game, the only information $\mathcal{D}$ has about C comes from the marking oracle hard-coding $x_0 = \mathsf{Enc}(a\|b\|c)$ in its answers. So if b is replaced by a random challenge $H(r)$, $C^{(\iota+1)}(\mathsf{PRG}(a))$ must still be a pre-image of b with non-negligible probability.

□

Claim: $\mathsf{view}(\mathsf{Hyb}_0^{\iota}) \overset{c}{\approx} \mathsf{view}(\mathsf{Hyb}_0^{\iota+1})$ for all $\iota \in [q]$.

Proof. The only difference between Hyb_0^{ι} and $\mathsf{Hyb}_0^{\iota+1}$ is the $(\iota+1)$-th answer by the mark oracle. We show that the mark oracle's answers are functionally equivalent in the two games, so indistinguishability follows from the security of $i\mathcal{O}$.

There are only two possible inputs on which $M^{(\iota+1)}$ may differ in Hyb_0^{ι} and $\mathsf{Hyb}_0^{\iota+1}$: namely, x_0 and x_1 due to the punctured correctness at non-punctured points of PE. We show that (with high probability) they respectively mapped to $C^{(\iota+1)}(x_0)$ and $C^{(\iota+1)}(x_1)$ without our changes, just as they do with our changes.

It holds that $\mathsf{PE.Dec}(\mathsf{sk}, x_1) = \perp$ with high probability since x_1 is uniformly random and PE satisfies sparseness. Thus, $M^{(\iota+1)}(x_1) = C^{(\iota+1)}(x_1)$ in Hyb_0^{ι}. This is also true in $\mathsf{Hyb}_0^{\iota+1}$ since $M^{(\iota+1)}\{x_0, x_1\}(x_1)$ goes to the punctured-points branch.

On the other hand, x_0 decrypts as $a\|b\|c\|i$, but by our previous claim, it cannot be the case that $H(C^{(\iota+1)}(\mathsf{PRG}(a))) = b$. Thus, $M^{(\iota+1)}(x_0) = C^{(\iota+1)}(x_0)$ in Hyb_0^{ι}.

□

We completed the proof of the lemma by the two claims. □

Lemma 1.6.6. *If C is a pPRF,* PE *satisfies the punctured correctness, and $i\mathcal{O}$ is a secure indistinguishability obfuscator, then* $\mathsf{view}(\mathsf{Hyb}_1) \overset{\mathsf{c}}{\approx} \mathsf{view}(\mathsf{Hyb}_2)$.

Proof of Lemma 1.6.6. We define auxiliary hybrid experiments as follows.

Hyb_1^1: Instead of choosing challenge program $\widetilde{C} \leftarrow i\mathcal{O}(M)$ where the program M is described in Figure 1-12, we now use punctured keys sk' and $C\{x_1\}$ and set $\widetilde{C} \leftarrow i\mathcal{O}(M_1\{x_0, x_1\})$ where $M_1\{x_0, x_1\}$ is defined in Figure 1-17, $y_0 := \mathsf{G}'(c) \oplus \mathsf{m}_i$ and $y_1 := C(x_1)$.

Constants: punctured PE decryption key $\underline{\mathsf{sk}' := \mathsf{sk}\{x_0, x_1\}}$, punctured pPRF key $\underline{C' = C\{x_1\}}$, $\underline{\text{values } x_0, x_1, y_0, y_1}$, message $\mathsf{m} = \mathsf{m}_1\|\cdots\|\mathsf{m}_w$

Inputs: $x \in \{0,1\}^n$

1. $\underline{\text{If } x = x_\sigma \text{ for } \sigma \in \{0,1\}, \text{ then output } y_\sigma.}$
2. Compute $\underline{a\|b\|c\|i \leftarrow \mathsf{PE.Dec}(\mathsf{sk}', x)}$, where $|a| = |b| = |c| = \ell/3$, and $i \in [w]$.
3. If $a\|b\|c\|i \neq \perp$ and $\underline{H(C'(\mathsf{G}(a))) = b}$, output $\underline{\mathsf{G}'(c) \oplus \mathsf{m}_i}$.
4. Otherwise, output $\underline{C'(x)}$.

Figure 1-17: Program $M_1\{x_0, x_1\}$ in Hyb_1^1

Hyb_1^2 : We choose uniformly random $y_1 \leftarrow \{0,1\}^m$ and hard-code it in the program $M\{x_0, x_1\}$.

Claim: $\mathsf{view}(\mathsf{Hyb}_1) \overset{c}{\approx} \mathsf{view}(\mathsf{Hyb}_1^1)$

Proof. Program $M_1\{x_0, x_1\}$ is functionally equivalent to Program M in Hyb_1, because we just hard-coded the values for y_0 and y_1 which would be output anyways. Also, replacing C by $C\{x_1\}$ does not change functionality because, by line 1, C is never evaluated at x_1.Thus, the claim holds due to the security of $i\mathcal{O}$. □

Claim: $\mathsf{view}(\mathsf{Hyb}_1^1) \overset{c}{\approx} \mathsf{view}(\mathsf{Hyb}_1^2)$

Proof. This follows from the pseudorandomness of $C\{x_1\}$ at x_1. □

Claim: $\mathsf{view}(\mathsf{Hyb}_1^2) \overset{c}{\approx} \mathsf{view}(\mathsf{Hyb}_2)$

Proof. In Hyb_2, C is un-punctured in the challenge program $i\mathcal{O}(M\{x_0, x_1\})$, but $M\{x_0, x_1\}$ is still functionally equivalent to the program in Hyb_1^2 due to line 1. Therefore, the claim holds due to the security of $i\mathcal{O}$. □

The proof of the lemma follows from these three claims. □

Lemma 1.6.7. *If* PE *satisfies ciphertext randomness, then* $\mathsf{view}(\mathsf{Hyb}_2) \overset{c}{\approx} \mathsf{view}(\mathsf{Hyb}_3)$.

Proof of Lemma 1.6.7. This reduces to the ciphertext randomness property of PE. If some PPT distinguisher $\mathcal{D}$ distinguishes Hyb_2 from Hyb_3, we construct a PPT $\mathcal{A}$ with non-negligible advantage in the ciphertext pseudorandomness game.

First, $\mathcal{A}$ chooses $a \leftarrow \{0,1\}^{\ell/3}$, $c \leftarrow \{0,1\}^{\ell/3}$, $C \leftarrow \mathcal{C}_\lambda$, and a UOWHF H, computes $b := H(C(\mathsf{PRG}(a)))$, and sends $m_0 := a \| b \| c \| i$ and uniformly random $m_1 \leftarrow \{0,1\}^{\ell+|w|}$ as a challenge. Then, the challenger of PE returns $(c_\sigma, c_{1-\sigma}, \mathsf{pk}, \mathsf{sk}')$ where $\sigma \in \{0,1\}$, $c_0 \leftarrow \mathsf{PE.Enc}(\mathsf{pk}, m_0)$, $c_1 \leftarrow \{0,1\}^n$, and $\mathsf{sk}' = \mathsf{PE.Puncture}(\mathsf{sk}, c_0, c_1)$.

Now, $\mathcal{A}$ can perfectly simulate Hyb_2 and Hyb_3 to $\mathcal{D}$, using c_σ as x_0. If $\sigma = 0$, then $\mathcal{A}$ perfectly simulates Hyb_2. If $\sigma = 1$, then $\mathcal{A}$ perfectly simulates Hyb_3. Thus, $\mathcal{A}$ can break the ciphertext pseudo-randomness by outputting whatever $\mathcal{D}$ outputs. □

Lemma 1.6.8. *If* PE *satisfies ciphertext randomness, then* $\mathsf{view}(\mathsf{Hyb}_3) \overset{c}{\approx} \mathsf{view}(\mathsf{Hyb}_4)$.

Proof. In Hyb_4, we change y_0 from $\mathsf{G}(a)$ to a truly random point. The indistinguishability of this change follows from the PRG security of G, since the adversary receives no other information about a. □

Lemma 1.6.9. *Under the assumptions of Thm 1.6.1,* $\mathsf{view}(\mathsf{Hyb}_4) \overset{c}{\approx} \mathsf{view}(\mathsf{Exp}^{\mathcal{D}}_{\mathsf{RAND}})$.

Proof of Lemma 1.6.9. This proof mirrors the proof of Lemma 1.6.5 and 1.6.6 (in reverse manner). □

Finally, Lemma 1.6.3 follows from Lemma 1.6.5, 1.6.6, 1.6.7, 1.6.8, and 1.6.9. □

1.7 Extensions and Variants of Watermarking

1.7.1 Stronger Unremovability in a Different Model

In this section, we show that if pPRF family $\mathcal{C}$ satisifies a special injective property, then the watermarking scheme for $\mathcal{C}$ in the previous section satisfies the strongest security (Definition 1.4.3).

There is only one part of the above security proof which does not transfer to a "CCA2" version of the unremovability game. This is the claim in the proof of Lemma 1.6.5, which states that the adversary cannot query the marking oracle on a program $C^{(\iota)}$ such that $H \circ C^{(\iota)}$ agrees with the $H \circ \widetilde{C}$ on a given point $\mathsf{PRG}(a)$, where $\widetilde{C}$ is the marked challenge program, H is a UOWHF, and a is a random string.

This clearly does not hold for queries made after seeing $\widetilde{C}$. Indeed, $\mathcal{D}$ could then query $\widetilde{C}$ itself. We show that if:

- The inputs to the mark oracle are pPRF keys instead of arbitrary circuits and
- The pPRF family satisfies a strong "key injectivity" property

then unremovability still holds.

In order to achieve the strongest notion of watermarking unremovability, we need to restrict ourselves to marking a pPRF family that satisfies the following key-injectivity condition. We further change the syntax of Mark, so that its input is no longer an arbitrary circuit, but is actually restricted to functions in the family $\mathcal{C}$.

Definition 1.7.1 (Key-Injective pPRFs).

$$\Pr_{F \leftarrow \mathcal{F}_\lambda}[\exists \alpha, F' \ s.\ t.\ F' \neq F \ \wedge \ F(\alpha) = F'(\alpha)] \leq \mathsf{negl}(\lambda)$$

In other words this says that with high probability over the choice of F, no other $F' \in \mathcal{F}$ agrees with F *anywhere.* See below for concrete instantiations. If we assume $\mathcal{C}$ satisfies the injective property in Definition 1.7.1, then there are only negligible fraction of inputs causes the collision $\widetilde{C}(\alpha) = C^{(\iota+1)}(\alpha)$, that is, Lemma 1.6.5 still holds.

Corollary 1.7.2. *Assuming the existence of injective one-way functions, and an indistinguishability obfuscator for all circuits, for all $\varepsilon(\lambda) = \frac{1}{2} + 1/\operatorname{poly}(\lambda)$, all message spaces $\mathcal{M} = \{0,1\}^w$, all integer functions $n(\lambda) = \Omega(\lambda)$ and $m(\lambda) = \Omega(\lambda)$ there exists a watermarking scheme with message space $\mathcal{M}$ which is ε-secure for every key-injective pPRF ensemble $\{\mathcal{C}_\lambda\}_{\lambda \in \mathbb{N}}$ such that functions C in $\mathcal{C}_\lambda$ map $\{0,1\}^{n(\lambda)} \to \{0,1\}^{m(\lambda)}$.*

Proposition 1.7.3. *(Informal) Assuming the DDH assumption or LWE assumption, there exist key-injective families of pPRFs.*

1.7.1.1 Key-Injective pPRF from LWE and DDH

A key-injective puncturable PRF can be constructed with a modification of the GGM pPRF by using an ensemble of left- and right-injective PRGs $\mathsf{PRG}^{(1)}, \ldots, \mathsf{PRG}^{(n)}$. When we say that $\mathsf{PRG}^{(i)}$ is left- and right-injective, we mean that if $\mathsf{PRG}^{(i)}$ is writen as $\mathsf{PRG}_0^{(i)} \| \mathsf{PRG}_1^{(i)}$, then both $\mathsf{PRG}_0^{(i)}$ and $\mathsf{PRG}_1^{(i)}$ are injective.

We also require the $\mathsf{PRG}^{(i)}$'s to have additive stretch. That is, there exists a polynomial p such that for each i, $\mathsf{PRG}^{(i)}$ maps $\{0,1\}^{\lambda+(i-1)\cdot p(\lambda)} \to \{0,1\}^{\lambda+i\cdot p(\lambda)}$. This ensures that, in the GGM construction, the size of the PRF output is bounded by

$n \cdot \text{poly}(\lambda)$. Such PRGs can be constructed from standard assumptions such as DDH or LWE.

Key-injective pPRFs from LWE. For example, using the learning with errors (LWE) assumption, we define $\mathsf{PRG}_{\boldsymbol{A}} : \mathbb{Z}_q^n \to \mathbb{Z}_p^m$ as $\mathsf{PRG}_{\boldsymbol{A}}(\boldsymbol{x}) := \lfloor \boldsymbol{A}^\mathsf{T} \cdot \boldsymbol{x} \rceil_p$ where operator $\lfloor \cdot \rceil_p$ returns the nearest integer (for each coordinate) modulo p. We can set $q := p^2 = 2^{2k}$ for some $k = O(\lambda)$ and $m := 4n + O(\lambda)$. Let $\boldsymbol{A} = \boldsymbol{A}_0 \| \boldsymbol{A}_1$ where $\boldsymbol{A}_0, \boldsymbol{A}_1 \in \mathbb{Z}_q^{n \times m/2}$, then $\mathsf{PRG}_b(x) = \lfloor \boldsymbol{A}_b^\mathsf{T} \boldsymbol{x} \rceil$. In this case, each $\mathsf{PRG}_b(x)$ is injective w.o.p. over the choice of $\boldsymbol{A}$ and it maps $2nk$ bits to $2nk + O(k\lambda)$ bits. See [BPR12] for details about the LWE assumption and proof of security of the above construction.

Key-injective pPRFs from DDH. Alternatively, it may seem that using DDH, we can set $\mathsf{PRG}_{g_1,g_2}(x) = g_1^x, g_2^x$ where g_1, g_2 are generators of some group $\mathbb{G}$ of prime order p. Unfortunately, the outputs cannot be directly used as PRG inputs in the next level of the tree since they are group elements rather than exponents and we do not know how to extract out two uniform values in $\mathbb{Z}_p$ from them. Nevertheless, this approach can be made to work by defining $\mathsf{PRG}_{g_1,g_2,g_3,h_0,h_1}(x) = h_0(g_1^x, g_2^x, g_3^x), h_1(g_1^x, g_2^x, g_3^x)$ where h_0, h_1 are universal hash functions that map $\mathbb{G}^3 \to \mathbb{Z}_{p'}$ for some p' such that $\log(p') = \log(p) + O(\lambda)$ and $\log(p') \le (3/2)\log(p) - \Omega(\lambda)$. This ensures injectivity (we are hashing p balls into p' bins and therefore for any fixed ball there is unlikely to be another ball colliding with it). It also ensures pseudorandomness security by thinking of h_0, h_1 as extractors via the leftover-hash lemma. In the context of the GGM construction we need a hierarchy of DDH groups of order $p_1, p_2, \ldots$ (one for each level) where $\log(p_{i+1}) = \log(p_i) + O(\lambda)$. Therefore the output does not get "too large".

1.7.2 Optimality of $(\frac{1}{2} + \frac{1}{\text{poly}(\lambda)})$-Unremovability

We now show that ε-unremovable message-embedding watermarking is impossible when $\varepsilon \le \frac{1}{2}$. This is because an adversary can obtain two independent uniformly sampled circuits $\tilde{C}_0$ and $\tilde{C}_1$, each marked with different messages (respectively m_0 and

m_1). The adversary then outputs a program C^* such that $C^* \cong_{1/2} \tilde{C}_0$ and $C^* \cong_{1/2} \tilde{C}_1$. Since C^* can be generated in a way which treats $\tilde{C}_0$ and $\tilde{C}_1$ symmetrically, we must have

$$\Pr\left[\mathsf{Extract}(C^*) = \mathsf{m}_0\right] = \Pr\left[\mathsf{Extract}(C^*) = \mathsf{m}_1\right] \leq \frac{1}{2}.$$

This impossibility clearly holds even in a setting where the adversary is extremely limited in e.g. the number and type of oracle queries he may make.

1.7.3 Variants

List decoding. We note that our construction could also be modified to satisfy ε-unremovability for *any* $\varepsilon = 1/\operatorname{poly}(\lambda)$ by relaxing the correctness requirement on Extract, allowing it to output a (small) list of possible messages rather than a single message. For unremovability, we only require that the correct message appear in the list. For example, in our construction, instead of outputting the "majority value" m such that $|\{i : \mathsf{m} = \mathsf{m}_i\}|$ is sufficiently large, we could just output all $O(1/\varepsilon^2)$ values of m_i. By signing the messages with a standard signature scheme, we can in a black-box way ensure that the list of messages output by the detection procedure *only* contain (in addition to the correct message) the messages that were embedded in some watermarked circuit by some previous call to the marking oracle.

Messageless watermarking. In the case of *messageless* watermarking, there is no challenge message. Instead, the message space is the singleton set $\mathcal{M} := \{\mathsf{marked}\}$. As a corollary of list-decodable watermarking scheme, we can achieve messageless watermarking with security against any $\varepsilon > 1/\operatorname{poly}(\lambda)$.

Marking PRFs With single-bit outputs. In our construction, we assumed we were marking a pPRF whose outputs was $\{0,1\}^m$ for $m = \Omega(\lambda)$. This assumption on m was not necessary. Indeed, any pPRF family mapping $\{0,1\}^n \to \{0,1\}$ can equally be construed as a pPRF family mapping $\{0,1\}^{n-\log m} \to \{0,1\}^m$, and can be marked as such. In doing so, we incur a loss in parameters. If the watermarking scheme for

m-bit outputs satisfied $(1-\varepsilon)$-unremovability, the watermarking scheme for single bit outputs will only satisfy $(1-\frac{\varepsilon}{m})$-unremovability.

Unforgeability. The classic Irish folk tale of "Clever Tom and the Leprechaun" [Kei70] tells of a farmer's son who one day captures a Leprechaun. The Leprechaun guides Tom through a field of bolyawn trees to the site of buried treasure. Before Tom goes to fetch a spade, he ties his red garter round the nearest bolyawn and forbids the Leprechaun from removing it. When Tom returns with the spade, *"lo an' behould, not a bolyawn in the field, but had a red garther, the very idintical model o' his own, tied about it."* Though the Leprechaun could not remove the garter, Tom had not forbade him from tying identical garters around the neighboring trees, making it impossible for Tom to discover the gold.

In their treatment of watermarking definitions, Hopper et al. [HMW07] define a notion of unforgeability that is dual to unremovability. Intended to prevent attacks like the Leprechaun's, unforgeability requires that the only marked programs circuits that an adversary can produce are functionally similar to circuits marked by a marking oracle. Whereas unremovability requires that a circuit is marked if it is ε-similar to some honestly-marked circuit, unforgeability requires that a circuit is marked *only if* it is δ-similar to an honestly-marked circuit, for some parameter $\delta < \varepsilon$.

Achieving unforgeability and unremovability simultaneously has proved challenging. Cohen et al. [CHV15] construct a watermarking scheme for puncturable PRFs which achieves weak notions of unforgeability and unremovability in a security model similar to ours: namely, against an adversary with a public extraction key and who can query the Mark oracle with arbitrary circuits as input. Subsequent works [BLW17, KW17, BKS17] have constructed watermarked PRF families that are both unforgeable and unremovable, albeit in much weaker security models (see Section 1.2.7 for futher discussion).

1.8 The Limits of Watermarking

A natural question is whether there are families of functions that for which there does not exist any watermarking scheme. Barak et al. [BGI+01a] observed that general-purpose indistinguishability obfuscation rules out a notion of watermarking that *exactly* preserves functionality, but not watermarking schemes that change functionality on even a negligible fraction of the domain (as in section 1.6). In this section, we demonstrate that some notion of *non-black-box* learnability implies that a family of functions is unwatermarkable. We demonstrate that there exist PRF families that cannot be watermarked (assuming only the existence of one-way functions), and that any family that is learnable with membership queries [KL93] is not watermarkable.

1.8.1 Impossibilities for Statistical Correctness

In this section, we discuss a number of conditions sufficient to prove that a family of circuits cannot even be watermarked—even for a significantly weakened form of unremovability. We modify the unremovability game (Definition 1.4.3): the adversary has no marking oracle, has neither a public extraction key nor an extraction oracle, and is not allowed to choose the message to be embedded in the challenge.We leave the syntax, statistical correctness, extraction correctness, and meaningfulness requirements of the watermarking definition (Definitions 1.4.1 and 1.4.2) are unchanged. In Section 1.8.2, we relax the statistical correctness condition.

Definition 1.8.1 (Weak ε-Unremovability Game). *The game* $\mathsf{Exp}^{\mathsf{nrmv}}_{\mathcal{A}}(\lambda, \varepsilon)$ *is defined as follows.*

1. *The challenger generates* $(\mathsf{xk}, \mathsf{mk}) \leftarrow \mathsf{Gen}(1^\lambda)$

2. *The challenger chooses a message* $\mathsf{m} \in \mathcal{M}_\lambda$ *arbitrarily, samples a circuit* $C \leftarrow \mathcal{C}_\lambda$ *uniformly at random and gives to the adversary* $\widetilde{C} \leftarrow \mathsf{Mark}(\mathsf{mk}, C, \mathsf{m})$.

3. *Finally, the adversary outputs a circuit* C^*. *If it holds* $C^* \cong_\varepsilon \widetilde{C} \wedge \mathsf{Extract}(\mathsf{xk}, C^*) \neq \mathsf{m}$ *then the experiment outputs* 1, *otherwise* 0.

Definition 1.8.2 (ε-Waterproof). *Let $\mathcal{F} = \{\mathcal{F}_\lambda\}_{\lambda \in \mathbb{N}}$ be a circuit ensemble. We say that $\mathcal{F}$ is* ε-waterproof *if there does not exist an weak ε-unremovable watermarking scheme for $\mathcal{F}$.*

Informally, if a function family is *non-black-box* learnable given an approximate circuit implementation (corresponding the the challenge watermarked circuit), then the family is waterproof. More formally, consider a family of circuits $\mathcal{F}_\lambda$ and some parameter $\rho = \rho(\lambda) \in [0,1]$. The learning algorithm will be given an (arbitrary) circuit g that ρ-approximates F, for a uniformly sampled circuit $F \leftarrow \mathcal{F}_\lambda$ from the family. The (randomized) learner will then output some "hypothesis" circuit h. If h is sufficiently close to F, then the learner can be used to reconstruct an unmarked circuit given a watermarking challenge. We conclude that the family $\mathcal{F}$ is waterproof.

We emphasize that we are interested in *non-black-box learning* in which the learning algorithm gets an (approximate) implementation of the function being learned. This is in contrast to the typical computational learning setting.

For the sake of clarity, we now define all the variants of learning we will consider. It may be best to read the definitions individually when required by the discussion that follows.

Definition 1.8.3 (Non-black-box Learnable Families). *Let $\mathcal{F} = \{\mathcal{F}_\lambda\}_{\lambda \in \mathbb{N}}$ be a circuit ensemble where each family $\mathcal{F}_\lambda = \{F\}$. Let $\rho = \rho(\lambda) \in [0,1]$. We say a distribution over circuits $\mathcal{C}_F$* ρ-strongly approximates $F \in \mathcal{F}_\lambda$ *if for all x,*

$$\Pr_{C \leftarrow \mathcal{C}_F}[C(x) \neq F(x)] \leq \rho.$$

Let $\{\mathcal{C}_F\}_{F \in \mathcal{F}_\lambda}$ be any collection of ρ-strongly approximating distributions for the circuits $F \in \mathcal{F}_\lambda$.

ROBUSTLY LEARNABLE:[11] *We say that $\mathcal{F}$ is* ρ-robustly learnable *if there exists an*

[10]The strong-approximation assumption on the distribution of the approximate implementation C arises from the statistical correctness requirement of Definition 1.4.2. Note that statistical correctness guarantees that for $F \in \mathcal{F}_\lambda$, the distribution $(\widetilde{F} \leftarrow \mathsf{Mark}(\mathsf{mk}, F) \ : \ \mathsf{mk} \leftarrow \mathsf{Setup}(1^\lambda))$ strongly-approximates F for some negligible function $\rho(\lambda)$.

efficient algorithm L outputting a circuit h, such that for all large enough $\lambda \in \mathbb{N}$, random $F \leftarrow \mathcal{F}_\lambda$, and random circuit $C \leftarrow \mathcal{C}_F$ (where $\mathcal{C}_F$ ρ-strongly approximates F):

$$\Pr[h \equiv F \mid h \leftarrow L(C, 1^\lambda)] \text{ is non-negligible.}$$

We say that $\mathcal{F}$ is robustly *learnable if it is ρ-robustly learnable for any negligible function $\rho(\lambda)$.*

PROPERLY LEARNABLE:[12] *Additionally, we say that $\mathcal{F}$ is* properly *learnable if for every function $F \in \mathcal{F}_\lambda$, and random $C \leftarrow \mathcal{C}_F$:*

$$\Pr[L(C, 1^\lambda) = F] \text{ is non-negligble.}$$

IMPLEMENTATION INDEPENDENTLY LEARNABLE: *Let $\mathcal{C}_F^1$ and $\mathcal{C}_F^2$ be two distributions that ρ-strongly approximate F. We say that L is* implementation independent *if for all $F \in \mathcal{F}_\lambda$ and for any two distributions $\mathcal{C}_F^1$ and $\mathcal{C}_F^2$ that ρ-strongly approximate F, the distributions $(L(C_1, 1^\lambda) : C_1 \leftarrow \mathcal{C}_F^1)$ and $(L(C_2, 1^\lambda) : C_2 \leftarrow \mathcal{C}_F^2)$ are computationally indistinguishable.*

ε-APPROXIMATELY LEARNABLE: *A weaker condition than the above, we say that $\mathcal{F}$ is* ε-approximately *learnable if instead, for all F and for random $C \leftarrow \mathcal{C}_F$:*

$$\Pr[h \cong_\varepsilon F \mid h \leftarrow L(C, 1^\lambda)] \text{ is non-negligible.}$$

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

[11]This is somewhat analogous to the notion of error-tolerance in computational learning [KL93], but in the non-black-box setting.

[12]This is stronger than simply requiring that $h \in \mathcal{F}_\lambda$. In particular, it implies that for every $F \in \mathcal{F}_\lambda$, there are only polynomially-many $F' \in \mathcal{F}_\lambda$ such that $F' \cong_{\rho/2} F$.

Proposition 1.8.4. *If $\mathcal{F}$ is robustly, properly learnable, then $\mathcal{F}$ is ε-waterproof for every $\varepsilon \in [0,1]$.*

Proof. Given a watermarking scheme for $\mathcal{F}$, let $\mathcal{C}_F = \{\mathsf{Mark}(\mathsf{mk}, F) : (\mathsf{xk}, \mathsf{mk}) \leftarrow \mathsf{Gen}(1^\lambda)\}$. There exists some negligible function $\rho(\lambda)$ such that $\mathcal{C}_F$ ρ-strongly approximates F for all circuits $F \in \mathcal{F}$, by the statistical correctness property. Suppose $\mathcal{F}$ is ρ-robustly, properly learnable with learning algorithm L. Given a challenge marked program $\widetilde{F} \leftarrow \mathsf{Mark}(\mathsf{mk}, F)$, evaluate $h \leftarrow L(\widetilde{F}, 1^\lambda)$. With noticeable probability, $h = F$. If $\mathsf{Extract}(\mathsf{xk}, F) = \mathsf{unmarked}$ with any noticeable probability, unremovability is violated. On the other hand, if $\mathsf{Extract}(\mathsf{xk}, F) \neq \mathsf{unmarked}$ with any noticeable probability, then meaningfulness is violated. □

Surprisingly, this proposition is also enough to construct a PRF family that is waterproof.

Proposition 1.8.5 ([BGI+12]). *Assuming one-way functions exist, there exists a pseudorandom function family $\mathcal{F}$ that is robustly, properly (non-black-box) learnable.*

Proof. In [BGI+12], the authors extend the impossibility of virtual-black box obfuscation to a notion of approximate obfuscation, where for every input x, the obfuscated circuit $\mathcal{O}(C)$ is required to agree with C on x with high probability over $\mathcal{O}$. They construct a "strongly unobfuscatable circuit ensemble" [BGI+12, Theorem 4.3], which has precisely we need: there exists an algorithm L which given any strongly approximate implementation of $F \in \mathcal{F}_\lambda$, efficiently outputs F with high probability. Additionally, their techniques can be extended to yield a family of strongly unobfuscatable PRFs [BGI+12, Section 4.2]. □

Corollary 1.8.6. *Assuming one-way functions, there exists a pseudorandom function family $\mathcal{F}$ which for every $\varepsilon \in [0,1]$ is ε-waterproof.*

Improper versus proper learning. What if the family is not properly learnable: instead of outputting F itself, the learning algorithm $L(C)$ can only output a circuit h that was functionally equivalent to F? One might think that this is indeed sufficient to prove Proposition 1.8.4, but the proof encounters a difficulty.

In the proper-learning setting, it was possible to sample a circuit which for which $\mathsf{Extract}(\mathsf{xk}, C) \neq \mathsf{unmarked}$ *independently of* mk, simply by picking $F \leftarrow \mathcal{F}$. In the improper-learning setting, we only know how to sample from this distribution by evaluating $L(\widetilde{F})$ *on the marked program* $\widetilde{F}$. To violate meaningfulness, we need to construct C such that $\mathsf{Extract}(\mathsf{xk}, C) \neq \mathsf{unmarked}$ with noticeable probability over both Gen and $\mathsf{Extract}$, suggesting that we should find such a C independently of mk.

To get around this issue, we consider families that are learnable with *implementation independence*; that is, for any strong approximate implementations $\mathcal{C}_F^1$ and $\mathcal{C}_F^2$ of F, the distributions $(L(C_1, 1^\lambda) : C_1 \leftarrow \mathcal{C}_F^1)$ and $(L(C_2, 1^\lambda) : C_2 \leftarrow \mathcal{C}_F^2)$ are computationally indistinguishable.[13]

Approximate versus exact learning. In the preceding, we required that an algorithm learning a family $\mathcal{F}$ is able to exactly recover the functionality F. What can we prove if $h = L(C, 1^\lambda)$ is only required to ε-approximate the original function F? For this case, the proof generalizes quite naturally to show that a family is ε-waterproof.

Proposition 1.8.7. *If $\mathcal{F}$ is robustly, ε-approximately learnable with implementation independence, then $\mathcal{F}$ is ε-waterproof.*

Proof. As before, we run the learner on the challenge program to get $h = L(\widetilde{F}, 1^\lambda)$. The circuit h is an ε-approximation of F with non-negligible probability. If $\mathsf{Extract}(\mathsf{xk}, h) = \mathsf{unmarked}$ with noticeable probability, then unremovability is violated. Therefore, it must be the case that $\mathsf{Extract}(\mathsf{xk}, h) \neq \mathsf{unmarked}$ with high probability (even conditioning on the case when h is an ε-approximation).

Observe that for any $F \in \mathcal{F}$, the singleton distribution $\{F\}$ is a strongly approximate implementation of F. To complete the above proof, consider $h' \leftarrow L(F, 1^\lambda)$ for random (unmarked) F (rather than on the marked $\widetilde{F}$). Implementation independence of L guarantees that the distributions of h and h' are indistinguishable and thus for general xk, $\mathsf{Extract}(\mathsf{xk}, h') \neq \mathsf{unmarked}$ with high probability. $\square$

[13]Weaker notions likely suffice because meaningfulness only requires noticeable probability of falsely extracting, whereas this argument gives us a high probability. We consider this input independence notion because it is a simple, natural and, as we will see, powerful case.

Corollary 1.8.8. *Any family that is (improperly, approximately) learnable with membership queries [KL93] is ε-waterproof for any non-negligible ε.*

Proof. An MQ learning algorithm L can be simulated with any approximate implementation C of F. Because $C \leftarrow \mathcal{C}_F$ for $\mathcal{C}_F$ a strongly approximating implementation of F, both C and F will agree on all the queries made by the MQ learner L with high probability. The views of L are statistically close for every approximating distribution $\mathcal{C}$, implying implementation independence. □

Additionally, this proposition captures the impossibility of exact watermarking originally presented in [BGI$^+$12].

Corollary 1.8.9. *Assuming the existence of indistinguishability obfuscation, exact watermarking schemes are impossible.*

Proof. Indistinguishability obfuscation implies a 0-robust, exact, implementation independent learning algorithm for all polynomial-sized circuits, where L simply obfuscates its input.[14] □

1.8.2 Impossibilities for Weak Statistical Correctness

It is possible to prove similar impossibility results even if we weaken the statistical correctness property of the watermarking scheme to only require that $\mathsf{Mark}(\mathsf{mk}, C, \mathsf{m})$ changes functionality at few points, but make no restrictions as to the distributions of these errors. We prove that for this weak setting (1) there exist waterproof PRFs and (2) PAC-learnable families are waterproof. The main difficulty in this setting is that Mark may now change the functionality on adversarially-chosen points, preventing a straightforward adaptation of Proposition 1.8.5 and Corollary 1.8.8.

We now consider watermarking schemes that satisfy only weak statistical correctness:

[14] Observed by Nir Bitansky.

Definition 1.8.10 (Weak Statistical Correctness:). *There is a negligible function $\nu(\lambda)$ such that for any circuit $C \in \mathcal{C}_\lambda$, and any message $\mathsf{m} \in \mathcal{M}_\lambda$:*

$$\mathsf{Mark}(\mathsf{mk}, C, \mathsf{m}) \cong_\nu C$$

We can adapt the learning definitions of the prequel to this weaker notion of statistical correctness. The main change in the definitions is that we no longer require strongly-approximating distributions of circuits $\mathcal{C}_F$ for a function F; an arbitrary circuit $C \cong_\rho F$ that is close to F suffices. This is a strictly more general setting.

Definition 1.8.11 (Learning from arbitrary approximate implementation). *For each of the learning definitions in Definition 1.8.3, we say that the learning algorithm works with* arbitrary approximate implementation *if instead of requiring a ρ-strongly approximate distribution $\mathcal{C}_F$ for F, the learning algorithm will work for arbitrary $C \cong_\rho F$.*

Modifying the definition of waterproof to require that the watermarking scheme only satisfies weak statistical correctness, both Proposition 1.8.7 and 1.8.4 still hold in this setting. Though membership query-learnability no longer suffices for waterproofness, PAC learnability [Val84] does.

Corollary 1.8.12. *Any family that is (improperly) PAC learnable is ε-waterproof (with weak statistical correctness) for any non-negligible ε.*

Proof. An PAC learning algorithm L can be simulated with random queries to arbitrary approximate implementation C of F. Because $C \cong_\rho F$, both C and F will agree on all the random queries seen by L with high probability. The views of L are statistically close for every C, implying implementation independence. □

The main technical contribution of this section is the following PRF construction.

Theorem 1.8.13. *Assuming one-way functions, there exists a pseudorandom function family $\mathcal{F}$ that is robustly, ε-approximately learnable with implementation independence* from arbitrary approximate implementations.

Corollary 1.8.14. *Assuming one-way functions, there exists a pseudorandom function family $\mathcal{F}$ which is ε-waterproof (with weak statistical correctness) for any non-negliglbe ε.*

We provide the proof of Theorem 1.8.13 in the next section.

1.8.3 Waterproof PRFs

The difficulty in this construction is dealing with *arbitrary approximate implementations.* If we try to use the PRF from [BGI$^+$12], changing the functionality on 1 specific point can destroy the learnability. This problem only arises in the case of weak statistical correctness.

We construct a PRF family that has an even stronger form of learnability: from arbitrary approximate implementation C of $f_k \in \mathcal{F}$ that may disagree on $\rho(\lambda) = \mathsf{negl}(\lambda)$ fraction of the domain, we efficiently construct an approximation C' that disagrees with f_k on $\varepsilon(\lambda) = \text{poly}(\lambda)$ fraction of the domain. It seems that we could have done better by simply outputting C! But C' (in particular, the erring inputs) are *completely independent of C*—guaranteeing implementation independence as required to prove that $\mathcal{F}$ is waterproof.

Our starting point is the constructions of unobfuscatable function families in [BGI$^+$12] and [BP13], and an understanding of those constructions will prove helpful towards understanding ours.

The former work was discussed in Proposition 1.8.5. 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, but sacrifice the total learnability of the earlier construction, instead learning only a single predicate of the PRF key. We require a notion of approximation stronger than [BGI$^+$12] but weaker than [BP13], and a notion of learnability weaker than [BGI$^+$12] but stronger than [BP13], and achieve this by adapting techniques from both works.

1.8.3.1 Preliminaries

The construction requires an invoker randomizable pseudorandom function [BGI+12] and a decomposable encryption schemes [BP13]. The following definitions and discussion are taken almost verbatim from those works.

Definition 1.8.15 (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 1.8.16 ([BGI+12]). *If pseudorandom functions exist, then there exist invoker-randomizable pseudorandom functions.*

Definition 1.8.17 (Decomposable Encryption [BP13]). *An encryption scheme* (Gen, Enc, Dec) *is* decomposable *if there exists an efficient algorithm* pub *that operates on ciphertexts and satisfies the following conditions:*

1. *For a ciphertext c,* pub(c) *is independent of the plaintext and samplable; that is, there exists an efficient sampler* 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 deterministeically defined by* 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 and for security parameter λ, the secret key is a random $sk \in \{0,1\}^\lambda$, and the encryption of a bit b is computed by sampling a random $r \leftarrow \{0,1\}^\lambda$ and outputting $(r, F_{sk}(r) \oplus b)$. This function satisfies a number of necessary properties [BP13]:

- 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 $\frac{1}{2} - \mathsf{negl}$) 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.

1.8.3.2 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 $i \in \{1, \ldots, 9\}$, 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 [BP13] 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 ε-close approximation for any $\varepsilon = 1 - \frac{1}{\mathrm{poly}(\lambda)}$, we simply augment the index i with an additional $\log(1/(1-\varepsilon))$

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 ε of the function. This establishes the theorem.[15]

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 cipher-texts 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 reencrypting, 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

[15] Note that the result is a PRF family that depends on the choice of ε. The argument would fail if ε 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 ε simultaneously) would be interesting.

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 $k = (\alpha, \beta, sk, s_1, s_2, s_e, s_h, s_b, s^*)$ 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 Theorem 1.8.13, and pseudorandomness.

1.8.3.3 Learnability

We must show that $F_\lambda = \{f_k\}$ is robustly, $\frac{1}{9}$-approximately learnable by an implementation independent algorithm, L from arbitrary approximate implementation.[16]

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

It suffices to show that, given any ρ-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 (by fixing some uniformly random q in the above). [17] 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 $\alpha \oplus F_{s_b}(\mathbf{0}, c, \bar{q})$. Evaluating $\mathbb{R}_b(\mathbf{0}, \mathsf{pub}(c_1), \ldots, \mathsf{pub}(c_n), \bar{q})$ 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) ρ-fraction of the domain for some negligible function $\rho(n)$? 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 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 uniform.

For $\mathbb{H}$, we leverage the invoker-randomizability of the PRF F_{s_h}, using the argument presented in [BGI+12, Proof of Theorem 4.3]. 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 64ρ, which is still negligible.

For $\mathbb{B}$ and $\mathbb{R}_b$, we leverage the properties of the decomposable symmetric-key

[17]This 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.

encryption scheme, using the argument presented in [BP13, Proof of Claim 3.8]. 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 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.

1.8.3.4 Pseudorandomness

Our proof that the family $\{f_k\}$ is pseudorandom follows that of [BP13]; 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 1.8.18. $\Pr_{k,\mathcal{A}}[E_1] = 0$

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

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 function. 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 functions). 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 1.8.20. *For all i,* $\Pr_{k,\mathcal{A}}[E_{\beta,i} | \neg E_{\alpha,i-1}] \leq \mathsf{negl}(\lambda)$

Proof. It suffices to show that for all i:

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

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 1.8.19, 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 function. 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 1.8.21. $\Pr_{k,\mathcal{A}}[E_{\alpha,i} | \neg E_{\beta,i-1}] \leq \mathsf{negl}(\lambda)$

Proof. It suffices to show that for all i:

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

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, Claim 3.6.1] and [BP13, Claim 3.3], 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. □

1.9 Conclusions

We showed how to watermark various cryptographic capabilities: PRF evaluation, ciphertext decryption, and message signing. For all of these, there is a natural and secret "true functionality" f_k that we would like to mark. Given a message m, we can distribute a "marked" circuit C which closely approximates f_k. Given C, any efficiently findable circuit C^* **which** even loosely approximates f_k must also contain m. Furthermore, in our scheme, the procedure for extracting m is entirely public-key. We show that unmarked circuits cannot approximate the marked capability to within an approximation factor of $\varepsilon = \frac{1}{2} + 1/\operatorname{poly}$ for any poly. If we allow *list decoding*, namely allow the extraction procedure to output a polynomial-sized list of messages containing m, then ε can be lowered to $1/\operatorname{poly}$.

There are several directions for further research. First, one could explore the connection between obfuscation and watermarking to see whether some form of obfuscation is *necessary* to achieve watermarking or if one can come up with constructions that avoid obfuscation. This was partially answered by Kim and Wu [KW17] since they presented a watermarking scheme with secret-key extraction from lattice-based assumptions. However, a watermarking scheme with *public-key* extraction without obfuscation remains open. Secondly, it would be interesting to achieve a fully public-key watermarking construction where both the marking and the detection procedure only use public keys. In the setting where the marking oracle takes keys as input, this kind of watermarking appears plausible. As usual with obfuscation, there is a heuristic construction which obfuscates the secret-key marking procedure to generate a public marking key. Proving such a scheme secure by only relying on iO (as opposed to VBB) appears to require significantly new techniques. Third, it would be interesting to explore weaker but meanigful models for watermarking such as the work by Baldmtsi et al. [BKS17] since there is a possibility to achieve watermarking based on standard cryptographic tools such as one-way functions. Finally, watermarking schemes for richer classes of programs seem to be beyond the reach of our techniques, but would be of obvious interest.

Chapter 2

Updatable Cryptography

2.1 Introduction

The last decade has seen the advent of a vast array of advanced cryptographic primitives such as attribute-based encryption [SW05, GPSW06], predicate encryption [BW07, SBC$^+$07, KSW08, GVW15a], fully homomorphic encryption [Gen09b], fully homomorphic signatures [ABC$^+$07, BF11, GVW15b], functional encryption [SW05, BSW11, O'N10, GGG$^+$14], constrained pseudorandom functions [BW13, BGI13, KPTZ13], witness encryption [GGSW13, GLW14], witness PRFs [Zha16], indistinguishability obfuscation [BGI$^+$01b, GGH$^+$13b], and many more. Most of these primitives can be viewed as "cryptographic circuit compilers" where a circuit C can be compiled into an encoding $\langle C \rangle$ and an input x can be encoded as $\langle x \rangle$ such that they can be evaluated together to compute $C(x)$. For example, in a functional encryption scheme, circuit compilation corresponds to the key generation process whereas input encoding corresponds to encryption. Over the recent years, cryptographic circuit compilers have revolutionized cryptography by providing non-interactive means of computing over inputs/data.

A fundamental limitation of these circuit compilers is that they only support *static* compilation. That is, once a circuit is compiled, it can no longer be modified. In reality, however, compiled circuits may need to undergo several updates over a

Based on "Cryptography with updates" with Prabhanjan Ananth and Abhishek Jain [ACJ17].

period of time. For example, consider an organization where each employee is issued a decryption key SK_P of an attribute-based encryption scheme where the predicate P corresponds to her access level determined by her employment status. However, if her employment status later changes, then we would want to update the predicate P associated with her decryption key. Known schemes, unfortunately, do not support this ability.

Motivated by the necessity of supporting updates in applications, we study and build *dynamic* circuit compilers. In a dynamic circuit compiler, it is possible to update a compiled circuit $\langle C \rangle$ into another compiled circuit $\langle C' \rangle$ by using an *encoded update string* whose size only depends on the difference between the plaintext circuits C and C'. For example, if the difference between C and C' is simply a single gate change, then this should be reflected in the size of the encoded update. Note that this rules out the trivial solution of simply releasing a new compiled circuit at the time of update.

Background: incremental cryptography. The study of cryptography with updates was initiated by Bellare, Goldreich and Goldwasser [BGG94] under the umbrella of *incremental cryptography.* They studied the problem of incremental digital signatures, where given a signature of a message m, it should be possible to efficiently compute a signature of a related message m', without having to recompute the signature of m' from scratch. Following their work, the study of incremental cryptography was extended to other basic cryptographic primitives such as encryption and hash functions [BGG94, BGG95, Mic97, Fis97, BM97, BKY01, MPRS], and more recently, indistinguishability obfuscation [GP15, AJS15b].

Our goal. In this chapter, we continue this line of research, and perform a systematic study of updatable cryptographic primitives. We take a unified approach towards adding updatability features to recently studied primitives such as attribute-based encryption, functional encryption and more generally, cryptographic circuit compilers. We, in fact, go further and also study updatability for classical protocols such as zero-knowledge proofs and secure multiparty computation.

To accomplish this goal, we introduce a new notion of *updatable randomized encodings* that extends the standard notion of randomized encoding [IK00] to incorporate updatability features. We show that updatable randomized encodings can be used to generically transform cryptographic primitives (discussed above) to their updatable counterparts.

2.1.0.1 Updatable Randomized Encodings

The notion of randomized encoding [IK00] allows one to encode a "complex" computation $C(x)$ into a "simple" randomized function $\mathsf{Encode}(C, x; r)$ such that given the output $\langle C(x) \rangle$ of the latter, it is possible to recover the value $C(x)$ (by running a public Decode algorithm) but it is impossible to learn anything else about C or x. The typical measure of complexity studied in the literature is parallel-time complexity or circuit depth. Such randomized encodings are known to exist for general circuits based on only the existence of one-way functions [AIK07] (also referred to as Yao's garbled circuits [Yao86], where $\mathsf{Encode}(C, x; r)$ is in $\mathbf{NC}^1$).

We study *updatable* randomized encodings (URE): given a randomized encoding $\langle C(x) \rangle$ of $C(x)$, we want the ability to update it to an encoding $\langle C'(x') \rangle$ of $C'(x')$, where C' and x' are derived from C and x by applying some "update" $\mathbf{u}$. For now, we may think of this update as some small modification to the circuit or input (e.g., change the output gate of C to AND and the second bit of x to 1). We require that the update $\mathbf{u}$ can be encoded as $\langle \mathbf{u} \rangle$ which can then be used to transform $\langle C(x) \rangle$ into $\langle C'(x') \rangle$, a randomized encoding of $C'(x')$. A bit more precisely, a URE scheme consists of the following algorithms:

$\mathsf{Encode}(C, x)$: takes as input a circuit C and an input x, and outputs an encoding $\langle C(x) \rangle$ and a secret state st.

$\mathsf{GenUpd}(\mathsf{st}, \mathbf{u})$: takes as input an update $\mathbf{u}$, and outputs an encoded update $\langle \mathbf{u} \rangle$ and a possibly updated state st'.

$\mathsf{ApplyUpd}(\langle C(x)\rangle, \langle \mathbf{u}\rangle)$: takes as input a randomized encoding $\langle C(x)\rangle$ and an update encoding $\langle \mathbf{u}\rangle$, and outputs an (updated) encoding $\langle C'(x')\rangle$.

$\mathsf{Decode}(\langle C(x)\rangle)$: takes as input a (possibly updated) randomized encoding $\langle C(x)\rangle$, and outputs the value $y = C(x)$.

If we make no additional requirements, the above could be easily achieved. For instance, let Encode output the state $\mathsf{st} = (C, x)$, and let GenUpd—which now has access to C and x from st in addition to the update $\mathbf{u}$—compute the updated C' and x' directly and output as the encoded update $\langle \mathbf{u}\rangle$ the standard randomized encoding of $\langle C'(x')\rangle$. $\mathsf{ApplyUpd}$ would correspondingly output $\langle \mathbf{u}\rangle = \langle C'(x')\rangle$. The drawback of this approach is that a fresh randomized encoding is computed during every evaluation of GenUpd, irrespective of whether $\mathbf{u}$ constitutes a minute or significant change to the underlying C and x.

Our key efficiency requirement is that the running time of the GenUpd algorithm must be a fixed polynomial size of the update (and a security parameter), and independent of the size of the circuit and input being updated. This, in particular, implies that the size of an update encoding $\langle \mathbf{u}\rangle$ is also a fixed polynomial in the size of $\mathbf{u}$ (and the security parameter).

The above discussion immediately generalizes to the setting of *multiple sequential updates.*[1] Let $\langle C_0(x_0)\rangle$ denote an initial randomized encoding. Let $\mathbf{u}_1, \ldots, \mathbf{u}_{\mathsf{q}}$ denote a sequence of updates and let $\langle \mathbf{u}_i\rangle$ denote an encoding of $\mathbf{u}_i$. In a URE scheme for multiple updates, $\langle C_0(x_0)\rangle$ can be updated to $\langle C_1(x_1)\rangle$ using $\langle u_1\rangle$; the result can then be updated into $\langle C_2(x_2)\rangle$ using $\langle u_2\rangle$, and so on, until we obtain $\langle C_{\mathsf{q}}(x_{\mathsf{q}})\rangle$. We allow the number of updates q to be an arbitrary polynomial in the security parameter.

Within this framework, two distinct notions naturally arise.

URE with multiple evaluations: Every intermediate encoding $\langle C_i(x_i\rangle$ can be decoded to obtain $C_i(x_i)$. For security, we require that given an initial randomized

[1]One may also consider an alternative notion of *parallel* updates, where every update $\langle \mathbf{u}_i\rangle$ is applied to the *original* encoding $\langle C_0(x_0)\rangle$. It turns out that URE with parallel updates is closely connected to the notion of reusable garbled circuits [GKP+13] (see Section 2.4.2).

encoding $\langle C_0(x_0)\rangle$ and a sequence of encoded updates $\{\langle \mathbf{u}_i\rangle\}_{i=1}^{\mathsf{q}}$, an adversary can learn only the outputs $\{C_i(x_i)\}_{i=0}^{\mathsf{q}}$, and nothing else.

URE with single evaluation: Only the final encoding $\langle C_{\mathsf{q}}(x_{\mathsf{q}})\rangle$ can be decoded. To enable this, we will consider an augmented decoding algorithm that additionally requires an "unlocking key."[2] This unlocking key is provided after all the updates are completed, allowing the user to decode the final encoding, but preventing her from decoding any intermediate values. For security, we require that given an initial randomized encoding $\langle C_0(x_0)\rangle$ and a sequence of encoded updates $\{\langle \mathbf{u}_i\rangle\}_{i=1}^{\mathsf{q}}$), an adversary can only learn the final output $C_{\mathsf{q}}(x_{\mathsf{q}})$, and nothing else.

Except where otherwise specified, we use URE to mean the multiple-evaluation variant. For both conceptual reasons and to minimize confusion, we in fact consider an alternative but equivalent formulation of single-evaluation URE which we call *updatable garbled circuits* (UGC). A garbled circuit [Yao86] is a "decomposable" randomized encoding, where a circuit C and an input x can be encoded separately. In an updatable garbled circuit scheme, given an encoding $\langle C_0\rangle$ of a circuit C_0 and a sequence of update encodings $\langle \mathbf{u}_1\rangle, \ldots, \langle \mathbf{u}_{\mathsf{q}}\rangle$, it is possible to compute updated circuit encodings $\langle C_1\rangle, \ldots, \langle C_{\mathsf{q}}\rangle$, where C_i is derived from C_{i-1} using $\mathbf{u}_i$. Once all the updates are completed, an encoding $\langle x\rangle$ for an input x is released. This input encoding can then be used to decode the final circuit encoding $\langle C_{\mathsf{q}}\rangle$ and learn $C_{\mathsf{q}}(x_{\mathsf{q}})$. Intuitively, the input encoding can be viewed as the unlocking key in single-evaluation URE.

It is easy to see that UGC is a weaker notion than multi-evaluation URE. In particular, since UGC only allows for decoding "at the end," it remains *single-use*, while multi-evaluation URE captures *reusability*.

We find the notions of URE and UGC to be of interest from a purely complexity-theoretic perspective. Further, as we discuss later, they have powerful applications to updatable cryptography.

[2]In the setting of bounded updates, this modification is unnecessary. We focus primarily on the unbounded setting.

2.1.1 Our Results

We initiate the study of updatable randomized encodings. We study both simulation and indistinguishability-based security definitions and obtain general positive results in a selective-security setting. We showcase URE as a central object for the study of updatable cryptography by demonstrating applications to other updatable cryptographic primitives. The technical ideas we develop for our constructions are quite general, and may be applicable to future works on updatable cryptography.

2.1.1.1 Multi-evaluation URE for General Updates

Before stating our positive results for multi-evaluation URE, we first informally describe which classes of updates we can support. An update $\mathbf{u} \in \mathcal{U}$ represents some way to modify any circuit C and an input x to some modified circuit C' and input x'. We denote by Update the procedure $(C', x') \leftarrow \mathsf{Update}(C, x, \mathbf{u})$ which applies the update to C and x. We consider all $\mathcal{U}$ and $\mathbf{u}$ subject to two restrictions: (1) $\mathbf{u}$ is computed by a (family) of circuits, one for every circuit size $|C|$, and (2) $\mathbf{u}$ preserves circuit size (i.e., $|C| = |C'|$). We refer to this very broad class of updates as *general circuit updates.*

For general circuit updates, we construct URE from *compact* functional encryption. The summary below focuses on indistinguishability-based security, and concludes with a remark on achieving simulation-based security.

Theorem 2.1.1 (Informal). *Assuming the existence of secret-key, compact functional encryption supporting a single key query and B ciphertexts, there exists a multi-evaluation* URE *scheme supporting B sequential general circuit updates.*

A compact functional encryption is one where the running time of the encryption algorithm for a message m is a fixed polynomial in the size of m and the security parameter, and independent of the complexity of the function family supported by the FE scheme.

For the case of **unbounded** updates, a recent work of Bitansky et al. [BNPW16] shows that secret-key compact functional encryption with unbounded-many ciphertexts implies exponentially-efficient indistinguishability obfuscation (XIO) [LPST16a]. Put together with the results of [LPST16a] and [AJ15, BV15a], it shows that *sub-exponentially* secure secret-key compact FE that supports a single function key query together with the learning with errors (LWE) assumption implies indistinguishability obfuscation.

In contrast, in Theorem 2.1.1, we require secret-key compact FE with only *polynomial security.* Such an FE scheme can be based on polynomial-hardness assumptions on multilinear maps using the results of [GGHZ14] and [BV15a, AJS15a].

For the case of **polynomially-bounded** updates, we can, in fact, relax our assumption to only *one-way functions.* We obtain this result by using a *stateful* single-key compact secret-key FE scheme for an a priori bounded number B of ciphertexts. A stateful single-key compact secret-key FE scheme can be constructed from garbled circuits: a functional key consists of B garbled circuits, i^{th} ciphertext consists of garbled wire keys corresponding to the i^{th} garbled circuit. This FE scheme is stateful since the encryption algorithm needs to store how many messages it has encrypted so far.

Plugging in such an FE scheme in Theorem 2.1.1 yields the following corollary.

Corollary 2.1.2 (Informal). *Assuming one-way functions, for any fixed polynomial B, there exists a multi-evaluation* URE *scheme supporting B sequential general circuit updates.*

On the necessity of functional encryption. It is natural to ask whether secret-key compact FE is necessary for building multi-evaluation URE with unbounded updates. We show that if a (multi-evaluation) URE scheme is *output compact*, then it implies XIO. Put together with the result of [LPST16a], we have that a URE scheme with output compactness together with LWE implies a public-key compact FE scheme that supports a single key query.

Theorem 2.1.3 (Informal). *Assuming LWE, a multi-evaluation* URE *scheme with unbounded output-compact updates implies a public-key compact FE scheme that supports a single key query.*

In an output-compact URE scheme, the running time of the GenUpd algorithm is independent of the output length of the updated circuit. We remark that the URE scheme obtained from Theorem 2.1.1 is, in fact, output compact. Our construction in Theorem 2.1.1 is in this sense tight.

On output compactness. We study both indistinguishability and simulation-based security notions for URE. In the context of FE, it is known from [AGVW13, CIJ$^+$13] that simulation-secure FE with output compactness is impossible for general functions. We observe that the same ideas as in [AGVW13, CIJ$^+$13] can be used to establish impossibility of simulation-secure URE with output compact updates.

However, when we consider indistinguishability-based security, URE with output compact updates is indeed possible. The results in Theorem 2.1.1 and Corollary 2.1.2 are stated for this case. Furthermore, using the trapdoor circuits technique of [CIJ$^+$13], one can generically transform output-compact URE with indistinguishability security to non-output-compact URE with simulation-based security.

2.1.1.2 Updatable Garbled Circuits with Gate-wise Updates

We now turn to updatable garbled circuits, an alternate formulation of single-evaluation URE. We consider the family of gate-wise updates, where an update $\mathbf{u}$ can modify a single gate of a circuit or add or delete a gate. Below, we consider the case of unbounded updates and bounded updates separately.

UGC **with unbounded updates from lattice assumptions.** Our first result is a construction of UGC for general circuits that supports an unbounded number of sequential updates from the family of gate-wise updates. We build such a scheme from worst-case lattice assumptions.

Theorem 2.1.4 (Informal). *Let $\mathcal{C}$ be a family of general circuits. Assuming the hardness of approximating either GapSVP or SIVP to within sub-exponential factors, there exists a* UGC *scheme for $\mathcal{C}$ that supports an unbounded polynomial number of sequential gate-wise updates.*

At the heart of this result is a new notion of *puncturable symmetric proxy reencryption scheme* that extends the well-studied notion of proxy reencryption [BBS98a]. In a symmetric proxy reencryption scheme, for any pair of secret keys SK_1, SK_2, it is possible to construct a reencryption key $\mathsf{rk}_{1\to 2}$ that can be used to publicly transform a ciphertext w.r.t. SK_1 into a ciphertext w.r.t. SK_2. In our new notion of puncturable proxy reencryption, reencryption keys can be "disabled" on ciphertexts CT^* (w.r.t. SK_1) s.t. the semantic security of CT^* holds even if the adversary is given the punctured key $\mathsf{rk}_{1\to 2}^{\mathsf{CT}^*}$ and SK_2. We give a construction of such a scheme based on the hardness of approximating either GapSVP or SIVP to within sub-exponential factors.

Given the wide applications of proxy reencryption (see, e.g., [AFGH05] for a discussion), we believe that our notion of puncturable proxy reencryption is of independent interest and likely to find new applications in the future.

UGC with bounded updates from one-way functions. For the case of a polynomially-bounded number of updates, we can relax our assumption to only *one-way functions.* We obtain this result by using a puncturable PRF scheme that can be based on one-way functions [GGM84, SW14].

Theorem 2.1.5 (Informal). *Let $\mathcal{C}$ be a family of general circuits, and λ be a security parameter. Assuming one-way functions, for any fixed polynomial q, there exists a* UGC *scheme for $\mathcal{C}$ that supports $\mathsf{q}(\lambda)$ sequential gate-wise updates. The size of the initial garbled circuit as well as each update encoding is independent of q. However, the initial circuit garbling time and update generation time grows with q.*

The construction of this scheme is quite simple and does not require a puncturable proxy reencryption scheme. We provide an informal description of this scheme in the technical overview section 2.2.1.

2.1.1.3 Applications

We next discuss applications of our results.

Updatable primitives with IND security. We start by discussing application of multi-evaluation URE to dynamic circuit compilers. Here, we demonstrate our main idea by a concrete example, namely, by showing how to use URE to transform any (key-policy) attribute-based encryption (ABE) scheme into *updatable ABE*. The same idea can be used in a generic way to build dynamic circuit compilers and obtain updatable functional encryption, updatable indistinguishability obfuscation, and so on. We refer the reader to Section 2.8.1 for the general case.

We briefly describe a generic transformation from any ABE scheme to one where the policies associated with secret keys can be updated. The setup and encryption algorithms for the updatable ABE scheme are the same as in the underlying ABE scheme. The key generation algorithm in the updatable ABE scheme works as follows: to compute an attribute key for a function f, we compute a URE $\langle C_f \rangle$ of a circuit C_f where C runs the key generation algorithm of the underlying ABE scheme using function f and outputs a key SK_f. To decrypt a ciphertext, a user can first decode $\langle C_f \rangle$ to compute SK_f and then use it to decrypt the ciphertext.

In order to update an attribute key for a function f to another key for function f', we can simply issue an update encoding $\langle \mathbf{u} \rangle$ for $\langle C_f \rangle$ where $\mathbf{u}$ captures the modification from f to f'. To compute the updated attribute key, a user can first update $\langle C_f \rangle$ using $\langle \mathbf{u} \rangle$ to obtain $\langle C_{f'} \rangle$, and then decode it to obtain an attribute key $\mathsf{SK}_{f'}$ for f'.

Let us inspect the efficiency of updates in the above updatable ABE scheme. As in URE, we would like the size (as well as the generation time) of an update encoding here to be independent of the size of the updated function. Note, however, that the output of the updated function $C_{f'}$ is very large—an entire attribute key $\mathsf{SK}_{f'}$! Thus, in order to achieve the aforementioned efficiency, we require that the URE scheme has updates with output compactness.

Recall that URE with output compact updates is only possible with indistinguishability based security. As such, the above idea is only applicable to cryptographic primitives with indistinguishability-based security.

Updatable primitives with SIM security. Next, we discuss applications of URE to cryptographic primitives with simulation-based security. In the main body of the chapter, we describe two concrete applications, namely, *updatable non-interactive zero-knowledge proofs* (UNIZK) and *updatable multiparty computation* (UMPC). A notable feature of these constructions is that they only require a URE scheme with *non-output-compact* updates and simulation-based security. Below, we briefly describe our main idea for constructing UNIZKs.

Let (x, w) denote an instance and witness pair for an **NP** language L. Let $\mathbf{u}$ denote an update that transforms (x, w) to another valid instance and witness pair (x', w'). In a UNIZK proof system for L, it should be possible for a prover to efficiently compute an encoding $\langle \mathbf{u} \rangle$ of $\mathbf{u}$ that allows a verifier to transform a valid proof π for x into a proof π' for x' and verify its correctness.

A proof π for (x, w) in the UNIZK scheme is computed as follows. We first compute a URE $\langle C_{x,w} \rangle$ for a circuit $C_{x,w}$ that checks whether (x, w) satisfies the **NP** relation associated with L and outputs 1 or 0 accordingly. We also compute a regular NIZK proof ϕ to prove that $\langle C_{x,w} \rangle$ is computed "honestly." To verify $\pi = (\langle C_{x,w} \rangle, \phi)$, a verifier first verifies ϕ and if the check succeeds, it decodes $\langle C_{x,w} \rangle$ and outputs its answer.

In order to update a proof π, we can simply issue an update encoding $\langle \mathbf{u} \rangle$ for the randomized encoding $\langle C_{x,w} \rangle$, along with a regular NIZK proof ϕ' that $\langle \mathbf{u} \rangle$ was computed honestly. Upon receiving the update $(\langle \mathbf{u} \rangle, \phi')$, a verifier can first verify ϕ' and then update $\langle C_{x,w} \rangle$ using $\langle \mathbf{u} \rangle$ to obtain $\langle C_{x',w'} \rangle$. Finally, it can decode the updated URE $\langle C_{x',w'} \rangle$ to learn whether x' is in the language L or not.

It should be easy to see that the above idea can, in fact, be also used to make *interactive* zero-knowledge proofs updatable. Finally, we note that the above is a slightly oversimplified description and we refer the reader to Sections 2.8.4 and 2.8.5 for further details on UNIZK and UMPC, respectively.

2.1.2 Related Work

Incremental cryptography. The area of incremental cryptography was pioneered by Bellare, Goldreich and Goldwasser [BGG94]. While their work dealt with signature schemes, the concept of incremental updates has been subsequently studied for other basic cryptographic primitives such as hash functions, semantically-secure encryption and deterministic encryption [BGG95, Mic97, Fis97, BKY01, MPRS]. To the best of our knowledge, all of these works only consider bit-wise updates, in which a single bit of the message is modified.

While our work shares much in spirit with these works, we highlight one important difference. In incremental cryptography, update operation is performed "in house," e.g., in the case of signatures, the entity who produces the original signature also performs the update. In contrast, we consider a *client-server* scenario where the client simply produces an update encoding, and the actual updating process is performed by the server. This difference stipulates different efficiency and security requirements. On the one hand, incremental cryptography necessarily requires efficient updating time for the notion to be non-trivial, while we consider the weaker property of efficient update encoding generation time. On the other hand, our security definition is necessarily stronger since we allow the adversary to view the update encodings—a property not necessary when the updating is done "in house."

Incremental and patchable obfuscation. Recently, [GP15] and [AJS15b] study the notion of updatability in the context of *indistinguishability obfuscation.* The work of [GP15] considers incremental (i.e., bit-wise) updates, while [AJS15b] allow for arbitrary updates, including those that may increase the size of the program (modeled as a Turing machine).

We note that one of our results, namely, URE with unbounded updates can be derived from [AJS15b] at the cost of requiring sub-exponentially secure $i\mathcal{O}$. In contrast, we obtain our result by using *polynomially secure* secret-key compact FE. On the other hand, our work only allows updates to single obfuscated circuits, whereas [AJS15b] considers the problem of updating many obfuscated Turing machines simultaneously.

Malleable NIZKs. Our notion of updatable NIZKs should be contrasted with the notion of malleable NIZKs proposed by Chase et al. [CKLM12]. In a malleable NIZK, it is possible to publicly "maul" a proof string π for a statement x into a a proof string π' for a related statement x'. In contrast, our notion of UNIZK only allows for privately generated updates. To the best of our knowledge, malleable NIZKs are only known either for a limited class of update relations from standard assumptions [CKLM12], or for general class of update relations based on non-falsifiable assumptions such as succinct non-interactive arguments [CKLM13]. In contrast, we show how to build UNIZK for unbounded number of general updates from compact secret-key FE and regular NIZKs, and for a bounded number of general updates from regular NIZKs.

Updatable codes. The concept of updating was also studied in the context of error correcting codes by [CKO14]. In this context, it is difficult to model the problem of updating—we should be able to change few bits of the code to correspond to a codeword of a different message and at the same time we want the distance between codewords of different messages to be far apart. We refer the reader to their work for discussion on this seemingly contradictory requirement. In a subsequent work, [DSLSZ15] studied this problem in the context of non-malleable codes.

2.2 Our Techniques

We start with the construction of UGC and present the main ideas underlying the construction. We then build upon the intuition developed in the construction of UGC, to construct (multi-evaluation) URE.

2.2.1 Construction of UGC

2.2.1.1 A *Lock-and-Release* Mechanism for Single Update

To begin, we construct a UGC scheme supporting a single gate update. Let C be a circuit comprised of s-many gates $C^1, \ldots, C^s$. To garble C, we simply compute a garbling of C using a standard gate-by-gate garbling scheme such as [Yao86].[3] We denote by $\langle C \rangle_{\mathsf{gc}}$ the garbled circuit for C, and by $\langle C \rangle^{i}_{\mathsf{gc}}$ the garbled gate corresponding to gate C^i. Encrypt each garbled gate, and output the resulting ciphertexts $\mathsf{CT}_1, \ldots, \mathsf{CT}_s$.

Now, suppose we wish to update C to C' by modifying the first gate. Suppose it was possible to generate a special decryption key that decrypts the ciphertexts $\mathsf{CT}_2, \ldots, \mathsf{CT}_s$ but not CT_1. The encoding of the update consists of this special decryption key, along with a garbled version of the new gate $\langle C' \rangle^{1}_{\mathsf{gc}}$. The receiver can use this to decrypt and recover the garbled gates $\langle C \rangle^{2}_{\mathsf{gc}}, \ldots, \langle C \rangle^{s}_{\mathsf{gc}}$. Together with $\langle C' \rangle^{1}_{\mathsf{gc}}$, this forms a complete garbled circuit for C'.

How can we implement the special decryption key? A naive approach is to encrypt each ciphertext with an independent encryption key, and then release the decryption key for every position $i \neq 1$. However, the size of the resulting update encoding is proportional to $s = |C|$, no better than garbling C' from scratch.

We could instead use a (secret key) *puncturable* encryption scheme. In a puncturable encryption scheme, it is possible to generate a *punctured decryption key* that can decrypt all ciphertexts but one. Punctured encryption schemes can be built from puncturable pseudorandom functions [SW14, BGI13, BW13, KPTZ13] (c.f. Waters [Wat15b]) which in turn can be based on any one-way function. Given such an encryption scheme, the above construction of single-update UGC achieves our efficiency goals.

Abstractly, the above construction can be though of as a *lock-and-release* mechanism. The encryption of the wire keys corresponding to C constitutes the *locking* step, while the dissemination of the punctured decryption key constitutes the (conditional) *release* step. We find this abstraction useful going forward.

2.2.1.2 A *Layered Lock-and-Release* Mechanism for Bounded Updates

The previous solution does not offer any security for even two sequential updates. Updates for two different gates would allow an adversary to recover a garbling of the

[3]In gate-by-gate garbling schemes such as [Yao86], each boolean gate can be garbled knowing only the circuit topology and the gate's functionality, independently of the remainder of the circuit.

original circuit. To support an a priori bounded number of updates q, we use a *layered* punctured encryption—abstractly, a *layered lock-and-release* mechanism.

When garbling the circuit C, instead of encrypting the garbled gates a single time, we use q onion layers of encryption scheme, each using a punctured encryption scheme. Let $\mathbf{u}_1, \ldots, \mathbf{u}_\mathsf{q}$ be a sequence of gate updates, each consisting of a gate $g \in [s]$ to change and a new gate type. To generate an updatable garbled circuit for C, first garble $\tilde{C}$ using a traditional gate-by-gate scheme. Sample q keys $\mathsf{SK}_1, \ldots, \mathsf{SK}_\mathsf{q}$ for a puncturable encryption scheme. Encrypt each garbled gate $\langle C \rangle^i_{\mathsf{gc}}$ of the garbled circuit in q layers, yielding a ciphertext $\mathsf{CT}_i = \mathsf{Enc}(\mathsf{SK}_1, \mathsf{Enc}(\mathsf{SK}_2, \ldots \mathsf{Enc}(\mathsf{SK}_\mathsf{q}, \langle C \rangle^i_{\mathsf{gc}})))$.

The encoding of the first update $\mathbf{u}_1 = (g_1, \mathsf{gateType}_1)$ is a decryption key for the outermost encryption layer that is punctured at CT_{g_1}, along with a layer $(\mathsf{q} - 1)$ encryption $\mathsf{CT}'_{g_i} = \mathsf{Enc}(\mathsf{SK}_2, \mathsf{Enc}(\mathsf{SK}_3, \ldots \mathsf{Enc}(\mathsf{SK}_\mathsf{q}, \langle C' \rangle^{g_i}_{\mathsf{gc}})))$, where $\langle C' \rangle^{g_i}_{\mathsf{gc}}$ is the new garbled gate. Likewise, an encoding of the i-th update $\mathbf{u}_i$ is a punctured decryption key for SK_i, and an $(i-1)$-layered encryption of the new garbled gate.

The use of layered punctured encryption ensures that the receiver cannot skip any update, and instead must apply all the updates one-by-one to peel off all the encryption layers from the garbled gates. Furthermore, since the encryption layers can only be removed in a prescribed order, the receiver must apply the updates *in order*. After applying all the updates the receiver only obtains a single garbled gate at every location in the circuit.

Theorem 2.1.5 claims that the garbled circuit and of each update encoding is independent of q. But a generic instantiation of the layered encryption described above would yield ciphertexts (and garbled circuits and updates) that grow with q. To achieve the promised efficiency, we use a punctured encryption scheme that supports layered encryption without ciphertext blowup.

2.2.1.3 A *Relock-and-Eventual-Release* Mechanism for Unbounded Updates

The previous construction inherently requires the number of updates to be bounded a priori. To support unbounded updates we develop a *relock-and-eventual-release*.

Instead of removing a lock at every step, our idea is to change the lock at every step by replacing That is, we replace the encryption with *reencryption* [BBS98a]. In a symmetric reencryption scheme, given two encryption keys SK_1 and SK_2, it is possible to generate a reencryption key $\mathsf{rk}_{1\to 2}$ that transforms any ciphertext under SK_1 into a ciphertext under SK_2.

As in the earlier constructions, we need the reencryption scheme to be puncturable. We define and construct a *puncturable symmetric reencryption scheme*. A punctured reencryption key $\mathsf{rk}_{1\to 2}^{\mathsf{CT}^*}$ allows one to transform any ciphertext under SK_1 into a ciphertext under SK_2, except the ciphertext CT^*. The semantic security of CT^* should hold against an adversary given a chain of reencryption keys $\mathsf{rk}_{1\to 2}, \ldots \mathsf{rk}_{\mathsf{q}-1\to \mathsf{q}}$ and the terminal secret key SK_q, so long as at least one of the reencryption keys is punctured at CT^*.

We can now modify the previous solution template to support unbounded updates. The garbling of a circuit C consists of $\tilde{C}$ as before. The garbled gates $\langle C\rangle_{\mathsf{gc}}^i$ are now encrypted using a puncturable reencryption scheme. Let SK_0 denote the secret key used to encrypt the garbled gates. In order to issue an update encoding for an update $\mathbf{u}_i$ for gate g, we release (a) a reencryption key $\mathsf{rk}_{i-1\to i}^{\mathsf{CT}_{i-1}^g}$ punctured at ciphertext CT_{i-1}^g, the encryption of $\langle C\rangle_{\mathsf{gc}}^g$ under SK_{i-1}, and (b) an encryption of $\langle C'\rangle_{\mathsf{gc}}^g$ under SK_i. For the final update q, we release the qth secret key SK_q.

This construction does not hide the location of the updates. Indeed, the evaluator needs to know the location being updated in order to correctly apply the updates. To address this, we provide a generic transformation from any UGC scheme (or in fact, any URE scheme) that does not achieve hide the location of updates into one that does. Our transformation uses non-interactive oblivious RAM in the same manner as in [GP15]. Finally, we note that while the above only discusses single-bit updates, our construction handles multi-bit updates as well.

2.2.1.4 Puncturable Symmetric Reencryption from Worst-case Lattice Assumptions

Boneh, Lewi, Montgomery, and Raghunathan construct symmetric reencryption schemes from *key homomorphic PRFs* [BLMR13b]. Key homomorphic PRFs have the property that for all x, K_1, and K_2, $\mathsf{PRF}(K_1, x) + \mathsf{PRF}(K_2, x) = \mathsf{PRF}(K_1 + K_2, x)$, where the keys and outputs of the PRF lie in appropriate groups. A secret key for the reencryption scheme is a PRF key. The encryption of a message m with secret key K_1 and randomness r is $\mathsf{CT} = (r, \mathsf{m} + \mathsf{PRF}(K_1, r))$.

A reencryption key for secret keys K_1 and K_2 is their difference: $\mathsf{rk}_{1\to 2} = K_2 - K_1$. The key-homomorphism provides a natural way to reencrypt ciphertexts: $(r, \mathsf{m} + \mathsf{PRF}(K_1, r) + \mathsf{PRF}(\mathsf{rk}_{1\to 2}, r)) = (r, \mathsf{m} + (K_2, r))$ is a ciphertext under K_2. Observe that successful reencryption of a ciphertext with randomness r relies on the ability to compute $\mathsf{PRF}(\mathsf{rk}_{1\to 2}, r)$.

We construct *puncturable* proxy reencryption scheme following the above approach, but instantiated with *constrained* key-homomorphic PRFs [BV15b]. A punctured reencryption key $\mathsf{rk}_{1\to 2}^{\mathsf{CT}^*}$ for a ciphertext CT^* with randomness r^* is the PRF key $K_2 - K_1$ punctured at the input r^*. This key, which can be used to evaluate $\mathsf{PRF}(K_2 - K_1, r)$ for all $r \neq r^*$, enables the reencryption of all ciphertexts except for the ciphertext CT^* with high probability.

We need semantic security of CT^* even given both $\mathsf{rk}_{1\to 2}^{\mathsf{CT}^*}$ and K_2. We reduce to the security of the constrained PRF, which guarantees that $y^* := \mathsf{PRF}(K_2 - K_1, r^*)$ is pseudorandom. The key idea is that (partial information about) y^* can be computed given CT^*, K_2, and (partial information about) the message m.

2.2.2 Construction of URE: *Relock-and-Release*

The main difference between UGC and URE is that UGC only allows for a single evaluation after a sequence of updates, while URE allows for evaluation after every update. As such, the relock-and-eventual-release mechanism discussed above is inadequate for URE. We develop a *relock-and-release* mechanism that performs both relocking and release at every step. Intuitively, relocking allows us to repeatedly apply updates, while the release mechanism allows us to evaluate the updated randomized encoding at every step.

2.2.2.1 Starting Idea: Garbled RAM with Persistent Memory

We implement the relock-and-release mechanism using garbled RAMs with persistent memory [LO13, GHL+14]. In a garbled RAM scheme, it is possible to encode a database D_0 and later issue encodings for RAM programs $M_1, \ldots, M_{\mathsf{q}}$. Each RAM program encoding $\widetilde{M_i}$ updates the database encoding from $\widetilde{D_{i-1}}$ to $\widetilde{D_i}$, and outputs the result of some computation on D_i.

Starting from a garbled RAM scheme and a standard randomized encodings scheme without updates [Yao86], we can construct URE as follows:

- Set the initial garbled RAM database D_0 to the circuit and input pair (C_0, x_0). The initial updatable randomized encoding of (C_0, x_0) is the garbled RAM encoding of D_0 along with an encoding of (C_0, x_0) computed under the standard randomized encoding scheme.

- To compute an encoding $\langle \mathbf{u}_i \rangle$ for an update $\mathbf{u}_i$, compute an garbled RAM encoding $\widetilde{M_i}$ of a machine M_i that has $\mathbf{u}_i$ hardcoded in it. The machine M_i on input $D_{i-1} = (C_{i-1}, x_{i-1})$ first updates the database to $D_i = (C_i, x_i)$, where $(C_i, x_i) \leftarrow \mathsf{Update}(C_{i-1}, x_{i-1}; \mathbf{u}_i)$, and outputs a fresh standard randomized encoding of (C_i, x_i).

Observe that M_i computes a fresh (standard) randomized encoding when the update is applied. To achieve the necessary efficiency guarantee for URE, we need the garbled RAM encoding time of M_i to be independent of its running time. Furthermore, since the output of M_i consists of a fresh randomized encoding, we also need that the time of encode M_i is independent of its output length. Such garbled RAM schemes are called, respectively, *succinct* [BGL+15, CHJV15] and *output-compressing* [AJ15, LPST16b].

2.2.2.2 Garbled RAM meets Delegation of Computation

Our goal is to base URE on polynomial hardness assumptions. Unfortunately, the only known constructions for output-compressing succinct garbled RAM are based on indistinguishability obfuscation ($i\mathcal{O}$), which is often believed to require sub-exponential hardness assumptions. In contrast, non-succinct garbled RAM schemes are known to exist based only on one-way functions. But if we use non-succinct garbled RAM, we need a new idea to achieve our efficiency goals.

We use secret-key functional encryption [SW05, BSW11, O'N10] to introduce an additional level of indirection. Roughly speaking, the initial encoding of $C_0(x_0)$ now corresponds to a non-succinct garbled RAM encoding of $D_0 = (C_0, x_0)$ along with FE functional key for a circuit P. P takes as input an update string $\mathbf{u}_i$ and outputs an encoding $\widetilde{M_i}$ of the machine M_i described before. Encoding the update $\mathbf{u}_i$ now corresponds to an FE encryption of $\mathbf{u}_i$.

For efficiency, we need the secret-key FE scheme to be *compact*: the running time of encrypting a message m is independent of the complexity of the FE function family. If this were not the case, then the encoding time for an update $\mathbf{u}_i$ in the above solution would depend on the size of the circuit C, which in turn depends on the running time and output length of M_i. Secret-key compact FE schemes with polynomial hardness can be built from polynomial hardness assumptions on multilinear maps using results of [GGHZ14] and [BV15a, AJS15a].

2.2.2.3 Challenges in Proving Security

While the above construction approach achieves correctness, it is not clear how to argue security. Note that the circuit P computed by an FE key in the above construction contains the garbling key of the garbled RAM scheme hardwired inside it. This is necessary for it to compute the encodings of M_i.

In order to leverage security of garbled RAM, one might remove the garbling key from the FE function key. However, maintaining functionality would require hardwiring the output of P, either in the FE key, or in the FE ciphertext. We cannot afford to hardwire the output in the ciphertext since that would violate the efficiency requirements of URE. Thus, our only option is to hardwire the output in the FE key. But to support multiple updates, would have to hardwire all the outputs in the FE key (one for each update). Doing so all at once in the proof would require putting a bound on the number of updates.

A better option is to hardwire the outputs one at a time, analogous to many proofs in the $i\mathcal{O}$ literature (see, e.g., [GLSW14, AJ15, BV15b]). Implementing this approach, however, would require puncturing the RAM garbling key, something we do not know how to do.

2.2.2.4 Using Cascaded Garbled Circuits

The key property of garbled RAM we are using is its ability to maintain the updated state in persistent memory. We do not need efficient random access to memory. We can implement the persistent memory more directly using cascaded garbled circuits, which will also help address the security issues discussed above.

Consider a circuit Q_i that has an update string $\mathbf{u}_i$ hardwired in its description. It takes as input (C_{i-1}, x_{i-1}) and outputs two values. The first value is a fresh randomized encoding of $C_i(x_i)$ where $(C_i, x_i) \leftarrow \mathsf{Update}(C_{i-1}, x_{i-1}; \mathbf{u}_i)$, and the second value is a set of wire keys for the string (C_i, x_i) corresponding to a garbling of the circuit Q_{i+1} (that is defined analogously to Q_i). The initial encoding of $C_0(x_0)$ now corresponds to the input wire keys for the string (C_0, x_0) corresponding to a garbling of circuit Q_1 as defined above, as well as an FE key for a function f that takes as input $\mathbf{u}_i$ and outputs a garbling a circuit Q_i. The encoding of an update $\mathbf{u}_i$ is an FE encryption of $\mathbf{u}_i$ as before.

We outline the indistinguishability security proof. Simulation-based security can be argued via a generic transformation following [CIJ$^+$13]. Let C_0^0, C_0^1, and x be the initial circuits and input, and let $(\mathbf{u}_1^0, \mathbf{u}_1^1), \ldots, (\mathbf{u}_q^0, \mathbf{u}_q^1)$ be the updates. There are two chains of updating processes: the 0^{th} chain starts from C_0^0 and 1^{st} chain starting from C_1^1. The i^{th} bead on the 0^{th} (resp., 1^{st}) chain corresponds to update $\mathbf{u}_i^0$ (resp., $\mathbf{u}_i^1$).

The security proof begins with the real experiment with challenge bit 0. Only the

0^{th} chain is active. In the next step, we introduce the 1^{st} chain while keeping the 0^{th} chain active. Namely, the randomized encoding at every step is generated using the 0^{th} chain. The next sequence of hybrids slowly switch the active chain from 0 to 1, one by one. In the i^{th} hybrid in the sequence, the first i randomized encodings are generated using the 1^{st} chain, the remainder using the 0^{th} chain. At the end, we remove the now inactive 0^{th} chain and altogether, recovering the real experiment with challenge bit 1.

The two chains described above are implemented in a sequence of garbled circuits, that we call *cascaded* garbled circuits. Every i^{th} garbled circuit in this sequence produces wire keys for the next garbled circuit. Every garbled circuit in this sequence is a result of ApplyUpd procedure and encapsulates, for some i, the i^{th} beads on both the chains. In order to move from the i^{th} intermediate step to $(i+1)^{th}$ intermediate step, we use the security of garbled circuits. But since these garbled circuits are not given directly, but instead produced by a FE key, we need to make use of security of FE to make this switch work.

2.3 Preliminaries

2.3.1 Circuits, Hardwired Circuits, and Updatable Circuits

A boolean circuit C is a directed acyclic graph of in-degree at most 2. Source vertices are called inputs, sink vertices are called outputs, and all other vertices are called gates. Each gate is associated with a type $\mathsf{gateType} \in \{\vee, \wedge, \neg\}$. The size of a circuit $s = |C|$ is the number of vertices in the graph, and each vertex is labeled by a unique index between 1 and s. A circuit with n inputs and m outputs represents a function $C : \{0,1\}^n \to \{0,1\}^m$. Evaluation of C on an input $x \in \{0,1\}^n$ is performed by assigning to each input vertex one bit of x (in index order), evaluating the circuit gate by gate, and returning the value $C(x) \in \{0,1\}^m$ assigned to the output gates (again, in index order). The depth of a circuit is the length of the longest path from an input to an output.

We associate with a pair (C, x) a corresponding *hardwired circuit* $C[x]$ which is identical to C except with its input vertices fixed to x. $C[x]$ takes no input and and always evaluates to $C(x)$. While not strictly necessary for our results, hardwired circuits are a very useful syntactic sugar. We occasionally abuse notation and use $C[x]$ to refer to both the hardwired circuit and output. Two hardwired circuits $C_0[x_0]$ and $C_1[x_1]$ are equivalent, denoted $C_0[x_0] \equiv C_1[x_1]$ if and only if $C_0(x_0) = C_1(x_1)$ and $|C_0| = |C_1|$.

A family of circuits is a collection $\mathcal{C} = \{\mathcal{C}_\lambda\}_{\lambda \in \mathbb{N}}$, where $\mathcal{C}_\lambda$ is a set of circuits $C : \{0,1\}^{n(\lambda)} \to \{0,1\}^{m(\lambda)}$ of size $s(\lambda)$, for polynomially-bounded functions n, m, and s.[4] We define the corresponding family of hardwired circuits $\mathcal{C}[X] = \{\mathcal{C}[X]_\lambda\}_{\lambda \in \mathbb{N}}$ in the natural way. Throughout this chapter, we will make computational indistinguishability statements of the form $\forall \{C_\lambda \in \mathcal{C}_\lambda\}_{\lambda \in \mathbb{N}}$, $\{F(C_\lambda)\}_{\lambda \in \mathbb{N}} \approx_{\mathsf{c}} \{G(C_\lambda)\}_{\lambda \in \mathbb{N}}$, where $F, G : \mathcal{C} \to \{0,1\}^*$ are some concrete randomized algorithms (possibly with additional arguments). We abuse notation and instead write: $\forall C \in \mathcal{C}_\lambda$, $\{F(C)\}_{\lambda \in \mathbb{N}} \approx_{\mathsf{c}} \{G(C)\}_{\lambda \in \mathbb{N}}$; or even more succinctly, $\forall C \in \mathcal{C}_\lambda$, $F(C) \approx_{\mathsf{c}} G(C)$.[5]

A family of circuits is *updatable* if there is some procedure for transforming a circuit C into another circuit C' as described by an *update* $\mathbf{u} \in \{0,1\}^{\mathrm{poly}(\lambda)}$. The size of $\mathbf{u}$ may be much smaller than $s(\lambda)$. For example, if $\mathbf{u}$ is "change all $\vee$ gates to $\wedge$ and vice-versa," its size is a constant independent of λ. An update for a hardwired circuit $C[x]$ may modify both the underlying circuit C or the input x.

We consider families of circuits that are closed under specific families of updates $\mathcal{U}$. A family of updates is a collection $\mathcal{U} = \{\mathcal{U}_\lambda\}_{\lambda \in \mathbb{N}}$ where $\mathcal{U}_\lambda \subseteq \{0,1\}^{\mathrm{poly}(n)}$ for some polynomial poly. The family $\mathcal{C}$ is closed under $\mathcal{U}$ if applying an update $\mathbf{u} \in \mathcal{U}$ to $C \in \mathcal{C}$ results in another circuit in $\mathcal{C}$.

Definition 2.3.1 (Updatable Circuits). *We say $\mathcal{C}$ is* $(\mathsf{Upd}, \mathcal{U})$-updatable *if for all $C \in \mathcal{C}_\lambda$ and all $\mathbf{u} \in \mathcal{U}_\lambda$, $\mathsf{Upd}(C, \mathbf{u}) \in \mathcal{C}_\lambda$. Likewise, we say that $\mathcal{C}[X]$ is $(\mathsf{Upd}, \mathcal{U})$-updatable if for all $C[x] \in \mathcal{C}[X]_\lambda$ and all $\mathbf{u} \in \mathcal{U}_\lambda$, $\mathsf{Upd}(C[x], \mathbf{u}) \in \mathcal{C}_\lambda$.*

[5]For instance, see Footnote 6.

[5]We assume that the class of all boolean circuits for every fixed size s and n inputs has an efficient binary representation $\mathsf{binary}(C) \in \{0,1\}^{O(s)}$. That is, there is an efficient algorithm that computes $C \mapsto (n, s, \mathsf{binary}(C))$, and its inverse.

This closure property enables us to consider chaining updates in sequence: $C_2[x_2] \leftarrow \mathsf{Upd}(\mathsf{Upd}(C[x], \mathbf{u}_1), \mathbf{u}_2)$.

2.3.2 Randomized Encodings

Randomized encodings, introduced by Ishai and Kushilevitz [IK00], are a way to represent relatively complex computations using simpler functions. We measure the complexity of a function by the depth of a circuit that computes it. A randomized encoding scheme RE consists of a pair of algorithms.

ENCODE: $\widetilde{C(x)} \leftarrow \mathsf{RE.Encode}(1^\lambda, C, x)$ takes as input a circuit C and an input x; it outputs the randomized encoding $\widetilde{C(x)}$.

DECODE: $C(x) \leftarrow \mathsf{RE.Decode}\left(\widetilde{C(x)}\right)$ is a deterministic algorithm. On input a randomized encoding $\widetilde{C(x)}$, it outputs the value $C(x)$.

A randomized encoding scheme must be correct, efficient, and secure.

CORRECTNESS: $\mathsf{RE.Decode}(\mathsf{RE.Encode}(1^\lambda, C, x) = C(x))$ for all C, x.

EFFICIENCY: $\mathsf{RE.Encode}$ is computable by a circuit in NC^1.

SECURITY: There exists a simulator Sim such that for all $C \in \mathcal{C}_\lambda$, $x \in \{0,1\}^{n(\lambda)}$,

$$\mathsf{RE.Encode}(1^\lambda, C, x) \approx_{\mathsf{c}} \mathsf{Sim}(1^\lambda, \phi(C), C(x)).^6$$

2.3.2.1 Garbled Circuits

Garbled circuits are a type of randomized encoding where C and x may be encoded separately using some shared state. We present a modified formalization of garbled circuits from [BHR12]. A garbled circuit scheme $\mathcal{C}$ is a tuple of algorithms $\mathsf{GC} = (\mathsf{GrbCkt}, \mathsf{GrbInp}, \mathsf{EvalGC})$ with the following syntax.

[6] Without the abuse of notation discussed on the previous page, we would instead write: There exists a simulator Sim such that for all $\{C_\lambda \in \mathcal{C}_\lambda\}_\lambda$ and $\{x_\lambda \in \{0,1\}^{n(\lambda)}\}_\lambda$, $\{\mathsf{RE.Encode}(1^\lambda, C_\lambda, x_\lambda)\}_\lambda \approx_{\mathsf{c}} \{\mathsf{Sim}(1^\lambda, \phi(C_\lambda), C_\lambda(x_\lambda))\}_\lambda$.

Circuit Garbling, $\left(\langle C\rangle_{\mathsf{gc}}, \mathsf{st}\right) \leftarrow \mathsf{GrbCkt}(1^\lambda, C)$: On input security parameter 1^λ and a circuit $C \in \mathcal{C}_\lambda$, outputs a garbled circuit $\langle C\rangle_{\mathsf{gc}}$ and state st.

Input Garbling, $\langle x\rangle_{\mathsf{gc}} \leftarrow \mathsf{GrbInp}(\mathsf{st}, x)$: On input state st, input $x \in \{0,1\}^{n(\lambda)}$, outputs an input encoding $\langle x\rangle_{\mathsf{gc}}$.

Evaluation, $\alpha \leftarrow \mathsf{EvalGC}\left(\langle C\rangle_{\mathsf{gc}}, \langle x\rangle_{\mathsf{gc}}\right)$: On input garbled circuit $\langle C\rangle_{\mathsf{gc}}$ and an input encoding $\langle x\rangle_{\mathsf{gc}}$, outputs the decoded value α.

As before, a garbled circuit scheme is correct if $C(x) = \mathsf{EvalGC}\left(\langle C\rangle_{\mathsf{gc}}, \langle x\rangle_{\mathsf{gc}}\right)$ for all $\langle C\rangle_{\mathsf{gc}}$ and $\langle x\rangle_{\mathsf{gc}}$ generated as above. We also require efficiency: that the algorithms GrbCkt and EvalGC be polynomial time in the input length and the security parameter λ, and that the time to generate (and the size of) the garbled input $\langle x\rangle_{\mathsf{gc}}$ is $\mathrm{poly}(\lambda, |x|)$ for some fixed poly. In particular, it must be independent of $|C|$.

Gate-by-gate schemes. In many schemes (including [Yao86]), the garbled circuit $\langle C\rangle_{\mathsf{gc}}$ consists of a collection of *garbled gates* $\langle C\rangle_{\mathsf{gc}}^g$, one for every gate $g \in [|C|]$ of the original circuit. More formally, there exists an algorithm $\mathsf{GrbGate}(\mathsf{st}, g, \mathsf{gateType})$ which outputs the gth garbled gate $\langle C\rangle_{\mathsf{gc}}^g$.

Security. Security is defined with respect to deterministic side-information function ϕ, which captures what information is leaked by the garbled circuit. For example, the side-information function in the original garbling scheme of Yao is ϕ_{topo}, the function that outputs the circuit topology, including size, number of inputs, and number of outputs.[7] One may consider both simulation-based and indistinguishability-based notions of security, but we restrict our attention to the latter. Yao [Yao86, LP09] constructed garbled circuits based on one-way functions satisfying the $\mathsf{PrivIND}_\phi$ security.

[7] Using universal circuits, the circuit topology can be hidden at the cost of a quadratic loss in efficiency. For further discussion on side-information, see [BHR12].

Definition 2.3.2 ($\mathsf{PrivIND}_\phi$). *A garbling scheme* GC *is* $\mathsf{PrivIND}_\phi$*-secure with leakage* ϕ *if for every pair of circuits* $C^0, C^1 \in \mathcal{C}_\lambda$*, and inputs* $x^0, x^1 \in \{0,1\}^{n(\lambda)}$*:*

$$\mathsf{Expt}_0^{\mathsf{GC}}(1^\lambda, C, x) \overset{c}{\approx} \mathsf{Expt}_1^{\mathsf{GC}}(1^\lambda, C, x)$$

where $\mathsf{Expt}_b^{\mathsf{GC}}$ *is defined below.*

$\mathsf{Expt}_b^{\mathsf{GC}}(1^\lambda, C, x)$:

- If $\phi(C^0) \neq \phi(C^1)$ or $C^0(x^0) \neq C^1(x^1)$, abort and return $\perp$.
- Compute the garbled circuit $\langle C^b \rangle_{\mathsf{gc}} \leftarrow \mathsf{GrbCkt}(1^\lambda, C^b)$, and the garbled input $\langle x^b \rangle_{\mathsf{gc}} \leftarrow \mathsf{GrbInp}(\mathsf{st}, x^b)$.
- Output $\left(\langle C^b \rangle_{\mathsf{gc}}, \langle x^b \rangle_{\mathsf{gc}}\right)$.

2.3.3 Private-Key Functional Encryption

A private-key functional encryption (FE) scheme FE over a message space $\mathcal{M} = \{\mathcal{M}_\lambda\}_{\lambda \in \mathbb{N}}$ and a function space $\mathcal{F} = \{\mathcal{F}_\lambda\}_{\lambda \in \mathbb{N}}$ is a tuple ($\mathsf{FE.Setup}$, $\mathsf{FE.KeyGen}$, $\mathsf{FE.Enc}$, $\mathsf{FE.Dec}$) of PPT algorithms with the following properties:

Setup, $\mathsf{FE.Setup}(1^\lambda)$: On input the security parameter in unary, outputs a secret key $\mathsf{FE.MSK}$.

Key Generation, $\mathsf{FE.KeyGen}(\mathsf{FE.MSK}, f)$: On input the secret key $\mathsf{FE.MSK}$ and a function $f \in \mathcal{F}_\lambda$, outputs a functional key $\mathsf{FE.SK}_f$.

Encryption, $\mathsf{FE.Enc}(\mathsf{FE.MSK}, m)$: On input the secret key $\mathsf{FE.MSK}$ and a message $\mathsf{m} \in \mathcal{M}_\lambda$, outputs a ciphertext CT.

Decryption, $\mathsf{FE.Dec}(\mathsf{FE.SK}_f, \mathsf{CT})$: On input a functional key $\mathsf{FE.SK}_f$ and a ciphertext CT, outputs $\mathsf{m} \in \mathcal{M}_\lambda \cup \{\perp\}$.

FE must be correct, compact, and function private. For correctness, we require that there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all sufficiently large $\lambda \in \mathbb{N}$, for every message $\mathsf{m} \in \mathcal{M}_\lambda$, and for every function $f \in \mathcal{F}_\lambda$ it holds that

$$\mathsf{FE.Dec}(\mathsf{FE.KeyGen}(\mathsf{FE.MSK}, f), \mathsf{FE.Enc}(\mathsf{FE.MSK}, \mathsf{m})) = f(\mathsf{m})$$

with probability at least $1 - \mathsf{negl}(\lambda)$, where $\mathsf{FE.MSK} \leftarrow \mathsf{FE.Setup}(1^\lambda)$, and the probability is taken over the random choices of all algorithms.

The FE scheme is compact if the complexity of $\mathsf{FE.Enc}$ is independent of the complexity of the function family. More formally, the running time of $\mathsf{FE.Enc}(\mathsf{FE.MSK}, \mathsf{m})$ must be bounded above by some polynomial $\text{poly}(\lambda, |\mathsf{m}|)$ for all m. In particular, a compact FE scheme allows for generating functional keys for circuits whose size and output-length are not a priori fixed.

The notion of function privacy is modeled as a game between a challenger and an adversary. In this game, the adversary may query pairs of functions, receiving in response a functional key corresponding to one of the functions. As long as the queried functions agree on a set of challenge message pairs, the functional keys must be indistinguishable. We focus on the selective-security setting, where the challenge messages are fixed in advance.

$\mathsf{Expt}^{\mathsf{FE}}_{\mathcal{A}}(\lambda, b \in \{0,1\})$:

- The challenger first executes $\mathsf{FE.Setup}(1^\lambda)$ to obtain $\mathsf{FE.MSK}$.
- Message queries: The adversary submits message pairs $((\mathsf{m}_1^{(0)}, \ldots, \mathsf{m}_q^{(0)}), (\mathsf{m}_1^{(1)}, \ldots, \mathsf{m}_q^{(1)}))$ to the challenger, where q is a polynomial in the security parameter λ. The challenger then sends $(c_1^*, \ldots, c_q^*)$ to $\mathcal{A}$, where $c_i^* \leftarrow \mathsf{FE.Enc}(\mathsf{FE.MSK}, \mathsf{m}_i^{(b)})$.
- Function queries: The adversary then submits polynomially many pairs of functions (f_0, f_1). If $\exists i$ such that $f_0(\mathsf{m}_i^{(0)}) \neq f_1(\mathsf{m}_i^{(1)})$, then the experiment aborts and outputs $\perp$. Otherwise the challenger returns $\mathsf{FE}.sk_{f_b} \leftarrow \mathsf{FE.KeyGen}(\mathsf{FE.MSK}, f_b)$
- $\mathcal{A}$'s output is returned by the experiment.

Definition 2.3.3 (Function-private selectively-secure FE). *A private-key functional encryption scheme* FE *is a* function-private selectively-secure *if for any PPT adversary* $\mathcal{A}$ *there exists a negligible function* $\mathsf{negl}(\lambda)$ *such that for all sufficiently large* $\lambda \in \mathbb{N}$*,*

$$\mathsf{Adv}^{\mathsf{FE}}_{\mathcal{A}}(\lambda) \triangleq \left|\Pr[\mathsf{Expt}^{\mathsf{FE}}_{\mathcal{A}}(1^\lambda, 0) = 1] - \Pr[\mathsf{Expt}^{\mathsf{FE}}_{\mathcal{A}}(1^\lambda, 1) = 1]\right| \leq \mathsf{negl}(\lambda).$$

We require a single-key secret key compact FE scheme in order to construct updatable randomized encodings. Currently, we know how to build this either from concrete assumptions on multilinear maps [GGHZ14] or based on iO [GGH+13b, Wat15b]. Although sub-exponentially secure, compact public-key FE is known to imply iO [AJ15, BV15a], the current approaches don't extend to the secret key setting (see [BV15a]). We emphasize that we only require *polynomially-secure, secret key, selectively-secure, single-key* compact FE for our constructions.

2.4 Defining Updatable Randomized Encodings

2.4.1 Sequential Updating

The semantics of URE depends on whether and how multiple updates interact. Can multiple updates be chained in sequence, or do we only allow a single update to be applied? We call the former option *sequential* updating and the latter *parallel* updating. We focus on the sequential setting and discuss parallel updates only briefly in Section 2.4.2. More complex updating semantics are not considered in this work.

Let $\mathcal{C}$ be an $(\mathsf{Upd}, \mathcal{U})$-updatable family of circuits, and $\mathcal{C}[X]$ be the corresponding updatable hardwired family. An updatable randomized encoding (URE) scheme for $\mathcal{C}$ is a tuple of algorithms $\mathsf{URE} = (\mathsf{Encode}, \mathsf{GenUpd}, \mathsf{ApplyUpd}, \mathsf{Decode})$ with the following syntax, and associated efficiency, correctness, and security properties discussed below.

Encode, $(\langle C[x]\rangle_{\mathsf{ure}}, \mathsf{st}) \leftarrow \mathsf{Encode}\left(1^\lambda, C, x\right)$: On input security parameter λ, circuit $C \in \mathcal{C}_\lambda$, input $x \in \{0,1\}^{n(\lambda)}$, it outputs an randomized encoding $\langle C[x]\rangle_{\mathsf{ure}}$ and state st.

Generating Secure Update, $(\langle \mathbf{u} \rangle_{\mathsf{ure}}, \mathsf{st}') \leftarrow \mathsf{GenUpd}(\mathsf{st}, \mathbf{u})$: On input state st, update $\mathbf{u} \in \mathcal{U}_\lambda$, output the secure update $\langle \mathbf{u} \rangle_{\mathsf{ure}}$ along with the new state st'.

Apply Secure Update, $\langle C'[x'] \rangle_{\mathsf{ure}} \leftarrow \mathsf{ApplyUpd}(\langle C[x] \rangle_{\mathsf{ure}}, \langle \mathbf{u} \rangle_{\mathsf{ure}})$: On input randomized encoding $\langle C[x] \rangle_{\mathsf{ure}}$, secure update $\langle \mathbf{u} \rangle_{\mathsf{ure}}$, output the updated randomized encoding $\langle C'[x'] \rangle_{\mathsf{ure}}$.

Evaluation, $\alpha \leftarrow \mathsf{Decode}(\langle C[x] \rangle_{\mathsf{ure}})$: On input randomized encoding $\langle C[x] \rangle_{\mathsf{ure}}$, output the decoded value α.

2.4.1.1 Correctness

For a hardwired circuit $C_0[x_0]$ from an $(\mathsf{Upd}, \mathcal{U})$-updatable family $\mathcal{C}[X]$ and a sequence of updates $\mathbf{u}_1, \ldots, \mathbf{u}_\mathsf{q}$, let $C_i[x_i]$ be recursively defined as the result of evaluating $\mathsf{Upd}(C_{i-1}[x_{i-1}], \mathbf{u}_i)$.[8] Each $C_i[x_i]$ may be evaluated, outputting $C_i(x_i)$.

Informally, a URE scheme is correct if it allows the $C_i(x_i)$ to be recovered from encodings. For $C_0[x_0]$ and $\mathbf{u}_1, \ldots, \mathbf{u}_\mathsf{q}$ as above, let $(\langle C[x] \rangle_{\mathsf{ure}}, \mathsf{st}_0) \leftarrow \mathsf{Encode}(1^\lambda, C, x)$ and $(\langle \mathbf{u}_i \rangle_{\mathsf{ure}}, \mathsf{st}_i) \leftarrow \mathsf{GenUpd}(\mathsf{st}_{i-1}, \mathbf{u}_i)$. Finally, for every $i \in [\mathsf{q}]$ let $\langle C_i[x_i] \rangle_{\mathsf{ure}} \leftarrow \mathsf{ApplyUpd}(\langle C_{i-1}[x_{i-1}] \rangle_{\mathsf{ure}}, \langle \mathbf{u}_i \rangle_{\mathsf{ure}})$.

Definition 2.4.1 (URE Correctness). URE *is* correct *if for all $\lambda \in \mathbb{N}$, $C_0[x_0] \in \mathcal{C}_\lambda[X]$, all sequences $\mathbf{u}_1, \ldots, \mathbf{u}_\mathsf{q}$ in $\mathcal{U}$, and all $i \in [\mathsf{q}]$:*

$$\mathsf{Decode}\Big(\langle C_i[x_i] \rangle_{\mathsf{ure}}\Big) = C_i(x_i).$$

2.4.1.2 Efficiency

We consider a range of efficiency goals.

Encoding Time: Computing the encoding $\mathsf{Encode}(1^\lambda, C, x)$ should be significantly simpler than evaluating $C(x)$. We require that the depth of the circuit computing Encode be smaller than the depth of C.

[8]By the closure property of Definition 2.3.1, each $C_i[x_i]$ is in the same updatable family, enabling this repeated use of Upd.

Secure Update Generation Time: Computing the encoding $\mathsf{GenUpd}(\mathsf{st}, \mathbf{u})$ of an update should take time poly$(\lambda, |\mathbf{u}|)$ independent of $|C|$. We call such URE schemes *output compact.*

Secure Update Size: The size of an encoded update $|\langle \mathbf{u} \rangle_{\mathsf{ure}}|$ should be poly$(\lambda, |\mathbf{u}|)$, independent of $|C|$. Note that any output compact URE satisfies this property.

State Size: The size of the state st maintained by the authority should be poly(λ), independent of $|C|$ and $|\mathbf{u}|$.

Runtime of Update: The time required to apply encoded updates to a randomized encoding should be polynomially related to the time required to apply the unencoded update to the unencoded circuit. Namely, $\mathsf{ApplyUpd}(\langle C[x] \rangle_{\mathsf{ure}}, \langle \mathbf{u} \rangle_{\mathsf{ure}})$ should take poly$(\lambda, t, |\mathbf{u}|)$ time, where t is the time taken to execute $\mathsf{Upd}(C[x], \mathbf{u})$.

Our constructions achieve most of these efficiency goals. On the negative side, our constructions do not satisfy the 'Runtime of Update' property. On the positive side, our main (indistinguishability-based) construction in Section 2.6 is output compact and achieves the 'Encoding Time' property. In Appendix 2.5.3, we provide a transformation from any output compact URE scheme to one that additionally satisfies the 'State Size' property. This transformation uses non-succinct garbled RAMs, and assumes only one-way functions. Our construction for simulation-based security does not achieve output compactness, but this is inherent [AGVW13, CIJ$^+$13].

2.4.1.3 Security

Our security notions attempt to capture the intuition that an updateable randomized encoding $\langle C_0[x_0] \rangle_{\mathsf{ure}}$ and a sequence of updates $\langle \mathbf{u}_1 \rangle_{\mathsf{ure}}, \ldots, \langle \mathbf{u}_{\mathsf{q}} \rangle_{\mathsf{ure}}$ should reveal only the outputs $C_0(x_0)$, $C_1(x_1)$,$\ldots C_{\mathsf{q}}(x_{\mathsf{q}})$ where C_i, x_i are as in Definition 2.4.1. In addition to hiding the circuits and inputs as in traditional randomized encodings, a URE additionally hides the sequence of updates. We study two different formalizations of this intuition: simulation-based and indistinguishability-based.

While our URE construction is update hiding, we could instead consider a relaxed notion in which updates are partially or wholly revealed. Indeed, this is what we will do in the context of updatable garbled circuits (Section 2.7). In Appendix 2.7.5, we provide a generic transformation from an update-revealing URE scheme to an update-hiding URE scheme, assuming only the existence of one-way functions.

Simulation-based security. We adopt the real world / ideal world paradigm in formalizing simulation-based security. In the real world, the adversary receives encodings of the circuit and update generated honestly as per the description of the scheme. In the ideal world, the adversary is provided simulated encodings and encodings of updates created by a PPT simulator Sim. A sequential updatable RE scheme is secure if no efficient adversary can distinguish real world from the ideal world.

Our definitions and constructions offer only *selective security*, wherein the circuit, input, and updates must be chosen in advance, before any encodings are generated. Even so, the simulator generates the encodings one at a time, using only limited knowledge of the circuits and updates. To begin, Sim gets as input the output of circuit $C_0(x_0)$ and $|C_0|$ (but neither C_0 nor x_0) and produces a simulated randomized encoding. Next, Sim simulates the update encodings in order. On input $|\mathbf{u}_i|$ and the output $C_i(x_i)$, it generates a simulated encoding of the update.

Definition 2.4.2 (SIM-secure Sequential URE). *A sequential URE scheme* URE *for* $(\mathsf{Upd},\mathcal{U})$*-updatable class of circuits* $\mathcal{C}$ *is* SIM secure *if there exists a PPT simulator* Sim *such that for every circuit* $C \in \mathcal{C}_\lambda$*, input* $x \in \{0,1\}^{n(\lambda)}$*, and every sequence of updates* $\mathbf{u}_1,\ldots,\mathbf{u}_\mathsf{q} \in \mathcal{U}_\lambda$*,*

$$\mathsf{IdealExpt}\left(1^\lambda, C, x, \{\mathbf{u}_i\}_{i\in[\mathsf{q}]}\right) \overset{c}{\approx} \mathsf{RealExpt}\left(1^\lambda, C, x, \{\mathbf{u}_i\}_{i\in[\mathsf{q}]}\right).$$

$\mathsf{IdealExpt}(1^\lambda, C, x, \{\mathbf{u}_i\}_{i\in[\mathsf{q}]})$:

- $(\langle C[x]\rangle_{\mathsf{ure}}, \mathsf{st}_0) \leftarrow \mathsf{Sim}(1^\lambda, 1^{|C|}, C(x))$
- $C_0[x_0] :=$ hardwired circuit of (C, x)
- $\forall i \in [\mathsf{q}]$, $C_i[x_i] \leftarrow \mathsf{Upd}(C_{i-1}[x_{i-1}], \mathbf{u}_i)$

- $\forall i \in [\mathsf{q}]$, $(\langle \mathbf{u}_i \rangle_{\mathsf{ure}}, \mathsf{st}_i) \leftarrow \mathsf{Sim}(\mathsf{st}_{i-1}, 1^{|\mathbf{u}_i|}, C_i(x_i))$.
- Output $\left(\langle C[x] \rangle_{\mathsf{ure}}, \langle \mathbf{u}_1 \rangle_{\mathsf{ure}}, \ldots, \langle \mathbf{u}_{\mathsf{q}} \rangle_{\mathsf{ure}} \right)$

$\mathsf{RealExpt}(1^\lambda, C, x, \{\mathbf{u}_i\}_{i \in [\mathsf{q}]})$:

- $(\langle C[x] \rangle_{\mathsf{ure}}, \mathsf{st}_0) \leftarrow \mathsf{Encode}\left(1^\lambda, C, x\right)$
- $\forall i \in [\mathsf{q}]$, $(\langle \mathbf{u} \rangle_{\mathsf{ure}}, \mathsf{st}_i) \leftarrow \mathsf{GenUpd}\left(\mathsf{st}_{i-1}, \mathbf{u}_i\right)$
- Output $\left(\langle C[x] \rangle_{\mathsf{ure}}, \langle \mathbf{u}_1 \rangle_{\mathsf{ure}}, \ldots, \langle \mathbf{u}_{\mathsf{q}} \rangle_{\mathsf{ure}} \right)$

Indistinguishability-based security. Our definition of IND security intends to capture the intuition that a updatable randomized encoding of (C, x) and a sequence of encoded updates $\{\mathbf{u}_i\}$ should be indistinguishable from a encodings of a different circuit, input, and sequence of updates as long as all the intermediate circuit evaluations agree.

Definition 2.4.3 only considers a selectively-secure version of this intuition where the circuits, inputs, and updates are fixed in advanced. For $\beta \in \{0, 1\}$, consider circuits C^β, inputs x^β, and update sequences $\{\mathbf{u}_i^\beta\}_{i \in [\mathsf{q}]}$. As long as all intermediate circuit evaluations are equal for $\beta = 0$ and $\beta = 1$, it should be impossible to distinguish between the corresponding sequences of encodings. Adaptive security, where an adversary may adaptively choose the updates introduces additional challenges not addressed in this work.

Definition 2.4.3 (IND-secure Sequential URE). *A sequential URE scheme* URE *for* $(\mathsf{Upd}, \mathcal{U})$*-updatable class of circuits* $\mathcal{C}$ *is* IND secure *if for every pair of circuits* $C^0, C^1 \in \mathcal{C}_\lambda$*, inputs* $x^0, x^1 \in \{0,1\}^{n(\lambda)}$*, and update sequences* $\{\mathbf{u}_i^0\}_{i \in [\mathsf{q}]}$ *and* $\{\mathbf{u}_i^1\}_{i \in [\mathsf{q}]}$*:*

$$\mathsf{Expt}_0^{\mathsf{SURE}}(1^\lambda, \mathsf{input}) \overset{\mathsf{c}}{\approx} \mathsf{Expt}_1^{\mathsf{SURE}}(1^\lambda, \mathsf{input}),$$

where $\mathsf{Expt}_b^{\mathsf{SURE}}$ *is defined below and* $\mathsf{input} = \left(C^0, C^1, x^0, x^1, \{\mathbf{u}_i^0\}_{i \in [\mathsf{q}]}, \{\mathbf{u}_i^1\}_{i \in [\mathsf{q}]}\right)$.

$\mathsf{Expt}_b^{\mathsf{SURE}}(1^\lambda, C^0, C^1, x^0, x^1, \{\mathbf{u}_i^0\}_{i \in [\mathsf{q}]}, \{\mathbf{u}_i^1\}_{i \in [\mathsf{q}]})$:

- For $\beta \in \{0,1\}$, let $C_0^\beta = C^\beta$ and $x_0^\beta = x^\beta$. For $i \in [\mathsf{q}]$, recursively evaluate $C_i^\beta[x_i^\beta] \leftarrow \mathsf{Upd}(C_{i-1}^\beta[x_{i-1}^\beta], \mathbf{u}_i^\beta)$.
- Check that $C^0(x) = C^1(x)$ and that $\forall i \in [\mathsf{q}]$, $C_i^0(x_i^0) = C_i^1(x_i^1)$. If not, abort and output $\perp$.
- $(\langle C^b[x^b]\rangle_{\mathsf{ure}}, \mathsf{st}_0^b) \leftarrow \mathsf{Encode}\left(1^\lambda, C^b, x^b\right)$.
- $\forall i \in [\mathsf{q}]$, $(\langle \mathbf{u}^b\rangle_{\mathsf{ure}}, \mathsf{st}_i^b) \leftarrow \mathsf{GenUpd}\left(\mathsf{st}_{i-1}^b, \mathbf{u}_i^b\right)$.
- Output $\left(\langle C^b[x^b]\rangle_{\mathsf{ure}}, \langle \mathbf{u}_1^b\rangle_{\mathsf{ure}}, \ldots, \langle \mathbf{u}_{\mathsf{q}}^b\rangle_{\mathsf{ure}}\right)$.

2.4.2 Parallel Updating

Given a randomized encoding $\langle C[x]\rangle_{\mathsf{ure}}$ and multiple patches, parallel updating allows for separately updating the original encoding using each of these patches. That is, given secure updates $\langle \mathbf{u}_1\rangle_{\mathsf{ure}}, \ldots, \langle \mathbf{u}_p\rangle_{\mathsf{ure}}$, we can update $\langle C[x]\rangle_{\mathsf{ure}}$ to obtain the respective updated encodings $\langle C_1[x_1]\rangle_{\mathsf{ure}}, \ldots, \langle C_p[x_p]\rangle_{\mathsf{ure}}$, where for each i, $\langle C_i[x_i]\rangle_{\mathsf{ure}} = \mathsf{ApplyUpd}(\langle C[x]\rangle_{\mathsf{ure}}, \langle \mathbf{u}_i\rangle_{\mathsf{ure}})$. We emphasize that this process does not immediately allow for updating the already updated randomized encoding $\langle C_i[x_i]\rangle_{\mathsf{ure}}$ again. This is in contrast with the sequential updating process where an updated randomized encoding can indeed be updated again.

2.4.2.1 Connection between Parallel URE and Reusable Garbled Circuits

The notions of parallel URE and reusable garbled circuits are closely connected. Specifically, consider an *input-updatable* family of hardwired circuits (defined formally below), in which an update is simply an input x', and $\mathsf{Upd}(C[x], x') = C[x']$. By a simple transformation, the existence of a parallel URE scheme for the hardwired circuit family $\{\mathcal{C}[X]_\lambda\}_{\lambda \in \mathbb{N}}$ is equivalent to the existence of a reusable garbled circuit scheme for $\mathcal{C}$. We present the transformation from a parallel URE scheme to a reusable garbling scheme; the reverse transformation is as straightforward.

Let $\mathcal{C} = \{\mathcal{C}_\lambda\}_{\lambda \in \mathbb{N}}$ be a family of circuits, and $\{\mathcal{C}[X]_\lambda\}_{\lambda \in \mathbb{N}}$ be the corresponding family of hardwired circuits. We $\{\mathcal{C}[X]_\lambda\}_{\lambda \in \mathbb{N}}$ is *input-updatable* if it is $(\mathcal{U}_{\mathsf{inp}}, \mathsf{Upd}_{\mathsf{inp}})$-updatable where $\mathcal{U}_{\mathsf{inp}} = \{0,1\}^*$, and $\mathsf{Upd}_{\mathsf{inp}}$ is as follows.

- $\mathsf{Upd}_{\mathsf{inp}}(C[x], x')$ takes as input a hardwired circuit $C[x] \in \mathcal{C}[X]$ and a update $x' \in \mathcal{U}_{\mathsf{inp}}$ indicating the new input to hardwired. If $|x| > \lambda$, then $\mathsf{Upd}_{\mathsf{inp}}$ outputs $\perp$. Otherwise, output the hardwired circuit $C[x']$.

Observe that in the sequential URE setting, the ability to compose updates implies that to update the input of a hardwired circuit, it suffices to support only bitwise updates, in which only a single bit is changed. In the setting of parallel updating, this is no longer the case.

Suppose $\mathsf{URE} = (\mathsf{Encode}, \mathsf{GenUpd}, \mathsf{ApplyUpd}, \mathsf{Decode})$ is a secure parallel URE scheme for the input-updatable class of hardwired circuits $\{\mathcal{C}[X]_\lambda\}_{\lambda \in \mathbb{N}}$. We construct a reusable garbled circuit scheme for $\mathcal{C}$ as follows:

- $\mathsf{GrbCkt}(1^\lambda, C) = \mathsf{Encode}(1^\lambda, C, 0^\lambda)$. This outputs a state $\mathsf{st}_{\mathsf{gc}} = \mathsf{st}_{\mathsf{ure}}$ and a garbled circuit $\langle C \rangle_{\mathsf{gc}} = \left\langle C[0^\lambda] \right\rangle_{\mathsf{ure}}$.
- $\mathsf{GrbInp}(\mathsf{st}, x) = \mathsf{GenUpd}(\mathsf{st}, x)$. This outputs a garbled input $\langle x \rangle_{\mathsf{gc}} = \langle x \rangle_{\mathsf{ure}}$.
- $\mathsf{EvalGC}(\langle C \rangle_{\mathsf{gc}}, \langle x \rangle_{\mathsf{gc}}) = \mathsf{Decode}\left(\mathsf{ApplyUpd}(\left\langle C[0^\lambda] \right\rangle_{\mathsf{ure}}, \langle x \rangle_{\mathsf{ure}})\right)$.

The security of the URE scheme transfers directly to the security of the resulting reusable garbling scheme.

2.5 Observations on the Defintion

2.5.1 IND to SIM-Security

We show how to transform IND-secure URE into SIM-secure URE generically. In the resulting SIM-secure scheme, the size of the updates depends on the output length $m(\lambda)$ of the circuit being encoded, even if we start with a IND-secure URE scheme where the size of the updates is independent m.

Let $\mathsf{URE}_{\mathrm{IND}}$ be an IND-secure URE scheme. We construct a SIM-secure URE scheme $\mathsf{URE}_{\mathsf{SIM}}$ following the template of de Caro et. al. [CIJ+13] for functional encryption.

$\mathsf{URE}_{\mathsf{SIM}}.\mathsf{Encode}\left(1^\lambda, C, x\right)$: On input security parameter λ, circuit $C \in \mathcal{C}_\lambda$, input $x \in \{0,1\}^\lambda$, it computes the encoding of circuit-input pair (C^*, x^*) with respect to $\mathsf{URE}_{\mathrm{IND}}$. That is, it computes the joint encoding w.r.t IND-secure scheme as follows,

$$\left(\langle C^*[x^*]\rangle_{\mathsf{ure}}^{\mathrm{IND}},\ \mathsf{st}_{\mathrm{IND}}\right) \leftarrow \mathsf{URE}_{\mathrm{IND}}.\mathsf{Encode}\left(1^\lambda, C^*, x^*\right)$$

Here, $x^* = (x, 0, 0)$ and C^* is a circuit that takes input (y, out, b) and outputs $C(y)$ if $b = 0$, else if $b = 1$ it outputs out, else if $b = 2$ it outputs $\perp$.

Output the encoding $\langle C[x]\rangle_{\mathsf{ure}} = \langle C^*[x^*]\rangle_{\mathsf{ure}}^{\mathrm{IND}}$ and state $\mathsf{st} = (\mathsf{st}_{\mathrm{IND}}, 1^m)$, where m is the output length of C.

$\mathsf{URE}_{\mathsf{SIM}}.\mathsf{GenUpd}\,(\mathsf{st}, \mathbf{u})$: On input state $\mathsf{st} = \mathsf{st}_{\mathrm{IND}}$, update $\mathbf{u} \in \mathcal{U}_\lambda$, compute the secure updates for $j \in [0, m+1]$, where m is the output length of C. In the following, we assign $\mathbf{v}_j = \mathbf{u}$ for every $j \in [m]$. We set $\mathbf{v}_0$ as the update that changes the mode b to value 2 and $\mathbf{v}_{m+1}$ as the update that changes the mode b to value 0. Looking ahead, in the proof, $\mathbf{v}_j$ would correspond to the j^{th} output bit of the (updated version of) C. For $j \in [0, m+1]$;

$$\left(\langle \mathbf{v}_j\rangle_{\mathsf{ure}}^{\mathrm{IND}}, \mathsf{st}_{\mathrm{IND}}^{j}\right) \leftarrow \mathsf{URE}_{\mathrm{IND}}.\mathsf{GenUpd}\left(\mathsf{st}_{\mathrm{IND}}^{j-1}, \mathbf{v}_j\right)$$

In the above, $\mathsf{st}_{\mathrm{IND}}^{-1} = \mathsf{st}_{\mathrm{IND}}$ and denote $\mathsf{st}'_{\mathrm{IND}}$ by $\mathsf{st}_{\mathrm{IND}}^{m}$. Output the secure update $\langle \mathbf{u}\rangle_{\mathsf{ure}} = \{\langle \mathbf{v}_j\rangle_{\mathsf{ure}}^{\mathrm{IND}}\}_{j \in [0, m+1]}$ and new state $\mathsf{st}' = \mathsf{st}'_{\mathrm{IND}}$.

$\mathsf{URE}_{\mathsf{SIM}}.\mathsf{ApplyUpd}\,(\langle C[x]\rangle_{\mathsf{ure}}, \langle \mathbf{u}\rangle_{\mathsf{ure}})$: On input randomized encoding $\langle C[x]\rangle_{\mathsf{ure}}$, secure update $\langle \mathbf{u}\rangle_{\mathsf{ure}} = \{\langle \mathbf{v}_j\rangle_{\mathsf{ure}}^{\mathrm{IND}}\}_{j \in [m]}$, compute the following for every $j \in [0, m+1]$, where m denotes the output length of C.

$$\langle C_j[x_j]\rangle_{\mathsf{ure}} \leftarrow \mathsf{URE}_{\mathrm{IND}}.\mathsf{ApplyUpd}\left(\langle C_{j-1}[x_{j-1}]\rangle_{\mathsf{ure}}, \langle \mathbf{v}_j\rangle_{\mathsf{ure}}^{\mathrm{IND}}\right)$$

In the above, $\langle C_{-1}[x_{-1}]\rangle_{\mathsf{ure}} = \langle C[x]\rangle_{\mathsf{ure}}$. Denote $\langle C'[x']\rangle_{\mathsf{ure}} = \langle C_{m+1}[x_{m+1}]\rangle_{\mathsf{ure}}$.

$\mathsf{URE}_{\mathsf{SIM}}.\mathsf{Decode}\,(\langle C[x]\rangle_{\mathsf{ure}})$: Return $\mathsf{URE}_{\mathrm{IND}}.\mathsf{Decode}(\langle C[x]\rangle_{\mathsf{ure}})$.

The correctness of the above scheme follows from the correctness of $\mathsf{URE}_{\mathrm{IND}}$. Suppose we have computed an updatable randomized encoding of (C, x). Applying the update $\mathbf{v}_0$ makes the randomized encoding to output $\perp$. Applying the updates $\mathbf{v}_1, \ldots, \mathbf{v}_m$ in sequence is equivalent to just applying the update $\mathbf{u}$, where $\mathbf{v}_1 = \cdots = \mathbf{v}_m = \mathbf{u}$. Finally applying $\mathbf{v}_{m+1}$ results in a randomized encoding of (C', x'), where (C', x') is the circuit-input pair obtained by updating (C, x) with $\mathbf{u}$.

To see why the above scheme is secure, let's take a simple scenario where the adversary only requests for one update. The adversary receives $\langle C[x] \rangle_{\mathsf{ure}} = \langle C^*[x^*] \rangle_{\mathsf{ure}}$ and update encoding $\langle \mathbf{u} \rangle_{\mathsf{ure}} = (\langle \mathbf{v}_0 \rangle_{\mathsf{ure}}^{\mathrm{IND}}, \ldots, \langle \mathbf{v}_{m+1} \rangle_{\mathsf{ure}}^{\mathrm{IND}})$. If $\langle \mathbf{u} \rangle_{\mathsf{ure}}$ is computed honestly then $\mathbf{v}_j$ is set to be $\mathbf{u}$ for all $j \in [m]$. Now, the IND-security of $\mathsf{URE}_{\mathrm{IND}}$ guarantees the following changes to the initial randomized encoding and the secure update encodings are computationally indistinguishable:

- We modify x^* to be $(\perp, \mathsf{out}, 0)$, where $\mathsf{out} = C(x)$. Recall that x^* was originally $(x, 0, 0)$.

- Let (C', x') be the hardwired circuit obtained by updating (C, x) using $\mathbf{u}$. Let $\mathsf{out}' = C'(x')$. We modify $\mathbf{v}_j$, for $j \in [m]$ to now update the input x^* from $(\perp,\ \mathsf{out}'_1 || \cdots || \mathsf{out}'_{j-1} || *,\ 2)$ to the new input $(\perp,\ \mathsf{out}'_1 || \cdots || \mathsf{out}'_j || *,\ 2)$. The update $\mathbf{v}_0$ is left intact (i.e., it changes the mode b to 2), while the update $\mathbf{v}_{m+1}$ now changes the mode b to value 1.

Note that the circuit C, x and $\mathbf{u}$ are "erased" from the system and hence, the above (modified) encodings can be simulated. The security thus follows.

2.5.2 On the Necessity of 1-Key Secret Key Compact FE

We show that any output-compact updatable randomized encoding scheme tolerating unbounded number of updates implies 1-key compact functional encryption scheme. We build upon a recent beautiful work of Bitansky et al. [BNPW16] to obtain this result.

We first show how to obtain ε-XiO [LPST16b] starting from any output-compact updatable randomized encodings scheme. Once we get XiO, we can apply the transformation of [LPST16b] to obtain 1-key compact public key compact FE. This additionally requires the learning with errors (LWE) assumption. Thus, overall, we conclude that assuming LWE, output-compact updatable randomized encodings imply 1-key public-key compact FE, which also implies 1-key secret-key compact FE.

XiO is an variant of indistinguishability obfuscation with the weaker efficiency requirement that dictates that the size of the obfuscated circuit should be sublinear in the size of the truth table associated with the circuit.

Definition 2.5.1. *(Exponentially-Efficient iO (XiO)) For a constant $\gamma < 1$, a machine* XiO *is a γ-compressing exponentially-efficient indistinguishability obfuscator (XiO) for a circuit class $\{\mathcal{C}_\lambda\}_{\lambda \in \mathbb{N}}$ if it satisfies the functionality and indistinguishability in Definition 1.3.1 and additionally:*

NON-TRIVIAL EFFICIENCY: *For every $\lambda \in \mathbb{N}$, every $C \in \mathcal{C}_\lambda$, we have that $|i\mathcal{O}(\lambda, C)| \leq 2^{n\gamma} \cdot \mathrm{poly}(\lambda, C)$, where n is the input length of C.*

2.5.2.1 Output-Compact URE implies XiO

Our transformation from output-compact updatable randomized encodings to XiO builds upon the recent work of [BNPW16]. Let URE be an output-compact URE scheme. Let c be a constant be such that the size of encoding of (C, x) with respect to URE scheme is $|C|^c \cdot \mathrm{poly}(\lambda)$. We construct XiO scheme using URE.

$\mathsf{XiO.Obf}(1^\lambda, C)$: To obfuscate C, compute a randomized encoding $(\langle G[y]\rangle_{\mathsf{ure}}, \mathsf{st}_1) \leftarrow \mathsf{URE.Encode}(1^\lambda, G, y)$, where y is a string of length $\lceil n(1 - \frac{1}{2c})\rceil$ and initially set $y = 0$. Here, G is defined as follows: G on input y outputs $\{C(i||y)\}_{i \in [2^{\lfloor \frac{n}{2c} \rfloor}]}$. Then for every $i \in \left[2^{\lceil n(1-\frac{1}{2c})\rceil}\right]$, compute $((\langle \mathbf{u}_{i+1}\rangle_{\mathsf{ure}}, \mathsf{st}_{i+1} \leftarrow \mathsf{GenUpd}(\mathsf{st}_i, \mathbf{u}_i)$, where $\mathbf{u}_i$ sets $y = i$ in the hardwired circuit $G[y]$. Output the following:

$$C' = (\langle G[y]\rangle_{\mathsf{ure}}, \{\langle \mathbf{u}_i\rangle_{\mathsf{ure}}\}_{i \in [2^?]})$$

$\mathsf{XiO.Eval}(C', x)$: On input x, first compute the truth table of the circuit obfuscated in C' as follows:

- Set $\langle G_0[y_0] \rangle_{\mathsf{ure}} = \langle G[y] \rangle_{\mathsf{ure}}$.
- For every $i \in \left[2^{\lceil n(1-\frac{1}{2c}) \rceil} - 1\right]$, compute $\langle G_{i+1}[y_{i+1}] \rangle_{\mathsf{ure}} \leftarrow \mathsf{ApplyUpd}(\langle G_i[y_i] \rangle_{\mathsf{ure}}, \langle \mathbf{u}_i \rangle_{\mathsf{ure}})$.
- For every $i \in \left[2^{\lceil n(1-\frac{1}{2c} \rceil}\right]$, evaluate $\langle G_i[y_i] \rangle_{\mathsf{ure}}$ to obtain the truth table of C.
- Let the output of the truth table on input x be α.

Output α.

The correctness of the underlying URE scheme implies the correctness of the above XiO scheme. Given any two functionally equivalent circuits C_0 and C_1, we can use the security of the updatable randomized encodings scheme to argue that the obfuscations of C_0 and C_1 are computationally indistinguishable.

We remark about the efficiency of the XiO scheme.

Size of obfuscated circuit C'. We first calculate the number of updates issued as part of the obfuscated circuit—there are $2^{\lceil n(1-\frac{1}{2c}) \rceil}$ ciphertexts with each one of them of size a fixed polynomial in the security parameter. The size of the randomized encoding of (G, y) is $(2^{\lfloor \frac{n}{2c} \rfloor})^c \cdot \mathrm{poly}(\lambda) = (2^{\lfloor \frac{n}{2} \rfloor}) \cdot \mathrm{poly}(\lambda)$. Thus, total size of the obfuscated circuit is $\max\{2^{\lceil n(1-\frac{1}{2c}) \rceil} \cdot \mathrm{poly}(\lambda),\ (2^{\lfloor \frac{n}{2} \rfloor}) \cdot \mathrm{poly}(\lambda)\} \leq 2^{n\beta} \cdot \mathrm{poly}(\lambda)$ for some $\beta < 1$.

2.5.3 Reducing the State of Authority

There is a generic approach to reduce the state size of the authority (i.e., the encoder in the definitions of URE and UGC) using garbled RAMs with persistent memory [GHL+14, GLOS15]. Interestingly, our approach works even if the garbled RAM scheme is *non-succinct*, i.e., if the size of the program encoding is dependent on the computation time. Since the existence of such garbled RAM schemes can be based on one-way functions, we only need to assume the existence of one-way functions for our transformation.

We remark that the idea of using garbled RAMs to reduce the state size of the encoder was observed in [AJS15b]. However, their work crucially use a strong notion of succinct garbled RAMs whose existence is known only from indistinguishability obfuscation. In contrast, here, we only rely on non-succinct garbled RAM.

Theorem 2.5.2. *Assuming that one-way functions exist, there exists an efficient transformation that transforms any URE (resp., UGC) scheme* URE *with efficient update generation into a new URE (resp., UGC) scheme with small states.*

The main idea behind proving the above theorem is that the authority delegates the job of computing the secure update to the client. This delegation process is carried out by initially garbling the circuit and input pair using the garbled database encoding algorithm. This will be shipped as part of the initial randomized encoding. That is the randomized encoding of (C, x) w.r.t URE^* comprises of the garbled database encoding of (C, x) and randomized encoding of (C, x) w.r.t URE. During the update phase, the update generation algorithm of URE^* essentially encodes the update generation algorithm of URE using the garbled program encoding algorithm. On the client's end, the evaluation of the program encoding is done on the database to obtain the secure update computed w.r.t the old scheme URE. Using this secure update, the randomized encoding of (C, x) w.r.t URE is updated. The database encoding is also correspondingly updated (this is done as part of garbled evaluation algorithm) to correspond to the updated circuit and input pair.

We refer the reader to [AJS15b] for further details.

2.6 Constructing URE

In this section, we present our main constructions of updatable randomized encodings.

2.6.1 Output compact, IND secure URE from FE

Our goal is to construct an updatable randomized encoding scheme, $\mathsf{URE} = (\mathsf{Encode}, \mathsf{GenUpd}, \mathsf{ApplyUpd}, \mathsf{Decode})$ for $\mathcal{C}$. The main tools we use in our construction are the following. We refer the reader to the preliminaries for the definitions of these primitives.

- Randomized encoding scheme, $\mathsf{RE} = (\mathsf{RE.Enc}, \mathsf{RE.Dec})$ for the same class of circuits $\mathcal{C}$.

- Functional encryption (FE) $\mathsf{FE} = (\mathsf{FE.Setup}, \mathsf{FE.KeyGen}, \mathsf{FE.Enc}, \mathsf{FE.Dec})$ that is compact, function private, 1-collusion resistant, secret key, and selectively secure.

- Garbling Scheme for circuits $\mathsf{GC} = (\mathsf{GrbCkt}, \mathsf{GrbInp}, \mathsf{EvalGC})$.

We assume, without loss of generality, that all randomized algorithms require only λ-many random bits. We use the above tools to design the algorithms of URE as given below.

The updatable randomized encoding of $C[x]$ will consist of a (standard) randomized encoding $C[x]$ and some additional information necessary to carry out the updating process. This additional information consists of (i) a garbled input encoding of (C, x) with respect to GC, and (ii) a FE secret key for a special function. The special function takes as input an update $\mathbf{u}$ and outputs a garbled circuit mapping $C[x]$ to (a) a randomized encoding $C'[x']$ and (b) a new garbled circuit input encodings of $C', x')$, where $C'[x'] \leftarrow \mathsf{Upd}(C[x], \mathbf{u})$. Henceforth, we denote by $s = |C[x]|$ the size of the hardwired circuit $C[x]$.

$\mathsf{Encode}\left(1^\lambda, C, x\right)$:

1. Execute the setup of FE, $\mathsf{FE.MSK} \leftarrow \mathsf{FE.Setup}(1^\lambda)$.

2. Compute a functional key $\mathsf{FE.SK}_{\mathsf{RRGarbler}} \leftarrow \mathsf{FE.KeyGen}(\mathsf{FE.MSK}, \mathsf{RRGarbler})$, where $\mathsf{RRGarbler}$ is as defined in Figure 2-2.

<u>$\mathsf{RelockRelease}_{i+1}$</u>

Input: $C_i^0[x_i^0]$, $C_i^1[x_i^1]$
Hard-coded values: $\mathbf{u}_{i+1}^0$, $\mathbf{u}_{i+1}^1$, $r_{\mathsf{gc},i+1}$, $r_{\mathsf{re},i+1}$, and mode

- Update both the hardwired circuits $C_i^b[x_i^b]$ using $\mathbf{u}_{i+1}^b$:

$$C_{i+1}^b[x_{i+1}^b] \leftarrow \mathsf{Upd}(C_i^b[x_i^b], \mathbf{u}_{i+1}^b)$$

- Encode the updated hardwired circuit $C_{i+1}^{\mathsf{mode}}[x_{i+1}^{\mathsf{mode}}]$:

$$\langle C_{i+1}^{\mathsf{mode}}[x_{i+1}^{\mathsf{mode}}]\rangle_{\mathsf{re}} \leftarrow \mathsf{RE.Enc}\Big(C_{i+1}^{\mathsf{mode}}[x_{i+1}^{\mathsf{mode}}];\ r_{\mathsf{re},i+1}\Big)$$

- Compute the randomized encoding of the input $(C_{i+1}^0[x_{i+1}^0],\ C_{i+1}^1[x_{i+1}^1])$:

$$\langle C_{i+1}^0[x_{i+1}^0],\ C_{i+1}^1[x_{i+1}^1]\rangle_{\mathsf{gc}} \leftarrow \mathsf{GrbInp}\Big(\ (C_{i+1}^0[x_{i+1}^0], C_{i+1}^1[x_{i+1}^1])\ ;\ r_{\mathsf{gc},i+1}\Big)$$

- Output $\Big(\langle C_{i+1}^{\mathsf{mode}}[x_{i+1}^{\mathsf{mode}}]\rangle_{\mathsf{re}},\ \langle C_{i+1}^0[x_{i+1}^0], C_{i+1}^1[x_{i+1}^1]\rangle_{\mathsf{gc}}\Big)$

Figure 2-1

<u>$\mathsf{RRGarbler}$</u>

Input: $(\mathbf{u}_{i+1}^0,\ \mathbf{u}_{i+1}^1,\ r_{\mathsf{gc},i},\ r_{\mathsf{gc},i+1},\ r_{\mathsf{re},i+1},\ \mathsf{mode})$

Compute the garbled circuit encoding of $\mathsf{RelockRelease}_{i+1}$*, which is defined in Figure 2-1:*

$$\langle \mathsf{RelockRelease}_{i+1}\rangle_{\mathsf{gc}} \leftarrow \mathsf{GrbCkt}\Big(\mathsf{RelockRelease}_{i+1};\ r_{\mathsf{gc},i}\Big)$$

Output $\langle \mathsf{RelockRelease}_{i+i}\rangle_{\mathsf{gc}}$.

Figure 2-2

3. Generate a randomized encoding of input (C, x). That is, evaluate $\langle C[x]\rangle_{\mathsf{re}} \leftarrow \mathsf{RE.Enc}(1^\lambda, C, x)$.

4. Let $s = |C[x]|$. Generate a garbled circuit input encoding of $(C[x], \perp)$ using randomness r_{gc}. That is, evaluate $\langle C[x], \perp\rangle_{\mathsf{gc}} \leftarrow \mathsf{GrbInp}(C[x], \perp; r_{\mathsf{gc}})$. Here we view $(C[x], \perp)$ as an *input* to the circuit $\mathsf{RelockRelease}$.

5. Output the state $\mathsf{st} = (\mathsf{FE.MSK},\ r_{\mathsf{gc}})$ and the updatable randomized encoding

$$\langle C[x] \rangle_{\mathsf{ure}} = \Big(\ \mathsf{FE.SK}_{\mathsf{RRGarbler}},\ \langle C[x] \rangle_{\mathsf{re}},\ \langle C[x], \perp \rangle_{\mathsf{gc}}\ \Big).$$

$\mathsf{GenUpd}\,(\mathsf{st}_i,\ \mathbf{u}_{i+1})$:

1. Let $\mathsf{st}_i = (\mathsf{FE.MSK}, r_{\mathsf{gc},i})$.
2. Let $\mathsf{mode} = 0$.
3. Sample random coins $r_{\mathsf{re},i+1}$ and $r_{\mathsf{gc},i+1}$.
4. Generate the FE ciphertext,

$$\mathsf{CT}_{i+1} \leftarrow \mathsf{FE.Enc}\,(\mathsf{FE.MSK},\ (\mathbf{u}_{i+1},\ \perp,\ r_{\mathsf{gc},i},\ r_{\mathsf{gc},i+1},\ r_{\mathsf{re},i+1},\ \mathsf{mode}))$$

5. Set the new state $\mathsf{st}_{i+1} = (\mathsf{FE.MSK},\ r_{\mathsf{gc},i+1})$.
6. Output $\langle \mathbf{u}_{i+1} \rangle_{\mathsf{ure}} = \mathsf{CT}_{i+1}$ and st_{i+1}.

$\mathsf{ApplyUpd}\,(\langle C_i[x_i] \rangle_{\mathsf{ure}}, \langle \mathbf{u}_{i+1} \rangle_{\mathsf{ure}})$: On input circuit encoding $\langle C_i[x_i] \rangle_{\mathsf{ure}}$

1. Let $\langle \mathbf{u}_{i+1} \rangle_{\mathsf{ure}} = \mathsf{CT}_{i+1}$. Parse the circuit encoding as:

$$\langle C_i[x_i] \rangle_{\mathsf{ure}} = \Big(\ \mathsf{FE.SK}_{\mathsf{RRGarbler}},\ \langle C_i[x_i] \rangle_{\mathsf{re}},\ \langle C_i[x_i], \perp \rangle_{\mathsf{gc}}\ \Big)$$

2. Execute the FE decryption, $\mathsf{FE.Dec}(\mathsf{FE.SK}_{\mathsf{RRGarbler}}, \mathsf{CT}_{i+1})$ to obtain:

$$\langle \mathsf{RelockRelease}_{i+1} \rangle_{\mathsf{gc}}.$$

3. Execute the decode algorithm of the garbling scheme,

$$(\langle C_{i+1}[x_{i+1}] \rangle_{\mathsf{re}}, \langle C_{i+1}[x_{i+1}], \perp \rangle_{\mathsf{gc}}) \leftarrow \mathsf{EvalGC}(\langle \mathsf{RelockRelease}_{i+1} \rangle_{\mathsf{gc}}, \langle C_i[x_i] \rangle_{\mathsf{gc}})$$

That is, the decode algorithm outputs the randomized encoding of updated hardwired circuit $C_{i+1}[x_{i+1}]$ and also wire keys of $(C_{i+1}[x_{i+1}], \perp)$ that will be input to the next level garbled circuit.

4. Output $\langle C_{i+1}[x_{i+1}]\rangle_{\mathsf{ure}} = \left(\mathsf{FE.SK}_{\mathsf{RRGarbler}}, \langle C_{i+1}[x_{i+1}]\rangle_{\mathsf{re}}, \langle C_{i+1}[x_{i+1}], \perp\rangle_{\mathsf{gc}}\right)$.

$\mathsf{Decode}\left(\langle C_i[x_i]\rangle_{\mathsf{ure}}\right)$: Parse the input as $\left(\mathsf{FE.SK}_{\mathsf{RRGarbler}}, \langle C_i[x_i]\rangle_{\mathsf{re}}, \langle C_i[x_i], \perp\rangle_{\mathsf{gc}}\right)$. Decode the term $\langle C_i[x_i]\rangle_{\mathsf{re}}$ by executing $\mathsf{RE.Dec}(\langle C_i[x_i]\rangle_{\mathsf{re}})$ to obtain α. Output α.

2.6.1.1 Correctness

Following the notation of Definition 2.4.1, we need to show that for all $i \in [\mathsf{q}]$ $\mathsf{Decode}\left(\langle C_i[x_i]\rangle_{\mathsf{ure}}\right) = C_i(x_i)$. We first define the following notation: We say that $\langle G\rangle_{\mathsf{gc}}$ is a valid garbled circuit of G if there exists randomness r such that $\langle G\rangle_{\mathsf{gc}} \leftarrow \mathsf{GrbCkt}(G;\ r)$. Further, we say that $\langle z\rangle_{\mathsf{gc}}$ is a valid garbled input encoding of z if there is randomness r' such that $\langle z\rangle_{\mathsf{gc}} \leftarrow \mathsf{GrbInp}(z; r')$.

By the correctness of FE, $\langle \mathsf{RelockRelease}_{i+1}\rangle_{\mathsf{gc}} \leftarrow \mathsf{FE.Dec}(\mathsf{FE.SK}_{\mathsf{RRGarbler}}, \mathsf{CT}_{i+1})$ is a valid garbled circuit of $\mathsf{RelockRelease}_{i+1}$ for every $i \in [\mathsf{q}]$. The following claim implies that the output of $\mathsf{Decode}\left(\langle C_i[x_i]\rangle_{\mathsf{ure}}\right)$ is $C_i(x_i)$, completing the proof of correctness.

Claim 2.6.1. *The output of evaluation of garbling scheme,*

$$\left(\langle C_{i+1}[x_{i+1}]\rangle_{\mathsf{re}}, \langle C_{i+1}[x_{i+1}], \perp\rangle_{\mathsf{gc}}\right) \leftarrow \mathsf{EvalGC}\left(\langle \mathsf{RelockRelease}_{i+1}\rangle_{\mathsf{gc}}, \langle C_i[x_i]\rangle_{\mathsf{gc}}\right)$$

yields a randomized encoding of $C_{i+1}[x_{i+1}]$ and a valid garbled input encoding $\langle C_{i+1}[x_{i+1}], \perp\rangle_{\mathsf{gc}}$ of $C_{i+1}[x_{i+1}]$.

Proof. We prove this by induction. Initially, the user is given a valid input encoding of $(C_0[x_0], \perp)$, i.e., $\langle C_0[x_0], \perp\rangle_{\mathsf{gc}}$. As observed above, the output of $\mathsf{FE.Dec}(\mathsf{FE.SK}_{\mathsf{RRGarbler}}, \mathsf{CT}_1)$ is a valid garbled circuit $\langle \mathsf{RelockRelease}_1\rangle_{\mathsf{gc}}$ of $\mathsf{RelockRelease}_1$ (this circuit corresponds to the first update). From the correctness of garbling schemes, it follows that the evaluation of $\langle \mathsf{RelockRelease}_1\rangle_{\mathsf{gc}}$ on the input encoding $\langle C_0[x_0], \perp\rangle_{\mathsf{gc}}$ is a valid garbled input encoding $\langle C_1[x_1], \perp\rangle_{\mathsf{gc}}$ and a randomized encoding of $C_1[x_1]$. This proves the base case.

Suppose the statement is true for some $i \in [\mathsf{q}]$. $\mathsf{FE.Dec}(\mathsf{FE.SK}_{\mathsf{RRGarbler}}, \mathsf{CT}_{i+1})$ is a valid garbled circuit $\langle \mathsf{RelockRelease}_{i+1} \rangle_{\mathsf{gc}}$ of $\mathsf{RelockRelease}_{i+1}$. From the correctness of garbling schemes, it follows that the output of evaluation of $\langle \mathsf{RelockRelease}_{i+1} \rangle_{\mathsf{gc}}$ on the input encoding $\langle C_i[x_i], \perp \rangle_{\mathsf{gc}})$ is a valid garbled input encoding $\langle C_{i+1}[x_{i+1}], \perp \rangle_{\mathsf{gc}}$ and a randomized encoding of $C_{i+1}[x_{i+1}]$. This proves the claim. □

2.6.1.2 Efficiency

The above scheme has state size that is a fixed polynomial in λ. It is also output compact. That is, the time to generate a secure update $\mathbf{u}$ is polynomial in $(\lambda, |\mathbf{u}|)$. This follows from the compactness of the underlying FE scheme, which guarantees that the running time of an encryption of m is a fixed polynomial in $(|m|, \lambda)$. This also implies that the size of the secure update is a fixed polynomial in $(\lambda, |\mathbf{u}|)$.

The algorithm Encode can be implemented in NC^1 if we additionally assume: (i) the FE scheme we employ has key generation in NC^1 and, (ii) RE has an encode algorithm in NC^1 and, (iii) GC has garbled input encoding algorithm in NC^1. We can instantiate (ii) and (iii) under standard cryptographic assumptions such as decisional Diffie-Helman and learning with errors. We can base (i) on the existence of a function-private *collusion-resistant* secret key FE [AJ15, BV15a, AJS15a].

2.6.2 IND Security

We prove the security of URE with respect to indistinguishability-based security definition. There are two "chains" of update. The 0^{th} chain starts with circuit C_0^0 and input x_0^0, and iteratively updates according to $\{\mathbf{u}_i^0\}$. The 1^{st} chain is analogous.

The security proof proceeds by switching from one chain to the other, step by step. In each hybrid and at every step, there will only be one "active" chain. We start with the real experiment where challenge bit 0 is used, where only the 0^{th} chain is present in the experiment. In the next step, we introduce the 1^{st} chain, along with the already present 0^{th}, into the experiment. However even in this step, 0^{th} chain is still active—that is, generating the randomized encoding at every step is performed using the 0^{th} chain. In the next q hybrids, we slowly switch from 0^{th} chain being activated to 1^{st} chain being activated. In the i^{th} intermediate step, the first i beads on 1^{st} chain are active and on the 0^{th} chain, all except the first i beads are active—this means that the first i updated randomized encodings are computed using the 1^{st} chain and the rest of them are computed using 0^{th} chain. At the end of this sequence of hybrids, we have the 1^{st} chain to be active and 0^{th} chain to be completely inactive, but present. Finally, we can remove the 0^{th} chain altogether, completing the proof.

The two chains described above are implemented in a sequence of garbled circuits, that we call *cascaded* garbled circuits. That is, every i^{th} garbled circuit in this sequence produces wire keys for the next garbled circuit. Every garbled circuit in this sequence is a result of ApplyUpd procedure and encapsulates, for some i, the i^{th} beads on both the chains. In order to move from the i^{th} intermediate step to $(i+1)^{th}$ intermediate step, we use the security of garbled circuits. But since these garbled circuits are not given directly, but instead produced by a FE key, we need to make use of security of FE to make this switch work. With this high level proof overview, we present the formal proof below.

Before we present the hybrids, we first introduce two lemmas relevant to the security of FE and garbled circuits that will be useful later.

2.6.2.1 FE Distributional Lemma

In the security proof, we will make frequent use of the following lemma (which is implicit in many previous works). In the context of secret key, function private, semantically secure functional encryption, the lemma gives us a way to switch from a secret key of some function f_0 and an encryption of a randomized message $M^{(0)}$ to a secret key of a different function f_1 and encryption of a different randomized message $M^{(1)}$, even when $f_0(M^{(0)})$ may not equal $f_1(M^{(1)})$. Whereas the security definition of (function private) functional encryption only makes explicit guarantees, we leverage the indistinguishability of the distributions $f_0(M^{(0)})$ and $f_1(M^{(1)})$, where the randomness is taken over the choice of the message, to switch from one to the other. Indistinguishability holds even in the presence of auxiliary information about

$M^{(0)}$ and $M^{(1)}$ and additional FE-ciphertexts for (randomized) messages $M_2, \ldots, M_{\mathsf{q}}$.

Definition 2.6.2 (Programmable Functions). *A function family $\mathcal{F}$ is said to be* programmable *if for all $f \in \mathcal{F}_\lambda$ and for all y in the co-domain of f, $f^y \in \mathcal{F}_\lambda$ where*

$$f^y(m) = \begin{cases} y & \text{if } m = \perp \\ f(m) & \text{otherwise.} \end{cases}$$

Lemma 2.6.3 (FE Distributional Lemma). *Let* $\mathsf{FE} = (\mathsf{FE.Setup}, \mathsf{FE.KeyGen}, \mathsf{FE.Enc}, \mathsf{FE.Dec})$ *be a 1-collusion, function-private, selectively secure functional encryption scheme over a programmable function space $\mathcal{F} = \{\mathcal{F}_\lambda\}_{\lambda \in \mathbb{N}}$ and a message space $\mathcal{M} = \{\mathcal{M}_\lambda\}_{\lambda \in \mathbb{N}}$. For each $\lambda \in \mathbb{N}$:*

- *Let $M_1^{(0)}$, $M_1^{(1)}$, and $M_2, \ldots, M_{\mathsf{q}}$ be random variables over $\mathcal{M}_\lambda$ and* aux_0 *and* aux_1 *be random variables over $\{0,1\}^*$, all efficiently sampleable. These random variables need not be independent.*

- *Let $(f_0, f_1) \in \mathcal{F}_\lambda$ be such that $\Pr_{M_i}[f_0(M_i) = f_1(M_i)] = 1$ for all $i \geq 2$.*

Suppose that $\mathcal{M}_\lambda$ contains a special message $\perp$ that is not in the support of any of the random variables M_i[9]*. Then it holds that: If*

$$\left(f_0(M_1^{(0)}), \mathsf{aux}_0\right)_\lambda \overset{c}{\approx} \left(f_1(M_1^{(1)}), \mathsf{aux}_1\right)_\lambda$$

then:

$$\begin{aligned} &\left(\mathsf{FE.SK}_0, \mathsf{FE.CT}_1^{(0)}, \mathsf{FE.CT}_2, \ldots, \mathsf{FE.CT}_{\mathsf{q}}, \mathsf{aux}_0\right)_\lambda \\ \overset{c}{\approx} &\left(\mathsf{FE.SK}_1, \mathsf{FE.CT}_1^{(1)}, \mathsf{FE.CT}_2, \ldots, \mathsf{FE.CT}_{\mathsf{q}}, \mathsf{aux}_1\right)_\lambda \end{aligned}$$

[9]This requirement along with requirement that the scheme be defined for programmable functions will be used in the proof of the lemma. Existing work in functional encryption (and obfuscation) that use similar arguments often make explicit use of "padding"—arbitrarily increasing the description size of the underlying function to allow for hardcoding. Exactly this type of padding construction can be used to imbue FE with this "closure under hardcoding" property, but we find it conceptually simpler to state the requirement explicitly.

where the probabilities are taken over the choice of $M_1^{(0)}$, $M_1^{(1)}$, $M_2, \ldots, M_{\mathsf{q}}$, *and the following probability experiment:*

$$\begin{aligned}
&\mathsf{FE.MSK} \leftarrow \mathsf{FE.Setup}(1^\lambda)\\
&\mathsf{FE.SK}_b \leftarrow \mathsf{FE.KeyGen}(\mathsf{FE.MSK}, f_b) \quad \forall b \in \{0,1\}\\
&\mathsf{FE.CT}_1^{(b)} \leftarrow \mathsf{FE.Enc}(\mathsf{FE.MSK}, M_1^{(b)}) \quad \forall b \in \{0,1\}\\
&\mathsf{FE.CT}_i \leftarrow \mathsf{FE.Enc}(\mathsf{FE.MSK}, M_i) \quad \forall i \in [2, \mathsf{q}]
\end{aligned}$$

Proof. The proof of the lemma proceeds by first hardwiring the output of the function $f_0(M_1^{(0)})$ in the functional key $\mathsf{FE.SK}_0$. Once this is done, the message in $\mathsf{FE.CT}_0$ can be switched to $\perp$ since this does not change the output of the function. After this, the hardwiring can be switched from $f_0(M_1^{(0)})$ to $f_1(M_1^{(1)})$ (along with auxiliary information). An analogous sequence of hybrids recovers the real distribution for $b = 1$. The hybrids below describe these intermediate distributions.

Hybrid $\mathbf{H}_0$: $\left(\mathsf{FE.SK}_0, \mathsf{FE.CT}_1^{(0)}, \mathsf{FE.CT}_2, \ldots, \mathsf{FE.CT}_{\mathsf{q}}, \mathsf{aux}_0\right)$, the real distribution with $b = 0$.

Hybrid $\mathbf{H}_1$: $\left(\mathsf{FE.SK}_0^{Y_0}, \mathsf{FE.CT}_1^{(0)}, \mathsf{FE.CT}_2, \ldots, \mathsf{FE.CT}_{\mathsf{q}}, \mathsf{aux}_0\right)$ where $Y_0 = f_0(M_1^{(0)})$ and $\mathsf{FE.SK}_0^{Y_0} \leftarrow \mathsf{FE.KeyGen}(\mathsf{FE.MSK}, f_0^{Y_0})$. Indistinguishability from $\mathbf{H}_0$ follows from the function-privacy of FE, because the functionalities agree on all encrypted inputs.

Hybrid $\mathbf{H}_2$: $\left(\mathsf{FE.SK}_0^{Y_0}, \mathsf{FE.CT}_1^{(\perp)}, \mathsf{FE.CT}_2, \ldots, \mathsf{FE.CT}_{\mathsf{q}}, \mathsf{aux}_0\right)$ where $\mathsf{FE.CT}_1^{(\perp)} \leftarrow \mathsf{FE.Enc}($ $\mathsf{FE.MSK}, \perp)$. Indistinguishability follows from the selective security of FE, as $f_0^{M_1^{(0)}}$ agrees on $M_1^{(0)}$ and $\perp$.

Hybrid $\mathbf{H}_3$: $\left(\mathsf{FE.SK}_0^{Y_1}, \mathsf{FE.CT}_1^{(\perp)}, \mathsf{FE.CT}_2, \ldots, \mathsf{FE.CT}_{\mathsf{q}}, \mathsf{aux}_1\right)$ where $Y_1 = f_1(M_1^{(1)})$ and $\mathsf{FE.SK}_0^{Y_1} \leftarrow \mathsf{FE.KeyGen}(\mathsf{FE.MSK}, f_0^{Y_1})$. This follows from the indistinguishability of (Y_0, aux_0) and (Y_1, aux_1).

Hybrid $\mathbf{H}_4$: $\left(\mathsf{FE.SK}_1^{Y_1}, \mathsf{FE.CT}_1^{(\perp)}, \mathsf{FE.CT}_2, \ldots, \mathsf{FE.CT}_{\mathsf{q}}, \mathsf{aux}_1\right)$ where $\mathsf{FE.SK}_1^{Y_1} \leftarrow \mathsf{FE.KeyGen}(\mathsf{FE.MSK}, f_1^{Y_1})$. Indistinguishability from $\mathbf{H}_3$ follows from the function-privacy of FE, because the functionalities agree on all encrypted inputs.

HYBRID $\mathbf{H}_5$: $\left(\mathsf{FE.SK}_1, \mathsf{FE.CT}_1^{(1)}, \mathsf{FE.CT}_2, \ldots, \mathsf{FE.CT}_\mathsf{q}, \mathsf{aux}_1\right)$ by an argument mirroring the indistinguishability of $\mathbf{H}_0$ and $\mathbf{H}_2$, the real distribution with $b = 1$. □

2.6.2.2 Cascaded Garbled Circuits

We start by explaining the notion of *cascaded circuits.* It is a sequence of circuits, initialized with an input, where the i^{th} circuit in the sequence produces input to the $(i+1)^{th}$ circuit. Every circuit also produces additional output that will not be part of the input to the next circuit in the sequence. We define this notion formally below.

Definition 2.6.4 (Cascaded Circuits). *Let $C_1, \ldots, C_\mathsf{q}$ be circuits mapping $\{0,1\}^n$ to $\{0,1\}^m$ for some $m > n$. We parse the output as $C_i(x_i) = x_{i+1} \| y_i$ where $x_{i+1} \in \{0,1\}^n$ and $y_i \in \{0,1\}^{m-n}$. We say that $(x_1, C_1, \ldots, C_\mathsf{q})$ are* cascaded circuits *if they are evaluated as follows:*

$$\begin{aligned} x_2 \| y_1 &\leftarrow C_1(x_1) \\ x_3 \| y_2 &\leftarrow C_2(x_2) \\ &\vdots \\ x_{\mathsf{q}+1} \| y_\mathsf{q} &\leftarrow C_\mathsf{q}(x_\mathsf{q}) \end{aligned}$$

The output of cascaded circuits is defined to be $(y_1, \ldots, y_\mathsf{q})$.

We can generalize this concept to the setting of garbled circuits, where the i^{th} garbled circuit in the sequence produces wire keys to the $(i+1)^{th}$ garbled circuit. As before, every garbled circuit also produces additional output. We associate with each C_i a circuit G_i that outputs a garbled input for the next garbled circuit $\langle G_{i+1} \rangle_\mathsf{gc}$, along with the additional output y_i. Namely, the cascaded garbled circuit is evaluated analogously to the cascaded circuits:

$$\begin{aligned}
\langle x_2\rangle_{\mathsf{gc}} \| \; y_1 &\leftarrow \mathsf{EvalGC}(\langle G_1\rangle_{\mathsf{gc}}, \langle x_1\rangle_{\mathsf{gc}}) \\
\langle x_3\rangle_{\mathsf{gc}} \| \; y_2 &\leftarrow \mathsf{EvalGC}(\langle G_2\rangle_{\mathsf{gc}}, \langle x_2\rangle_{\mathsf{gc}}) \\
&\vdots \\
\langle x_{\mathsf{q}+1}\rangle_{\mathsf{gc}} \| \; y_{\mathsf{q}} &\leftarrow \mathsf{EvalGC}(\langle C_{\mathsf{q}}\rangle_{\mathsf{gc}}, \langle x_{\mathsf{q}}\rangle_{\mathsf{gc}})
\end{aligned}$$

and the output is defined to be $(y_1, \ldots, y_{\mathsf{q}})$.

Definition 2.6.5 (Cascaded Garbled Circuits). *Let* $\mathsf{GC} = (\mathsf{GrbCkt}, \mathsf{GrbInp}, \mathsf{EvalGC})$ *be a circuit garbling scheme. For a circuit* C_i *and a garbling key* $r \leftarrow \mathsf{Gen}(1^\lambda)$, *define the circuit* $G[C_i, r]$ *on input* x_i *to be:*

$$G[C_i, r](x_i) = \mathsf{GrbInp}(r, x_{i+1}) \| y_i,$$

with x_i *and* y_i *as above. For the cascaded circuits* $(x_1, C_1, \ldots, C_{\mathsf{q}})$, *the corresponding* cascaded garbled circuits *are*

$$\left(\langle x_1\rangle_{\mathsf{gc}}, \langle G_1\rangle_{\mathsf{gc}}, \ldots, \langle G_{\mathsf{q}}\rangle_{\mathsf{gc}}\right) \leftarrow (\mathsf{GrbInp}(r_1, x_1), \mathsf{GrbCkt}(r_1, G_1), \ldots, \mathsf{GrbCkt}(r_{\mathsf{q}}, G_{\mathsf{q}}))$$

where $r_i \leftarrow \mathsf{Gen}(1^\lambda)$ *and* $G_i = G[C_i, r_{i+1}]$. *The output of cascaded garbled circuits is* $(y_1, \ldots, y_{\mathsf{q}})$.

The following lemma about the security of cascaded garbled circuits has been implicitly used in several different contexts such as secure computation on the web [HLP11, GHV10], garbled RAM [LO13, GHL$^+$14, GLOS15, GLO15], indistinguishability obfuscation for Turing machines [BGL$^+$15] and adaptive garbled circuits [HJO$^+$15].

Lemma 2.6.6 (SIM-security of Cascaded Garbled Circuits). *For every* $\lambda \in \mathbb{N}$, *let* $(x_1, C_1, \ldots, C_{\mathsf{q}})$ *be a cascaded circuit with output* $(y_1, \ldots, y_{\mathsf{q}})$. *Suppose that* GC *is a SIM-secure garbled circuits scheme. There exists a PPT simulator* Sim *such that,* *the distribution over the cascaded garbled circuit*

$$\left(\langle x_1\rangle_{\mathsf{gc}}, \langle G_1\rangle_{\mathsf{gc}}, \langle G_2\rangle_{\mathsf{gc}}, \ldots, \langle G_{\mathsf{q}-1}\rangle_{\mathsf{gc}}, \langle G_{\mathsf{q}}\rangle_{\mathsf{gc}}\right)_\lambda \overset{c}{\approx} \mathsf{Sim}\left(1^\lambda, \phi(C_1), \ldots, \phi(C_{\mathsf{q}}), y_1, \ldots, y_{\mathsf{q}}\right)_\lambda,$$

where $\phi(C_i) = \phi_{\mathsf{topo}}(C_i)$ *denotes the topology of* C_i.

Proof. We define the simulator Sim using the PPT simulator of garbled circuits, $\mathsf{Sim}_{\mathsf{gc}}$. Sim uses $\mathsf{Sim}_{\mathsf{gc}}$ to simulate the garbled circuits in reverse order as follows. Let m be the output length of G_{q} and let $\phi_\perp$ be the topology of an arbitrary circuit with m-bit outputs (e.g., $\phi_\perp = \phi(G_{\mathsf{q}})$).

- Let $(x^{\mathsf{sim}}_{\mathsf{q}+1}, G^{\mathsf{sim}}_{\mathsf{q}+1}) \leftarrow \mathsf{Sim}_{\mathsf{gc}}(\phi_\perp, 0^m)$.
- For i from q to 1, let $(x^{\mathsf{sim}}_i, G^{\mathsf{sim}}_i) \leftarrow \mathsf{Sim}_{\mathsf{gc}}(\phi(G_i), x^{\mathsf{sim}}_{i+1} \| y_i)$.
- Output $\left(x^{\mathsf{sim}}_1, G^{\mathsf{sim}}_1, \ldots, G^{\mathsf{sim}}_{\mathsf{q}}\right)$.

The proof of computational indistinguishability proceeds by a sequence of hybrids. The initial hybrid is the real cascaded garbled circuit, and the final hybrid is the simulated cascaded garbled circuit. The intermediate hybrids are defined by simulating the garbling of x_1, along with the first i circuits $G_1, \ldots, G_i$, while using GC to garble the remaining circuits $G_{i+1}, \ldots, G_{\mathsf{q}}$. The final reduction uses the SIM security of GC to establish that each adjacent pair of hybrids are indistinguishable. We briefly describe the hybrids below.

HYBRID $\mathbf{H}_1$: $\left(\langle x_1\rangle_{\mathsf{gc}}, \langle G_1\rangle_{\mathsf{gc}}, \langle G_2\rangle_{\mathsf{gc}}, \ldots, \langle G_{\mathsf{q}-1}\rangle_{\mathsf{gc}}, \langle G_{\mathsf{q}}\rangle_{\mathsf{gc}}\right)$. All the garbled circuits are computed according to GC.

HYBRID $\mathbf{H}_{2.i}$: $\left(x^{\mathsf{sim}}_1, G^{\mathsf{sim}}_1, G^{\mathsf{sim}}_2, \ldots, G^{\mathsf{sim}}_i, \langle G_{i+1}\rangle_{\mathsf{gc}}, \ldots, \langle G_{\mathsf{q}-1}\rangle_{\mathsf{gc}}, \langle G_{\mathsf{q}}\rangle_{\mathsf{gc}}\right)$. The first i garbled circuits are simulated and the rest are computed according to GC. $\mathbf{H}_{2.0}$ is identical to $\mathbf{H}_1$. The indistinguishability of $\mathbf{H}_{2.i}$ and $\mathbf{H}_{2.i+1}$ follows from the security of garbled circuits.

HYBRID $\mathbf{H}_3$: $\left(x^{\mathsf{sim}}_1, G^{\mathsf{sim}}_1, \ldots, G^{\mathsf{sim}}_{\mathsf{q}}\right)$. All the garbled circuits are simulated. The hybrid $\mathbf{H}_{2.\mathsf{q}+1}$ is identical to $\mathbf{H}_3$. □

2.6.2.3 Proof of IND Security

We prove that the above is a IND-secure sequential URE scheme. We first the sequence of hybrids, and then demonstrate their indistinguishability. Let q be the number of update queries $(\mathbf{u}_i^0, \mathbf{u}_i^1)$ the adversary makes in the IND security game in Definition 2.4.3.

The first hybrid corresponds to the IND-experiment $\mathsf{Expt}_{\mathcal{A}}^{\mathsf{SURE}}(1^\lambda, 0)$ given in Definition 2.4.3. Then we move to a hybrid in which the security experiment is run using modified versions of Encode and GenUpd, with outputs corresponding to updatable randomized encoding of circuit C^0, input x^0, and updates $\mathbf{u}_i^0$.

We then proceed to change the encodings one-by-one until all all the encodings correspond to circuit C^1, input x^1, and updates $\mathbf{u}_i^1$. Finally we revert to the unmodified versions of Encode and GenUpd, yielding an experiment corresponding to the IND-experiment $\mathsf{Expt}_{\mathcal{A}}^{\mathsf{SURE}}(1^\lambda, 1)$.

$\mathbf{H}_{\text{real}}^0$: Run the indistinguishability experiment $\mathsf{Expt}_{\mathcal{A}}^{\mathsf{SURE}}(1^\lambda, 0)$.

$\mathbf{H}_{\text{proof}}^0$: As above, except replacing $\mathsf{Encode}(1^\lambda, C^0, x^0)$ with $\mathsf{Encode}_{\text{proof}}^0(1^\lambda, C^0, x^0, C^1, x^1)$ (Figure 2-3), and replacing all executions of $\mathsf{GenUpd}(\mathsf{st}_i, \mathbf{u}_{i+1}^0)$ with $\mathsf{GenUpd}_0(\mathsf{st}_i, \mathbf{u}_{i+1}^0, \mathbf{u}_{i+1}^1)$ (Figure 2-4).

$\mathbf{H}_0$: As above, except replacing $\mathsf{Encode}_{\text{proof}}^0(1^\lambda, C^0, x^0, C^1, x^1)$ with $\mathsf{Encode}_{\text{proof}}^1(1^\lambda, C^0, x^0, C^1, x^1)$ (Figure 2-3). This hybrid still uses GenUpd_0.

$\mathbf{H}_h$ FOR $1 \leq h \leq \mathsf{q}$: As above, except using GenUpd_h (Figure 2-4).

$\mathbf{H}_{\text{real}}^1$: Run the indistinguishability experiment $\mathsf{Expt}_{\mathcal{A}}^{\mathsf{SURE}}(1^\lambda, 1)$. That is, replace $\mathsf{Encode}_{\text{proof}}^1(1^\lambda, C^0, x^0, C^1, x^1)$ with $\mathsf{Encode}(1^\lambda, C^1, x^1)$ and all executions of $\mathsf{GenUpd}_{\mathsf{q}}(\mathsf{st}_i, \mathbf{u}_{i+1}^0, \mathbf{u}_{i+1}^1)$ with $\mathsf{GenUpd}(\mathsf{st}_i, \mathbf{u}_{i+1}^1)$.

2.6.2.4 Indistinguishability of Hybrids

Moving from $\mathbf{H}_{\text{real}}^0$ to $\mathbf{H}_{\text{proof}}^0$ involves two changes which we address in turn. First, when generating the initial updatable randomized encoding, we switch from using

$\underline{\mathsf{Encode}^b_{\text{proof}}\left(1^\lambda, C^0, x^0, C^1, x^1\right)}$:

1. Execute the setup of FE, $\mathsf{FE.MSK} \leftarrow \mathsf{FE.Setup}(1^\lambda)$.
2. Compute a functional key $\mathsf{FE.SK} \leftarrow \mathsf{FE.KeyGen}(\mathsf{FE.MSK}, \mathsf{RRGarbler})$, where $\mathsf{RRGarbler}$ is as defined in Figure 2-2.
3. Generate a randomized encoding of input (C^b, x^b). That is, evaluate $\langle C^b[x^b]\rangle_{\mathsf{re}} \leftarrow \mathsf{RE.Enc}(1^\lambda, C^b, x^b)$.
4. Generate a garbled circuit input encoding of $(C^0[x^0], C^1[x^1])$ using randomness r_{gc}. That is, evaluate $\langle C^0[x^0], C^1[x^1]\rangle_{\mathsf{gc}} \leftarrow \mathsf{GrbInp}(C^0[x^0], C^1[x^1]; r_{\mathsf{gc}})$
5. Output as the randomized encoding the tuple $\langle C[x]\rangle_{\mathsf{ure}} = \Big(\ \mathsf{FE.SK},\ \langle C^b[x^b]\rangle_{\mathsf{re}},\ \langle C^0[x^0], C^1[x^1]\rangle_{\mathsf{gc}}\ \Big)$ and set the state to be $\mathsf{st} = (\mathsf{FE.MSK},\ r_{\mathsf{gc}})$.

Figure 2-3

$\underline{\mathsf{GenUpd}_h\left(\mathsf{st}_i,\ \mathbf{u}^0_{i+1},\ \mathbf{u}^1_{i+1}\right)}$:

1. Let $\mathsf{st}_i = (\mathsf{FE.MSK}, r_{\mathsf{gc},i})$.
2. If $i + 1 \leq h$, let $\mathsf{mode} = 1$. Otherwise, let $\mathsf{mode} = 0$.
3. Sample random coins $r_{\mathsf{re},i+1}$ and $r_{\mathsf{gc},i+1}$.
4. Generate the FE ciphertext CT_{i+1} as $\mathsf{FE.Enc}\left(\mathsf{FE.MSK}, \left(\mathbf{u}^0_{i+1}, \mathbf{u}^1_{i+1}, r_{\mathsf{gc},i}, r_{\mathsf{gc},i+1}, r_{\mathsf{re},i+1}, \mathsf{mode}\right)\right)$.
5. Set the new state $\mathsf{st}_{i+1} = (\mathsf{FE.MSK},\ r_{\mathsf{gc},i+1})$.
6. Output $\langle \mathbf{u}_{i+1}\rangle_{\mathsf{ure}} = \mathsf{CT}_{i+1}$ and st_{i+1}.

Figure 2-4

$\mathsf{Encode}(1^\lambda, C^0, x^0)$ to using $\mathsf{Encode}^0(1^\lambda, C^0, x^0, C^1, x^1)$. The resulting change in the actual URE encoding is that initial garbled input is changed from $w = \langle C^0[x^0], \perp\rangle_{\mathsf{gc}}$ to $w' = \langle C^0[x^0], C^1[x^1]\rangle_{\mathsf{gc}}$. This causes cascading changes to the garbled inputs as each update is generated and applied, but everything else remains unchanged (namely, the garbled $\mathsf{RelockRelease}_i$ circuits, the FE secret key, and all the RE encodings of the updated $C^0_i[x^0_i]$). We first observe that

$$\left(\langle C^0[x^0]\rangle_{\mathsf{re}}, w, \langle \mathsf{RelockRelease}_1\rangle_{\mathsf{gc}}, \ldots, \langle \mathsf{RelockRelease}_{\mathsf{q}}\rangle_{\mathsf{gc}}\right)$$

$$\overset{c}{\approx} \left(\langle C^0[x^0] \rangle_{\mathsf{re}}, w', \langle \mathsf{RelockRelease}_1 \rangle_{\mathsf{gc}}, \dots, \langle \mathsf{RelockRelease}_{\mathsf{q}} \rangle_{\mathsf{gc}} \right) \quad (2.1)$$

by a direct application of Lemma 2.6.6, because the outputs of the cascaded garbled circuits are unchanged. Now Lemma 2.6.3 establishes indistinguishability, taking aux_0 and aux_1 to be the two tuples above, respectively. The function f is $\mathsf{RRGarbler}$ and the messages M_i are those corresponding to the ciphertexts $\langle \mathbf{u}_i \rangle_{\mathsf{ure}} = \mathsf{FE.CT}_i$.

The second step in switching from $\mathbf{H}^0_{\text{real}}$ to $\mathbf{H}^0_{\text{proof}}$ is to switch GenUpd to GenUpd_0, ultimately changing each of the URE encoded updates from $\langle \mathbf{u}_i \rangle_{\mathsf{ure}} = \mathsf{FE.CT}_i$ to $\langle \mathbf{u}'_i \rangle_{\mathsf{ure}} = \mathsf{FE.CT}'_i$ containing both $\mathbf{u}^0_i$ and $\mathbf{u}^1_i$. While the FE key for $\mathsf{RRGarbler}$ is unchanged, the new ciphertexts result in output garbled circuits $\langle \mathsf{RelockRelease}'_i \rangle_{\mathsf{gc}}$, each of which now has both $\mathbf{u}^0_i$ and $\mathbf{u}^1_i$ hard-coded. This results in changes to the garbled input after each update is applied, but all of the RE encodings $\langle C^0_i[x^0_i] \rangle_{\mathsf{re}}$ output by $\mathsf{RelockRelease}_i$ are exactly the same. Therefore, by Lemma 2.6.6, for each $j \in [\mathsf{q}]$:

$$\left(\langle C^0[x^0] \rangle_{\mathsf{re}},\ w',\ \begin{matrix} \langle \mathsf{RelockRelease}_1 \rangle_{\mathsf{gc}}, \dots, \langle \mathsf{RelockRelease}_j \rangle_{\mathsf{gc}}, \\ \langle \mathsf{RelockRelease}'_{j+1} \rangle_{\mathsf{gc}}, \dots, \langle \mathsf{RelockRelease}'_{\mathsf{q}} \rangle_{\mathsf{gc}} \end{matrix} \right)$$
$$\overset{c}{\approx} \left(\langle C^0[x^0] \rangle_{\mathsf{re}},\ w',\ \begin{matrix} \langle \mathsf{RelockRelease}_1 \rangle_{\mathsf{gc}}, \dots, \langle \mathsf{RelockRelease}_{j-1} \rangle_{\mathsf{gc}}, \\ \langle \mathsf{RelockRelease}'_j \rangle_{\mathsf{gc}}, \dots, \langle \mathsf{RelockRelease}'_{\mathsf{q}} \rangle_{\mathsf{gc}} \end{matrix} \right)$$

For each j, we apply Lemma 2.6.3 to switch $\langle \mathbf{u}_j \rangle_{\mathsf{ure}}$ to $\langle \mathbf{u}'_j \rangle_{\mathsf{ure}}$, thereby establishing the indistinguishability of $\mathbf{H}^0_{\text{real}}$ and $\mathbf{H}^0_{\text{proof}}$.

Showing $\mathbf{H}^0_{\text{proof}} \approx_{\mathsf{c}} \mathbf{H}_0$ is much simpler. Changing Encode^0 to Encode^1 switches the initial URE encoding from $\left(\mathsf{FE.SK}, \langle C^0[x^0] \rangle_{\mathsf{re}}, \langle C^0[x^0], C^1[x^1] \rangle_{\mathsf{gc}} \right)$ to $\left(\mathsf{FE.SK}, \langle C^1[x^1] \rangle_{\mathsf{re}}, \langle C^0[x^0], C^1[x^1] \rangle_{\mathsf{gc}} \right)$, and all URE encoded updates $\langle \mathbf{u}_i \rangle_{\mathsf{ure}}$ are unchanged. Notice that the randomness used to generate $\langle C^b[x^b] \rangle_{\mathsf{re}}$ is independent of everything in the adversary's view except for $\langle C^b[x^b] \rangle_{\mathsf{re}}$ itself. Given $\langle C^b[x^b] \rangle_{\mathsf{re}}$ along with C^0, C^1, x^0, and x^1, the view of the adversary can be exactly simulated. Therefore, the security of RE enables the switch from $\langle C^0[x^0] \rangle_{\mathsf{re}}$ to $\langle C^1[x^1] \rangle_{\mathsf{re}}$ that we require.

We now show that $\mathbf{H}_h \approx_{\mathsf{c}} \mathbf{H}_{h+1}$ for all $h \in [\mathsf{q}-1]$. In the latter hybrid, GenUpd_{h+1}

is used, while GenUpd_h is used in the former. The only change in the view of the adversary is therefore the mode bit's setting in $\langle \mathbf{u}_{h+1} \rangle_{\mathsf{ure}} = \mathsf{FE.CT}_{h+1}$, which is set to 0 in the former and 1 in the latter. When evaluated using the FE functional key $\mathsf{FE.SK}$ for $\mathsf{RRGarbler}$, this results in the garbled $\mathsf{RelockRelease}_{h+1}$ outputting a randomized encoding $\langle C^1_{h+1}[x^1_{h+1}] \rangle_{\mathsf{re}}$ of $C^1_{h+1}[x^1_{h+1}]$ instead of $C^0_{h+1}[x^0_{h+1}]$; all other URE-encoded updates (FE ciphertexts) are unchanged. Recall in Defintion 2.4.3 we require that $C^0_{h+1}(x^0_{h+1}) = C^1_{h+1}(x^1_{h+1})$. This requirement, along with the security of RE, guarantees that

$$\begin{aligned}&\left(\langle C^1_0[x^1_0] \rangle_{\mathsf{re}}, \ldots, \langle C^1_h[x^1_h] \rangle_{\mathsf{re}}, \langle C^0_{h+1}[x^0_{h+1}] \rangle_{\mathsf{re}}, \langle C^0_{h+2}[x^0_{h+2}] \rangle_{\mathsf{re}}, \ldots, \langle C^0_{\mathsf{q}}[x^0_{\mathsf{q}}] \rangle_{\mathsf{re}}\right)\\ \overset{\mathsf{c}}{\approx}\ &\left(\langle C^1_0[x^1_0] \rangle_{\mathsf{re}}, \ldots, \langle C^1_h[x^1_h] \rangle_{\mathsf{re}}, \langle C^1_{h+1}[x^1_{h+1}] \rangle_{\mathsf{re}}, \langle C^0_{h+2}[x^0_{h+2}] \rangle_{\mathsf{re}}, \ldots, \langle C^0_{\mathsf{q}}[x^0_{\mathsf{q}}] \rangle_{\mathsf{re}}\right)\end{aligned}$$

Applying Lemma 2.6.6 followed by Lemma 2.6.3 establishes indistinguishability of $\mathbf{H}_h$ and $\mathbf{H}_{h+1}$.

The indistinguishability of $\mathbf{H}_{\mathsf{q}}$ and $\mathbf{H}^1_{\mathrm{real}}$ follows essentially the same argument as $\mathbf{H}^0_{\mathrm{proof}}$ and $\mathbf{H}^0_{\mathrm{real}}$ but with the bit b swapped in $\mathsf{Expt}^{\mathsf{SURE}}_{\mathcal{A}}(1^\lambda, b)$, Encode^b, and the setting of mode used by GenUpd.

2.7 Updatable Garbled Circuits

In this section, we present definitions and a construction of updatable garbled circuits for the family of "gate-wise updates" which allow for changing the functionality of any single circuit gate. These gate-wise updates can be composed in sequence to perform general circuit modifications. The main tool in our construction is a puncturable proxy reencryption scheme, a notion that we define and construct in this section.

We start by giving a formal definition the class of gate-wise updates and of updatable garbled circuits in Section 2.7.1. Next, in Section 2.7.2 we define and a construct a puncturable proxy reencryption scheme based on puncturable (almost) key-homomorphic PRFs of [BV15b]. Finally, in Section 2.7.3, we give our construction of updatable garbled circuits for gate-wise updates from a puncturable proxy reencryption scheme.

2.7.1 Definition of Sequential Updatable Garbled Circuits

2.7.1.1 Gate-wise Updates

We consider updates which change the description of the gates of the circuit, which we term "gate-wise" updating. Changing the output gate from $\vee$ to $\neg$ is one such example. Composing many such updates suffices for many interesting applications. One could also consider more general updates such as adding gates, deleting gates and so on which we leave as a direction to future exploration.

Definition 2.7.1 (Gate-wise updatable circuits). *A family of circuits $\mathcal{C}$ is* gate-wise *updatable if it is $(\mathsf{Upd}_{\mathsf{gate}}, \mathcal{U}_{\mathsf{gate}})$-updatable, where:*

- $\mathcal{U}_{\mathsf{gate}} = \{(g, \mathsf{gateType}) \in \mathbb{Z} \times \{\vee, \wedge, \neg\}\}$ *is the set of updates consisting of a gate index g and a new gate type* $\mathsf{gateType}$.[10]

- $\mathsf{Upd}_{\mathsf{gate}}(C \in \mathcal{C}, \mathbf{u})$ *takes as input a circuit C and an update $\mathbf{u} = (g, \mathsf{gateType})$. If $g > |C|$, or if g corresponds to a source node in the circuit, then* $\mathsf{Upd}_{\mathsf{gate}}$ *outputs $\perp$. Otherwise, output the circuit that results from changing gate g of C to gate type* $\mathsf{gateType}$.

2.7.1.2 Syntax

A scheme $\mathsf{UGC} = (\mathsf{GrbCkt}, \mathsf{GrbInp}, \mathsf{GenUpd}, \mathsf{ApplyUpd}, \mathsf{EvalGC})$ for a $(\mathsf{Upd}, \mathcal{U})$-updatable class of circuits $\mathcal{C} = \{\mathcal{C}_\lambda\}_{\lambda \in \mathbb{N}}$ is defined below.

Circuit Garbling, $\langle C \rangle_{\mathsf{gc}}, \mathsf{st} \leftarrow \mathsf{GrbCkt}(C)$: On input circuit $C \in \mathcal{C}_\lambda$, output the garbled circuit $\langle C \rangle_{\mathsf{gc}}$.

Generate Secure Update, $\langle \mathbf{u} \rangle_{\mathsf{gc}}, \mathsf{st}' \leftarrow \mathsf{GenUpd}(\mathsf{st}, \mathbf{u})$: On input state st, update $\mathbf{u} \in \mathcal{U}_\lambda$, output the secure update $\langle \mathbf{u} \rangle_{\mathsf{gc}}$.

[10] Any universal gate set would do, and any could be supported.

APPLY SECURE UPDATE, $\langle C' \rangle_{\mathsf{gc}} \leftarrow \mathsf{ApplyUpd}\left(\langle C \rangle_{\mathsf{gc}}, \langle \mathbf{u} \rangle_{\mathsf{gc}}\right)$: On input the (old) garbled circuit $\langle C \rangle_{\mathsf{gc}}$, secure update $\langle \mathbf{u} \rangle_{\mathsf{gc}}$, output the updated garbled circuit $\langle C' \rangle_{\mathsf{gc}}$.

INPUT GARBLING, $\langle x \rangle_{\mathsf{gc}} \leftarrow \mathsf{GrbInp}(\mathsf{st}, x \in \{0,1\}^{\lambda})$: On input state st, input $x \in \{0,1\}^{n(\lambda)}$, output the garbled input $\langle x \rangle_{\mathsf{gc}}$.

EVALUATION, $\alpha \leftarrow \mathsf{EvalGC}(\langle C \rangle_{\mathsf{gc}}, \langle x \rangle_{\mathsf{gc}})$: On input garbled circuit $\langle C \rangle_{\mathsf{gc}}$, garbled input $\langle x \rangle_{\mathsf{gc}}$, output the result α.

2.7.1.3 Efficiency

We desire the same efficiency properties as in the definition of URE, except with the requirement on garbling (encoding) time modified appropriately to the setting of garbled circuits.

- *Garbling Time*: We require that the time to generate (and thus the size of) the garbled input $\langle x \rangle_{\mathsf{gc}}$ should be polynomial in $|x|$ and λ, and independent of the size of C and the number of updates. The time to generate the garbled circuit $\langle C \rangle_{\mathsf{gc}}$ should be polynomial in $|C|$ and λ.

- *Secure Update Generation Time*, *Secure Update Size*, *Runtime of Update*, and *State Size*: Same efficiency goals as URE.

While we would like to achieve all the efficiency goals simultaneously, in our construction of UGC presented in Section 2.7.3, the 'Runtime of Update' and the secret 'State Size' will both depend on $|C|$. Using the transformation described in Appendix 2.5.3, it is possible to remove the dependence on $|C|$ for the state size.

2.7.1.4 Correctness

The notion of sequential updating for updatable garbled circuits closely mirrors the URE notion. The key difference is that in the UGC setting, the encoded input is released only after all the encoded updates. Below, we define correctness and security of sequential updating.

Let $C_0 \in \mathcal{C}_\lambda, x \in \{0,1\}^\lambda$. Let $(\mathbf{u}_1, \ldots, \mathbf{u}_{\mathsf{q}}) \in (\mathcal{U}_\lambda)^{\mathsf{q}}$, where $\mathsf{q}(\lambda)$ is a polynomial in λ. We require that $\mathsf{EvalGC}(\langle C \rangle_{\mathsf{gc}_\mathsf{q}}, \langle x \rangle_{\mathsf{gc}}) = C_{\mathsf{q}}(x)$, where ${}_{\mathsf{q}}(x)$ is generated by recursively evaluating $C_i = \mathsf{Upd}(C_{i-1}, \mathbf{u}_i)$, and $\mathsf{EvalGC}(\langle C \rangle_{\mathsf{gc}_\mathsf{q}}, \langle x \rangle_{\mathsf{gc}})$ is generated as:

- Let $\langle C \rangle_{\mathsf{gc}_0}, \mathsf{st}_0 \leftarrow \mathsf{GrbCkt}(C_0)$.
- For $i \in [\mathsf{q}]$, let $\langle \mathbf{u} \rangle_{\mathsf{gc}_i}, \mathsf{st}_i \leftarrow \mathsf{GenUpd}(\mathbf{u}_i, \mathsf{st}_{i-1})$
- For $i \in [\mathsf{q}]$, let $\langle C \rangle_{\mathsf{gc}_i} \leftarrow \mathsf{ApplyUpd}(\langle C \rangle_{\mathsf{gc}_{i-1}}, \langle \mathbf{u} \rangle_{\mathsf{gc}_i})$.
- Let $\langle x \rangle_{\mathsf{gc}} \leftarrow \mathsf{GrbInp}(\mathsf{st}_{\mathsf{q}}, x)$.

2.7.1.5 IND-Security of Sequential Updating

As in the case of URE, we can consider indistinguishability-based and simulation-based security definitions. We focus on the former. We denote the challenger by Ch and the PPT adversary by $\mathcal{A}$.

As for traditional garbled circuits, security is defined with respect to deterministic side-information functions ϕ, which capture what information is leaked by the encoded updates. Ideally, a UGC scheme would be *update hiding*, meaning that $\phi(\mathbf{u}) = \bot$. In our main construction of updatable garbled circuits for gatewise updates $\mathbf{u} = (g, \mathsf{gateType})$, the side-information function $\phi_{\mathsf{gate}}(g, \mathsf{gateType}) = g$ will leak the index of the gate being updated. In particular, to execute $\mathsf{ApplyUpd}$, the evaluator will have to know which gate of the circuit representation is being modified (though the old and new types of this gate will remain secret). In Appendix 2.7.5, we describe a generic transformation from a non-update-hiding UGC scheme into one that achieves update hiding using a (non-interactive) oblivious RAM.

Definition 2.7.2 (IND-secure Sequential UGC). *A sequential UGC scheme* UGC *for* $(\mathsf{Upd}, \mathcal{U})$*-updatable class of circuits* $\mathcal{C}$ *is* IND secure *with leakage* ϕ *if for every pair of circuits* $C^0, C^1 \in \mathcal{C}_\lambda$*, inputs* $x^0, x^1 \in \{0,1\}^{n(\lambda)}$*, and update sequences* $\{\mathbf{u}_i^0\}_{i \in [\mathsf{q}]}$ *and* $\{\mathbf{u}_i^1\}_{i \in [\mathsf{q}]}$*:*

$$\mathsf{Expt}_0^{\mathsf{UGC}}(1^\lambda, \mathsf{input}) \overset{c}{\approx} \mathsf{Expt}_1^{\mathsf{UGC}}(1^\lambda, \mathsf{input}),$$

where $\mathsf{Expt}_b^{\mathsf{UGC}}$ *is defined below and* $\mathsf{input} = \left(C^0, C^1, x^0, x^1, \{\mathbf{u}_i^0\}_{i \in [\mathsf{q}]}, \{\mathbf{u}_i^1\}_{i \in [\mathsf{q}]}\right)$.

$\mathsf{Expt}_b^{\mathsf{UGC}}(1^\lambda, C^0, C^1, x^0, x^1, \{\mathbf{u}_i^0\}_{i\in[\mathsf{q}]}, \{\mathbf{u}_i^1\}_{i\in[\mathsf{q}]})$:

- Check that $\phi(C^0) = \phi(C^1)$ and that for all $i \in [\mathsf{q}]$, $\phi(\mathbf{u}_i^0) = \phi(\mathbf{u}_i^1)$. If not, abort and output $\perp$.
- For $\beta \in \{0,1\}$, let $C_0^\beta = C^\beta$. For $i \in [\mathsf{q}]$, recursively evaluate $C_i^\beta \leftarrow \mathsf{Upd}(C_{i-1}^\beta, \mathbf{u}_i^\beta)$.
- Check that $C_{\mathsf{q}}^0(x^0) = C_{\mathsf{q}}^1(x^1)$. If not, abort and output $\perp$.
- Compute the garbled circuit. $(\langle C^b \rangle_{\mathsf{ugc}}, \mathsf{st}_{\mathsf{ugc}}) \leftarrow \mathsf{GrbCkt}(1^\lambda, C^b)$. Compute the garbled updates $\langle \mathbf{u}_i^b \rangle_{\mathsf{ugc}}, \mathsf{st}_{\mathsf{ugc}}^i \leftarrow \mathsf{GenUpd}(\mathbf{u}_i^b, \mathsf{st}_{\mathsf{ugc}}^{i-1})$. Compute the garbled input $\langle x^b \rangle_{\mathsf{ugc}} \leftarrow \mathsf{GrbInp}(\mathsf{st}_{\mathsf{ugc}}^{\mathsf{q}}, \langle x^b \rangle_{\mathsf{ugc}})$.
- Output $\left(\langle C^b \rangle_{\mathsf{ugc}}, \langle \mathbf{u}_1^b \rangle_{\mathsf{ugc}}, \ldots, \langle \mathbf{u}_{\mathsf{q}}^b \rangle_{\mathsf{ure}}, \langle x^b \rangle_{\mathsf{ugc}}\right)$.

2.7.2 Puncturable Proxy Reencryption Scheme

En route to constructing updatable garbled circuits, we introduce a tool called *puncturable symmetric proxy reencryption scheme*. As in a standard proxy reencryption scheme, our notion allows for generation of reencryption keys. However, unlike a standard proxy reencryption scheme, our notion allows for puncturing of reencryption keys on ciphertexts such that the reencryption mechanism fails on the punctured ciphertexts.

We will augment the symmetric proxy reencryption scheme of [BLMR13b]. By using key-homomorphic *puncturable* PRFs [BFP+15, BV15b], a strengthening of the key-homomorphic PRFs used in [BLMR13b], we are able to imbue the reencryption scheme with additional properties.[11] The most significant change is that we introduce an additional algorithm to the scheme to generate "punctured" reencryption keys.

We begin by defining puncturable symmetric proxy reencryption, along with identifying a number of additional properties our construction will satisfy that are useful in the construction of updatable garbled circuits. Our definition is tailored to our

[11]The definition and constructions are both related to, though incomparable with to the "proxy reencryption with fine-grained access control" of [BFP+15]. Neither our work nor theirs provides a satisfactory general definition for a (ciphertext-) constrained proxy reencryption scheme.

needs, and is not intended to be the most general definition of puncturable proxy reencryption schemes. In Section 2.7.4, we prove:

Theorem 2.7.3. *Assuming the hardness of approximating either* GapSVP *or* SIVP *to within sub-exponential factors, there exists a symmetric puncturable reencryption scheme satisfying all the properties described next.*

Remark 2.7.4. Chapter 3 discusses proxy reencryption in greater depth, focusing on a shortcoming of the most well-studied security notion. The use of proxy reencryption for updatable garbled circuits inspired that chapter. It is worth remarking on the differences between our use of proxy reencryption in the two chapters. In this chapter, we use bidirectional, symmetric, puncturable proxy reencryption. In Chapter 3, we consider unidirectional, asymmetric, non-puncturable proxy reencryption. Unidirectionaly is the most relevant distinction—the observations of the next chapter do not apply in the bidirectional setting. However, notice the resemblance between fresh-stale indistinguishability and reencryption simulatability. Though incomparable, both are used to prove security with respect to a recipient who knows the ultimate secret key for decryption.

2.7.2.1 Syntax

A puncturable, symmetric proxy reencryption scheme is a tuple of algorithms $\mathsf{ReEnc} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{ReKeyGen}, \mathsf{Enc}, \mathsf{ReEnc}, \mathsf{Dec})$ with the following syntax.

SETUP: $\mathsf{PP} \leftarrow \mathsf{Setup}(1^\lambda)$: On input a security parameter λ in unary, output the public parameters PP of the ReEnc.

KEY GENERATION: $\mathsf{SK} \leftarrow \mathsf{KeyGen}(\mathsf{PP})$: On input the public parameters PP, sample a ReEnc secret key SK.

ENCRYPTION: $\mathsf{CT} \leftarrow \mathsf{Enc}(\mathsf{SK}, \mathsf{m})$: On input a secret key SK and a message $\mathsf{m} \in \mathcal{M}$, output a ciphertext CT.

DECRYPTION: $\mathsf{m} \leftarrow \mathsf{Dec}(\mathsf{SK}, \mathsf{CT})$: On input a secret key SK and a ciphertext CT, output a message $\mathsf{m} \in \mathcal{M}$.

PUNCTURED RE-ENCRYPTION KEY GENERATION: $\mathsf{rk}^{\pi}_{1,2} \leftarrow \mathsf{ReKeyGen}(\mathsf{SK}_1, \mathsf{SK}_2, \pi)$: On input two secret keys SK_1 and SK_2 and a boolean-valued circuit π whose input is the space of ciphertexts, output a reencryption key $\mathsf{rk}^{\pi}_{1,2}$. Informally, $\mathsf{rk}^{\pi}_{1,2}$ should only allow the reencryption of ciphertexts CT for which $\pi(\mathsf{CT}) = 1$ to be reencrypted from SK_1 to SK_2. We overload the notation $\mathsf{ReKeyGen}$, and will simply let $\mathsf{ReKeyGen}(\mathsf{SK}_1, \mathsf{SK}_2) := \mathsf{ReKeyGen}(\mathsf{SK}_1, \mathsf{SK}_2, \mathbf{1})$ where $\mathbf{1}$ is the constant 1 circuit.

REENCRYPTION: $\mathsf{CT}_2 \leftarrow \mathsf{ReEnc}(\mathsf{rk}_{1,2}\ ,\ \mathsf{CT}_1)$: On input a reencryption key $\mathsf{rk}_{1,2} \leftarrow \mathsf{ReKeyGen}(\mathsf{SK}_1, \mathsf{SK}_2)$ and a ciphertext $\mathsf{CT}_1 \leftarrow \mathsf{Enc}(\mathsf{SK}_1, \mathsf{m})$, output a ciphertext SK_2.

Informally, $\mathsf{rk}^{\pi}_{1,2}$ should only allow the reencryption of ciphertexts CT for which $\pi(\mathsf{CT}) = 1$ to be reencrypted from SK_1 to SK_2.

For our application, we will consider only a restricted class of constraint circuits π, and ciphertexts of a particular form. First, ciphertexts in ReEnc are of the form $\mathsf{CT} = (r, \mathsf{pl})$, where r is the randomness used in the encryption algorithm, and pl is the "payload." The randomness comes from a finite set $\mathcal{X}$ with description length polynomial in the security parameter.[12] Second, reencryption preserves the randomness of the ciphertext. That is, for any reencryption key rk and ciphertext (r, pl), $\mathsf{ReEnc}(\mathsf{rk}, (r, \mathsf{pl})) = (r, \mathsf{pl}')$ for some pl'. Third, we restrict our attention to constraints:

$$\pi_{r^*}(r, \mathsf{pl}) = \begin{cases} 0 & \text{if } r = r^* \\ 1 & \text{if } r \neq r^* \end{cases}$$

We abuse notation and identify r^* with the constraint $\pi(r^*)$, and additionally primitives with such constraints "punctured at r^*."

2.7.2.2 Correctness

The reencryption scheme ReEnc satisfies a modified correctness property that reflects the informal goal that reencryption keys punctured at r^* should enable reencryption

[12]Instantiating the PRF as we do, $\mathcal{X}$ will be the set of binary strings of appropriate length.

only of ciphertexts with randomness $r \neq r^*$.

Definition 2.7.5 (Ciphertext Punctured Reencryption: Correctness). *We will say the ciphertext constrained reencryption scheme* ReEnc *is correct if, under public parameters* $\mathsf{PP} \leftarrow \mathsf{Setup}(1^\lambda)$*, secret key* $\mathsf{SK} \leftarrow \mathsf{KeyGen}(1^\lambda)$*, and for all* $\mathsf{m} \in \mathcal{M}$*, two conditions hold:*

- $\mathsf{Dec}(\mathsf{SK}, \mathsf{Enc}(\mathsf{SK}, \mathsf{m})) = \mathsf{m}$
- *For all* $T = \text{poly}(\lambda)$*, and for any sequence of secret keys* $\mathsf{SK}_1, \ldots, \mathsf{SK}_T \leftarrow \mathsf{KeyGen}(1^\lambda)$*, and reencryption keys* $\mathsf{rk}^{r_i}_{i,i+1} \leftarrow \mathsf{ReKeyGen}(\mathsf{SK}_i, \mathsf{SK}_{i+1}, r_i)$ *for all* $i \in [1, T-1]$*, for all messages* $\mathsf{m} \in \mathcal{M}$*, and all sequences of ciphertexts* $\mathsf{CT}_1 \leftarrow \mathsf{Enc}(\mathsf{SK}_1, \mathsf{m})$*,* $\mathsf{CT}_2 \leftarrow \mathsf{ReEnc}(\mathsf{rk}^{r_2}_{1,2}, \mathsf{CT}_1)$*, …,* $\mathsf{CT}_T \leftarrow \mathsf{ReEnc}(\mathsf{rk}^{r_{T-1}}_{T-1,T}, \mathsf{CT}_{T-1})$ *it holds that either:*
  1. $\mathsf{Dec}(\mathsf{SK}_T, \mathsf{CT}_T) = \mathsf{m}$*, or*
  2. $\exists i \in [1, T-1]$ *such that* $r = r_i$*, where* r *is the randomness of the ciphertext* CT_1 *(and thus of all ciphertexts* CT_j*).*

2.7.2.3 Security

The semantic security definition of [BLMR13b] does not suffice for our needs. We require a different, incomparable security definition that captures the power of the constrained reencryption keys. In formalizing our security requirement, we diverge significantly from [BLMR13b]. A security definition that both suffices for the application to updatable garbled circuits and generalizes the [BLMR13b] definition would unnecessarily complicate our discussion and is outside the scope of this chapter. Below, we present a security definition that is tailored to our requirements.

Informally, we wish to guarantee security when the adversary gets constrained reencryption keys *and a terminal secret key*, if the constraint prevents "honest reencryption" of the challenge ciphertext.[13] In a very simplified form, this setting can be seen in the following game between an adversary $\mathcal{A}$ and a challenger Ch:

[13]Corresponding to the last condition in the correctness properties above.

- $\mathcal{A}$ chooses messages m^0 and m^1.
- Ch chooses a random bit $b \leftarrow \{0,1\}$, two random secret keys $\mathsf{SK}_1, \mathsf{SK}_2 \leftarrow \mathsf{KeyGen}(1^\lambda)$, and sends $c^* = (r^*, \mathsf{pl}^*) \leftarrow \mathsf{Enc}(\mathsf{SK}_1, \mathsf{m}^b)$.
- Ch generates $\mathsf{rk}_{1,2}^{r^*}$, and sends both $\mathsf{rk}_{1,2}^{r^*}$ and SK_2 to $\mathcal{A}$.
- $\mathcal{A}$ attempts to guess b with probability significantly better than 1/2.

Our actual security definition is somewhat more complex. The adversary will receive encryptions of many messages, a chain of punctured reencryption keys, and a single secret key. We also introduce the additional complexity that the messages may originally be encrypted with any of the secret keys $\mathsf{SK}_0, \ldots, \mathsf{SK}_\mathsf{q}$. Lastly, we must impose a non-triviality condition—which we term "validity"—described in the definition and further discussed below.

$\mathsf{Expt}_{\mathcal{A}}^{\mathsf{ReEnc}}(1^\lambda, b \in \{0,1\})$:

- The challenger samples and sends $\mathsf{PP} \leftarrow \mathsf{Setup}(1^\lambda)$ to $\mathcal{A}$.
- The adversary selects two sequences of messages $(\mathsf{m}_1^0, \ldots, \mathsf{m}_\ell^0)$ and $(\mathsf{m}_1^1, \ldots, \mathsf{m}_\ell^1)$ for some $\ell = \text{poly}(\lambda)$, along with a sequence of q "punctured reencryption key requests:" $(p_1, \ldots, p_\mathsf{q}) \in [\ell]$, where each p_j is distinct. Additionally, the adversary specifies a tuple of key-indices $(k_1, \ldots, k_\ell) \in [\mathsf{q}]$.
- The challenger checks the validity condition. If it is violated, the experiment aborts.
  - For all $i \in [\ell]$, if $\forall j \ : \ \big(p_j \neq i\big) \vee \big(j < k_i\big)$, then $\mathsf{m}_i^0 = \mathsf{m}_i^1$.
- The challenger samples secret keys $\mathsf{SK}_i \leftarrow \mathsf{KeyGen}(1^\lambda)$ for $i \in [\mathsf{q}+1]$, and sends the following to the adversary:
  - Ciphertexts $\mathsf{CT}_i \leftarrow \mathsf{Enc}(\mathsf{SK}_{k_i}, \mathsf{m}_i^b)$, for all $i \in [\ell]$. Let r_i be the randomness used in CT_i.
  - Punctured reencryption keys $\mathsf{rk}_{j,j+1}^{r_{p_j}} \leftarrow \mathsf{ReKeyGen}(\mathsf{SK}_j, \mathsf{SK}_{j+1}, r_{p_j})$, for each $j \in [\mathsf{q}]$, punctured to not work on CT_{p_j}.

  - The final secret key $\mathsf{SK}_{\mathsf{q}+1}$.

- The adversary outputs b'. The output of the experiment is 1 if $b' = b$, and 0 if $b' \neq b$.

Definition 2.7.6. *A scheme* ReEnc *is* secure *if for any PPT adversary $\mathcal{A}$ there exists a negligible function* negl

$$\left|\mathsf{Pr}\left[1 \leftarrow \mathsf{Expt}_{\mathcal{A}}^{\mathsf{ReEnc}}(1^{\lambda}, 0)\right] - \mathsf{Pr}\left[1 \leftarrow \mathsf{Expt}_{\mathcal{A}}^{\mathsf{ReEnc}}(1^{\lambda}, 1)\right]\right| \leq \mathsf{negl}(\lambda).$$

Notice that the definition would be trivially unsatisfiable without the validity check. Unless a reencryption key after k_i is punctured at the ciphertext, then given the reencryption keys along with SK_{q}, the adversary can recover m_i^b. Therefore, we require the additional condition that for any such i, $\mathsf{m}_i^0 = \mathsf{m}_i^1$.

2.7.2.4 Fresh-Stale Indistinguishability

Our scheme will enjoy one more security property: that for secret keys SK_1 and SK_2 with reencryption key $\mathsf{rk}_{1,2}$, "fresh" ciphertexts produced with SK_2 are indistinguishable from ciphertexts (of the same message) encrypted under SK_1 and then reencrypted using $\mathsf{rk}_{1,2}$, even *given* SK_1 *and* SK_2.

Definition 2.7.7 (Fresh-stale indistinguishability)**.** ReEnc *is said to satisfy* fresh-stale indistinguishability *if, for all messages* m *and randomness r^*:*

$$\left(\mathsf{Enc}(\mathsf{SK}_2, \mathsf{m}),\ \mathsf{SK}_1,\ \mathsf{SK}_2,\ \mathsf{PP},\ \mathsf{m},\ r^*\right) \overset{c}{\approx} \left(\mathsf{ReEnc}(\mathsf{rk}_{1,2}^{r^*}, \mathsf{Enc}(\mathsf{SK}_1, \mathsf{m})),\ \mathsf{SK}_1,\ \mathsf{SK}_2,\ \mathsf{PP},\ \mathsf{m},\ r^*\right)$$

where the probabilities are taken over $\mathsf{PP} \leftarrow \mathsf{Setup}(1^{\lambda})$, $\mathsf{SK}_1, \mathsf{SK}_2 \leftarrow \mathsf{KeyGen}$, $\mathsf{rk}_{1,2}^{r^*} \leftarrow \mathsf{ReKeyGen}(\mathsf{SK}_1, \mathsf{SK}_2, r^*)$*, along with* Enc *and* ReEnc.

Note that by a hybrid argument, fresh-stale indistinguishability as we defined for SK_1 and SK_2 can be extended for any polynomial-length chain of secret keys. That is, a fresh encryption under SK_{q} is indistinguishable from q-many reencryptions of a ciphertext through the keys $\mathsf{SK}_1, \ldots, \mathsf{SK}_{\mathsf{q}+1}$, even when given all the keys $\mathsf{SK}_1, \ldots, \mathsf{SK}_{\mathsf{q}+1}$.

Looking ahead, fresh-stale indistinguishability will be useful in proving security (Definition 2.7.6). In Section 2.7.4.1, we construct a puncturable proxy reencryption scheme as described in this section from almost key-homomorphic puncturable PRFs. If we instead instantiated our construction with (perfectly) key homomorphic PRFs, then our scheme would immediately satisfy fresh-stale indistinguishability (in fact, the distributions would be equivalent).

2.7.3 Construction of UGC

Our construction of UGC for general circuits and gate-wise updates proceeds in two steps:

1. First, we present a construction of UGC for general circuits and gate-wise updates that does not achieve update hiding. See Section 2.2.1 for a high-level overview of our techniques.

2. In Section 2.7.5, we present a generic transformation from a non-update-hiding UGC scheme into one that achieves update hiding. Applying this transformation on our UGC scheme constructed in the first step, we obtain an update-hiding UGC scheme for general circuits and gate-wise updates.

We prove the following theorem in this Section:

Theorem 2.7.8. *Suppose* ReEnc *is a puncturable symmetric proxy reencryption scheme and* GC *is an efficient* $\mathsf{PrivIND}_\phi$*-secure gate-by-gate garbling scheme for all circuits. Then there exists an IND-secure, sequentially updatable garbled circuits scheme* UGC *for the class* $\mathcal{C}$ *of updatable circuits with gate-wise updates* $(\mathsf{Upd}_{\mathsf{gate}}, \mathcal{U}_{\mathsf{gate}})$*, which is not update-hiding.*

Applying the transformation in Appendix 2.7.5, we obtain our main result of UGC:

Corollary 2.7.9. *Under the same assumptions as Theorem 2.7.8, there is an update-hiding, IND-secure, sequentially updatable garbled circuits scheme* UGC *for the same class of circuits and updates.*

Remark 2.7.10. Our techniques can be easily adapted to realize updatable garbled circuits for the class of "bit-wise" updates, in which updating a circuit C modifies some subset of the bits of the underlying binary representation $\mathsf{binary}(C)$ of the circuit. To achieve this, the updatable garbling of C consists of a garbled universal circuit, along with puncturable proxy reencryption ciphertexts of the garbled input labels corresponding to $\mathsf{binary}(C)$. The remainder of the construction is modified appropriately.

The rest of this Section is devoted to the proof of Theorem 2.7.8.

2.7.3.1 Proof of Theorem 2.7.8

Let ReEnc be a puncturable symmetric proxy reencryption scheme with the properties outlined in Section 2.7.2, and let GC be an efficient, $\mathsf{PrivIND}_\phi$-secure projective garbling scheme for all circuits (Definition 2.3.2). We construct a non-update-hiding updatable garbling scheme $\mathsf{UGC} = (\mathsf{GrbCkt}, \mathsf{GrbInp}, \mathsf{GenUpd}, \mathsf{ApplyUpd}, \mathsf{EvalGC})$ as follows.

$\mathsf{GrbCkt}(1^\lambda, C)$: On input a circuit C, do the following.

1. Generate the garbled circuit and state $(\langle C \rangle_{\mathsf{gc}}, \mathsf{st}_{\mathsf{gc}}) \leftarrow \mathsf{GC.GrbCkt}(C)$.
2. Sample randomness $\mathbf{r}_{\mathsf{reEnc}} = (r_1, \ldots, r_s)$ for ReEnc encryption. Sample public parameters $\mathsf{PP} \leftarrow \mathsf{ReEnc.Setup}(1^\lambda)$ and secret key $\mathsf{SK} \leftarrow \mathsf{ReEnc.KeyGen}(\mathsf{PP})$ for the reencryption scheme.
3. Let $s = |C|$ be the number of gates in C. For each gate $g \in [s]$, let $\langle C \rangle_{\mathsf{gc}}^g$ be the gth garbled gate. For each $g \in [s]$, encrypt $\mathsf{CT}_g = \mathsf{Enc}(\mathsf{SK}, \langle C \rangle_{\mathsf{gc}}^g;\ r_g)$, using r_g as the encryption randomness.
4. Output $\langle C \rangle_{\mathsf{ugc}} := (\mathsf{PP}, \mathsf{CT}_1, \ldots, \mathsf{CT}_s)$ and $\mathsf{st}_{\mathsf{ugc}} = (\mathsf{st}_{\mathsf{gc}}, \mathsf{PP}, \mathsf{SK}, \mathbf{r}_{\mathsf{reEnc}})$.

$\mathsf{GenUpd}(\mathsf{st}_{\mathsf{ugc}}, \mathbf{u})$: On input a state $\mathsf{st} = (\mathsf{st}_{\mathsf{gc}}, \mathsf{SK}, \mathbf{r}_{\mathsf{reEnc}})$ and update $\mathbf{u} = (g, \mathsf{gateType})$, do the following.

1. Sample a fresh secret key $\mathsf{SK}' \leftarrow \mathsf{KeyGen}(1^\lambda)$. Generate the reencryption key $\mathsf{rk} \leftarrow \mathsf{ReKeyGen}(\mathsf{SK}, \mathsf{SK}', r_g)$, punctured at r_g, the gth element of $\mathbf{r}_{\mathsf{reEnc}}$.

2. Sample new randomness r'_g for $\mathsf{ReEnc.Enc}$. Let $\mathbf{r}'_{\mathsf{reEnc}}$ be $\mathbf{r}_{\mathsf{reEnc}}$ with r_g replaced by r'_g.

3. Compute the new garbled gate $\langle C' \rangle^g_{\mathsf{gc}} = \mathsf{GC.GrbGate}(\mathsf{st}, g, \mathsf{gateType})$, and encrypt the new garbled gate $\mathsf{CT}'_g = \mathsf{Enc}(\mathsf{SK}, \langle C' \rangle^g_{\mathsf{gc}}; r'_g)$.

4. Output the garbled update $\langle \mathbf{u} \rangle_{\mathsf{ugc}} := (g, \mathsf{CT}'_g, \mathsf{rk}^{r_g})$ and the updated state $\mathsf{st}'_{\mathsf{ugc}} = (\mathsf{st}_{\mathsf{gc}}, \mathsf{SK}', \mathbf{r}'_{\mathsf{reEnc}})$.

$\mathsf{ApplyUpd}(\langle C \rangle_{\mathsf{ugc}}, \langle \mathbf{u} \rangle_{\mathsf{ugc}})$: On input a UGC-garbled circuit $\langle C \rangle_{\mathsf{ugc}} = (\mathsf{CT}_1, \ldots, \mathsf{CT}_s)$ and a garbled update $\langle \mathbf{u} \rangle_{\mathsf{ugc}} = (g, \mathsf{CT}'_g, \mathsf{rk})$:

1. For each $j \neq g$, let $\mathsf{CT}'_j = \mathsf{ReEnc}(\mathsf{rk}, \mathsf{CT}_j)$.

2. Output $\langle C \rangle'_{\mathsf{ugc}} := (\mathsf{CT}'_1, \ldots, \mathsf{CT}'_s)$

$\mathsf{GrbInp}(\mathsf{st}_{\mathsf{ugc}}, x)$: On input UGC state $\mathsf{st}_{\mathsf{ugc}} = (\mathsf{st}_{\mathsf{gc}}, \mathsf{SK}, \mathbf{r}_{\mathsf{reEnc}})$ and an input $x \in \{0,1\}^\lambda$:

1. Using $\mathsf{st}_{\mathsf{gc}}$, generate the garbled input $\langle x \rangle_{\mathsf{gc}} \leftarrow \mathsf{GC.GrbInp}(\mathsf{st}_{\mathsf{gc}}, x)$.

2. Return $\langle x \rangle_{\mathsf{ugc}} = (\langle x \rangle_{\mathsf{gc}}, \mathsf{SK})$.

$\mathsf{GC.EvalGC}(\langle C \rangle_{\mathsf{ugc}}, \langle x \rangle_{\mathsf{gc}})$: On input input $\langle C \rangle_{\mathsf{ugc}} = (\mathsf{CT}_1, \ldots, \mathsf{CT}_s)$ and $\langle x \rangle_{\mathsf{ugc}} := (\langle x \rangle_{\mathsf{gc}}, \mathsf{SK})$, do as follows.

1. Decrypt the garbled circuit: $\langle C \rangle_{\mathsf{gc}} = (\mathsf{Dec}(\mathsf{SK}, \mathsf{CT}_1), \ldots, \mathsf{Dec}(\mathsf{SK}, \mathsf{CT}_s))$.

2. Return $\mathsf{EvalGC}(\langle C \rangle_{\mathsf{gc}}, \langle x \rangle_{\mathsf{gc}})$.

Correctness. Consider a circuit $C \in \mathcal{C}_\lambda$, input $x \in \{0,1\}^\lambda$, and a sequence of updates $\mathbf{u}_1, \ldots, \mathbf{u}_{\mathsf{q}}$. We need to show that $\mathsf{EvalGC}\left(\langle C_{\mathsf{q}} \rangle_{\mathsf{ugc}}, \langle x \rangle_{\mathsf{ugc}}\right) = C_{\mathsf{q}}(x)$.

Proof. We define notation used in the secure updating process. The state for each $i \in [\mathsf{q}]$, $\mathsf{st}^i_{\mathsf{ugc}} = (\mathsf{st}^i, \mathsf{SK}_i, \mathbf{r}^i_{\mathsf{reEnc}})$. For each i, the ith garbled update consists of $\langle \mathbf{u}_i \rangle_{\mathsf{ugc}} = (g_i, \mathsf{CT}^i_{g_i}, \mathsf{rk}^{r^i_{g_i}}_{i,i+1})$, where $g_i \in [s]$ is the gate of C updated in $\mathbf{u}_i$, and $r^i_{g_i}$ is the randomness r_{g_i} from $\mathbf{r}^i_{\mathsf{reEnc}}$ on which $\mathsf{rk}_{i,i+1}$ is punctured, and $\mathsf{CT}^i_{g_i}$ is an encryption of the new garbled gate for the input g_i encrypted under the key SK_i.

By the correctness of the circuit garbling scheme GC, it suffices to show that $\langle C_{\mathsf{q}} \rangle_{\mathsf{gc}} = (\mathsf{Dec}(\mathsf{SK}_{\mathsf{q}}, \mathsf{CT}_1^{\mathsf{q}}), \ldots, \mathsf{Dec}(\mathsf{SK}_{\mathsf{q}}, \mathsf{CT}_s^{\mathsf{q}}))$ computed during EvalGC are indeed the correct garbled gates. We will show correctness for $\mathsf{CT}_1^{\mathsf{q}}$; the argument extends naturally to the remaining ciphertexts.

$\mathsf{CT}_1^{\mathsf{q}}$ is computed by a series of reencryptions using the various punctured reencryption keys starting from a "source" encryption under some earlier "source" key. This source encryption may have been computed using SK_0, or using SK_j if $\mathbf{u}_j$ modified the first gate C. By construction, this source ciphertext encrypted the correct garbled gate $\langle C_j \rangle_{\mathsf{gc}}^1 := \mathsf{GC.GrbGate}(\mathsf{st}, g, \mathsf{gateType})$. Therefore we must show that the various reencryptions, and the final decryption with key SK_{q}, do not introduce any errors.

Let r_1^* be the randomness used to encrypt the source ciphertext under the source key. By the correctness of the proxy reencryption scheme ReEnc and because q is polynomially bounded, $\mathsf{Dec}(\mathsf{SK}_{\mathsf{q}}, \mathsf{CT}_1^{\mathsf{q}})$ correctly outputs $\langle C_{\mathsf{q}} \rangle_{\mathsf{gc}}^1$ as long as none of the reencryption keys were punctured at r_1^*.

With probability at least $1 - (s + \mathsf{q})/(2^\lambda) = 1 - \mathsf{negl}(\lambda)$, all choices of randomness r ever sampled for $\mathsf{ReEnc.Enc}$ are unique, where $s + \mathsf{q}$ is the total number of fresh encryptions computed. Conditioning on uniqueness (i.e., with high probability), none of the reencryption keys are punctured at r_1^*, completing the proof. $\square$

Efficiency. We lay out different efficiency properties associated with the above scheme.

- *Garbling Time*: The parallel-time (depth) to compute $\mathsf{UGC.GrbCkt}(1^\lambda, C)$ is equal to $T_{\mathsf{GrbCkt}}(1^\lambda, \mathcal{C}) + \mathrm{poly}(\lambda)$, where $T_{\mathsf{GrbCkt}}(1^\lambda, \mathcal{C})$ is the time needed to garble the universal circuit for $\mathcal{C}$. The second term captures the overhead of encrypting each garbled gate (which may be done in parallel).

  Using an underlying garbling scheme with GrbCkt in NC^1 (for example, Yao's garbling scheme [Yao86]), the parallel-time (depth) for computing $\mathsf{UGC.GrbCkt}(1^\lambda, C)$ is $\mathrm{poly}(\lambda, \log |C|)$.

The parallel-time to compute $\mathsf{UGC.GrbInp}(\mathsf{st}_{\mathsf{ugc}}, x)$ is $T_{\mathsf{GrbInp}}(\mathsf{st}_{\mathsf{gc}}, x) + \mathrm{poly}(\lambda)$, where $T_{\mathsf{GrbInp}}(\mathsf{st}_{\mathsf{gc}}, x)$ is the time needed to garble the input x. Using Yao's garbling scheme, the time to compute the garbled input is $\mathrm{poly}(\lambda, |x|)$, independent of $|C|$ and the number of updates.

- *Secure Update Generation Time*: The time to generate an update is $\mathrm{poly}(\lambda)$, independent of the size of the circuit C.

- *Secure Update Size*: The size of a garbled update is $|\langle \mathbf{u} \rangle_{\mathsf{ugc}}| = \mathrm{poly}(\lambda)$, independent of $|C|$ and the history of updates.

- *State Size*: The size of garbler's state is $|C| \cdot \mathrm{poly}(\lambda)$. Though it grows with $|C|$, it is independent of the history of updates. Using the transformation described in Appendix 2.5.3, it is possible to remove the dependence on $|C|$.

- *Runtime of Update*: The runtime of $\mathsf{ApplyUpd}$ is $\mathrm{poly}(|C|, \lambda)$. It depends on $|C|$, but is independent of the number of updates performed.

IND-security. The security of UGC follows directly from the security of the proxy reencryption scheme ReEnc.

In the security game, the adversary sends circuits C^0 and C^1, inputs x^0 and x^1, and sequences of updates $\{(\mathbf{u}_j^0, \mathbf{u}_j^1)\}_{j=1}^{\mathsf{q}}$ such that the gates being updated match (i.e., $\phi(\mathbf{u}_j^0) = \phi(\mathbf{u}_j^1)$ for all j). In response, the adversary receives a sequence of ciphertexts, punctured reencryption keys, the secret key $\mathsf{SK_q}$, and a garbled input.

For each $\mathsf{gateType} \in \{\vee, \wedge, \neg\}$, and gate $g \in [s]$, let $\langle \mathsf{gateType} \rangle_{\mathsf{gc}}^{g} =$ $\mathsf{GC.GrbGate}(\mathsf{st}, g, \mathsf{gateType})$ be the gth garbled gate for type $\mathsf{gateType}$. Then for each $g \in [s]$, the adversary receives a series of encryptions of $\langle \mathsf{gateType} \rangle_{\mathsf{gc}}^{g}$: one ciphertext from the initial garbling GrbCkt, and subsequent ciphertexts generated by GenUpd whenever an update $\mathbf{u}$ alters the gth gate of C.

By construction, each update for the gth gate of C consists of a new encryption of $\langle \mathsf{gateType} \rangle_{\mathsf{gc}}^{g}$ for some $\mathsf{gateType}$, along with a reencryption key that is punctured at the previous ciphertext corresponding to gate g. For every g, there is a single "most up-to-date" encryption CT^*_g of $\langle \mathsf{gateType}^b_{\mathsf{q}} \rangle^g_{\mathsf{gc}}$, where $\mathsf{gateType}^b_{\mathsf{q}}$ is the gth gate of the final circuit C^b_{q}.

It suffices to show that the adversary's view in the real security game is indistinguishable from the view in a modified security game in which all other ciphertexts CT'_g (those that are not the most up-to-date) are in fact generated as $\mathsf{CT}'_g \leftarrow \mathsf{Enc}(\mathsf{SK}, 0)$, encryptions of 0.

That this is sufficient follows directly from the $\mathsf{PrivIND}_\phi$ security of the garbling scheme GC. The adversary's view in the modified game can be generated given $\langle C^b_{\mathsf{q}} \rangle_{\mathsf{gc}} \leftarrow \mathsf{GrbCkt}(1^\lambda, C^b_{\mathsf{q}})$ and $\langle x^b \rangle_{\mathsf{gc}} \leftarrow \mathsf{GrbInp}(1^\lambda, x^b)$, where C^b_{q} is C^b with all updates $\mathbf{u}^b_j$ applied. Because $C^0(x^0) = C^1(x^1)$, the $\mathsf{PrivIND}_\phi$ security of GC implies that the bit b is computationally hidden.

Indistinguishability of these two games follows from the security of ReEnc. Observe that for any CT'_g that is not the most up-to-date, there exists a later update that modifies the gate g. By construction, when that update is issued, the corresponding reencryption key will be punctured at the ciphertext CT'_g. Indistinguishability maps directly to the security of ReEnc. One sequence of messages are the real sequence of garbled gates used to generate the encryptions of CT^*_g and CT'_g for each g, while the second is the sequence of labels where all but the most up-to-date are replaced with 0. Similarly, the sequence of punctured key requests $(p_1, \ldots, p_{\mathsf{q}})$ and the key-indices $(k_1, \ldots, k_\ell)$ (where $\ell = s + \mathsf{q}$) as defined in Definition 2.7.6 correspond to the sequence reencryption keys and fresh encryptions generated by GenUpd.

2.7.4 Construction of Puncturable PRE

We use puncturable almost key-homomorphic PRFs [BLMR13b, BV15b] to build updatable garbled circuits. In [BLMR13b], the authors construct a symmetric proxy reencryption scheme from any almost key-homomorphic PRF. We similarly construct a *puncturable*, symmetric proxy reencryption scheme given a *puncturable* almost key-homomorphic PRF, which can be based on worst-case lattice assumptions as shown in the recent work of [BV15b]. Our definition of puncturable, almost key-homomorphic PRFs adapted from [BLMR13b] and recall the main theorem of [BV15b]. We restrict our attention to PRFs with the particular co-domain $\mathbb{Z}_p$ for some integer and p.

Definition 2.7.11 (Puncturable, γ-Almost Key Homomorphic PRF). *Let* $\mathsf{PRF} = (\mathsf{PRF.Setup}, \mathsf{PRF.KeyGen}, \mathsf{PRF.Punct}, F)$ *be point-puncturable PRF family,*[14] *with key space* $\mathcal{K}_\lambda$, *domain* $\mathcal{X}_\lambda$, *and co-domain* $\mathbb{Z}_p$, *for* $\lambda \in \mathbb{N}$, *and some integer* p. *Suppose that* $(\mathcal{K}_\lambda, +)$ *and* $(\mathcal{Y}, +)$ *are groups. The puncturable PRF family is* γ-almost key homomorphic *if for all points* x_1^*, *and* x_2^*, *and all keys* K_1 *and* K_2, *and all inputs* $x \neq X_1^*, x^*, 2$, *there exists an* $e \in [0, \gamma]$ *such that*

$$F(\mathsf{PRF.Punct}(K_1, x_1), x) + F(\mathsf{PRF.Punct}(K_2, x_2^*), x) = F(K_1 + K_2, x) \pm e \pmod{p}.$$

The security requirements of punctured PRFs vary from construction to construction (e.g. the number of punctured keys, the selectivity / adaptivity of punctured points). We consider a weak security notion: selective security with respect to only a single punctured key. Such PRFs were constructed in [BV15b].

Theorem 2.7.12 ([BV15b], Theorem 5.1). *For* $c > 0$, *a single-key, selectively secure point-puncturable, 2-almost key homomorphic PRF with domain* $\{0,1\}^n$ *and co-domain* $\mathbb{Z}_p$ *can be constructed based on the hardness of approximating either* GapSVP *or* SIVP *to within a factor of* $2^{\tilde{O}(n^{1/c})}$, *where* $n = (\lambda \log \lambda)^c$, $p = 2^{\tilde{O}(n^{1/c})}$.

Proof. We observe that the family of circuits that check whether an input $r \in \{0,1\}^\lambda$ is equal to a specific r^* can be uniformly generated, and each such circuit has depth $O(\log \lambda)$ and a description of size λ (the description being r^* itself).

Applying the choice of parameters in Section 5.2 of [BV15b], along with the observations in Section 5.5 of that work yields the corollary. □

Finally, we remark that this construction has a property that will be important for us in the proof of security for ReEnc. Namely, for all $\mathsf{PP} \leftarrow \mathsf{PRF.Setup}(1^\lambda)$, the following two distributions over the keyspace $\mathcal{K}_\lambda$ are identical:

$$\mathsf{PRF.KeyGen}(\mathsf{PP}) \equiv \mathsf{Uniform}(\mathcal{K}_\lambda) \tag{2.2}$$

[14]A puncturable PRF family is a constrained PRF family (see [BW13, BV15b]) for the family of constraints which are equal to 1 except at a single "punctured" point.

2.7.4.1 Construction of Puncturable Symmetric Proxy Reencryption

We now present a puncturable reencryption scheme that satisfies all the properties discussed in Section 2.7.2, closely modeled on [BLMR13b].[15] The main ingredient in the construction is a puncturable almost key-homomorphic PRF scheme that additionally satisfies the property stated in equation 2.2.

Let PRF be the family of puncturable, 2-almost key homomorphic PRFs from Corollary 2.7.12. The domain and range of PRF are $\{0,1\}^n$ and $\mathbb{Z}_p$ respectively, where $n = \text{poly}(\lambda)$. Let $B = \lambda^{\log \lambda}$ and let u be an integer such that $\lfloor p/u \rfloor > 6B$ (recall that $p = 2^{\widetilde{O}(n^{1/c})}$).

Construction. We define $\mathsf{ReEnc} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{ReKeyGen}, \mathsf{Enc}, \mathsf{ReEnc}, \mathsf{Dec})$ to be an encryption scheme with message space $\mathbb{Z}_u$ as follows:

- $\mathsf{Setup}(1^\lambda)$: Sample and output the public parameters of the PRF $\mathsf{PP} \leftarrow \mathsf{PRF.Setup}(1^\lambda)$.
- $\mathsf{KeyGen}(1^\lambda)$: Output a secret key $\mathsf{SK} \leftarrow \mathsf{PRF.KeyGen}(1^\lambda)$.
- $\mathsf{Enc}(\mathsf{SK}, \mathsf{m})$: Sample $r \leftarrow \{0,1\}^n$ and noise $\eta \leftarrow [-B, B]$, and output (r, pl) where $\mathsf{pl} = m \cdot \lfloor p/u \rfloor + F(\mathsf{SK}, r) + \eta$.
- $\mathsf{Dec}(\mathsf{SK}, (r, \mathsf{pl}))$: Output $\lfloor \mathsf{pl} - F(\mathsf{SK}, r) \pmod p \rceil_u$, where $\lfloor \cdot \rceil_u$ denotes rounding to the nearest multiple of $\lfloor p/u \rfloor$.
- $\mathsf{ReKeyGen}(\mathsf{SK}_1, \mathsf{SK}_2, r^*)$: Output $\mathsf{rk}_{1,2}^{r^*} \leftarrow \mathsf{PRF.Punct}(\mathsf{SK}_2 - \mathsf{SK}_1, r^*)$.
- $\mathsf{ReEnc}(\mathsf{rk}, (r, \mathsf{pl}))$: Output $(r, \mathsf{pl} + F(\mathsf{rk}, r))$.

Correctness. Correctness follows in a straightforward manner from the γ-almost key-homomorphic guarantee of PRF and the choice of u. This is because the magnitude of the total accumulated error (from $T = \text{poly}(\lambda)$ reencryptions and the encryption noise η) cannot exceed $(2T + B) < 3B$ and thus the error does not affect decryption correctness.

[15] As in that work, the scheme would be simplified by using key homomorphic PRFs, rather than *almost* key homomorphic PRFs. Because only the latter are known to exist from lattice assumptions, we present only that construction.

Fresh-stale indistinguishability. We wish to show that for any m, SK_1, SK_2, and r^* it is infeasible to determine whether a given ciphertext $\mathsf{CT} = (r, \mathsf{pl})$ was generated according to $\mathsf{CT} = \mathsf{CT}_2 \leftarrow \mathsf{Enc}(\mathsf{SK}_2, \mathsf{m})$, or according to $\mathsf{CT} = \mathsf{CT}_1 \leftarrow \mathsf{ReEnc}(\mathsf{ReKeyGen}(\mathsf{SK}_1, \mathsf{SK}_2, r^*), \mathsf{Enc}(\mathsf{SK}_1, \mathsf{m}))$. Note that in our scheme, both $\mathsf{ReKeyGen}$ and ReEnc are deterministic algorithms. Intuitively, the error from reencrypting the ciphertext instead of computing it fresh is drowned out by the noise η added to the ciphertext during encryption.

Let $c, B, \Delta \in \mathbb{Z}_p$ such that $B < p/4$ and $c < p/4$. Let D_2 be the uniform distribution over $[c - B, +B]$, and let D_1 be uniform over $[c - B + \Delta, c + B + \Delta]$. The statistical distance of these distributions is $\mathsf{SD}(D_1, D_2) = \Delta/(2B + 1)$.

Fix any choice of randomness r for Enc, and any choice of SK_1, SK_2, r^* and m. Let $c = m \cdot \lfloor p/u \rfloor + F(\mathsf{SK}_2, r)$ and $\Delta = |F(\mathsf{SK}_2, r) - (F(\mathsf{SK}_1, r) + F(\mathsf{PRF.Punct}(\mathsf{SK}_2 - \mathsf{SK}_1, r^*), r))|$ $\leq 2\gamma$. Consider the distribution (induced by the choice of noise $\eta \in [-B, B]$) over ciphertexts $\mathsf{CT}_1 = (r, \mathsf{pl}_1)$ and $\mathsf{CT}_2 = (r, \mathsf{pl}_2)$ defined as above. CT_1 is distributed according to D_1 and CT_2 is distributed according to D_2. Therefore, the distributions over CT_1 and CT_2 induced by the choice of noise η by Enc have statistical distance bounded by $2\gamma/(2B + 1)$. Because B is chosen to be superpolynomial, this statistical distance is negligible.

Security. Suppose for contradiction that there existed an adversary $\mathcal{A}$ for which $\mathsf{Expt}_{\mathcal{A}}^{\mathsf{ReEnc}}(1^\lambda, 0)$ and $\mathsf{Expt}_{\mathcal{A}}^{\mathsf{ReEnc}}(1^\lambda, 1)$ were distinguishable. First, we observe that fresh-stale indistinguishability implies that, without loss of generality we may assume that for all $i \in [\ell]$, $k_i = 0$ (i.e. all the messages m_i^b are originally encrypted under SK_1 instead of SK_{k_i}). This follows directly from the observation that fresh-stale indistinguishability extends to chains of secret keys of polynomial length (see note after Definition 2.7.7), along with the requirement that if $p_j = i$, then $j > i$. If some $k_i > 0$, then the view of that adversary can be indistinguishably simulated by using $k_i' = 1$, getting the encryption of m_i^b, then using the reencryption keys to compute a ciphertext with respect to key SK_{k_i}. Because the reencryption keys $\mathsf{rk}_{1,2}, \ldots, \mathsf{rk}_{i-1,i}$ are not punctured at the ciphertext CT_i (by the restriction on p_j previously mentioned), this simulation will succeed.

Second, by a simple hybrid argument, we may also assume without loss of generality that the sequences of messages $(\mathsf{m}_1^0, \ldots, \mathsf{m}_\ell^0)$ and $(\mathsf{m}_1^1, \ldots, \mathsf{m}_\ell^1)$ differ at exactly 1 index i^*. For all $i \neq i^*$, let $\mathsf{m}_i = \mathsf{m}_i^0 = \mathsf{m}_i^1$. We may also assume that there exists $j^* \in [\mathsf{q}]$ such that $p_{j^*} = i^*$; otherwise $\mathcal{A}$'s challenge is invalid and the challenger aborts.

We use $\mathcal{A}$ to violate the single-key, selective security of the punctured PRF, yielding a contradiction. Given the public parameters PP of PRF, sample a uniform PRF input r^*, and receive in turn the punctured PRF challenge: a punctured key K^{r^*} and y^*, which is either $F(K, r^*)$ or a uniformly random value in the co-domain $\mathbb{Z}_p$.

Pass along the public parameters to $\mathcal{A}$ and receive in return $(\mathsf{m}_1, \ldots, \mathsf{m}_{i^*-1}, \mathsf{m}_{i^*}^0, \mathsf{m}_{i^*}^1, \mathsf{m}_{i^*+1}, \ldots, \mathsf{m}_\ell)$, and $(p_1, \ldots, p_\mathsf{q})$ with $p_{j^*} = i^*$.

For $j \neq j^*$, sample PRF keys $\mathsf{rk}_{j,j+1} \leftarrow \mathsf{PRF.KeyGen}(\mathsf{PP})$ (which we identify with reencryption keys), and a final PRF key $\mathsf{SK}_{\mathsf{q}+1} \leftarrow \mathsf{PRF.KeyGen}(\mathsf{PP})$ (interpreted as the terminal secret key in the ReEnc security game). Lastly, set $\mathsf{rk}_{j^*,j^*+1}^{r^*} = K^{r^*}$, the challenge punctured PRF key.

For each $i \neq i^*$, sample encryption randomness r_i uniformly and compute the ciphertext $\mathsf{CT}_i^{\mathsf{q}+1} = \mathsf{Enc}(\mathsf{SK}_{\mathsf{q}+1}, \mathsf{m}_i; r_i)$. Then, compute

$$\mathsf{CT}_i = \mathsf{CT}_i^{\mathsf{q}+1} - F(\mathsf{rk}_{j^*,j^*+1}^{r^*}, r_i) - \sum_{j \neq j^*} F(\mathsf{rk}_{j,j+1}, r_i) \tag{2.3}$$

For i^*, pick $b \leftarrow \{0,1\}$ uniformly, and compute the ciphertext $\mathsf{CT}_i^{\mathsf{q}+1} = \mathsf{Enc}(\mathsf{SK}_{\mathsf{q}+1}, \mathsf{m}_i^b; r_i)$. Then compute

$$\mathsf{CT}_{i^*} = \mathsf{CT}_{i^*}^{\mathsf{q}+1} - y^* - \sum_{j \neq j*1} F(\mathsf{rk}_{j,j+1}, r_{i^*})$$

Finally, compute the punctured reencryption keys $\mathsf{rk}_{j,j+1}^{r_{p_j}} \leftarrow \mathsf{PRF.Punct}(\mathsf{rk}_{j,j+1}, r_{p_j})$.

As in the security definition for punctured proxy reencryption, return to $\mathcal{A}$ the CT_i for all i, $\mathsf{rk}_{j,j+1}^{r_{p_j}}$ for all j, and $\mathsf{SK}_{\mathsf{q}+1}$. By Equation 2.2, the reencryption keys $\mathsf{rk}_{j,j+1}^{r_{p_j}}$ and the secret key $\mathsf{SK}_{\mathsf{q}+1}$ are distributed exactly as in the actual ReEnc security game. On the other hand, the ciphertexts CT_i are not distributed like honest ciphertexts, even for $i \neq i^*$. This is because the PRF is only *almost* key-homomorphic—there's no guarantee that $F(\sum \mathsf{rk}_{j,j+1}, r_i)$ is indistinguishable from $\sum F(\mathsf{rk}_{j,j+1}, r_i)$ when also given the (punctured) reencryption keys.

We again use fresh-stale indistinguishability. If instead of generating the $\mathsf{CT}_i^{\mathsf{q}+1}$ as we did by encrypting directly using $\mathsf{SK}_{\mathsf{q}+1}$, the $\mathsf{CT}_i^{\mathsf{q}+1}$ had been generated as reencryptions of ciphertexts $\mathsf{CT}_i \leftarrow \mathsf{Enc}(\mathsf{SK}_1, \mathsf{m}_i)$ using the punctured reencryption keys (i.e. if the $\mathsf{CT}_i^{\mathsf{q}+1}$ were "stale"), then the transformation we compute in (2.3) would exactly recover the fresh ciphertexts CT_i. On the other hand, the reduction used in the proof uses "fresh" ciphertexts $\mathsf{CT}_i^{\mathsf{q}+1}$. Therefore an adversary $\mathcal{A}$ who could distinguish its view in the proof, where the CT_i are generated as in (2.3) from fresh encryptions $\mathsf{CT}_i^{\mathsf{q}+1} \leftarrow \mathsf{Enc}(\mathsf{SK}_{\mathsf{q}+1}, \mathsf{m}_i)$ could be used to violate fresh-stale indistinguishability. Given a fresh-or-stale ciphertext $\mathsf{CT}_i^{\mathsf{q}+1}$ and all the secret keys SK_j, it is easy to generate the reencryption keys in the view of the adversary.

Therefore, we are able to conclude that the view of the adversary in this proof of ReEnc security is indistinguishable from the real-world view, and the adversary (by assumption) must succeed with non-negliglbe probability. Now, if punctured PRF challenge value $y^* = F(K, r^*)$, then the view of the adversary is indistinguishable from its view in $\mathsf{Expt}_{\mathcal{A}}^{\mathsf{ReEnc}}(1^\lambda, b)$ with random b. On the other hand, if y^* is random, then CT_{i^*} is distributed uniformly, and hides the value of the bit b. Therefore, if $\left|\Pr[1 \leftarrow \mathsf{Expt}_{\mathcal{A}}^{\mathsf{ReEnc}}(1^\lambda, 0)] - \Pr[1 \leftarrow \mathsf{Expt}_{\mathcal{A}}^{\mathsf{ReEnc}}(1^\lambda, 1)]\right|$ is non-negliglbe, we are able to distinguish the case when $y^* = F(K, r^*)$ or is random, violating the security of PRF and completing the proof.

2.7.5 Achieving Update Hiding Generically

We describe how to achieve update hiding generically. We give this transformation for the general case of updatable randomized encodings but the same transformation works even for updatable garbled circuits. The main idea is to use a non-interactive write-only oblivious RAM scheme to achieve this. We define this notion below.

Non-interactive write-only ORAM. A scheme $\mathsf{wNIORAM}$ consists of the following algorithms.

Database Encoding, $\mathsf{EncDB}(1^\lambda, D)$: On input security parameter λ, database D, it produces an encoded $\widetilde{D}$ and secret key osk. The database encoding is given to the server by the client. The client hides osk from the server.

Query encoding, $\mathsf{EncQ}(\mathsf{osk}, q)$: On input secret key osk and write query q, it produces program encoding $\widetilde{q}$. The client sends the query encoding to the server. Here, we only consider write queries of the form $(index, b)$.

Updating, $\mathsf{Upd}(\widetilde{q}, \widetilde{D})$: On input query encoding $\widetilde{q}$ and database encoding $\widetilde{D}$, it produces an updated database encoding $\widetilde{D'}$. Note that the server can execute this procedure non-interactively and in particular, this does not involve any communication with the client.

Correctness is as in an oblivious RAM scheme—updating an encoding of the database using an encoded query is equivalent to updating the underlying database and then encoding it. The security guarantee (as in any ORAM scheme) is that the access pattern does not leak any information about the underlying query.

We are interested in $\mathsf{wNIORAM}$ schemes satisfying two properties:

1. The query encoding $\widetilde{q}$ is a tuple of pairs of the form $(index, b)$ and the updating algorithm substitutes b in $index^{th}$ position of $\widetilde{D}$ for every $(index, b)$ in $\widetilde{q}$. We call such schemes *bitwise* $\mathsf{wNIORAM}$ schemes.

2. Another property we are interested is *decodability*: given a valid database $\widetilde{D}$ and secret key osk, there is a public algorithm to recover D correctly.

The garbled RAM scheme of [GLOS15] yields a bitwise $\mathsf{wNIORAM}$ schemes that is decodable. Furthermore, their scheme is based on one-way functions. We thus have the following theorem.

Theorem 2.7.13. *There exists decodable bitwise* $\mathsf{wNIORAM}$ *schemes assuming one-way functions.*

2.7.5.1 Update-Hiding Transformation

Suppose we have an updatable randomized encoding scheme that is not update hiding, denoted $\mathsf{URE}_{\mathsf{nuh}}$. We show how to construct update hiding updatable RE URE

$\mathsf{Encode}\left(1^\lambda, C, x\right)$: First, it computes $(\widetilde{(C,x)}, \mathsf{osk}) \leftarrow \mathsf{EncDB}(1^\lambda, (C,x))$. It executes $(\mathsf{URE}_{\mathsf{nuh}}.\langle C^*[\bot]\rangle_{\mathsf{ure}}, \mathsf{st}) \leftarrow \mathsf{URE}_{\mathsf{nuh}}.\mathsf{Encode}(1^\lambda, C^*, \bot)$, where C^* (with $\widetilde{(C,x)}$ and osk hardwired into it) is defined as follows. C^* first decodes $\widetilde{(C,x)}$ using osk to recover (C,x) (using decodability) and then it outputs $C(x)$. Finally, it outputs $\mathsf{URE}_{\mathsf{nuh}}.\langle C^*[\bot]\rangle_{\mathsf{ure}}$ as the randomized encoding and it sets the state $\mathsf{st} = \mathsf{osk}$.

$\mathsf{GenUpd}\left(\mathsf{st}, \mathbf{u}\right)$: It first encodes $\mathbf{u}$ using the query encode algorithm EncQ to obtain $\widetilde{\mathbf{u}}$. It then compiles $\widetilde{\mathbf{u}}$ into an encoding $\mathsf{URE}_{\mathsf{nuh}}.\langle\widetilde{\mathbf{u}}\rangle_{\mathsf{ure}}$ by executing $\langle\widetilde{\mathbf{u}}\rangle_{\mathsf{ure}} \leftarrow \mathsf{URE}_{\mathsf{nuh}}.\mathsf{GenUpd}(\mathsf{st}, \widetilde{\mathbf{u}})$. The encoding of update is set to be $\mathsf{URE}_{\mathsf{nuh}}.\langle\widetilde{\mathbf{u}}\rangle_{\mathsf{ure}}$. The new state is the same as the old state.

$\mathsf{ApplyUpd}\left(\langle C^*[x]\rangle_{\mathsf{ure}}, \langle\widetilde{\mathbf{u}}\rangle_{\mathsf{ure}}\right)$: This procedure essentially executes $\mathsf{ApplyUpd}$ of the scheme $\mathsf{URE}_{\mathsf{nuh}}$. It executes $\mathsf{URE}_{\mathsf{nuh}}.\mathsf{ApplyUpd}(\mathsf{URE}_{\mathsf{nuh}}.\langle C^*[\bot]\rangle_{\mathsf{ure}}, \mathsf{URE}_{\mathsf{nuh}}.\langle\widetilde{\mathbf{u}}\rangle_{\mathsf{ure}})$ to obtain $\mathsf{URE}_{\mathsf{nuh}}.\langle C^{*\prime}[\bot]\rangle_{\mathsf{ure}}$. It outputs the updated randomized encoding $\mathsf{URE}_{\mathsf{nuh}}.\langle C^{*\prime}[\bot]\rangle_{\mathsf{ure}}$. Here, $C^{*\prime}$ contains the updated ORAM and similar to circuit C^*—it first decrypts the ORAM to get (C', x') and performs the computation $C'(x')$.

$\mathsf{Decode}\left(\langle C[x]\rangle_{\mathsf{ure}}\right)$: This procedure executes the decoding procedure of $\mathsf{URE}_{\mathsf{nuh}}$.

The correctness of $\mathsf{URE}_{\mathsf{nuh}}$ as well as the decodibility of $\mathsf{wNIORAM}$ implies the correctness of the above URE scheme. Furthermore, we can invoke the security of $\mathsf{URE}_{\mathsf{nuh}}$ and the access pattern hiding of $\mathsf{wNIORAM}$ to show the security of the above scheme.

2.8 Updatable Cryptography

We define the notion of updatability in the context of several cryptographic primitives. In Section 2.8.1, we present a generic indistinguishability-based approach through the lens of *circuit compilers* and *dynamic circuit compilers*. This framework captures updatable versions of attribute based encryption, non-interactive witness indistinguishable proofs, and indistinguishability obfuscation. We then move on to two example applications with simulation-based security: updatable non interactive zero knowledge in Section 2.8.4 and updatable multiparty computation in Section 2.8.5.

2.8.1 Circuit Compilers

We introduce the notion of circuit compilers below. It consists of the algorithms $\mathsf{CC} = (\mathsf{Gen}, \mathsf{Compile}, \mathsf{Encode}, \mathsf{Decode})$. Its associated with a class of circuits $\mathcal{C}$.

GENERATION OF PARAMETERS, $\mathsf{Gen}(1^\lambda)$: On input security parameter λ, outputs parameters $(\mathsf{cktSK}, \mathsf{inpSK})$ and public parameters pp.

CIRCUIT COMPILATION, $\mathsf{Compile}(\mathsf{cktSK}, C)$: On input secret parameters cktSK and circuit C, it outputs an encoding of circuit $\langle C \rangle$ and state st.

INPUT ENCODING, $\mathsf{Encode}(\mathsf{inpSK}, x)$: On input parameters inpSK and input x, it outputs an encoding of input $\langle x \rangle$.

EVALUATION, $\mathsf{Eval}(\langle C \rangle, \langle x \rangle)$: On input encodings $\langle C \rangle$ and $\langle x \rangle$, it outputs the decoded value α.

There are two properties associated with a circuit compiler—correctness and security.

Correctness. Consider a circuit C and an input x. We require that the evaluation of encoding $\langle C \rangle$ on $\langle x \rangle$ yields $C(x)$.

ρ-IND security. We define a generic indistinguishability-based selective security notion associated with a ρ-admissibility property. For example, in Yao's garbled circuits, indistinguishability is only guaranteed for circuits that agree on the input and have the same topology. ρ is some boolean function with input $\mathsf{aux}_{\mathsf{cc}}$ consisting of: (i) a pair of circuits C^0, C^1, (ii) additional circuits $C_1, \ldots, C_{\ell_f}$, and (iii) a pair of input sequences $(x_0^0, \ldots, x_{\ell_{\mathsf{inp}}}^0)$, $(x_0^1, \ldots, x_{\ell_{\mathsf{inp}}}^1)$.

Definition 2.8.1 (ρ-IND Security). *A circuit compiler scheme* CC *is* ρ-IND secure *(selectively) for all* $\lambda \in \mathbb{N}$ *and all* $\mathsf{aux}_{\mathsf{cc}}$ *as above,* $\mathsf{Expt}_0^{\mathsf{CC}}(1^\lambda, \mathsf{aux}_{\mathsf{cc}}) \approx_{\mathsf{c}} \mathsf{Expt}_1^{\mathsf{CC}}(1^\lambda, \mathsf{aux}_{\mathsf{cc}})$.

$\mathsf{Expt}_b^{\mathsf{CC}}(1^\lambda, \mathsf{aux}_{\mathsf{cc}})$:

1. If $\rho(\mathsf{aux}_{\mathsf{cc}}) = 0$, abort and output $\perp$.
2. Let $\mathsf{pp}, (\mathsf{cktSK}, \mathsf{inpSK}) \leftarrow \mathsf{Gen}(1^\lambda)$.
3. Generate the circuit encodings $(\langle C^b \rangle, \mathsf{st}_0) \leftarrow \mathsf{Compile}(\mathsf{cktSK}, C^b)$ and $(\langle C_i \rangle, \mathsf{st}_{i,0}) \leftarrow \mathsf{Compile}(\mathsf{cktSK}, C_i)$ for $i \in [\ell_f]$. (The states $\mathsf{st}_{i,0}$ are never used.)
4. Generate the input encodings $\langle x_k^b \rangle \leftarrow \mathsf{Encode}(\mathsf{inpSK}, x_k^b)$ for $k \in [\ell_{\mathsf{inp}}]$.
5. Output
$$\left(\langle C^b \rangle,\ \{\langle C_i \rangle\}_{i \in [\ell_f]},\ \{\langle x_k^b \rangle\}_{k \in [\ell_{\mathsf{inp}}]}\right).$$

2.8.1.1 Instantiations

Several advanced cryptographic primitives can be seen in the form of circuit compilers. We give a couple of examples.

Attribute Based Encryption (ABE): We take $\mathsf{Gen} = \mathsf{Setup}$ (setup), $\mathsf{Compile} = \mathsf{KeyGen}$ (key generation), $\mathsf{Encode} = \mathsf{Enc}$ (encryption), $\mathsf{Eval} = \mathsf{Dec}$ (decryption). Every circuit $C \in \mathcal{C}$ associated with an ABE scheme is of the form: $C(x = (\mathsf{attr}, m)) := m$ if and only if $C'(\mathsf{attr}) = 1$ for some circuit C' hardwired in C. Depending on whether the ABE scheme we are considering is a public key scheme or not, we can assign inpSK to be either public parameters or secret key parameters.

For security, ρ interprets x_i^b as $(\mathsf{attr}_i^b, m_i^b)$ and checks if $\mathsf{attr}_i^0 = \mathsf{attr}_i^1$. It also checks if $C^0 = C^1$. If either of the checks fail, ρ outputs 0.

Non Interactive Witness Indistinguishable Proof Systems (WI): Consider a non interactive witness indistinguishable system $\mathsf{WI} = (P, V)$ associated with a NP relation R. We denote by P the circuit representing the prover. Suppose x is the NP instance and w is the witness associated with x. We take $\mathsf{Compile} = P(x, w; r)$ (prover circuit with instance, witness, randomness hardwired), $\mathsf{Eval} = V$ (verifier circuit), and Gen and Encode are undefined,

Indistinguishability Obfuscation (iO): We take $\mathsf{Compile} = \mathsf{Obf}$ (obfuscate), $\mathsf{Eval} = \mathsf{iOEval}$ (evaluation of obfuscation), while Gen and Encode are undefined. For security, ρ checks if $C^0 \equiv C^1$ and $C_i = \perp$ for $i \in [\ell_f]$, $x_k = \perp$ for $k \in [\ell_{\mathsf{inp}}]$. If the check fails it outputs 0, else it outputs 1.

2.8.2 Dynamic Circuit Compilers

The notion of dynamic circuit compilers additionally have the algorithms GenUpd (generation of secure update) and $\mathsf{ApplyUpd}$ (apply secure update) associated with it. It consists of the algorithms $\mathsf{DCC} = (\mathsf{Gen}, \mathsf{Compile}, \mathsf{Encode}, \mathsf{GenUpd}, \mathsf{ApplyUpd}, \mathsf{Decode})$. Its associated with a class of circuits $\mathcal{C}$.

GENERATION OF UPDATES, $\mathsf{GenUpd}(\mathsf{cktSK}, \mathsf{st}, \mathbf{u})$: On input secret parameters cktSK, state st, update $\mathbf{u}$, it outputs an encoding of update $\langle \mathbf{u} \rangle$ and an updated state st'.

APPLY UPDATE, $\mathsf{ApplyUpd}(\mathsf{pp}, \langle C \rangle, \langle \mathbf{u} \rangle)$: On input public parameters pp, circuit encoding $\langle C \rangle$, secure update $\langle \mathbf{u} \rangle$, it outputs an updated circuit encoding $\langle C' \rangle$.

Correctness. Consider a circuit $C = C_0$, input x and a sequence of updates $\mathbf{u}_1, \ldots, \mathbf{u}_{\mathsf{q}}$. Let C_i be the circuit by updating C_{i-1} using update $\mathbf{u}_i$. We require that the evaluation of encoding $\langle C_i \rangle$ on $\langle x \rangle$ yields $C_i(x)$, where $\langle C_i \rangle$ is obtained by updating $\langle C_{i-1} \rangle$ using $\langle \mathbf{u}_i \rangle$ (encoding of $\mathbf{u}_i$) and is associated with a new secret key $\mathsf{CC.cktSK}_i$. The encoding $\langle x \rangle$ is computed using the new secret key $\mathsf{CC.cktSK}_i$.

We emphasize that once the circuit is updated, the secret key associated with the compiled circuit could potentially change. Hence, we require that correctness to be satisfied only for input encodings created using the new secret key.

ρ-IND Security. We define a generic indistinguishability-based selective security notion associated with a ρ-admissibility property. ρ is some boolean function with input $\mathsf{aux}_{\mathsf{dcc}}$ consisting of: (i) a pair of circuits C^0, C^1, (ii) additional circuits C_1, $\dots$, C_{ℓ_f}, (iii) a pair of input sequences $(x_0^0, \dots, x_{\ell_{\mathsf{inp}}}^0)$, $(x_0^1, \dots, x_{\ell_{\mathsf{inp}}}^1)$, and (iv) a pair of update sequences $(\mathbf{u}_0^0, \dots, \mathbf{u}_{\mathsf{q}}^0)$, $(\mathbf{u}_0^1, \dots, \mathbf{u}_{\mathsf{q}}^1)$.

Definition 2.8.2 (ρ-IND Security). *A dynamic circuit compiler scheme* DCC *is* ρ-IND secure *(selectively) for all* $\lambda \in \mathbb{N}$ *and all* $\mathsf{aux}_{\mathsf{dcc}}$ *as above,* $\mathsf{Expt}_0^{\mathsf{DCC}}(1^\lambda, \mathsf{aux}_{\mathsf{dcc}}) \approx_{\mathsf{c}} \mathsf{Expt}_1^{\mathsf{DCC}}(1^\lambda, \mathsf{aux}_{\mathsf{dcc}})$.

$\mathsf{Expt}_b^{\mathsf{DCC}}(1^\lambda, \mathsf{aux}_{\mathsf{dcc}})$:

1. If $\rho(\mathsf{aux}_{\mathsf{dcc}}) = 0$, abort and output $\perp$.
2. Let $\mathsf{pp}, (\mathsf{cktSK}, \mathsf{inpSK}) \leftarrow \mathsf{Gen}(1^\lambda)$.
3. Generate the circuit encodings $(\langle C^b \rangle, \mathsf{st}_0) \leftarrow \mathsf{Compile}(\mathsf{cktSK}, C^b)$ and $(\langle C_i \rangle, \mathsf{st}_{i,0}) \leftarrow \mathsf{Compile}(\mathsf{cktSK}, C_i)$ for $i \in [\ell_f]$. (The states $\mathsf{st}_{i,0}$ are never used.)
4. Generate the update encodings $\langle \mathbf{u}_j^b \rangle \leftarrow \mathsf{GenUpd}(\mathsf{cktSK}, \mathsf{st}_{j-1}, \mathbf{u}_j)$ for $j \in [\mathsf{q}]$.
5. Generate the input encodings $\langle x_k^b \rangle \leftarrow \mathsf{Encode}(\mathsf{inpSK}, x_k^b)$ for $k \in [\ell_{\mathsf{inp}}]$.
6. Output
$$\left(\langle C^b \rangle,\ \{\langle C_i \rangle\}_{i \in [\ell_f]}, \{\langle \mathbf{u}_j^b \rangle\}_{j \in [\mathsf{q}]},\ \{\langle x_k^b \rangle\}_{k \in [\ell_{\mathsf{inp}}]} \right).$$

2.8.2.1 Instantiations

We can consider the updatability versions of several cryptographic primitives via the lens of dynamic circuit compilers. For example, we can consider the notion of updatable ABE: In updatable ABE, an attribute key associated with a function, say sk_C, can sequentially updated.

Remark 2.8.3. Since we consider dynamic circuit compilers, where only one circuit can be updated in the security game, this correspondingly leads to defining updatable

ABE scheme where only one attribute key is updated in the security game. However, for other primitives such as updatable indistinguishability obfuscation and updatable non interactive witness indistinguishable systems, it suffices to just consider dynamic circuit compilers where only a single circuit is updated.

One can also consider the more general setting where multiple circuits are updated simultaneously using the same sequence of updates. Such a setting was studied in the context of indistinguishability obfuscation [AJS15b]. We do not deal with this setting in this thesis.

2.8.3 Construction of Dynamic Circuit Compilers

We sketch a construction of dynamic circuit compilers using output-compact updatable randomized encodings. We start with a circuit compiler, not necessarily supporting updatability, and show how to transform it into a dynamic circuit compilers scheme. Let the circuit compiler scheme be denoted by $\mathsf{CC} = (\mathsf{CC.Gen}, \mathsf{CC.Compile}, \mathsf{CC.Encode}, \mathsf{CC.Eval}$ We construct $\mathsf{DCC} = (\mathsf{Gen}, \mathsf{Compile}, \mathsf{Encode}, \mathsf{GenUpd}, \mathsf{ApplyUpd}, \mathsf{Eval})$ as follows.

$\mathsf{Gen}(1^\lambda)$: Execute $\mathsf{CC.Gen}(1^\lambda)$ and also the setup of updatable randomized encoding scheme. The parameters output by Gen is the joint parameters output by both $\mathsf{CC.Gen}$ and the setup of URE.

$\mathsf{Compile}$: On input secret parameters and circuit C, execute an (updatable) randomized encoding of $\mathsf{Compile}(\cdot, \cdot; \cdot)$ and input $(\mathsf{CC.cktSK}, C, r)$, where $\mathsf{CC.cktSK}$ is the secret parameters output by CC and r is the randomness used in the circuit compilation process.

GenUpd: On input secret parameters and update $\mathbf{u}$, generate a secure update $\langle(\mathbf{u}, \mathsf{CC.cktSK}', r')\rangle$, where $\mathsf{CC.cktSK}'$ is the new secret key and r' is a new randomness to be used in the circuit compilation algorithm, computed with respect to updatable randomized encodings scheme.

$\mathsf{ApplyUpd}$: Update the randomized encoding (part of the compiled circuit) using the secure update. Output the updated randomized encoding.

Encode: Identical to CC.Encode.

Eval: Decodes the randomized encoding to get a compiled circuit, computed with respect to CC. Then executes the evaluation algorithm CC.Eval.

We omit the proof of correctness argument since it follows directly from the proof of correctness of the underlying circuit compilers and updatable randomized encodings scheme. We sketch the proof of security below.

In terms of efficiency, the output-compactness of the underlying URE scheme guarantees that the size of the updates are small. Note that if not for the output-compactness, the size of the updates could be proportional to the output length of the circuit which is proportional to the size of the circuit being compiled.

2.8.3.1 Security

The main steps in the security argument are as follows: Let (C_0, C_1) be the circuit pair, $(x_1^0, x_1^1), \ldots, (x_{\ell_{\mathsf{inp}}}^0, x_{\ell_{\mathsf{inp}}}^1)$ be the input pairs and update pairs $(\mathbf{u}_1^0, \mathbf{u}_1^1) \ldots, (\mathbf{u}_{\mathsf{q}}^0, \mathbf{u}_{\mathsf{q}}^1)$.

1. In the first step, instead of computing the randomized encoding of circuit compile algorithm Compile on just C_0, it does the following: consider the circuit G which is computed as follows:

   $$G(\mathsf{temp} = 0, \mathsf{CC.cktSK}, C^0, r, C^1, r') \text{ outputs } \mathsf{Compile}(\mathsf{CC.cktSK}, C^0; r).$$

   Furthermore, the secure updates are computed as an encoding of $(i, \mathbf{u}_i^0, r_i^0, \mathbf{u}_i^1, r_i^1, \mathsf{CC.cktSK}_i)$, where $(\mathbf{u}_i^0, r_i^0)$ (resp., $(\mathbf{u}_i^1, r_i^1)$) will be used to update C_{i-1}^0 to C_i^0 (resp., C_{i-1}^1 to C_i^1). The value i, as part of update encoding, is used to update the value temp from 0 to i. Finally, $\mathsf{CC.cktSK}_i$ is the new secret key.

2. In a sequence of steps, the code of G is switched from computing a compiled circuit of C^0 to computing a compiled circuit of C^1. This is done in a sequence

of steps: in the j^{th} step, for $j \in [\mathsf{q}+1]$, G is of the following form:

$$G(\mathsf{temp}=0, \mathsf{CC.cktSK}, C^0, r, C^1, r') := \begin{cases} \mathsf{Compile}(\mathsf{CC.cktSK}, C^0; r) & \text{if } \mathsf{temp} \geq j, \\ \mathsf{Compile}(\mathsf{CC.cktSK}, C^1; r) & \text{otherwise} \end{cases}$$

The j^{th} step is switched to $(j+1)^{th}$ step is done in the following steps: a third branch is introduced in G, for the case when $\mathsf{temp} = j$, a hardwired value V is output. For all values of temp, G remains unchanged as in j^{th} step. The value V is set to be $\mathsf{Compile}(\mathsf{CC.cktSK}_j, C_j^0; r_j^0)$, where C_j^0 is the j^{th} updated circuit. Furthermore, in the j^{th} update, the secret key $\mathsf{CC.cktSK}_j$ and randomness (r_j^0, r_j^1) is removed from the description of updates. Now, we invoke the security of underlying dynamic circuit compiler scheme (since $\mathsf{CC.cktSK}_j$ is "removed" from the system) to switch from $\mathsf{Compile}(\mathsf{CC.cktSK}_j, C_j^0; r_j^0)$ to $\mathsf{Compile}(\mathsf{CC.cktSK}_j, C_j^1; r_j^1)$. Once this is done, we can switch the description of G from having the instruction "$\mathsf{temp} \geq j$" to "$\mathsf{temp} \geq j+1$". We invoke the security of URE to make this change.

3. In the $(\mathsf{q}+1)^{th}$ step, G outputs a compiled circuit of C^1 for every value of temp. Thus, C^0 and updates $\{\mathbf{u}_i^0\}$ can now be removed from the system.

2.8.4 Updatable Non-Interactive Zero Knowledge

2.8.4.1 Definition

Syntax. Let R be an efficiently computable relation that consists of pairs (x, w), where x is called the statement and w is the witness. Let L denote the language consisting of statements in R. We say that the relation R is $(\mathsf{Upd}, \mathcal{U})$-updatable if for any $(x, w) \in R$ and any update string $\mathbf{u} \in \mathcal{U}$, $\mathsf{Upd}(x, w, \mathbf{u}) = (x', w')$ s.t. $(x', w') \in R$.

An updatable non-interactive zero-knowledge (UNIZK) proof system for a language L with a $(\mathsf{Upd}, \mathcal{U})$-updatable relation R consists of a setup algorithm CRSGen, a prover algorithm Prove and a verifier algorithm Verify similar to a standard NIZK proof system. However, it also comes equipped with two additional algorithms, namely,

GenUpd and $\mathsf{ApplyUpd}$. We formally describe the algorithms below:

SETUP, $\mathsf{crs} \leftarrow \mathsf{CRSGen}(1^\lambda)$: On input a security parameter λ, it outputs a common reference string crs.

PROVE, $(\pi, \mathsf{st}) \leftarrow \mathsf{Prove}(\mathsf{crs}, x, w)$: On input a common reference string crs and a statement x along with a witness w, it first checks whether $(x, w) \in R$; if so, it produces a proof string π along with a state st, else it outputs `fail`.

GENERATE UPDATE, $\langle \mathbf{u} \rangle \leftarrow \mathsf{GenUpd}(\mathbf{u}, \mathsf{st})$: On input an update $\mathbf{u} \in \mathcal{U}$ and state st, it outputs an update encoding $\langle \mathbf{u} \rangle$ and a new state st'.

APPLY UPDATE, $\pi' \leftarrow \mathsf{ApplyUpd}(\mathsf{crs}, \pi, \langle \mathbf{u} \rangle)$: On input a common reference string crs, a proof string π and an update encoding $\langle \mathbf{u} \rangle$, it outputs an updated proof string π'.

VERIFY, $b \leftarrow \mathsf{Verify}(\mathsf{crs}, x, \pi)$: On input a common reference string crs, a statement x and a proof string π, it outputs $b = 1$ if the proof is valid, and $b = 0$ otherwise.

Definition 2.8.4 (Updatable NIZKs). *An updatable non-interactive zero-knowledge (*UNIZK*) proof system for a language L with a* PPT *relation R and update family $\mathcal{U}$ with updating algorithm* Upd *is a tuple of* PPT *algorithms* $(\mathsf{CRSGen}, \mathsf{Prove}, \mathsf{GenUpd}, \mathsf{ApplyUpd}, \mathsf{Verify})$ *such that the following properties hold:*

EFFICIENCY: *For any update $\mathbf{u} \in \mathcal{U}$, the running time of* $\mathsf{GenUpd}(\mathbf{u}, \mathsf{st})$ *is $p(\lambda, |\mathbf{u}|)$, where p is an a priori fixed polynomial. This implies that the size of the resultant update encoding $\langle \mathbf{u} \rangle \leftarrow$* $\mathsf{GenUpd}(\mathbf{u}, \mathsf{st})$ *is also a fixed polynomial in λ and $|\mathbf{u}|$.*

COMPLETENESS: *For every $(x_0, w_0) \in R$ and every sequence of updates $\mathbf{u}_1 \ldots, \mathbf{u}_\mathsf{q} \in \mathcal{U}$, it holds that for every $0 \leq i \leq \mathsf{q}$:*

$$\Pr[\mathsf{Verify}(\mathsf{crs}, x_i, \pi_i) = 1] = 1$$

where $\mathsf{crs} \leftarrow \mathsf{CRSGen}(1^\lambda)$, $(\pi_0, \mathsf{st}_0) \leftarrow \mathsf{Prove}(\mathsf{crs}, x_0, w_0)$, $(\langle \mathbf{u}_i \rangle, \mathsf{st}_i) \leftarrow \mathsf{GenUpd}(\mathbf{u}_i, \mathsf{st}_{i-1})$, $\pi_i \leftarrow \mathsf{ApplyUpd}(\pi_{i-1}, \langle \mathbf{u}_i \rangle)$ *and the probability is taken over the coins of* CRSGen, Prove, GenUpd, $\mathsf{ApplyUpd}$ *and* Verify.

SOUNDNESS AGAINST SEQUENTIAL UPDATES: *For every adversary $\mathcal{A}$, there exists a negligible function* $\mathsf{negl}(\lambda)$ *s.t.*

$$\Pr[1 \leftarrow \mathsf{Expt}_{\mathcal{A}}^{\mathsf{pu\text{-}sound}}(1^\lambda)] \leq \mathsf{negl}(\lambda)$$

where $\mathsf{Expt}_{\mathcal{A}}^{\mathsf{pu\text{-}sound}}(1^\lambda)$ *is defined as follows:*

1. Ch *computes* $\mathsf{crs} \leftarrow \mathsf{CRSGen}(1^\lambda)$ *and sends* crs *to* $\mathcal{A}$.

2. $\mathcal{A}$ *outputs* $(x_0, \pi_0, \{x_i, \langle \mathbf{u} \rangle_i\}_{i=1}^{\mathsf{q}})$ *to* Ch.

3. *The output of the experiment is* 1 *if:* either $\mathsf{Verify}(\mathsf{crs}, x_0, \pi_0) = 1 \wedge x_0 \notin L$, or *there exists* $1 \leq i \leq \mathsf{q}$ *s.t.* $\mathsf{Verify}(\mathsf{crs}, x_i, \pi_i) = 1 \wedge x_i \notin L$ *where* $\pi_i \leftarrow \mathsf{ApplyUpd}(\pi_{i-1}, \langle \mathbf{u}_i \rangle)$. *Otherwise, the output is* 0.

If the soundness only holds against PPT *adversaries, then we call it an* argument *system.*

ZERO KNOWLEDGE AGAINST SEQUENTIAL UPDATES: *There exists a PPT simulator* $\mathcal{S} = (\mathcal{S}_1, \mathcal{S}_2, \mathcal{S}_3)$ *s.t. for every* $(x_0, w_0) \in R$*, every auxiliary input* $z \in \{0,1\}^*$ *and every sequence of updates* $\mathbf{u}_1, \ldots, \mathbf{u}_{\mathsf{q}} \in \mathcal{U}$*, it holds that*

$$\mathsf{IDEAL}\Big(1^\lambda, x_0, w_0, z, \{\mathbf{u}_i\}_{i=1}^{\mathsf{q}}\Big) \overset{\mathsf{c}}{\approx} \mathsf{REAL}\Big(1^\lambda, x_0, w_0, z, \{\mathbf{u}_i\}_{i=1}^{\mathsf{q}}\Big)$$

where

$\underline{\mathsf{IDEAL}\left(1^\lambda, x_0, w_0, z, \{\mathbf{u}_i\}_{i=1}^{\mathsf{q}}\right)}$:	$\underline{\mathsf{REAL}\left(1^\lambda, x_0, w_0, z, \{\mathbf{u}_i\}_{i=1}^{\mathsf{q}}\right)}$:
1. $(\mathsf{crs}, \mathsf{st}_1) \leftarrow \mathcal{S}_1(1^\lambda)$	1. $\mathsf{crs} \leftarrow \mathsf{CRSGen}(1^\lambda)$
2. $(\pi_0, \mathsf{st}_2) \leftarrow \mathcal{S}_2(x_0, \mathsf{st}_1)$	2. $(\pi_0, \mathsf{st}_0) \leftarrow \mathsf{Prove}(\mathsf{crs}, x_0, w_0)$
3. $\forall i \in [\mathsf{q}]$, $(x_i, w_i) \leftarrow \mathsf{Upd}(x_0, w_0, \mathbf{u}_i)$	3. $\forall i \in [\mathsf{q}]$, $\langle \mathbf{u}_i \rangle \leftarrow \mathsf{GenUpd}\left(\mathbf{u}_i, \mathsf{st}_{i-1}\right)$
4. $\forall i \in [\mathsf{q}]$, $\langle \mathbf{u}_i \rangle \leftarrow \mathcal{S}_3(\mathsf{st}_2, 1^{\lvert \mathbf{u}_i \rvert}, x_i)$.	
Output $(x_0, z, \pi_0, \{\langle \mathbf{u} \rangle_i\}_{i=1}^{p})$	Output $\left(x_0, z, \pi_0, \{\langle \mathbf{u}_i \rangle\}_{i=1}^{\mathsf{q}}\right)$

2.8.4.2 Construction of UNIZK

In this section, we construct a UNIZK proof system for NP. Let L be any language in NP with a $(\mathsf{Upd}, \mathcal{U})$-updatable PPT relation R. We construct a UNIZK proof system $(\mathsf{CRSGen}, \mathsf{Prove}, \mathsf{GenUpd}, \mathsf{ApplyUpd}, \mathsf{Verify})$ for L.

Let $R[x, w]$ denote a hardwired circuit corresponding to R with inputs (x, w). Let $R[X, W]$ denote the corresponding hardwired circuit family. We will use the following ingredients in our construction:

1. A stateless[16] URE scheme $(\mathsf{URE.Encode}, \mathsf{URE.GenUpd}, \mathsf{URE.ApplyUpd}, \mathsf{URE.Decode})$ for an $(\mathsf{Upd}_{\mathsf{ure}}, \mathcal{U}_{\mathsf{ure}})$-updatable class of hardwired circuits $R[X, W]$ where $\mathcal{U}_{\mathsf{ure}} = \mathcal{U}$ and $\mathsf{Upd}_{\mathsf{ure}}$ is s.t. for any $\mathbf{u} \in \mathcal{U}_{\mathsf{ure}}$, $\mathsf{Upd}_{\mathsf{ure}}(R[x, w], \mathbf{u}) = R[x', w']$ where $(x', w') \leftarrow \mathsf{Upd}(x, w, \mathbf{u})$.

2. A non-interactive perfectly binding commitment scheme Com. Such a scheme can be based on the existence of injective one-way functions.[17]

3. A NIZK proof of knowledge (NIZKPOK) system $(\mathsf{NIZK.CRSGen}, \mathsf{NIZK.Prove}, \mathsf{NIZK.Verify})$ for NP.

[16]Our construction also works if we start with a stateful URE scheme. For simplicity of exposition, however, we present our construction using a stateless URE scheme. As a result, we actually construct a stateless UNIZK scheme.

[17]We can, in fact, use a two round statistically binding commitment scheme that can be based on standard one-way functions. For simplicity of exposition, however, we present our construction using non-interactive commitments.

$\mathsf{CRSGen}(1^\lambda)$: Sample a common reference string $\mathsf{crs}_{\mathsf{nizk}} \leftarrow \mathsf{NIZK.CRSGen}(1^\lambda)$ for the NIZKPOK system. Output $\mathsf{crs} = \mathsf{crs}_{\mathsf{nizk}}$.

$\mathsf{Prove}(x, w)$: Perform the following sequence of steps:

- Compute $\langle R[x,w]\rangle_{\mathsf{ure}} \leftarrow \mathsf{URE.Encode}\left(1^\lambda, R, (x,w)\,; r_{\mathsf{ure}}\right)$ using randomness a random string r_{ure}.
- Sample a random string r_{com} and compute a commitment $C \leftarrow \mathsf{Com}(r_{\mathsf{ure}}; r_{\mathsf{com}})$ to r_{ure} using randomness r_{com}.
- Compute a proof $\pi_{\mathsf{nizk}} \leftarrow \mathsf{NIZK.Prove}(x_{\mathsf{nizk}}, w_{\mathsf{nizk}})$ for the statement $x_{\mathsf{nizk}} = (x, \langle R[x,w]\rangle_{\mathsf{ure}}, C)$ using witness $w_{\mathsf{nizk}} = (w, r_{\mathsf{ure}}, r_{\mathsf{com}})$ where $(w', r'_{\mathsf{ure}}, r'_{\mathsf{com}})$ is a valid witness for x_{nizk} iff all of the following hold:
  - $\langle R[x,w]\rangle_{\mathsf{ure}} \leftarrow \mathsf{URE.Encode}\left(1^\lambda, R, (x,w')\,; r'_{\mathsf{ure}}\right)$
  - $C \leftarrow \mathsf{Com}(r'_{\mathsf{ure}}; r'_{\mathsf{com}})$

Finally, set $\mathsf{st} = (r_{\mathsf{ure}}, C, r_{\mathsf{com}})$ and output $\pi = (\langle R[x,w]\rangle_{\mathsf{ure}}, C, \pi_{\mathsf{nizk}})$.

$\mathsf{GenUpd}(\mathbf{u}, \mathsf{st})$: Perform the following sequence of steps:

- Parse $\mathsf{st} = (r_{\mathsf{ure}}, C, r_{\mathsf{com}})$.
- Sample a random string r_{upd} and compute $\langle \mathbf{u}\rangle_{\mathsf{ure}} \leftarrow \mathsf{URE.GenUpd}(\mathbf{u}, r_{\mathsf{ure}}; r_{\mathsf{upd}})$ using randomness r_{upd}.
- Compute a proof $\pi_{\mathsf{nizk}} \leftarrow \mathsf{NIZK.Prove}(x_{\mathsf{nizk}}, w_{\mathsf{nizk}})$ for the statement $x_{\mathsf{nizk}} = (\langle \mathbf{u}\rangle_{\mathsf{ure}}, C)$ using witness $w_{\mathsf{nizk}} = (\mathbf{u}, r_{\mathsf{upd}}, r_{\mathsf{ure}}, r_{\mathsf{com}})$ where $(\mathbf{u}', r'_{\mathsf{upd}}, r'_{\mathsf{ure}}, r'_{\mathsf{com}})$ is a valid witness for x_{nizk} iff all of the following hold:
  - $\langle \mathbf{u}\rangle_{\mathsf{ure}} \leftarrow \mathsf{URE.GenUpd}\left(\mathbf{u}', r'_{\mathsf{ure}}; r'_{\mathsf{upd}}\right)$
  - $C \leftarrow \mathsf{Com}(r'_{\mathsf{ure}}; r'_{\mathsf{com}})$

Output $\langle \mathbf{u}\rangle = (\langle \mathbf{u}\rangle_{\mathsf{ure}}, C, \pi_{\mathsf{nizk}})$.[18]

[18]Note that it is not really necessary for the update encoding to include C since the verifier, who runs the ApplyUpd and Verify algorithms, can remember C. We have added C to the update encoding for clarity.

$\mathsf{ApplyUpd}(\mathsf{crs}, \pi, \langle \mathbf{u} \rangle)$: Perform the following steps:

- Parse $\langle \mathbf{u} \rangle = (\langle \mathbf{u} \rangle_{\mathsf{ure}}, C, \pi^{\mathsf{upd}}_{\mathsf{nizk}})$ and $\mathsf{crs} = \mathsf{crs}_{\mathsf{nizk}}$. Let $x^{\mathsf{upd}}_{\mathsf{nizk}} = (\langle \mathbf{u} \rangle_{\mathsf{ure}}, C)$. If $\mathsf{NIZK.Verify}(\mathsf{crs}_{\mathsf{nizk}}, x^{\mathsf{upd}}_{\mathsf{nizk}}, \pi^{\mathsf{upd}}_{\mathsf{nizk}}) = 0$, then output $\perp$.
- If π is a level-0 proof, then parse $\pi = (\langle R[x, w] \rangle_{\mathsf{ure}}, C, \pi_{\mathsf{nizk}})$. Else, parse $\pi = \langle R[x, w] \rangle_{\mathsf{ure}}$.
- Compute $\langle R[x', w'] \rangle_{\mathsf{ure}} \leftarrow \mathsf{URE.ApplyUpd}(\langle R[x, w] \rangle_{\mathsf{ure}}, \langle \mathbf{u} \rangle_{\mathsf{ure}})$.
- Output $\langle R[x', w'] \rangle_{\mathsf{ure}}$.

$\mathsf{Verify}(\mathsf{crs}, x, \pi)$: Perform the following steps:

- Parse $\mathsf{crs} = \mathsf{crs}_{\mathsf{nizk}}$.
- If π is a level-0 proof, then parse $\pi = (\langle R[x, w] \rangle_{\mathsf{ure}}, C, \pi_{\mathsf{nizk}})$. Let $x_{\mathsf{nizk}} = (x, \langle R[x, w] \rangle_{\mathsf{ure}}, C)$. Output 1 if both $\mathsf{NIZK.Verify}(\mathsf{crs}_{\mathsf{nizk}}, x_{\mathsf{nizk}}, \pi_{\mathsf{nizk}})$ and $\mathsf{URE.Decode}(\langle R[x, w] \rangle_{\mathsf{ure}})$ return 1.
- Else, parse $\pi = \langle R[x, w] \rangle_{\mathsf{ure}}$. Output $\mathsf{URE.Decode}(\langle R[x, w] \rangle_{\mathsf{ure}})$.

2.8.4.3 Proof Sketch

We begin with efficiency. Recall that the computation of an update encoding for UNIZK involves two main steps. First, we compute an update encoding for the underlying URE scheme. Next, we compute a fresh proof string for the underlying NIZKPOK system to prove that the URE update encoding was computed honestly. From the efficiency of the URE, it follows that the first step only requires time polynomial in the update size and the security parameter. Further, it follows from the standard efficiency of NIZKPOKs that the second step also only requires time polynomial in the size of the update and the security parameter. Putting the above together, we have that GenUpd satisfies the efficiency requirement.

Correctness of the above construction is easy to verify. In order to argue soundness, we leverage the proof of knowledge of the underlying NIZKPOK. Let us assume that the construction is not sound, i.e., there exists an efficient adversary for the soundness security game that outputs $(x_0, \pi_0, \{x_i, \langle \mathbf{u}_i \rangle\}_{i=1}^{\mathsf{q}})$ for some q s.t. all the (updated) proof strings are accepted by the honest verification algorithm, yet at least one of the (updated) statements is false. We obtain a contradiction as follows: for every i, let $\langle \mathbf{u}_i \rangle = (\langle \mathbf{u}_i \rangle_{\mathsf{ure}}, \pi^i_{\mathsf{nizk}})$ and let x^i_{nizk} be the statement corresponding to π^i_{nizk}. Then, starting with $i = \mathsf{q}$ and proceeding backwards, we apply the NIZKPOK extractor on each π^i_{nizk} to extract a valid witness w^i_{nizk} for x^i_{nizk}. From the description of the scheme, it follows that w^i_{nizk} consists of an update $\mathbf{u}_i$ as well as randomness that can be used to verify that $\mathbf{u}_i$ was indeed used to compute the update encoding honestly. For $i = 0$, the extractor also returns the witness w_0 for the original statement x_0. Putting all of this together, we can recover a witness w_i for every updated statement x_i, which leads to a contradiction.

Zero-knowledge follows from a simple hybrid argument. The first hybrid H_0 corresponds to the real world experiment. In the next hybrid H_1, we first simulate $\mathsf{crs}_{\mathsf{nizk}}$ and all the proof strings π^i_{nizk} that are part of the original proof π_0 and the update encodings $\mathbf{u}_i$. Next, in hybrid H_2, we switch the commitment C to be commitment of all zeros. Finally, in H_3, we use the simulator of URE to simulate the URE encoding $\langle R[x, w] \rangle_{\mathsf{ure}}$ in π_0 as well as the URE update encodings $\langle \mathbf{u} \rangle^i_{\mathsf{ure}}$ in every update $\langle \mathbf{u}_i \rangle$. Note that this experiment corresponds to the ideal world.

The indistinguishability of H_0 and H_1 follows from the zero-knowledge of the underlying NIZKPOK. The indistinguishability of H_1 and H_2 follows from the hiding of the commitment scheme Com. Finally, the indistinguishability of H_2 and H_3 follows from the security of the URE scheme. This finishes the proof sketch of our construction.

2.8.5 Updatable Multiparty Computation

We consider the setting of n parties $\mathcal{P} = \{P_1, ..., P_n\}$ who wish to jointly compute any PPT function over their private inputs. We are interested in the scenario where after performing a computation of any function f over an input vector $\boldsymbol{x} = x_1, \ldots, x_n$, the parties wish to perform another computation over an *updated* function f' and input vector $\boldsymbol{x}'$. Note that if the parties were to simply use a standard MPC protocol to perform a fresh computation over the updated function and input vector, then the communication complexity of this computation will depend on $|f'|$ and $|\boldsymbol{x}'|$. We instead consider the scenario where after performing the initial computation of f over $\boldsymbol{x}$, the parties can run a less-expensive *update phase* whose communication complexity only depends on the description size of the update and not on the size of the updated function and input vector. We refer to a protocol with this efficiency as *updatable MPC* (UMPC).

We consider the setting of multiple updates where the parties can perform multiple updated computations in a *sequential* manner. We now proceed to formally define UMPC with sequential updating using the real/ideal paradigm.

2.8.5.1 Additional Notation

We study the notion of UMPC with respect to a class of updatable hardwired circuits. Below, we first extend the notation for updatable hardwired circuits (as described in Section 2.4) to meet our requirements for UMPC. We consider two changes: (a) First, we consider computation over n different inputs, as opposed to a single input. (b) Second, we consider updates $\mathbf{u}$ of the form $\mathbf{u} = u_1, \ldots, u_n$ where each u_i is contributed by party P_i. For simplicity of exposition, we restrict our discussion to the case where all the parties receive the same output.

Let $C : \{0,1\}^\lambda \times \ldots \times \{0,1\}^\lambda \to \{0,1\}^{\ell(\lambda)}$ be any n-input circuit that the parties are interested in computing. Then, for any input vector $\boldsymbol{x} = x_1, \ldots, x_n$, where $x_i \in \{0,1\}^\lambda$, the corresponding hardwired circuit is denoted as $C[\boldsymbol{x}]$. The corresponding hardwired circuit family is denoted as $\{\mathcal{C}[X]_\lambda\}_{\lambda \in \mathbb{N}}$ where $X = \{0,1\}^{n \times \lambda}$.

For any set system of strings $\mathcal{U} = \{\mathcal{U}_\lambda\}_{\lambda \in \mathbb{N}}$, we say that $\mathcal{C}[X]$ is $(\mathsf{Upd}, \mathcal{U})$-updatable if $C'[\boldsymbol{x}'] \leftarrow \mathsf{Upd}\,(C[\boldsymbol{x}], \mathbf{u})$, where $C[\boldsymbol{x}] \in \mathcal{C}[X]_\lambda, \mathbf{u} = (u_1, \ldots, u_n)$, $u_i \in \mathcal{U}_\lambda$, is such that $C'[\boldsymbol{x}']$ is also a hardwired circuit.

2.8.5.2 Security

We consider polynomial-time adversaries who can statically corrupt up to $n-1$ parties, and with abort.

We say that a protocol Π is a UMPC protocol if any adversary, who corrupts a subset of parties and runs the protocol with honest parties, gains *no information* about the inputs of the honest parties beyond the protocol outputs that correspond to sequential evaluations of $C[\boldsymbol{x}], C[\boldsymbol{x}]_1, \ldots, C[\boldsymbol{x}]_{\mathsf{q}}$ where $C[\boldsymbol{x}]_\ell = \mathsf{Upd}(C[\boldsymbol{x}], \mathbf{u}_\ell)$, $\mathbf{u}_\ell = (u_{1,\ell}, \ldots, u_{n,\ell})$.

Definition 2.8.5 (Updatable MPC). *A protocol* $\Pi = (\Pi_{\mathsf{init}}, \Pi_{\mathsf{upd}})$ *is a secure n-party* UMPC *protocol for a* $(\mathsf{Upd}, \mathcal{U})$*-updatable circuit family* $\mathcal{C}$ *if for every PPT adversary* $\mathcal{A}$ *in the real world, there exists a PPT adversary* $\mathcal{S}$ *corrupting the same parties in the ideal world such that for every initial input vector* $\boldsymbol{x}$*, every auxiliary input* z*, and every sequence of updates* $\{\mathbf{u}_\ell\}_{\ell=1}^{\mathsf{q}}$ *where* $\mathbf{u}_\ell = (u_{1,\ell}, \ldots, u_{n,\ell})$ *and* $u_{i,\ell} \in \mathcal{U}$*, it holds that*

$$\mathsf{IDEAL}_{\mathcal{S},M}\Big(1^\lambda, \boldsymbol{x}, z, \{\mathbf{u}_\ell\}_{\ell=1}^{\mathsf{q}}\Big) \overset{c}{\approx} \mathsf{REAL}^{\Pi}_{\mathcal{A},M}\Big(1^\lambda, \boldsymbol{x}, z, \{\mathbf{u}_\ell\}_{\ell=1}^{\mathsf{q}}\Big),$$

where $\mathsf{IDEAL}_{\mathcal{S},M}$ *and* $\mathsf{REAL}_{\mathcal{A},M}$ *are defined next.*

Ideal world. We start by describing the ideal world for UMPC. Let $C \in \mathcal{C}$ be the initial circuit that the parties wish to compute.

INPUTS: Each party P_i obtains an initial input x_i. The adversary $\mathcal{S}$ is given auxiliary input z. $\mathcal{S}$ selects a subset of the parties $M \subset \mathcal{P}$ to corrupt, and is given the inputs x_ℓ of each party $P_\ell \in M$.

SENDING INPUTS TO TRUSTED PARTY: Each honest party P_i sends its input x_i to the trusted party. For each corrupted party $P_i \in M$, the adversary may select any value x_i^* and send it to the ideal functionality.

TRUSTED PARTY COMPUTES OUTPUT: Let $x_1^*, ..., x_n^*$ be the inputs that were sent to the trusted party. Let $C[\boldsymbol{x}^*]_0$ be the hardwired circuit corresponding to the circuit C and input vector $\boldsymbol{x}^* = x_1^*, \ldots, x_n^*$. The trusted party sends the evaluation of $C[\boldsymbol{x}^*]_0$ to the adversary who replies with either `continue` or `abort`. If the adversary's message is `abort`, then the trusted party sends $\perp$ to all honest parties. Otherwise, it sends the evaluation of $C[\boldsymbol{x}^*]_0$ to all honest parties.

ℓTH UPDATE PHASE: For every $\ell \in [\mathsf{q}]$, where q is chosen by the adversary, the following is repeated sequentially:

- Each party P_i sends an update string $u_{i,\ell} \in \mathcal{U}_\lambda$ to the trusted party.
- The trusted party computes $C[\boldsymbol{x}^*]_\ell \leftarrow \mathsf{Upd}(C[\boldsymbol{x}^*]_{\ell-1}, \mathbf{u}_\ell)$ where $\mathbf{u}_\ell = (u_{1,\ell}, \ldots, u_{n,\ell})$. It sends the evaluation of $C[\boldsymbol{x}^*]_\ell$ to the adversary who replies with either `continue` or `abort`. If the adversary's message is `abort`, then the trusted party sends $\perp$ to all honest parties. Otherwise, it sends the evaluation of $C[\boldsymbol{x}^*]_\ell$ to all honest parties.

OUTPUTS: Honest parties output all the messages they obtained from the ideal functionality. Malicious parties may output an arbitrary PPT function of the adversary's view.

The overall output of the ideal-world experiment consists of the outputs of all parties. For any ideal-world adversary $\mathcal{S}$ with auxiliary input $z \in \{0,1\}^*$, input vector $\boldsymbol{x}$, any arbitrary polynomial set of updates $\{\mathbf{u}_\ell\}_{\ell=1}^{\mathsf{q}}$, and security parameter λ, we denote the output of the corresponding ideal-world experiment by

$$\mathsf{IDEAL}_{\mathcal{S},M}\left(1^\lambda, \boldsymbol{x}, z, \{\mathbf{u}_\ell\}_{\ell=1}^{\mathsf{q}}\right).$$

Real world. The real world execution begins by an adversary $\mathcal{A}$ selecting any arbitrary subset of parties $M \subset \mathcal{P}$ to corrupt. The parties then engage in an execution of a real n-party updatable MPC protocol $\Pi = (\Pi_{\mathsf{init}}, \Pi_{\mathsf{upd}})$ for initial circuit $C \in \mathcal{C}$ that consists of two stages, namely, (a) an initial computation phase, (b) an *update phase*, where the latter can be repeated polynomially many times. Throughout the execution of Π, the adversary $\mathcal{A}$ sends all messages on behalf of the corrupted parties, and may follow an arbitrary polynomial-time strategy. In contrast, the honest parties follow the instructions of Π.

INITIAL COMPUTATION PHASE: Let x_i be the initial input of party P_i. In this phase, all the parties execute protocol Π_{init}, where each honest party $P_i \in \mathcal{P} \setminus M$ acts in accordance with its input x_i. At the end of the protocol, each honest party computes an initial output as well as state $\mathsf{st}_{i,0}$ in accordance with the protocol. If the protocol computation ends in an abort, then each honest party sets its output and state to $\perp$.

ℓTH UPDATE PHASE: Whenever the parties wish to perform a computation over an updated circuit and input vector, they run an execution of Π_{upd}, where each honest party acts in accordance with its input $(\mathsf{st}_{i,\ell-1}, u_{i,\ell})$. At the end of the protocol, each honest party computes an output and an updated state st_ℓ, both of which may be set to $\perp$ if the protocol ends in an abort.

At the conclusion of all the update phases, each honest party P_i outputs all the outputs it obtained in the computations. Malicious parties may output an arbitrary PPT function of the view of $\mathcal{A}$.

For any adversary $\mathcal{A}$ with auxiliary input $z \in \{0,1\}^*$, input vector $\boldsymbol{x}$, any arbitrary polynomial set of updates $\{\mathbf{u}_\ell\}_{\ell=1}^{\mathsf{q}}$, and security parameter λ, we denote the output of the multi-function MPC protocol $\Pi = (\Pi_{\mathsf{init}}, \Pi_{\mathsf{Upd}})$ by

$$\mathsf{REAL}^{\Pi}_{\mathcal{A},M}\left(1^\lambda, \boldsymbol{x}, z, \{\mathbf{u}_\ell\}_{\ell=1}^{\mathsf{q}}\right).$$

Efficiency. We require that the total communication complexity of any update phase ℓ is a fixed polynomial in the size of the update length $|\mathbf{u}_\ell|$ and the security parameter.

2.8.5.3 Construction of UMPC

In this section, we construct a UMPC protocol for general circuits.

Let $\mathcal{C}$ be a n-input $(\mathsf{Upd}, \mathcal{U})$-updatable circuit family and let $\mathcal{C}[X]$ denote the corresponding updatable hardwired circuit family. In order to construct a UMPC protocol for $\mathcal{C}$, we will use the following ingredients in our construction:

1. A stateless[19] URE scheme (URE.Encode, URE.GenUpd, URE.ApplyUpd, URE.Decode) for an $(\mathsf{Upd}_{\mathsf{ure}}, \mathcal{U}_{\mathsf{ure}})$-updatable class of hardwired circuits $\mathcal{C}[X]$ where $\mathcal{U}_{\mathsf{ure}} = \mathcal{U}^n$

and $\mathsf{Upd}_{\mathsf{ure}}$ is the same as Upd: for any $C[\boldsymbol{x}] \in \mathcal{C}[X]$ and any $\mathbf{u} \in \mathcal{U}_{\mathsf{ure}}$ where $\mathbf{u} = (u_1, \ldots, u_n)$, $\mathsf{Upd}_{\mathsf{ure}}(C[\boldsymbol{x}], \mathbf{u}) = \mathsf{Upd}(C[\boldsymbol{x}, \mathbf{u})$.

2. A standard n-party MPC protocol Π_{mpc} for general circuits that is secure against arbitrary, static corruptions.

PROTOCOL Π_{init}: Let $C_0 \in \mathcal{C}$ be the level-0 circuit that the parties wish to compute and $x_{i,0}$ denote the level-0 input of party P_i. Protocol Π_{init} consists of the following two steps:

1. First, each party P_i privately samples a random string r_i.
2. Next, the parties engage in an execution of Π_{mpc} for computing the following function f_0: it takes as input $(C_0, x_{i,0}, r_i)$ from party P_i and computes $\langle C_0[\boldsymbol{x}_0]\rangle_{\mathsf{ure}} \leftarrow \mathsf{URE.Encode}(C_0[\boldsymbol{x}_0]; r)$ using randomness $r = r_1 \oplus \cdots \oplus r_n$, where $\boldsymbol{x}_0 = (x_{1,0}, \ldots, x_{n,0})$. The output of f is $\langle C_0[\boldsymbol{x}_0]\rangle_{\mathsf{ure}}$.
3. At the end of Π_{mpc}, each party computes $y_0 \leftarrow \mathsf{URE.Decode}(\langle C_0[\boldsymbol{x}_0]\rangle_{\mathsf{ure}})$ and outputs y_0. Each party P_i stores $\mathsf{st}_{i,0} = (\langle C[\boldsymbol{x}]\rangle_{\mathsf{ure}}, r_i)$.

PROTOCOL Π_{upd}: Let $u_{i,\ell}$ denote the ℓth update string corresponding to party P_i. Let $\mathsf{st}_{i,\ell} = (\langle C_{\ell-1}[\boldsymbol{x}_{\ell-1}]\rangle_{\mathsf{ure}}, r_i)$ be the state of party P_i at the start of the ℓth update phase. Then, the ℓth execution of protocol Π_{upd} consists of the following two steps:

1. First, the parties engage in an execution of Π_{mpc} for computing the following (randomized) function f_ℓ: it takes as input $(u_{i,\ell}, r_i)$ from party P_i and computes $\langle \mathbf{u}_\ell\rangle_{\mathsf{ure}} \leftarrow \mathsf{URE.GenUpd}(\mathbf{u}_\ell, r)$ where $\mathbf{u}_\ell = (u_{1,\ell}, \ldots, u_{n,\ell})$ and $r = r_1 \oplus \cdots \oplus r_n$. The output of f_ℓ is $\langle \mathbf{u}_\ell\rangle_{\mathsf{ure}}$.
2. At the end of Π_{mpc}, each party computes

$$\langle C_\ell[\boldsymbol{x}_\ell]\rangle_{\mathsf{ure}} \leftarrow \mathsf{URE.ApplyUpd}(\langle C_{\ell-1}[\boldsymbol{x}_{\ell-1}]\rangle_{\mathsf{ure}}, \langle \mathbf{u}_\ell\rangle_{\mathsf{ure}}).$$

[19]Our construction also works if we start with a stateful URE scheme. For simplicity of exposition, however, we present our construction using a stateless URE scheme.

3. Next, each party computes $y_\ell \leftarrow \mathsf{URE.Decode}(\langle C_\ell[\boldsymbol{x}_\ell]\rangle_{\mathsf{ure}})$ and outputs y_ℓ. Finally, P_i updates its state to $\mathsf{st}_i = (\langle C_\ell[\boldsymbol{x}_\ell]\rangle_{\mathsf{ure}}, r_i)$.

2.8.5.4 Proof Sketch

It is easy to see that the above construction satisfies our desired efficiency. Specifically, since the ℓ^{th} update phase involves the execution of a standard MPC protocol π_{mpc} to compute an update encoding $\langle \mathbf{u}_\ell\rangle_{\mathsf{ure}}$, it follows from the efficiency of URE that the the size of the function f_ℓ computed by the MPC protocol is a fixed polynomial in the size of $|\mathbf{u}_\ell|$ and the security parameter. It follows then from the efficiency of a standard MPC that the total communication complexity of the ℓ^{th} update phase is a fixed polynomial in the size of $|\mathbf{u}_\ell|$ and the security parameter.

Next, we argue security of our construction. We start with hybrid H_0 that corresponds to the real world experiment. In hybrid H_1 we all the executions in π_{mpc}. Note that here we still use the inputs and the updates of the honest parties. Finally, in H_2 we simulate the output of each execution of π_{mpc}, i.e., we simulate the URE encoding $\langle C_0[\boldsymbol{x}_0]\rangle_{\mathsf{ure}}$ and the update encodings $\langle \mathbf{u}_\ell\rangle_{\mathsf{ure}}$ using the URE simulator who is provided the outputs using the trusted party in the ideal world. This experiment corresponds to the simulator's algorithm.

The indistinguishability of H_0 and H_1 follows easily from the security of the MPC protocol π_{mpc}. The indistinguishability of H_1 and H_2 follows from the security of URE. This completes our proof sketch.

Chapter 3

The Inadequacy of CPA Security for Proxy Reencryption

3.1 Introduction

Consider three parties: Alice, Bob, and Polly Proxy. Alice keeps encrypted data (created with a public key) that she can decrypt with a secret key known only to her. She wants to communicate some of the data to Bob, but not to Polly (nor anybody else). Assuming Alice and Polly know Bob's public key, how can she communicate the data to him?

If she is willing to entrust Bob with all her secrets, past and future, Alice might try to tell Bob her secret decryption key by encrypting it using Bob's public key. We call this the *Trivial Scheme.* A less trusting Alice can instead decrypt the data and reencrypt it using Bob's public key. But what if Alice does not want to do the work of decrypting and reencrypting large amounts of data?

Proxy reencryption (PRE) provides an elegant solution: Alice creates and gives to Polly a *reencryption key* that will enable Polly to reencrypt her data under Bob's public key for his use, but that will not reveal the data to Polly. Proxy reencryption guarantees that Bob can recover the data uncorrupted (correctness) and that Polly cannot learn anything about Alice's data (security). The most widely-studied model

Based on "What about Bob? The inadequacy of CPA security for proxy re-encryption" [Coh19].

of security for proxy reencryption is called *CPA security*, named after the corresponding notion from standard encryption on which it is based.

But what about Bob? As already observed, if we do not require any security against Bob, proxy reencryption is trivial: Alice uses the Trivial Scheme, simply sending Bob her encrypted secret key. This is undesirable, unsatisfying, and insufficient for a number of supposed applications of proxy reencryption (Section 3.2).

Surprisingly, *the Trivial Scheme is a CPA secure proxy reencryption scheme* when the public key encryption scheme used is circularly secure [BHHO08]! Bob completely learns Alice's secret key, and circular security is used only to prove security against a malicious Polly. Furthermore, the CPA-security of any proxy reencryption scheme remains uncompromised if Polly attaches the reencryption key to every reencrypted ciphertext sent to Bob, even though this would enable Bob to decrypt messages encrypted under Alice's public key (Section 3.3.1).

These "constructions" of CPA-secure proxy reencryption demonstrate the inadequacy of CPA security for proxy reencryption. If they had been observed previously, perhaps CPA security would not have gained the traction that it has.

Throughout this chapter, we use CPA (respectively, CCA and HRA) to refer to the security notion for proxy reencryption, and Enc-CPA (resp., Enc-CCA) to refer to the security notion for standard encryption. We restrict our attention to *unidirectional* proxy reencryption, where the reencryption key allows Alice's ciphertexts to be reencrypted to Bob's key, but not the reverse. In a bidirectional scheme, Bob— using his own secret key and Alice's public key— is able compute the Alice-to-Bob reencryption key himself; thus a lack of security against Bob is inherent.

3.1.1 CPA and CCA Security of Proxy Reencryption

First considered by Blaze, Bleumer, and Strauss [BBS98b], proxy reencryption has received significant and continuous attention in the last decade, including definitions [ID03, AFGH06, CH07, NAL15b], number-theoretical constructions [ABH09, LV08, CWYD10], lattice-based constructions [Gen09a, ABW$^+$13, PWA$^+$16, FL17], implementations [LPK10, HHY11, PRSV17, BPR$^+$17], and early success in program obfuscation [HRSV07, CCL+14].

Adapting notions from standard encryption, this literature considers two main indistinguishability-based security notions for proxy reencryption: security under *chosen plaintext attacks (CPA)* [ABH09] and *chosen ciphertext attacks (CCA)* [CH07]. While CCA security is considered the gold-standard, CPA security has received significant attention [AFGH06, ABH09, HRSV07], especially in latticed-based constructions [Gen09a, ABW+13, PWA+16, PRSV17]. CPA security has been used as a testing ground for new techniques for proxy reencryption and in settings where efficiency concerns make the added security of CCA undesirable.

We now briefly describe the definitions of CPA and CCA security for proxy reencryption, with the goal of communicating the underlying intuition. For this informal description, we restrict our attention to the limited three party setting of Alice, Bob, and Polly and strip away many of the complexities of the full definition. For a full definitions of CPA and CCA security, see Definitions 3.3.3 and 3.6.1 respectively.

Both notions are typically defined using a security game between an adversary and a challenger in which the adversary's task is to distinguish between encryptions of two messages. Both notions allow the adversary to corrupt either Bob (learning $\mathsf{sk}_{\mathsf{bob}}$) or Polly (learning the reencryption key rk). CCA and CPA security differ in the additional power granted to the adversary.

CCA security grants the adversary access to two oracles:

- $\mathcal{O}_{\mathsf{Dec}}$: The decryption oracle takes as input a ciphertext along with the public key of either Alice or Bob, and outputs the decryption of the ciphertext using the corresponding secret key.

- $\mathcal{O}_{\mathsf{ReEnc}}$: The reencryption oracle takes as input a ciphertext $\mathsf{ct}_{\mathsf{alice}}$ and outputs a reencrypted ciphertext $\mathsf{ct}_{\mathsf{bob}}$.

These oracles come with restrictions to prevent the adversary from simply reencrypting or decrypting the challenge ciphertext, adapting replayable chosen-ciphertext security of standard encryption (Enc-CCA) in the natural way.

CPA security of proxy reencryption, however, removes both oracles.[1] Why? First, to adapt chosen-plaintext security from standard encryption (Enc-CPA) to proxy reencryption, we must of course do away with $\mathcal{O}_{\mathsf{Dec}}$. Secondly, it seems we must also remove $\mathcal{O}_{\mathsf{ReEnc}}$: otherwise, by corrupting Bob it seems that the adversary can use the combination of $\mathcal{O}_{\mathsf{ReEnc}}$ and $\mathsf{sk}_{\mathsf{bob}}$ to simulate $\mathcal{O}_{\mathsf{Dec}}$. Removing both decryption and reencryption oracles, according to [ABH09], naturally extends the Enc-CPA security to proxy reencryption, yielding CPA security.

As we observe in this thesis, a natural definition is not always a good definition. Not only is the above intuition for removing $\mathcal{O}_{\mathsf{ReEnc}}$ false (Theorem 3.6.4), but CPA security as defined above guarantees little against a honest-but-curious Bob, even under *normal operation.* The definition only requires that the adversary will not win the game as long as it never sees any reencrypted ciphertexts. It guarantees nothing if Bob sees even a single reencrypted ciphertext. This vulnerability is not purely theoretical: in the CPA secure scheme of [PRSV17], Bob can recover Alice's secret key with significant probability from a single reencrypted ciphertext (Theorem 3.5.1).

This makes CPA security ill-suited for the most commonly cited applications of proxy reencryption, including forwarding of encrypted email and single-writer, many-reader encrypted storage (Section 3.2). CPA security is inadequate for proxy reencryption and must be replaced.

3.1.2 Security Against Honest Reencryption Attacks

What minimal guarantees should proxy reencryption provide? First, it should offer security against a dishonest proxy Polly when Alice and Bob are honest and using the proxy reencryption as intended. Polly's knowledge of a reencryption key from Alice to Bob (or vice versa) should not help her learn anything about the messages underlying ciphertexts encrypted under $\mathsf{pk}_{\mathsf{alice}}$ or $\mathsf{pk}_{\mathsf{bob}}$. Such security against the corrupted proxy is guaranteed by CPA.

[1]This description is an oversimplification. In the many party setting, the adversary has access to a reencryption oracle which will reencrypt ciphertexts between two uncorrupted parties or between two corrupted parties, but not from an honest party to a corrupted party.

Second, proxy reencryption should offer security against a dishonest receiver Bob when Alice and Polly are honest and using the proxy reencryption as intended. Bob's knowledge of *honestly reencrypted ciphertexts* (that were honestly generated to begin with) should not help him learn anything about the messages underlying other ciphertexts encrypted under $\mathsf{pk}_{\mathsf{alice}}$ that have not been reencrypted. As we show, such security against the corrupted receiver is not guaranteed by CPA.

Generalizing these dual guarantees to many possibly colluding parties, we want security as long as the adversary only sees honestly reencrypted ciphertexts. In Section 3.4, we formalize this notion as proxy reencryption security against *honest reencryption attacks (HRA)*. Recall that CCA security provides the adversary with both $\mathcal{O}_{\mathsf{Dec}}$ and $\mathcal{O}_{\mathsf{ReEnc}}$ while CPA provides neither oracle. In contrast, HRA security provides the adversary with a restricted reencryption oracle which will only reencrypt honestly generated ciphertexts.

By guaranteeing security of both kinds described above, HRA is a strengthening of CPA security that better captures our intuitions for security of proxy reencryption. Furthermore, HRA guarantees more meaningful security in the most common applications of proxy reencryption (Section 3.4.1). HRA security is an appropriate goal when developing new techniques for proxy reencryption and in settings where full CCA security is undesirable or out of reach.

Security of existing schemes. Can we construct a proxy reencryption scheme that is HRA secure? HRA security is a strict strengthening of CPA security,so it is not immediately clear that any existing constructions are HRA secure without redoing the proofs from scratch. Indeed, the CPA secure scheme of [PRSV17] is not HRA secure (Theorem 3.5.1).

In Section 3.5, we identify a property—*reencryption simulatablity*—which is sufficient to boost CPA security to HRA security. Very roughly, reencryption simulatability means that ciphertexts resulting from computing $\mathsf{ReEnc}(\mathsf{rk}_{\mathsf{alice}\to\mathsf{bob}}, \mathsf{ct}_{\mathsf{alice}})$ can be simulated without knowledge of the secret key $\mathsf{sk}_{\mathsf{alice}}$ (but with knowledge of the plaintext message m). Reencryption simulatability allows a reduction with access to the CPA oracles to efficiently implement the honest reencryption oracle, thereby reducing HRA security to CPA security.

We the examine the simple construction of proxy reencryption from any fully-homomorphic encryption [Gen09a], and the pairing-based construction of [AFGH06]. In the first case, if the fully-homomorphic encryption secure is circuit private, then the resulting proxy reencryption scheme is reencryption simulatable. In the second case, rerandomizing reencrypted ciphertexts suffices for reencryption simulation.[2]

3.1.3 Related Work

The below mentioned works are just the most directly relevant. There is by now an extensive research literature on proxy reencryption, presenting a zoo of definitions. There have been three main approaches to defining security: CPA, CCA, and (to a much lesser extent) obfuscation-based. The CPA notion, in one form or another, is by far the most well studied. We make the deliberate choice to address the core CPA definition, not to present an ultimate definition of security for proxy reencryption nor to address the vast array of different criticisms or strengthenings of CPA security that have been or may be considered. We hope that doing so will make these ideas more understandable and adaptable.

3.1.3.1 RIND-CPA Security

In concurrent and independent work defining and constructing forward-secure proxy reencryption, Derler, Krenn, Lorünser, Ramacher, Slamanig, and Striecks identify the same problem with CPA security as discussed in this chapter [DKL$^+$18, Definition 14]. As in our work, they address the problem with CPA security by defining a new security notion—RIND-CPA security—which expands the power of the adversary. They additionally separate RIND-CPA and CPA security with a construction that is essentially our Concatenation Scheme.

However, this is where the resemblance between [DKL$^+$18] and our work ends.

[2]While we don't examine every pairing-based construction of proxy reencryption, we suspect that rerandomizing reencryption will suffice for reencryption simulation in many, if not all.

In the RIND-CPA game offered by [DKL+18], the adversary gets access to an reencryption oracle that works on all inputs (not just honestly generated ones), but only before the challenge ciphertext is generated.[3] In contrast, HRA allows reencryption both before and after the challenge, but only for honestly generated ciphertexts.

RIND-IND is inadequate as a replacement for CPA security in the research literature: its usefulness in applications is unclear, and it appears too strong to provide a useful testing ground for the development of new techniques for constructing proxy reencryption.

In the course of normal operation of a proxy reencryption in applications, an adversarial party will typically see many reencrypted ciphertexts. These ciphertexts may come at any time—both before and after other ciphertexts whose contents should remain secret. HRA is meaningful in many such applications—many more than CPA security. But because RIND-CPA restricts the reencryption oracle to the period before the challenge ciphertext, its usefulness in applications is not clear.

The challenge of proving CCA security for encryption (proxy or otherwise) is demonstrating that an adversary cannot use dishonestly generated, malformed ciphertexts to win in the security game. In this respect, RIND-CPA security is much more akin to CCA security than to CPA security. HRA, on the other hand, makes minimal assumptions about the distribution of plaintext messages by allowing the adversary to choose messages itself, just as in Enc-CPA for standard encryption.

Appendix 3.7 discusses RIND-CPA security in more depth, expanding on the arguments above and proving that RIND-CPA and HRA security are incomparable.

3.1.3.2 Subsequent Work

Two subsequent works continue the study of HRA secure proxy reencryption. Fuchsbauer, Kamath, Klein, and Pietrzak study CPA and HRA secure proxy reencryption in an adaptive corruption model [FKKP18]. As in our work, they prove the HRA security of their construction by first proving CPA security and then lifting it to full HRA security using a version of reencryption simulatability.

[3] Appendix 3.6 discusses the related definition of $\mathsf{ind\text{-}cca}_{0,1}$ security from [NAL15b]

More recently, Dottling and Nishimaki study the problem of converting ciphertexts between different public-key encryption schemes, a problem they call universal proxy reencryption [DN18]. They define security by extending HRA security to the universal setting. [DN18] extends Theorem 3.5.3 to show that a computational version of reencryption simulatability suffices to lift CPA to HRA security. However, they prove HRA security directly rather, finding that proving computational reencryption simulatability is not much more simple than proving HRA security itself.

3.1.3.3 Other Related Work

Our dual-guarantee conception of proxy reencryption security mirrors the security requirements of what Ivan and Dodis call CPA security [ID03]. Their notion differs substantially from what is now referred to by that name. The [ID03] conception of CPA security is only defined in a proof in the appendix of that work and seems to have been completely overlooked by the later works proposing the modern notion of CPA security (beginning with [AFGH06] and then in its present form in [ABH09]). If, however, Ivan and Dodis had undertaken to revisit proxy reencryption after [ABH09], they might have proposed a security definition similar to HRA.

In [NAL15b], Nuñez, Agudo, and Lopez provide a parameterized family of CCA-type attack models for proxy reencryption. Their weakest model corresponds to CPA security and their strongest to full CCA security. That work is partially a response to a claimed construction of CCA-1 secure proxy reencryption in a security model that does not allow reencryption queries. [NAL15b] provide an attack on the construction in the presence of a reencryption oracle consisting of carefully constructed, dishonestly generated queries which leak the reencryption key. They do not consider restricting the reencryption oracle in the security game to honestly generated ciphertexts. We discuss [NAL15b] further in Appendix 3.6.3.

In an independent work [LPZ19], Liu, Pan, and Zhang discover an HRA-style vulnerability in an NTRU based proxy reencryption scheme proposed in [NAL15a]. They show that a recipient of many honestly reencrypted ciphertexts can recover the sender's secret key.

A parallel line of work initiated by Hohenberger, Rothblum, shelat, Vaikuntanathan which studies proxy reencryption using an obfuscation-based definition that is incomparable to CPA security [HRSV07]. Their definition requires that the functionality of the obfuscated reencryption circuit be statistically close to that of the ideal reencryption functionality: namely, that $\mathsf{ReEnc}(\mathsf{rk}_{i\to j}, \mathsf{Enc}(\mathsf{pk}_i, \mathsf{m})) \approx_s \mathsf{Enc}(\mathsf{pk}_j, \mathsf{m})$. Thus the definition of [HRSV07] (and even the relaxed correctness found in [CCL$^+$14]) imply *reencryption simulatability* defined in Section 3.5.

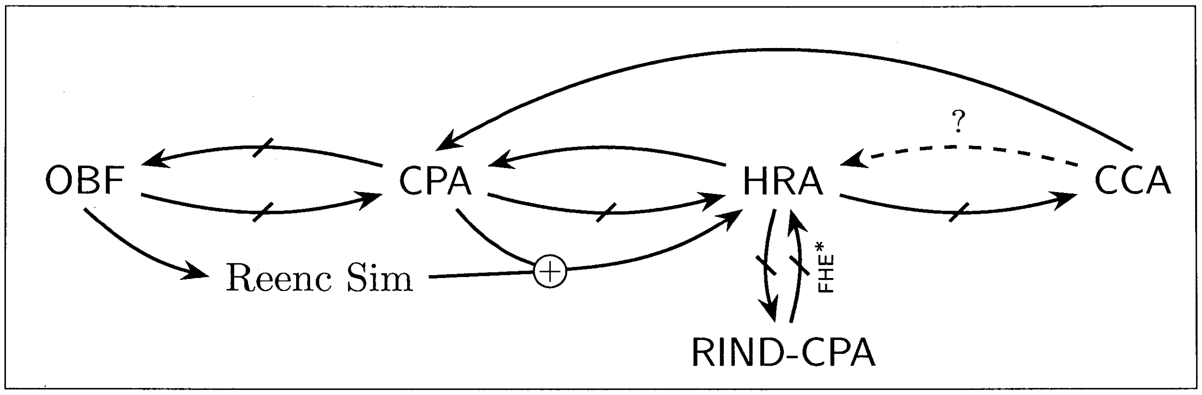


Figure 3-1: Summary of relationships between various security notions. Plain arrows denote implications and slashed arrows denote separations. OBF refers to the obfuscation-based notion of [HRSV07] and Reenc Sim refers to Definition 3.5.2. The separation between RIND-CPA and HRA assumes the existence of a special FHE scheme. CPA and Reenc Sim together imply HRA. Whether CCA implies HRA is unknown.

Organization

We begin by discussing applications of proxy reencryption and identifying the weaknesses of CPA security in those applications (Section 3.2). Then we present the existing CPA security definition and further demonstrate its weaknesses with two new schemes: the Trivial Scheme and Concatentation Scheme (Section 3.3). We propose a new security notion to overcome those weaknesses: security against honest reenecryption attacks (HRA) (Section 3.4). We examine the relationship between CPA and HRA security and the HRA security (or insecurity) of existing reencryption schemes (Section 3.5). Finally, discuss HRA security's relation to CCA security (Section 3.6) and RIND-CPA security (Section 3.7).

3.2 Insufficiency of CPA Security for Applications

In Section 3.3, we recall the definition of CPA security of proxy reencryption from [ABH09] and formalize the Trivial Scheme from the introduction satisfying the notion. In the Trivial Scheme, Bob learns Alice's secret key after receiving a single reencrypted ciphertext.

We are faced with a choice: accept the existing definition of CPA security, or reject it and seek a definition that better captures our intuitions. In support of the latter, we describe a number of applications of proxy reencryption proposed in the literature for which CPA security (as implemented by the Trivial Scheme) is potentially unsatisfactory, but for which full CCA security may not always be necessary.[4] We revisit these applications in Section 3.4.1 after proposing our new security notion.

Encrypted Email Forwarding [BBS98b, Jak99, AFGH06]. Forwarding of encrypted email without requiring the sender's participation might be desirable for temporary delegation during a vacation [Jak99] or for spam filtering [AFGH06]. Does the Trivial Scheme suffice? The Trivial Scheme enables Bob, the receiver of Alice's forwarded (and reencrypted) email, to recover Alice's secret key. If Alice trusts Bob enough to use the Trivial Scheme, she could instead reveal her secret key. The Trivial Scheme might be preferable in very specific trust or interaction models, but it does not offer meaningful security against Bob if Alice only wishes to forward a subset of emails (for example, from particular senders or during a specific time period).

Key Escrow [ID03]. Similar to email forwarding, Ivan and Dodis describe the application of key escrow as follows: "The problem is to allow the law enforcement agency to read messages encrypted for a set of users, for a limited period of time, without knowing the users' secrets. The solution is to locate a key escrow

[4]We might also appeal for support to [ID03], the only paper in the proxy reencryption literature of which we are aware adopting a security definition providing a reencryption oracle without a decryption oracle. One could look to the originators of proxy reencryption for guidance, but the shortcoming we identify does not manifest in the original setting of [BBS98b] (there is only Alice and Bob; there is no Proxy). It is therefore of little help.

agent between the users and the law enforcement agency, such that it controls which messages are read by the law enforcement agencies." As in email forwarding, the "for a limited period of time" requirement suggests that Ivan and Dodis would not have been satisfied with the Trivial Scheme.[5]

Single-Writer, Many-Reader Encrypted Storage [AFGH06, KHP06, LPK10, PRSV17]. Under different monikers (including DRM and publish/subscribe systems), these works describe systems in which a single privileged writer encrypts data and determines an access control policy for readers. A semi-honest proxy server is entrusted with reencryption keys and is tasked with enforcing the access control policy. Whether the Trivial Scheme suffices for these applications depends on what sort of access control policies are envisioned. If the access is *all or nothing* (i.e., all readers may access all data), the Trivial Scheme suffices; if the access is *fine grained* (i.e., each reader may access only a specific subset of the data), then it does not. Existing works are often unclear on which is envisioned.

For completeness, we mention that CPA security does suffice for two important applications of proxy reencryption: namely, key rotation for encrypted cloud storage [BLMR13a, EPRS17] and bootstrapping fully homomorphic encryption [Gen09a].

3.3 Security Against Chosen Plaintext Attacks

In this section, we recall the definition of CPA security for proxy reencryption and illustrate its shortcomings. We begin with the definitions of syntax, correctness, and CPA security from [ABH09, Definition 2.2] (with minor changes in notation and

[5]Note that Ivan and Dodis do not adopt the CPA definition used elsewhere, but a definition much closer to our own. There is no gap between their security guarantees and the requirements of their briefly-described application.

Though primarily focused on the setting where the key escrow agent enforces the limited time requirement by eventually refusing to reencrypt, [ID03] considers the possibility of dividing time into epochs and enforcing the time limitation technically. Such a proxy reencryption is called *temporary* in [AFGH06]. We do not discuss temporary proxy reencryption further.

presentation, and the change noted in Remark 3.3.5 at the end of this subsection). The syntax and correctness requirements are common to CPA, HRA, and CCA security.

For the sake of concreteness, simplicity, and brevity, we restrict the discussion to unidirectional, single-hop schemes. In multi-hop schemes, reencryption keys $\mathsf{rk}_{A\to B}$ and $\mathsf{rk}_{B\to C}$ can be used to reencrypt a ciphertext ct_A from pk_A to pk_C. In single-hop schemes, they cannot. Single-hop or multi-hop schemes each have their benefits and drawbacks, and works typically focus on one or the other notion.[6] To the best of our knowledge, our observations and results can all be adapted to the multi-hop setting.

Definition 3.3.1 (Proxy Reencryption: Syntax [ABH09]). *A proxy reencryption scheme for a message space $\mathcal{M}$ is a tuple of algorithms* $\mathsf{PRE} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{ReKeyGen}, \mathsf{Enc}, \mathsf{ReEnc}, \mathsf{Dec})$*:*

SETUP: $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$ *takes as input the security parameter in unary; it outputs the public parameters* pp.

KEY GENERATION: $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(\mathsf{pp})$ *takes as input the public parameters; it outputs a public key* pk *and a secret key* sk. *For ease of notation, we assume that both* pk *and* sk *include* pp *and refrain from including* pp *as input to other algorithms.*

REENCRYPTION KEY GENERATION: $\mathsf{rk}_{i\to j} \to \mathsf{ReKeyGen}(\mathsf{sk}_i, \mathsf{pk}_j)$ *takes as input a secret key* sk_i *and a public key* pk_j*, where $i \neq j$, the* reencryption key generation *algorithm; it outputs a reencryption key* $\mathsf{rk}_{i\to j}$.

[6]The literature is divided about whether "single-hop" is merely a correctness property (i.e., able to reencrypt at least once, but agnostic about whether reencrypting more than once is possible) or if it is also a security property (i.e., a ciphertext can be reencrypted once, but never twice). This distinction manifests in the security definition. In works that consider only single-hop correctness [AFGH06, ABH09, HRSV07, NAL15b], the oracle $\mathcal{O}_{\mathsf{ReKeyGen}}$ in the security game will not accept queries from honest users to corrupted users (i.e., (i, j) such that $i \in \mathsf{Hon}$ and $j \in \mathsf{Cor}$). We adopt this formalism in Definitions 3.3.3 and 3.4.1 for simplicity of presentation only.

In works that consider single-hop security [LV08, CWYD10, FL17], the oracle will answer such queries, but the challenge ciphertext must be encrypted under the key of an honest user i^* for which no such reencryption key was generated (which can be formalized in a number of ways).

We adopt the simplest model, requiring only one hope of correctness, but neither requiring nor forbidding additional functionality.

ENCRYPTION: $\mathsf{ct}_i \leftarrow \mathsf{Enc}(\mathsf{pk}_i, \mathsf{m})$ *takes as input a public key* pk_i *and a message* $\mathsf{m} \in \mathcal{M}$*; it outputs a ciphertext* ct_i.

REENCRYPTION: $\mathsf{ct}_j \leftarrow \mathsf{ReEnc}(\mathsf{rk}_{i\to j}, \mathsf{ct}_i)$ *takes as input an i-to-j reencryption key* $\mathsf{rk}_{i\to j}$ *and a ciphertext* ct_i*; it outputs a ciphertext* ct_j *or the error symbol* $\perp$.

DECRYPTION: $\mathsf{m} \leftarrow \mathsf{Dec}(\mathsf{sk}_j, \mathsf{ct}_j)$ *takes as input a secret key* sk_j *and a ciphertext* ct_j*; it outputs a message* $\mathsf{m} \in \mathcal{M}$ *or the error symbol* $\perp$.

Definition 3.3.2 (Proxy Reencryption: Correctness [ABH09]). *A proxy reencryption scheme* PRE *is correct with respect to message space* $\mathcal{M}$ *if for all* $\lambda \in \mathbb{N}$, $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$*, and* $\mathsf{m} \in \mathcal{M}$*:*

- *For all* $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(\mathsf{pp})$*:*

$$\mathsf{Dec}(\mathsf{sk}, \mathsf{Enc}(\mathsf{pk}, \mathsf{m})) = \mathsf{m}.$$

- *For all* $(\mathsf{pk}_i, \mathsf{sk}_i), (\mathsf{pk}_j, \mathsf{sk}_j) \leftarrow \mathsf{KeyGen}(\mathsf{pp})$, $\mathsf{rk}_{i\to j} \leftarrow \mathsf{ReKeyGen}(\mathsf{sk}_i, \mathsf{pk}_j)$*:*

$$\mathsf{Dec}(\mathsf{sk}_j, \mathsf{ReEnc}(\mathsf{rk}_{i\to j}, \mathsf{Enc}(\mathsf{pk}_i, \mathsf{m}))) = \mathsf{m}.$$

Security is modeled by a game played by an adversary $\mathcal{A}$. $\mathcal{A}$ has the power to corrupt a set of users Cor (learning their secret keys) while learning only the public keys for a set of honest users Hon. Additionally, $\mathcal{A}$ may reencrypt ciphertexts (either by getting a reencryption key or calling a reencryption oracle) between pairs of users in Hon or in Cor, or from Cor to Hon, but not from Hon to Cor. This is in contrast to the simplified three-party setting discussed in the introduction, where the $\mathcal{A}$ could not reencrypt whatsoever.

Definition 3.3.3 (Proxy Reencryption: Security Game for Chosen Plaintext Attacks (CPA) [ABH09]). *Let* λ *be the security parameter and* $\mathcal{A}$ *be an oracle Turing machine. The CPA game consists of an execution of* $\mathcal{A}$ *with the following oracles. The game consists of three phases, which are executed in order. Within each phase, each oracle can be executed in any order,* $\text{poly}(\lambda)$ *times, unless otherwise specified.*

Phase 1:

SETUP: *The public parameters are generated and given to* $\mathcal{A}$. *A counter* $\mathsf{numKeys}$ *is initialized to 0, and the sets* Hon *(of honest, uncorrupted indices) and* Cor *(of corrupted indices) are initialized to be empty. This oracle is executed first and only once.*

UNCORRUPTED KEY GENERATION: *Obtain a new key pair* $(\mathsf{pk}_{\mathsf{numKeys}}, \mathsf{sk}_{\mathsf{numKeys}}) \leftarrow \mathsf{KeyGen}(\mathsf{pp})$ *and give* $\mathsf{pk}_{\mathsf{numKeys}}$ *to* $\mathcal{A}$. *The current value of* $\mathsf{numKeys}$ *is added to* Hon *and* $\mathsf{numKeys}$ *is incremented.*

CORRUPTED KEY GENERATION: *Obtain a new key pair* $(pk_{\mathsf{numKeys}}, \mathsf{sk}_{\mathsf{numKeys}}) \leftarrow \mathsf{KeyGen}(\mathsf{pp})$ *and given to* $\mathcal{A}$. *The current value of* $\mathsf{numKeys}$ *is added to* Cor *and* $\mathsf{numKeys}$ *is incremented.*

Phase 2: *For each pair* $i, j \leq \mathsf{numKeys}$, *compute the reencryption key* $\mathsf{rk}_{i \to j} \leftarrow \mathsf{ReKeyGen}(\mathsf{sk}_i, \mathsf{pk}_j)$.

REENCRYPTION KEY GENERATION $\mathcal{O}_{\mathsf{ReKeyGen}}$: *On input* (i, j) *where* $i, j \leq \mathsf{numKeys}$, *if* $i = j$ *or if* $i \in \mathsf{Hon}$ *and* $j \in \mathsf{Cor}$, *output* $\perp$. *Otherwise return the reencryption key* $\mathsf{rk}_{i \to j}$.

REENCRYPTION $\mathcal{O}_{\mathsf{ReEnc}}$: *On input* (i, j, ct_i) *where* $i, j \leq \mathsf{numKeys}$, *if* $i = j$ *or if* $i \in \mathsf{Hon}$ *and* $j \in \mathsf{Cor}$, *output* $\perp$. *Otherwise return the reencrypted ciphertext* $\mathsf{ReEnc}(\mathsf{rk}_{i \to j}, \mathsf{ct}_i)$.

CHALLENGE ORACLE: *On input* $(i, \mathsf{m}_0, \mathsf{m}_1)$ *where* $i \in \mathsf{Hon}$ *and* $\mathsf{m}_0, \mathsf{m}_1 \in \mathcal{M}$, *sample a bit* $b \leftarrow \{0, 1\}$ *uniformly at random, and return the challenge ciphertext* $\mathsf{ct}^* \leftarrow \mathsf{Enc}(\mathsf{pk}_i, \mathsf{m}_b)$. *This oracle can only be queried once.*

Phase 3:

DECISION: *On input a bit* $b' \in \{0, 1\}$, *return* win *if* $b = b'$.

The CPA advantage *of $\mathcal{A}$ is defined as*

$$\mathsf{Adv}^{\mathcal{A}}_{\mathsf{cpa}}(\lambda) = \Pr[\mathsf{win}],$$

where the probability is over the randomness of $\mathcal{A}$ and the oracles in the CPA game.

Definition 3.3.4 (Proxy Reencryption: CPA Security [ABH09])**.** *Given a security parameter 1^λ, a proxy reencryption scheme is CPA secure if for all probabilistic polynomial-time adversaries $\mathcal{A}$, there exists a negligible function* negl *such that*

$$\mathsf{Adv}^{\mathcal{A}}_{\mathsf{cpa}}(\lambda) < \frac{1}{2} + \mathsf{negl}(\lambda)$$

Remark 3.3.5. Another definitional subtlety of proxy reencryption worth discussing affects not just CPA security, but HRA and CCA security as well. Consider the specification of $\mathcal{O}_{\mathsf{ReKeyGen}}$ and $\mathcal{O}_{\mathsf{ReEnc}}$ in Definition 3.3.3. As defined, the reencryption keys $\mathsf{rk}_{i \to j}$ are *persistent*: the same key is used each time a pair (i, j) is queried. This follows [ABH09, Definition 2.5] and [ABW+13, FL17], though we find our formalization somewhat simpler.

Contrast this with [ABH09, Definition 2.2] in which reencryption keys are *ephemeral*: they are generated afresh each time either oracle is invoked on the same pair (i, j). [BLMR13a, PWA+16, CH07] similarly use ephemeral keys in their definitions. In the remaining papers we examined, it was less clear whether reencryption keys were ephemeral or persistent.

Neither definition implies the other; any scheme secure with persistent keys can be modified into one that is insecure with ephemeral keys, and vice-versa. The definitions, however, are not in serious tension; any scheme secure with persistent keys and having deterministic ReKeyGen is also secure with ephemeral keys, and ReKeyGen can in general be derandomized using pseudorandom functions. Of course, one can easily define a hybrid notion stronger than both by allowing the adversary to specify for each query whether it wants to use reencryption keys that are new or old.

We adopt the persistent formalization as it better captures 'typical' use. To the best of our knowledge, all our claims can be adapted to the ephemeral setting.

Remark 3.3.6. The power of the adversary above can be strengthened by allowing *adaptive* corruptions instead of dividing the game into phases. Our definitions of CPA and HRA security follow the convention of [ABH09] primarily because it is most common in the research literature. For an examination of CPA and HRA security in the adaptive setting, see the subsequent work of Fuchsbauer, Kamath, Klein, and Pietrzak [FKKP18]. Adaptive-secure, bidirectional, CCA secure proxy reencryption has been studied in [CH07, NAL15b].

3.3.1 Concatenation Scheme and Trivial Scheme

The weakness of CPA security lies in the specification of $\mathcal{O}_{\mathsf{ReEnc}}$, which does not reencrypt any ciphertexts from honest to corrupt users. Said another way, $\mathcal{O}_{\mathsf{ReEnc}}$ reencrypts between only those pairs keys for which $\mathcal{O}_{\mathsf{ReKeyGen}}$ outputs a reencryption key (rather than returning $\perp$). We now describe two schemes that are CPA secure, but are insecure against a dishonest receiver of reencrypted ciphertexts. In both schemes, a single ciphertext reencrypted from an honest index to a corrupted index enables the decryption of messages encrypted under the honest index's public key.

3.3.1.1 Concatenation Scheme

Let $\mathsf{PRE} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{ReKeyGen}, \mathsf{ReEnc})$ be a CPA-secure proxy reencryption scheme. Define a new scheme by modifying only reencryption and decryption:

$$\mathsf{ReEnc}'(\mathsf{rk}, \mathsf{ct}) := \mathsf{ReEnc}(\mathsf{rk}, \mathsf{ct}) \| \mathsf{rk}$$

$$\mathsf{Dec}'(\mathsf{sk}, \mathsf{ct}) := \begin{cases} \mathsf{Dec}(\mathsf{sk}, \mathsf{ct}^1) & \text{if } \mathsf{ct} = \mathsf{ct}^1 \| \mathsf{ct}^2 \\ \mathsf{Dec}(\mathsf{sk}, \mathsf{ct}) & \text{otherwise} \end{cases}$$

Theorem 3.3.7. *Let* $\mathsf{PRE} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{ReKeyGen}, \mathsf{ReEnc})$ *be a CPA-secure proxy reencryption scheme. The corresponding Concatenation Scheme* $\mathsf{PRE}' = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec}', \mathsf{ReKeyGen}, \mathsf{ReEnc}')$ *is a CPA-secure proxy reencryption scheme.*

Proof. For any probabilistic, polynomial-time algorithm $\mathcal{A}'$ (the CPA adversary against PRE'), we construct an efficient algorithm $\mathcal{A}$ such that $\mathsf{Adv}^{\mathcal{A}}_{\mathsf{cpa}} = \mathsf{Adv}^{\mathcal{A}'}_{\mathsf{cpa}}$. By the CPA security of PRE, this advantage is negligible, completing the proof.

$\mathcal{A}$ runs $\mathcal{A}'$ and simulates the CPA security game for PRE' (if $\mathcal{A}'$ does not follow the specification of the game, $\mathcal{A}$ simply aborts). Except for calls to $\mathcal{O}_{\mathsf{ReEnc}}$, all oracle calls by $\mathcal{A}'$ are passed along unaltered by $\mathcal{A}$, along with their responses.

$\mathcal{A}$ begins Phase 2 by requesting all admissible reencryption keys $\mathsf{rk}_{i \to j}$ from its own reencryption key generation oracle. To answer oracle calls from $\mathcal{A}'$ to $\mathcal{O}_{\mathsf{ReEnc}}$, $\mathcal{A}$ first queries its own reencryption oracle, which returns ct^1. If $\mathsf{ct}^1 = \perp$, then $\mathcal{A}'$ returns $\perp$. Otherwise, $\mathcal{A}'$ concatenates the appropriate reencryption key rk to form the new ciphertext $\mathsf{ct} = \mathsf{ct}^1 \| \mathsf{rk}$. This is possible because if $\mathsf{ct}^1 \neq \perp$, then $\mathcal{A}$ is able to get the corresponding reencryption key at the beginning of Phase 2.

$\mathcal{A}$ perfectly implements the CPA security game for PRE', and $\mathcal{A}'$ wins that game if and only if $\mathcal{A}$ wins the corresponding game for PRE. Therefore, $\mathsf{Adv}^{\mathcal{A}}_{\mathsf{cpa}} = \mathsf{Adv}^{\mathcal{A}'}_{\mathsf{cpa}}$. Finally, the running time of $\mathcal{A}$ is polynomially related to that of $\mathcal{A}'$. □

3.3.1.2 The Trivial Scheme

While the Concatenation Scheme builds upon any CPA-secure proxy reencryption scheme, the Trivial Scheme presented next makes use of public-key encryption enjoying *circular security.* Informally, circular security guarantees that encryptions of messages that are a function of the secret key(s) are as secure as encryptions of messages that are independent of the secret key(s), a security property that does not follow from standard semantic security.

In the Trivial Scheme, the reencryption key from party i to j is simply $\mathsf{rk}_{i \to j} = \mathsf{Enc}(\mathsf{pk}_j, \mathsf{sk}_i)$. CPA security of the resulting proxy reencryption scheme requires security against an adversary who has both $\mathsf{rk}_{i \to j}$ and $\mathsf{rk}_{j \to i}$. This requires that the underlying encryption scheme is circular secure.

Because existing definitions and constructions of circular secure encryption schemes based on standard assumptions (e.g., [BHHO08] from DDH) require a bound on the total number of public keys n, the corresponding Trivial Scheme will only satisfy a bounded-key variant of CPA security. Any circular secure encryption scheme without this limitation would yield a Trivial Scheme secure according to Definition 3.3.4.

This section establishes the security of the Trivial Scheme for proxy reencryption. We present the construction, state the theorem, define circular security, and finally prove the theorem.

Let $(\mathsf{KeyGen}_{\mathsf{circ}}, \mathsf{Enc}_{\mathsf{circ}}, \mathsf{Dec}_{\mathsf{circ}})$ be an public key encryption scheme. Let $\mathsf{Setup} \equiv \perp$, $\mathsf{KeyGen} \equiv \mathsf{KeyGen}_{\mathsf{circ}}$; $\mathsf{Enc} \equiv \mathsf{Enc}_{\mathsf{circ}}$;

$$\mathsf{ReKeyGen}(\mathsf{sk}_i, \mathsf{pk}_j) := \mathsf{Enc}_{\mathsf{circ}}(\mathsf{pk}_j, \mathsf{sk}_i)$$

$$\mathsf{ReEnc}(\mathsf{rk}_{i\to j}, \mathsf{ct}_i) := \mathsf{ct}_i \| \mathsf{rk}_{i\to j}$$

$$\mathsf{Dec}(\mathsf{sk}, \mathsf{ct}) := \begin{cases} \mathsf{Dec}_{\mathsf{circ}}(\mathsf{Dec}_{\mathsf{circ}}(\mathsf{sk}, \mathsf{ct}^2), \mathsf{ct}^1) & \text{if } \mathsf{ct} = \mathsf{ct}^1 \| \mathsf{ct}^2 \\ \mathsf{Dec}_{\mathsf{circ}}(\mathsf{sk}, \mathsf{ct}) & \text{otherwise} \end{cases} .$$

Theorem 3.3.8. *If* $(\mathsf{KeyGen}_{\mathsf{circ}}, \mathsf{Enc}_{\mathsf{circ}}, \mathsf{Dec}_{\mathsf{circ}})$ *is n-way circularly secure, then the corresponding Trivial Scheme* PRE *is an n-way CPA secure proxy reencryption scheme.*[7]

The following description and definition of circular security is adapted with slight modification from [BHHO08]. Let $(\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ be a public-key encryption scheme with key space $\mathcal{K}$ and message space $\mathcal{M}$ such that $\mathcal{K} \subseteq \mathcal{M}$. Let $n > 0$ be an integer and let $\mathcal{C}$ be the set of functions $\mathcal{C} = \{f : \mathcal{K}^n \to \mathcal{M}\}$ consisting of

- all $|\mathcal{M}|$ constant functions $f_m(z) = m$ for all $z \in \mathcal{K}^n$, and
- all n selector functions $f_i(x_1, \ldots, x_n) = x_i$ for $1 \leq i \leq n$.

We define circular security using the following game between a challenger and an adversary $\mathcal{A}$. For an integer $n > 0$ and a security parameter λ, the game proceeds as follows:

INITIALIZATION: The challenger chooses a random bit $b \leftarrow \{0, 1\}$. It generates $(\mathsf{pk}_1, \mathsf{sk}_1), \ldots, (\mathsf{pk}_n, \mathsf{sk}_n)$ by running $\mathsf{KeyGen}(1^\lambda)$ n times, and sends $(\mathsf{pk}_1, \ldots, \mathsf{pk}_n)$ to $\mathcal{A}$.

[7]In fact, the proof extends the case when there are n uncorrupted keys and any number of corrupted keys.

QUERIES: The adversary repeatedly issues queries where each query is of the form (i, f) with $1 \leq i \leq n$ and $f \in \mathcal{C}$. The challenger responds by setting $y = f(\mathsf{sk}_1, \ldots, \mathsf{sk}_n)$ and sends ct to $\mathcal{A}$, where

$$\mathsf{ct} \leftarrow \begin{cases} \mathsf{Enc}(\mathsf{pk}_i, y) & \text{if } b = 0 \\ \mathsf{Enc}(\mathsf{pk}_i, 0^{|y|}) & \text{if } b = 1 \end{cases}.$$

FINISH: Finally, the adversary outputs a bit $b' \in \{0, 1\}$.

We say that $\mathcal{A}$ wins the game if $b = b'$. Let win be the event that $\mathcal{A}$ wins the game and define $\mathcal{A}$'s advantage as

$$\mathsf{Adv}^{\mathcal{A}}_{\mathsf{circ},n}(\lambda) = \Pr[\mathsf{win}].$$

Definition 3.3.9 (n-Circular Security)**.** *We say that a public-key encryption scheme* $(\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ *is* n-way circular secure *if for all probabilistic polynomial time adversaries $\mathcal{A}$, there exists a negligible function* negl *such that*

$$\mathsf{Adv}^{\mathcal{A}}_{\mathsf{circ},n}(\lambda) < \frac{1}{2} + \mathsf{negl}(\lambda).$$

Because existing constructions of circularly secure encryption schemes based on standard assumptions require a bound on the total number of public keys n, the corresponding Trivial Scheme will only satisfy a bounded-key variant of CPA security, defined next.

Definition 3.3.10 (Proxy Reencryption: n-CPA Security)**.** *For $n \in \mathbb{N}$, the n-CPA security game is identical to the CPA security game in Definition 3.3.3 except for the following underlined modifications. Recall that* $\mathsf{numKeys}$ *is initialized to 0 and is incremented after every key generation call in the security game.*

UNCORRUPTED KEY GENERATION: *If* $\underline{\mathsf{numKeys} = n}$, *return $\perp$. Otherwise,* *obtain a new key pair* $(\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{KeyGen}(\mathsf{pp})$. *$\mathcal{A}$ is given* pk_i. *The current value of* $\mathsf{numKeys}$ *is added to* Hon *and* $\mathsf{numKeys}$ *is incremented.*

Corrupted Key Generation: *If* $\mathsf{numKeys} = n$*, return* $\perp$*. Otherwise, obtain a new key pair* $(pk_i, \mathsf{sk}_i) \leftarrow \mathsf{KeyGen}(\mathsf{pp})$*.* $\mathcal{A}$ *is given* $(\mathsf{pk}_i, \mathsf{sk}_i)$*. The current value of* $\mathsf{numKeys}$ *is added to* Cor *and* $\mathsf{numKeys}$ *is incremented.*

The corresponding n-CPA advantage *of* $\mathcal{A}$ *is denoted* $\mathsf{Adv}^{\mathcal{A}}_{\mathsf{cpa},n}(\lambda)$*. A proxy reencryption scheme is* n-CPA secure *if for all probabilistic polynomial-time adversaries* $\mathcal{A}$*, there exists a negligible function* negl *such that*

$$\mathsf{Adv}^{\mathcal{A}}_{\mathsf{cpa},n}(\lambda) < \frac{1}{2} + \mathsf{negl}(\lambda)$$

of Theorem 3.3.8. For all $n \in \mathbb{N}$ and any probabilistic, polynomial-time algorithm $\mathcal{A}$ (the adversary against the trivial scheme), we construct an efficient algorithm $\mathcal{A}_{\mathsf{circ}}$ such that $\mathsf{Adv}^{\mathcal{A}_{\mathsf{circ}}}_{\mathsf{circ},n} = \frac{1}{2} \cdot \mathsf{Adv}^{\mathcal{A}}_{\mathsf{cpa},n}$. By the hypothesis, this advantage is negligible, completing the proof.

At the beginning of the game, the circular security challenger picks a random bit b. If $b = 0$, then the Queries oracle encrypts all messages correctly; if $b = 1$, then the Queries oracle encrypts only the message 0. $\mathcal{A}_{\mathsf{circ}}$ runs $\mathcal{A}$ and simulates the CPA security game for PRE (if $\mathcal{A}$ does not follow the specification of the game, $\mathcal{A}_{\mathsf{circ}}$ simply aborts).

At the start of Phase 1, $\mathcal{A}_{\mathsf{circ}}$ calls its Initialization oracle in the circular security game. In return it receives the public keys $(\mathsf{pk}^{\mathsf{circ}}_1, \ldots, \mathsf{pk}^{\mathsf{circ}}_n)$. To answer an Uncorrupted Key Generation query, $\mathcal{A}_{\mathsf{circ}}$ gives to $\mathcal{A}$ the first unused public key $\mathsf{pk}^{\mathsf{circ}}_i$ from this list. To answer a Corrupted Key Generation query, $\mathcal{A}_{\mathsf{circ}}$ generates a new key pair $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}$ and gives $(\mathsf{pk}, \mathsf{sk})$ to the adversary.

$\mathcal{A}$ begins Phase 2 by using its Queries oracle to learn the reencryption keys for all pairs of uncorrupted keys generated. Using its knowledge of the corrupted secret keys, it also computes reencryption keys for all the pairs of corrupted keys generated. Oracle calls from $\mathcal{A}$ to $\mathcal{O}_{\mathsf{ReKeyGen}}$ are answered with the corresponding reencryption key (or with $\perp$). To answer oracle calls from $\mathcal{A}$ to $\mathcal{O}_{\mathsf{ReEnc}}$, computes the appropriate response; namely, it concatenates the reencryption key to the ciphertext (or returns $\perp$).

At some point, $\mathcal{A}$ queries the Challenge oracle with an honest key index i and a pair of messages $(\mathsf{m}_0, \mathsf{m}_1)$. $\mathcal{A}_{\mathsf{circ}}$ chooses a random one of the messages m and queries its own Queries oracle with the pair (i, m), returning the resulting ciphertext to $\mathcal{A}$.

Finally, $\mathcal{A}$ guesses whether $\mathsf{m} = \mathsf{m}_0$ or m_1. If $\mathcal{A}$ guesses correctly, $\mathcal{A}_{\mathsf{circ}}$ guesses the bit $b' = 0$. Otherwise, $\mathcal{A}_{\mathsf{circ}}$ guesses a random $b' \leftarrow \{0, 1\}$. Conditioned on $b = 0$, $\mathcal{A}_{\mathsf{circ}}$ perfectly simulates the PRE security game, and guesses $b' = 0$ with probability $\mathsf{Adv}^{\mathcal{A}}_{\mathsf{cpa},n}$. It follows that $\mathsf{Adv}^{\mathcal{A}_{\mathsf{circ}}}_{\mathsf{circ},n} = \frac{1}{2} \cdot \mathsf{Adv}^{\mathcal{A}}_{\mathsf{cpa},n}$. □

3.4 Security Against Honest Reencryption Attacks

We seek a definition of security which holds as long as the adversary only sees honestly reencrypted ciphertexts, unless the set of corrupt parties can trivially violate security (by some chain of reencryption keys from an uncorrupted public key to a corrupted public key).

We term this notion security against *honest reencryption attacks (HRA)*. To formalize it, we model the ability of an adversary to see honest reencryptions by granting it access to a modified reencryption oracle $\mathcal{O}_{\mathsf{ReEnc}}$. Instead of taking a ciphertext as input, the modified $\mathcal{O}_{\mathsf{ReEnc}}$ takes as input a reference to a previously generated ciphertext (either as the output of $\mathcal{O}_{\mathsf{Enc}}$ or $\mathcal{O}_{\mathsf{ReEnc}}$ itself).

To implement such an oracle, we introduce to the security game a key-value store $\mathcal{C}$ as additional state: the values are ciphertexts ct which are keyed by a pair of integers (i, k), where i denotes the index of the key pair $(\mathsf{pk}_i, \mathsf{sk}_i)$ under which ct was (re)encrypted, and k is a unique index assigned to ct.

As described, this new $\mathcal{O}_{\mathsf{ReEnc}}$ admits a trivial attack: simply reencrypt the challenge ciphertext to a corrupted key and decrypt. To address this issue, we adapt an idea from [CH07]'s definition of CCA security: we rule out the trivial attack by storing a set Deriv of ciphertexts derived from the challenge by reencrypting, and rejecting queries to $\mathcal{O}_{\mathsf{ReEnc}}$ for ciphertexts in Deriv and a corrupted target key. We might have instead chosen to forbid any reencryptions of the challenge ciphertext, even between uncorrupted keys. Though this would have simplified the definition, it would have been unsatisfactory. For example, in the single-writer, many-reader encrypted storage application the contents of a message m that gets reencrypted from Alice to Charlie should be hidden from Bob.

We now present the honest reencryption attacks security game. The game is similar to the CPA security game with some modifications to Setup, Challenge, and $\mathcal{O}_{\mathsf{ReEnc}}$, and the addition of an Enc oracle $\mathcal{O}_{\mathsf{Enc}}$ to Phase 2. $\mathcal{O}_{\mathsf{Enc}}$ may be executed poly(λ) times and in any order relative to the other oracles in Phase 2. For the sake of clarity we reproduce the full game below and mark the modified oracles with a $\star$.

Definition 3.4.1 (Proxy Reencryption: Security Game for Honest Reencryption Attacks (HRA))**.** *Let λ be the security parameter and $\mathcal{A}$ be an oracle Turing machine. The HRA game consists of an execution of $\mathcal{A}$ with the following oracles.*

Phase 1:

$\star$ SETUP: *The public parameters are generated and given to $\mathcal{A}$. A counter* $\mathsf{numKeys}$ *is initialized to 0, and the sets* Hon *(of honest, uncorrupted indices) and* Cor *(of corrupted indices) are initialized to be empty.*

Additionally the following are initialized: a counter numCt *to 0, a key-value store $\mathcal{C}$ to empty, and a set* Deriv *to be empty. This oracle is executed first and only once.*

UNCORRUPTED KEY GENERATION: *Generate key* $(\mathsf{pk}_{\mathsf{numKeys}}, \mathsf{sk}_{\mathsf{numKeys}}) \leftarrow \mathsf{KeyGen}(\mathsf{pp})$ *and give* $\mathsf{pk}_{\mathsf{numKeys}}$ *to $\mathcal{A}$. The current value of* $\mathsf{numKeys}$ *is added to* Hon *and* $\mathsf{numKeys}$ *is incremented.*

CORRUPTED KEY GENERATION: *Generate keys* $(pk_{\mathsf{numKeys}}, \mathsf{sk}_{\mathsf{numKeys}}) \leftarrow \mathsf{KeyGen}(\mathsf{pp})$ *and give it to $\mathcal{A}$. The current value of* $\mathsf{numKeys}$ *is added to* Cor *and* $\mathsf{numKeys}$ *is incremented.*

Phase 2: *For each pair $i, j \leq$* $\mathsf{numKeys}$*, compute the reencryption key* $\mathsf{rk}_{i \to j} \leftarrow \mathsf{ReKeyGen}(\mathsf{sk}_i, \mathsf{pk}_j)$.

REENCRYPTION KEY GENERATION $\mathcal{O}_{\mathsf{ReKeyGen}}$: *On input* (i, j) *where* $i, j \leq \mathsf{numKeys}$, *if* $i = j$ *or if* $i \in \mathsf{Hon}$ *and* $j \in \mathsf{Cor}$, *output* $\perp$. *Otherwise return the reencryption key* $\mathsf{rk}_{i \to j}$.

⋆ ENCRYPTION $\mathcal{O}_{\mathsf{Enc}}$: *On input* (i, m), *where* $i \leq \mathsf{numKeys}$, *compute* $\mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{pk}_i, \mathsf{m})$ *and increment* numCt. *Store the value* ct *in* $\mathcal{C}$ *with key* (i, numCt). *Return* $(\mathsf{numCt}, \mathsf{ct})$.

⋆ CHALLENGE ORACLE: *On input* $(i, \mathsf{m}_0, \mathsf{m}_1)$ *where* $i \in \mathsf{Hon}$ *and* $\mathsf{m}_0, \mathsf{m}_1 \in \mathcal{M}$, *sample a bit* $b \leftarrow \{0, 1\}$ *uniformly at random, compute the challenge ciphertext* $\mathsf{ct}^* \leftarrow \mathsf{Enc}(\mathsf{pk}_i, \mathsf{m}_b)$, *and increment* numCt. *Add* numCt *to the set* Deriv. *Store the value* ct^* *in* $\mathcal{C}$ *with key* (i, numCt). *Return* $(\mathsf{numCt}, \mathsf{ct}^*)$. *This oracle can only be queried once.*

⋆ REENCRYPTION $\mathcal{O}_{\mathsf{ReEnc}}$: *On input* (i, j, k) *where* $i, j \leq \mathsf{numKeys}$ *and* $k \leq \mathsf{numCt}$, *if* $j \in \mathsf{Cor}$ *and* $k \in \mathsf{Deriv}$ *return* $\perp$. *If there is no value in* $\mathcal{C}$ *with key* (i, k), *return* $\perp$.

Otherwise, let ct_i *be that value in* $\mathcal{C}$, *let* $\mathsf{ct}_j \leftarrow \mathsf{ReEnc}(\mathsf{rk}_{i \to j}, \mathsf{ct}_i)$, *and increment* numCt. *Store the value* ct_j *in* $\mathcal{C}$ *with key* (j, numCt). *If* $k \in \mathsf{Deriv}$, *add* numCt *to the set* Deriv.

Return $(\mathsf{numCt}, \mathsf{ct}_j)$.

Phase 3:

DECISION: *On input a bit* $b' \in \{0, 1\}$, *return* win *if* $b = b'$.

The HRA advantage *of* $\mathcal{A}$ *is defined as*

$$\mathsf{Adv}^{\mathcal{A}}_{\mathsf{hra}}(\lambda) = \Pr[\mathsf{win}],$$

where the probability is over the randomness of $\mathcal{A}$ *and the oracles in HRA game.*

Definition 3.4.2 (Proxy Reencryption: HRA Security). *Given a security parameter* 1^λ*, a proxy reencryption scheme is HRA secure if for all probabilistic polynomial-time adversaries* $\mathcal{A}$*, there exists a negligible function* negl *such that*

$$\mathsf{Adv}^{\mathcal{A}}_{\mathsf{hra}}(\lambda) < \frac{1}{2} + \mathsf{negl}(\lambda)$$

The Concatenation Scheme demonstrates that CPA security does not necessarily imply HRA security. Together with following theorem, we see that HRA security is a strict strengthening of CPA security.

Theorem 3.4.3. *Let* PRE *be an HRA secure proxy reencryption scheme. Then* PRE *is CPA secure.*

Proof. From any probabilistic, polynomial-time algorithm $\mathcal{A}$ (the CPA adversary), we construct an efficient algorithm $\mathcal{A}'$ such that $\mathsf{Adv}^{\mathcal{A}'}_{\mathsf{hra}} = \mathsf{Adv}^{\mathcal{A}}_{CPA}$. By the HRA security of PRE this advantage is negligible, completing the proof.

$\mathcal{A}'$ runs $\mathcal{A}$ and simulates the CPA security game (if $\mathcal{A}$ does not follow the specification of the CPA security game, $\mathcal{A}'$ simply aborts). Except for calls to $\mathcal{O}_{\mathsf{ReEnc}}$, all oracle calls by $\mathcal{A}'$ are passed along unaltered by $\mathcal{A}$ to the corresponding HRA oracles, along with their responses.

$\mathcal{A}'$ begins Phase 2 by requesting all (admissible) reencryption keys $\mathsf{rk}_{i\to j}$ from $\mathcal{O}_{\mathsf{ReKeyGen}}$. racle calls from $\mathcal{A}$ to $\mathcal{O}_{\mathsf{ReEnc}}$ are answered by $\mathcal{A}'$ by computing the reencryption using the appropriate reencryption key; this is possible because $\mathcal{O}_{\mathsf{ReEnc}}$ returns $\perp$ if and only if $\mathcal{A}'$ is unable to get the corresponding reencryption key.

$\mathcal{A}'$ prefectly implements the CPA security game, and $\mathcal{A}$ wins that game if and only if $\mathcal{A}'$ wins the HRA security game. Therefore $\mathsf{Adv}^{\mathcal{A}'}_{\mathsf{hra}} = \mathsf{Adv}^{\mathcal{A}}_{CPA}$. Finally, the running time of $\mathcal{A}'$ is polynomially related to the that of $\mathcal{A}$. $\square$

3.4.1 Sufficiency of HRA Security for Applications

Returning to the applications of proxy reencryption described in Section 3.2, we observe that HRA security provides substantially stronger security guarantees.

Encrypted email forwarding Using proxy reencryption with HRA security, Alice can forward encrypted email to Bob for a short period of time (during a vacation, say) and be sure that Bob cannot read her email after she returns.

Key escrow Similar to encrypted email forwarding, proxy reencryption with HRA can be used to enable law enforcement to read certain encrypted messages without compromising the secrecy of documents outside the scope of a search warrant or subpoena, for example.

Single-writer, many-reader encrypted storage Whereas proxy reencryption with CPA security can only support all or nothing access (i.e., all readers may access all data), HRA security can support fine grained access control (i.e., each reader may access only a specific subset of the data).

There is no question that HRA does not provide as much security as CCA, and that CCA-secure proxy reencryption would yield more robust applications. HRA security, however, can provide meaningful guarantees in the above applications.

Encrypted email forwarding If Alice is forwarding all emails to Bob, then Bob could certainly mount an attack outside the honest reencryption model. On the other hand, if Alice is forwarding only those emails from a third-party sender Charlie, then such an attack is impossible without the involvement of Charlie (supposing of course that the sender of an email can be authenticated).

Key escrow The hypothetical legal regime that establishes the government's power of exceptional access by way of key escrow could additionally prohibit the government from mounting chosen-ciphertext attacks. In the United States, a Constitutional argument could perhaps be made that law-enforcement use of chosen-ciphertext attacks must be limited.

Single-writer, many-reader encrypted storage The only ciphertexts being reencrypted are those uploaded by the single-writer to the proxy server (hence the name). It is by no means a stretch to require that the proxy server does not allow writes by unauthorized parties (i.e., the readers). If the honest writer only uploads honestly generated ciphertexts, HRA is appropriate.

3.5 Security of Existing CPA Secure Schemes

Can we construct HRA-secure proxy reencryption? The most natural place to begin is with existing CPA secure schemes.

We begin by demonstrating that the construction of [PRSV17] is not HRA secure. Although CPA security is strictly weaker than HRA security, we might hope that the existing CPA secure schemes already satisfy the more stringent definition. To this end, we identify a natural property—*reencryption simulatability*—sufficient to boost CPA security to HRA security.[8]

We examine the simple construction of CPA secure proxy reencryption from any fully-homomorphic encryption (FHE) presented in [Gen09a]. While the resulting proxy reencryption may not be HRA secure in general, if the FHE is *circuit private*—a property Gentry imbues into his FHE by rerandomization—the same construction will be HRA secure. We then examine pairing-based schemes, finding there too that rerandomization provides a direct path to HRA security.[9]

3.5.1 HRA Insecurity of [PRSV17]

Though it is easy to construct CPA secure encryption schemes that are not HRA secure, the question remains whether any previously proposed schemes fail to satisfy HRA security. In this section, we show that the construction of Polyakov, Rohloff, Sahu, and Vaikuntanathan [PRSV17, Section 5] is one such scheme. Their construction is based on a public key encryption scheme of Brakerski and Vaikuntanathan [BV11] and is CPA secure assuming the hardness of Ring Learning With

[8]Some existing notions in the proxy reencryption literature seem powerful enough to elevate CPA security to HRA security, including proxy invisibility [AFGH06], unlinkability [FL17], and punctured security [ACJ17]. However, these notions are not sufficiently well defined to draw any concrete conclusions. The notion of key-privacy [ABH09] does not in general suffice for HRA security.

[9]While we do not examine every pairing-based construction of proxy reencryption, we suspect that rerandomizing reencryption will suffice for reencryption simulation in many, if not all.

Errors (RLWE).

As with the Trivial and Concatenation schemes, the HRA attack is simple yet severe: any single honestly generated ciphertext enables the recipient Bob to recover the sender Alice's secret key with significant probability.

Theorem 3.5.1. *The proxy reencryption scheme of [PRSV17, Section 5] is not HRA secure.*

Proof. Except where noted, the notation used below is consistent with [PRSV17]; we restrict our description to those facts necessary to describe the HRA attack.

For n a power of 2 and prime $q \equiv 1 \mod 2n$, let $R_q = \mathbb{Z}_q[x]/(x^n + 1)$ be a ring of degree $(n-1)$ polynomials with coefficients in $\mathbb{Z}_q$. The sender Alice's secret key is s, and the recipient Bob's secret key is s^*. Bob's public key includes $O(\log q) = \text{poly}(n)$ RLWE samples $\theta_i^* = \beta_i \cdot s^* + pe_i$, where p is a public prime and the β_i and e_i are ring elements sampled by Bob.[10] Ciphertexts are pairs of ring elements $(c_0, c_1) \in R_q^2$. By [BV11, Lemma 4], the distribution of c_1 is statistically close to uniform over R_q. By [LPR13, Lemma 2.25], c_1 is invertible with probability at least $e^{-1} - \mathsf{negl}(n)$. The result of reencrypting (c_0, c_1) is a pair (c_0', c_1') such that $c_0' - s^* \cdot c_1' = c_0 - s \cdot c_1 + pE_1$, where E_i is computable given c_1 and the e_i. This fact is used by [PRSV17] to prove the correctness of their scheme. Rearranging the above, we see that

$$s \cdot c_1 = c_0 + pE_1 - c_0' + s^* \cdot c_1'.$$

Given any ciphertext (c_0, c_1) and its reencryption (c_0', c_1'), Bob can evaluate the above and compute $s \cdot c_1$. With probability at least $e^{-1} - \mathsf{negl}(n)$, c_1 is invertible and Bob can recover the secret s. $\square$

3.5.2 Reencryption Simulatability

While HRA is a strictly stronger security notion than CPA, we now show that if a CPA secure proxy reencryption scheme has an additional property we call *reencryp-*

[10] [PRSV17] separate the computation of θ_i^* from Bob's public key, but this is only a matter of presentation.

tion simulatability, then it must also be HRA secure. Very roughly, reencryption simulatability means that ciphertexts resulting from computing $\mathsf{ReEnc}(\mathsf{rk}_{i \to j}, \mathsf{ct}_i)$ can be simulated without knowledge of the sender's secret key sk_i (but with knowledge of the plaintext message m and the recipient's secret key sk_j). Note that ReEnc uses $\mathsf{rk}_{i \to j}$ which is a function of sk_i.

Reencryption simulatability allows an algorithm with access to the CPA oracles to efficiently implement the honest reencryption oracle. For intuition, consider the following approach to reducing HRA security to CPA security. Suppose there existed an adversary $\mathcal{A}_{\mathsf{hra}}$ violating the HRA security of a scheme; the reduction plays the roles of both the CPA adversary and the challenger in the HRA security game, and attempts to violate CPA security. To succeed, the reduction must be able to answer honest reencryption queries from uncorrupted keys to corrupted keys. Though the reduction knows the messages being reencrypted, it does not know the appropriate reencryption key. However, if it could indistinguishably simulate these reencryptions then it could indeed leverage $\mathcal{A}_{\mathsf{hra}}$ to violate CPA security.

We emphasize that the goal of Definition 3.5.2 is to capture a large swath of possible schemes while still enabling very simple proofs of simulatability (and thus of HRA security for existing CPA secure schemes). It is not intended to be the only avenue for proving HRA security of new or existing constructions. Reencryption simulatability is not necessary for HRA security of proxy reencryption. In particular, analogous versions of Theorem 3.5.3 hold with computational simulatability guarantees, but are more complicated [DN18, Foonote 7 and Appendix A].

Definition 3.5.2 (Reencryption Simulatability). *A proxy reencryption scheme* PRE *is* reencryption simulatable *if there exists a probabilistic, polynomial-time algorithm* $\mathsf{ReEncSim}$ *such that with high probability over* aux, *for all* $\mathsf{m} \in \mathcal{M}$*:*

$$(\mathsf{ReEncSim}(\mathsf{aux}), \mathsf{aux}) \approx_s (\mathsf{ReEnc}(\mathsf{rk}_{a \to b}, \mathsf{ct}_a), \mathsf{aux}),$$

where $\approx_s$ *denotes statistical indistinguishability, and* ct_a *and* aux *are sampled accord-*

ing to

$$\begin{aligned}
&\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda),\\
&(\mathsf{pk}_a, \mathsf{sk}_a) \leftarrow \mathsf{KeyGen}(\mathsf{pp}),\\
&(\mathsf{pk}_b, \mathsf{sk}_b) \leftarrow \mathsf{KeyGen}(\mathsf{pp}),\\
&\mathsf{rk}_{a\to b} \leftarrow \mathsf{ReKeyGen}(\mathsf{sk}_a, \mathsf{pk}_b),\\
&\mathsf{ct}_a \leftarrow \mathsf{Enc}(\mathsf{pk}_a, \mathsf{m}),\\
&\mathsf{aux} = (\mathsf{pp}, \mathsf{pk}_a, \mathsf{pk}_b, \mathsf{sk}_b, \mathsf{ct}_a, \mathsf{m}).
\end{aligned}$$

A special case of the above is when $\mathsf{ReEncSim}(\mathsf{aux}) = \mathsf{Enc}(\mathsf{pk}_b, \mathsf{m})$ simply computes a fresh encryption of the message. That is, if reencrypted ciphertexts are distributed like fresh ciphertexts, then the scheme is reencryption simulatable.

Theorem 3.5.3. *Let* PRE *be an CPA secure, reencryption simulatable, proxy reencryption scheme. Then* PRE *is HRA secure.*

Outline. The proof proceeds according to the intuition above. From any probabilistic, polynomial-time algorithm $\mathcal{A} = \mathcal{A}_{\mathsf{hra}}$ (the HRA adversary), we construct an algorithm $\mathcal{A}' = \mathcal{A}_{\mathsf{cpa}}$ such that $\mathsf{Adv}^{\mathcal{A}'}_{\mathsf{cpa}}(\lambda) \geq \mathsf{Adv}^{\mathcal{A}}_{\mathsf{hra}}(\lambda) - \mathsf{negl}(\lambda)$; by the CPA security of PRE this advantage is negligible, completing the proof.

$\mathcal{A}_{\mathsf{cpa}}$ runs $\mathcal{A}_{\mathsf{hra}}$ and simulates the HRA security game (if $\mathcal{A}_{\mathsf{hra}}$ does not follow the specification of the HRA security game, $\mathcal{A}_{\mathsf{cpa}}$ simply aborts). To answer oracle calls by $\mathcal{A}_{\mathsf{hra}}$ to any oracle other than $\mathcal{O}_{\mathsf{ReEnc}}$, $\mathcal{A}_{\mathsf{cpa}}$ simply passes the calls and answers unaltered to the corresponding CPA oracles.

To answer oracle calls to $\mathcal{O}_{\mathsf{ReEnc}}$ between two uncorrupted keys or two corrupted keys, $\mathcal{A}_{\mathsf{cpa}}$ uses the corresponding reencryption key. On the other hand, for calls to $\mathcal{O}_{\mathsf{ReEnc}}$ from an uncorrupted key to a corrupted key, $\mathcal{A}_{\mathsf{cpa}}$ simulates the reencryption using ReEncSim. This is possible because $\mathcal{A}_{\mathsf{cpa}}$ knows the underlying m (along with the other information in aux).

Reencryption simulatability implies that the views of $\mathcal{A}_{\mathsf{hra}}$ in the real security game (using the real $\mathcal{O}_{\mathsf{ReEnc}}$) and the simulated security game (using ReEncSim) are statistically close. $\mathcal{A}_{\mathsf{cpa}}$ wins the CPA security game if and only if $\mathcal{A}_{\mathsf{hra}}$ wins in the simulated HRA game described above. □

3.5.3 Fully Homomorphic Encryption and Proxy Reencryption

There is an intimate connection between FHE and proxy reencryption: a sufficiently powerful somewhat homomorphic encryption scheme implies CPA secure proxy reencryption, which can be used to "bootstrap" the scheme to achieve fully homomorphic encryption [Gen09a]. For the relevant FHE definitions, see [Gen09a, Section 2].

Let $\mathsf{FHE} = (\mathsf{Setup}_{\mathsf{FHE}}, \mathsf{KeyGen}_{\mathsf{FHE}}, \mathsf{Enc}_{\mathsf{FHE}}, \mathsf{Dec}_{\mathsf{FHE}}, \mathsf{Eval}_{\mathsf{FHE}})$ be an FHE scheme. Proxy reencryption can be constructed as follows (compare to the Trivial Scheme):

$\mathsf{KeyGen}_{\mathsf{PRE}}$, $\mathsf{Enc}_{\mathsf{PRE}}$ AND $\mathsf{Dec}_{\mathsf{PRE}}$: Identical to their FHE counterparts.

$\mathsf{ReKeyGen}_{\mathsf{PRE}}(\mathsf{sk}_i, \mathsf{pk}_j)$: Output $\mathsf{rk}_{i\to j} = \mathsf{Enc}_{\mathsf{FHE}}(\mathsf{pk}_j, \mathsf{sk}_i)\|\mathsf{pk}_j$. The reencryption key is an encryption of sk_i under pk_j, along with the target public key pk_j.

$\mathsf{ReEnc}_{\mathsf{PRE}}(\mathsf{rk}_{i\to j}, \mathsf{ct}_i)$: Let $\mathsf{ct}_{i\to j} \leftarrow \mathsf{Enc}_{\mathsf{FHE}}(\mathsf{pk}_j, \mathsf{ct}_i)$. Homomorphically compute the FHE decryption function $\mathsf{Dec}_{\mathsf{FHE}}(\mathsf{sk}_i, \mathsf{ct}_i)$ using the corresponding ciphertexts $\mathsf{rk}_{i\to j}$ and $\mathsf{ct}_{i\to j}$ (under pk_j). Namely, $\mathsf{ct}_j = \mathsf{Eval}_{\mathsf{FHE}}(\mathsf{pk}_j, \mathsf{Dec}_{\mathsf{FHE}}, \mathsf{rk}_{i\to j}, \mathsf{ct}_{i\to j})$.

The correctness of the FHE implies the correctness of the resulting proxy reencryption:

$$\mathsf{Dec}_{\mathsf{PRE}}(\mathsf{sk}_j, \mathsf{ct}_j) = \mathsf{Dec}_{\mathsf{FHE}}(\mathsf{sk}_j, \mathsf{ct}_j) = \mathsf{Dec}_{\mathsf{FHE}}(\mathsf{sk}_i, \mathsf{ct}_i) = \mathsf{Dec}_{\mathsf{PRE}}(\mathsf{sk}_i, \mathsf{ct}_i).$$

Furthermore, the proxy reencryption scheme is CPA secure.

To demonstrate that the construction might not be HRA secure, consider the following fully homomorphic encryption scheme FHE' constructed from any existing scheme FHE. The only modification made in FHE' is to $\mathsf{Eval}_{\mathsf{FHE}'}$:

$$\mathsf{Eval}_{\mathsf{FHE}'}(\mathsf{pk}_j, C, \mathsf{ct}_1, \mathsf{ct}_2) := \mathsf{Eval}_{\mathsf{FHE}}(\mathsf{pk}_j, C, \mathsf{ct}_1, \mathsf{ct}_2)\|\mathsf{ct}_1.$$

Note that FHE' does not violate FHE compactness if ct_1 (in the proxy reencryption construction, $\mathsf{rk}_{i\to j}$) is always of some size bounded by a polynomial in the security parameter of the FHE scheme; this suffices for our purpose. Instantiating the proxy reencryption construction with FHE' essentially results in the Concatenation Scheme, which is not HRA secure.

3.5.3.1 Circuit Privacy

An FHE scheme is *circuit private* if ciphertexts resulting from FHE evaluations are statistically indistinguishable from fresh ciphertexts [Gen09a]. Namely, if for any circuit C and any ciphertexts $\mathsf{ct}_1, \ldots, \mathsf{ct}_t$:

$$\mathsf{Enc}_{\mathsf{FHE}}(\mathsf{pk}, C(\mathsf{ct}_1, \ldots, \mathsf{ct}_t)) \approx_s \mathsf{Eval}_{\mathsf{FHE}}(\mathsf{pk}, C, \mathsf{ct}_1, \ldots, \mathsf{ct}_t).$$

In [Gen09a], an FHE scheme without circuit privacy is modified to be circuit private by rerandomizing the ciphertexts resulting from $\mathsf{Eval}_{\mathsf{FHE}}$.

While our proxy reencryption construction above is not in general HRA secure, it is easy to see that if the underlying FHE is circuit private, then the proxy reencryption is reencryption simulatable (Definition 3.5.2). By Theorem 3.5.3, the resulting scheme is therefore HRA secure.

3.5.4 Pairing-Based Proxy Reencryption

Many constructions of proxy reencryption are based on the hardness of Diffie-Hellman-type problems over certain bilinear groups, including [AFGH06, CH07, LV08, ABH09, HRSV07].

A prototypical construction is that of [AFGH06], which itself is based on the original scheme of [BBS98b]. For every λ, let G_1 and G_2 be groups of prime order $q = \Theta(2^\lambda)$, and let g be a generator of G_1. Let e be a non-degenerate bilinear map $e : G_1 \times G_1 \to G_2$ (i.e., for all $h \in G_1$ and $a, b \in \mathbb{Z}_q$, $e(h^a, h^b) = e(h,h)^{ab}$, and for all generators g of G_1, $e(g,g) \neq 1$). Let $Z = e(g,g)$. The message-space of the scheme is G_2.

$\mathsf{Setup}(1^\lambda)$: Output $\mathsf{pp} = (q, g, G_1, G_2, e)$.

$\mathsf{KeyGen}(\mathsf{pp})$: Sample $a \leftarrow \mathbb{Z}_q$ uniformly at random. Output $\mathsf{sk} = a$ and $\mathsf{pk} = g^a$.

$\mathsf{Enc}(\mathsf{pk}, \mathsf{m})$: Sample $k \leftarrow \mathbb{Z}_q$ uniformly at random. Output $\mathsf{ct} = (\mathsf{pk}^k, \mathsf{m}Z^k) = (g^{ak}, \mathsf{m}Z^k)$.

$\mathsf{ReKeyGen}(\mathsf{sk}_A = a, \mathsf{pk}_B = g^b)$: Output $\mathsf{rk}_{A\to B} = g^{b/a}$.

$\mathsf{ReEnc}(\mathsf{rk}_{A\to B}, \mathsf{ct}_A)$: Let $\mathsf{ct}_A = (\alpha_1, \alpha_2)$. Output

$$\mathsf{ct}_B = (e(\alpha_1, \mathsf{rk}_{A\to B}), \alpha_2) = (e(g^{ak}, g^{b/a}), \mathsf{m}Z^k) = (Z^{bk}, \mathsf{m}Z^k).$$

$\mathsf{Dec}(\mathsf{sk}, \mathsf{ct})$: Let $\mathsf{ct} = (\alpha_1, \alpha_2)$. If $\alpha_1 \in G_2$ (i.e., if it is the result of ReEnc), then output $\alpha_2/\alpha_1^{1/a} = \mathsf{m}Z^k/Z^k = \mathsf{m}$. Otherwise $\alpha_1 \in G_1$ (i.e., it is a fresh ciphertext); output $\alpha_2/e(\alpha_1, g)^{1/a} = \mathsf{m}Z^k/e(g^{ak}, g)^{1/a} = \mathsf{m}Z^k/Z^k = \mathsf{m}$.

Is this scheme HRA secure? It is tempting to say that the scheme is reencryption simulatable; after all, given a message m it is indeed straightforward to sample $(Z^{bk}, \mathsf{m}Z^k)$ for random $k \leftarrow \mathbb{Z}_q$. However $\mathsf{ct}_A = (g^{ak}, \mathsf{m}Z^k)$ and $\mathsf{ct}_B = \mathsf{ReEnc}(\mathsf{rk}_{A\to B}, \mathsf{ct}_A) = (Z^{bk}, \mathsf{m}Z^k)$ share the randomness k. Given $\mathsf{ct}_A = (g^{ak}, \mathsf{m}Z^k)$ and m, it is not clear how to output $(g^{bk}, \mathsf{m}Z^k)$.

3.5.4.1 Rerandomization

A minor modification to the scheme above guarantees reencryption simulatability and therefore HRA security. Replace ReEnc above with ReEnc':

$\mathsf{ReEnc}'(\mathsf{rk}_{A\to B}, \mathsf{ct}_A)$: Compute $(Z^{bk}, \mathsf{m}Z^k) = \mathsf{ReEnc}(\mathsf{rk}_{A\to B}, \mathsf{ct}_A)$. Sample $k' \leftarrow \mathbb{Z}$ uniformly at random, and output $(Z^{bk}\cdot e(g^b, g^{k'}),\ \mathsf{m}Z^k\cdot e(g, g^{k'})) = (Z^{b(k+k')}, \mathsf{m}Z^{k+k'})$.

The resulting proxy reencryption scheme maintains the CPA security of the original, as the only modification is the rerandomization of reencrypted ciphertexts (which can be done by anyone with knowledge of the public parameters).

Furthermore, the scheme is now reencryption simulatable. To see why, observe that the resulting reencrypted ciphertexts are uniformly distributed in $\{(\mathsf{ct}_1, \mathsf{ct}_2) \in G_2 \times G_2 : \mathsf{ct}_2/\mathsf{ct}_1^{1/b} = \mathsf{m}\}$, independent of all other keys and ciphertexts. Such ciphertexts are easily sampled with knowledge of pp, $\mathsf{pk}_B = g^b$ and m as follows.

$\mathsf{ReEncSim}(\mathsf{pp}, \mathsf{pk}_B, \mathsf{m})$: Sample $k' \leftarrow \mathbb{Z}_q$ uniformly at random, and output

$$(e(\mathsf{pk}_B, g^{k'}), \mathsf{m} \cdot e(g, g^{k'})) = (Z^{bk'}, \mathsf{m}Z^{bk'}).$$

Thus, by Theorem 3.5.3, the modified scheme is HRA secure. Observe that rerandomization was the key to achieving circuit privacy (and thereby HRA security) in the FHE-based proxy reencryption construction as well.

The pairing-based schemes of [ABH09] and [HRSV07] already incorporate rerandomization during reencryption. In the former case, it is used to achieve "key privacy;" in the latter, to achieve obfuscation of the reencryption functionality. In each, it is straightforward to show that the schemes are also reencryption simulatable and therefore HRA secure.

3.6 CCA Security

It may seem that CCA security for proxy reencryption should imply HRA security, but the situation is not so simple. While the relationship between CCA and HRA security is still open, CCA security does imply CPA security. By Theorem 3.5.3, any CCA secure proxy reencryption scheme satisfying reencryption simulatability is also HRA secure. CCA security is discussed in Section 3.6. In this section, we define CCA security for proxy reencryption and describe the challenge in proving that CCA security implies HRA security. Finally, we construct a proxy reencryption scheme that illustrates the problem with the intuition which motivated the original CPA definition which also separates the security models $\mathsf{ind\text{-}cca}_{0,1}$ and $\mathsf{ind\text{-}cca}_{2,0}$ defined in [NAL15b], proving a converse to their Theorem 4.6.

3.6.1 Definition

The definition below is adapted from [CH07, Definition 2.4], but modified to simplify comparison to the other definitions presented in this chapter. First, while [CH07] focuses on bidirectional PRE, we consider unidirectional PRE. Secondly, we modify the definition to use persistent, rather than ephemeral, reencryption keys (see Remark 3.3.5). Finally, we add an intialization stage Setup and generally adapt the syntax to coincide with the notation used throughout this chapter.[11]

The core concept in the definition is that of *derivatives* of the challenge. Informally, a pair (i, ct) is a derivative of the challenge if the decryption $\mathsf{Dec}(\mathsf{sk}_j, \mathsf{ct})$ or the reencryption $\mathsf{ReEnc}(\mathsf{rk}_{i \to j}, \mathsf{ct})$ to some corrupted key index j would give the adversary "illegitimate information" about the challenge ciphertext. The precise formalization (Definition 3.6.2) is reminiscent of *replayable* CCA security for standard encryption [CKN03].

Definition 3.6.1 (Proxy Reencryption: Security Game for Chosen Ciphertext Attacks (CCA) [CH07]). *Let λ be the security parameter and $\mathcal{A}$ be an oracle Turing machine. The HRA game consists of an execution of $\mathcal{A}$ with the following oracles, which can be invoked multiple times in any order, subject to the constraints below:*

SETUP: *The public parameters are generated and given to $\mathcal{A}$. A counter* $\mathsf{numKeys}$ *is initialized to 0, and the sets* Hon *(of honest / uncorrupted indices) and* Cor *(of corrupted indices) are initialized to be empty. This oracle is executed first and only once.*

CHALLENGE ORACLE: *On input $(i^*, \mathsf{m}_0, \mathsf{m}_1)$ where $i^* \in \mathsf{Hon}$ and $\mathsf{m}_0, \mathsf{m}_1 \in \mathcal{M}$, sample a bit $b \leftarrow \{0, 1\}$ uniformly at random, compute the challenge ciphertext $\mathsf{ct}^* \leftarrow \mathsf{Enc}(\mathsf{pk}_{i^*}, \mathsf{m}_b)$. Return ct^*. This oracle can only be queried once.*

UNCORRUPTED KEY GENERATION: *Generate keys $(\mathsf{pk}_{\mathsf{numKeys}}, \mathsf{sk}_{\mathsf{numKeys}}) \leftarrow \mathsf{KeyGen}(\mathsf{pp})$ and give $\mathsf{pk}_{\mathsf{numKeys}}$ to $\mathcal{A}$. The current value of* $\mathsf{numKeys}$ *is added to* Hon *and*

[11] Unlike the CPA definition of [ABH09], the CCA definition in [CH07] does not divide the security game into three distinct phases. Rather, it allows calls Corrupted / Uncorrupted Key Generation calls to be made at any time.

$\mathsf{numKeys}$ *is incremented.*

CORRUPTED KEY GENERATION *Generate keys* $(pk_{\mathsf{numKeys}}, \mathsf{sk}_{\mathsf{numKeys}}) \leftarrow \mathsf{KeyGen}(\mathsf{pp})$ *and given to* $\mathcal{A}$. *The current value of* $\mathsf{numKeys}$ *is added to* Cor *and* $\mathsf{numKeys}$ *is incremented.*

REENCRYPTION KEY GENERATION $\mathcal{O}_{\mathsf{ReKeyGen}}$: *On input* (i, j) *where* $i, j \leq \mathsf{numKeys}$, *if* $i = j$ *or if* $i \in \mathsf{Hon}$ *and* $j \in \mathsf{Cor}$, *output* $\perp$. *If* $\mathcal{O}_{\mathsf{ReEnc}}$ *has not been executed on input* (i, j), *compute and store* $\mathsf{rk}_{i \to j} \leftarrow \mathsf{ReKeyGen}(\mathsf{sk}_i, \mathsf{pk}_j)$. *Output the reencryption key* $\mathsf{rk}_{i \to j}$

REENCRYPTION $\mathcal{O}_{\mathsf{ReEnc}}$: *On input* (i, j, ct_i) *where* $i, j \leq \mathsf{numKeys}$, *if* $j \in \mathsf{Cor}$ *and* (i, ct_i) *is a* derivative *of* (i^*, ct^*), *return* $\perp$. *Otherwise, let* $\mathsf{ct}_j \leftarrow \mathsf{ReEnc}(\mathsf{rk}_{i \to j}, \mathsf{ct}_i)$, *and return* ct_j.

DECRYPTION ORACLE: *On input* (i, ct) *where* $i \leq \mathsf{numKeys}$, *if the pair* (i, ct) *is a* derivative *of* (i^*, ct^*), *then return* $\perp$. *Otherwise, return* $\mathsf{Dec}(\mathsf{sk}_i, \mathsf{ct})$.

DECISION: *On input a bit* $b' \in \{0, 1\}$, *return* win *if* $b = b'$.

The CCA advantage *of* $\mathcal{A}$ *is defined as*

$$\mathsf{Adv}^{\mathcal{A}}_{\mathsf{cca}}(\lambda) = \Pr[\mathsf{win}],$$

where the probability is over the randomness of $\mathcal{A}$ *and the oracles in CCA game.*

Definition 3.6.2 (Derivative). Derivatives *of* (i^*, ct^*) *are defined inductively as follows.*

- (i^*, ct^*) *is a derivative of itself.*
- *If* $\mathcal{O}_{\mathsf{ReEnc}}$ *has been queried on input* (i, j, ct_i), *returning output* ct_j, *then* (j, ct_j) *is a derivative of* (i, ct_i).
- *If* (i, ct) *is a derivative of* (i^*, ct^*), *and* (i', ct') *is a derivative of* (i, ct), *then* (i', ct') *is also a derivative of* (i^*, ct^*).

- *If* $\mathcal{O}_{\mathsf{ReKeyGen}}$ *has been queried on* (i,j) *and* $\mathsf{Dec}(j,\mathsf{ct}_j) \in \{\mathsf{m}_0,\mathsf{m}_1\}$*, then* (j,ct_j) *is a derivative of* (i,ct_i) *for all* ct_i*.*

The first three conditions prevent an adversary from learning the bit b' by a chain of reencryption queries resulting ending to a corrupted key or ending with a decryption query. The purpose of the fourth condition is the same: it applies the same protections to ciphertexts that the adversary reencrypts locally.

Definition 3.6.3 (Proxy Reencryption: CCA Security). *Given a security parameter* 1^λ*, a proxy reencryption scheme is CCA secure if for all probabilistic polynomial-time adversaries* $\mathcal{A}$*, there exists a negligible function* negl *such that*

$$\mathsf{Adv}^{\mathcal{A}}_{\mathsf{cca}}(\lambda) < \frac{1}{2} + \mathsf{negl}(\lambda)$$

3.6.2 CCA and HRA Security

We do not resolve the question of whether CCA security implies HRA security. It may seem that CCA security should imply HRA security. Intuitively, CCA security allows the adversary to make relatively unrestricted queries to both $\mathcal{O}_{\mathsf{ReEnc}}$ and $\mathcal{O}_{\mathsf{Dec}}$, whereas HRA restricts the adversary to making only honest reencryption queries to $\mathcal{O}_{\mathsf{ReEnc}}$. However the fourth type of derivative in the CCA definition restricts the CCA adversary in a way that stymies a naive attempt at a reduction.

The CCA definition of derivative is over-inclusive: it includes all ciphertexts that could in principle be derived from the challenge. On the other hand, the HRA security game restricts reencryption queries only when the ciphertext is actually a derivative of the challenge. The adversary may reencrypt other encryptions of the challenge messages, so long as those encryptions were honestly generated independently from the challenge ciphertext.

3.6.3 Separating $\mathsf{ind\text{-}cca}_{0,1}$ and $\mathsf{ind\text{-}cca}_{2,0}$

The definition of CCA security presented above grants the adversary access to $\mathcal{O}_{\mathsf{Dec}}$ and $\mathcal{O}_{\mathsf{ReEnc}}$ both before and after receiving the challenge ciphertext. Just as in the case of Enc-CCA-1 and Enc-CCA-2 security for standard encryption, it is natural to consider how the definition is altered by restricting the adversary's access to one or both oracles to the period before the challenge.

The work of [NAL15b] formalize this problem by considering a family of security definitions $\mathsf{ind\text{-}cca}_{d,r}$ parameterized by a pair of numbers $d, r \in \{0, 1, 2\}$. The parameter $d = 2$ means $\mathcal{O}_{\mathsf{Dec}}$ is *unrestricted*, $d = 1$ means that $\mathcal{O}_{\mathsf{Dec}}$ is *restricted* to before the challenge, and $d = 0$ means that $\mathcal{O}_{\mathsf{Dec}}$ is unavailable. Similarly, the parameter r defines the availability of $\mathcal{O}_{\mathsf{ReEnc}}$. $\mathsf{ind\text{-}cca}_{2,2}$ corresponds to CCA security as defined above, whereas $\mathsf{ind\text{-}cca}_{0,0}$ corresponds to CPA security.

In Theorems 4.6, [NAL15b] show that $\mathsf{ind\text{-}cca}_{2,0} \not\Longrightarrow \mathsf{ind\text{-}cca}_{0,1}$. That is, even if a PRE scheme is secure with unrestricted access to $\mathcal{O}_{\mathsf{Dec}}$, it may be insecure with restricted access to $\mathcal{O}_{\mathsf{ReEnc}}$. We now prove a (stronger) converse. Our construction also demonstrates the failure of the intuition—described in the Introduction and motivating the original definition of CPA security—that access to $\mathcal{O}_{\mathsf{ReEnc}}$ is as powerful as $\mathcal{O}_{\mathsf{Dec}}$.

Theorem 3.6.4 ($\mathsf{ind\text{-}cca}_{0,2} \not\Longrightarrow \mathsf{ind\text{-}cca}_{1,0}$). *If there exists a PRE scheme that is* $\mathsf{ind\text{-}cca}_{0,2}$ *secure, then there exists a PRE scheme that is* $\mathsf{ind\text{-}cca}_{0,2}$ *secure but not* $\mathsf{ind\text{-}cca}_{1,0}$ *secure.*

Proof. Suppose PRE is $\mathsf{ind\text{-}cca}_{0,2}$ secure, and let $\top$ be a special symbol that is not a valid ciphertext. Define a new scheme PRE' by modifying decryption as follows:

$$\mathsf{Dec}'(\mathsf{sk}, \mathsf{ct}) := \begin{cases} \mathsf{Dec}(\mathsf{sk}, \mathsf{ct}) & \text{if } \mathsf{ct} \neq \top \\ \mathsf{sk} & \text{if } \mathsf{ct} = \top \end{cases}.$$

PRE' is $\mathsf{ind\text{-}cca}_{0,2}$ secure: without access to $\mathcal{O}_{\mathsf{Dec}}$, the view of the adversary is independent of whether the challenger uses PRE or PRE'.

PRE' is not $\mathsf{ind\text{-}cca}_{1,0}$ secure: observe that a single call to $\mathcal{O}_{\mathsf{Dec}}(i^*, \top)$ allows the adversary to learn the challenge secret key sk_{i^*} and thereby distinguish encryptions of m_0 and m_1. □

3.7 Comparison to RIND-CPA

The concurrent work of Derler, Krenn, Lorünser, Ramacher, Slamanig, and Striecks identify the same problem with CPA security as discussed [DKL+18]. They define a new security notion—RIND-CPA security—as an additional property that proxy reencryptions schemes should guarantee.

In this section, we compare the approach of [DKL+18] with ours. We begin by describing RIND-CPA security as defined by [DKL+18]. Next, we compare RIND-CPA with HRA security informally, arguing that HRA provides the better generalization of Enc-CPA security to the PRE setting. Finally, we show that HRA and RIND-CPA security are incomparable notions.

The key feature of RIND-CPA security is that the adversary gets access to an unrestricted ReEnc oracle, but only before seeing the challenge ciphertext. The definition is similar to $\mathsf{ind\text{-}cca}_{0,1}$ of [NAL15b]. The definition of the RIND-CPA security experiment is from [DKL+18, Experiment 8].

Definition 3.7.1 (RIND-CPA Security Experiment).

$\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda), (\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(\mathsf{pp}), b \leftarrow \{0, 1\}$

$(\mathsf{pk}^*, \mathsf{st}) \leftarrow \mathcal{A}(\mathsf{pp}, \mathsf{pk})$

$\mathsf{rk} \leftarrow \mathsf{ReKeyGen}(\mathsf{sk}, \mathsf{pk}^*)$

$(\mathsf{m}_0, \mathsf{m}_1, \mathsf{st}) \leftarrow \mathcal{A}^{\{\mathsf{ReEnc}(\mathsf{rk}, \cdot)\}}(\mathsf{st})$

$b^* \leftarrow \mathcal{A}(\mathsf{st}, \mathsf{Enc}(\mathsf{pk}, \mathsf{m}_b))$

if $b = b^*$ *return 1, else return 0.*

RIND-CPA security requires that for all efficient adversaries, the probability of outputting 1 in the experiment is $\frac{1}{2} \pm \mathsf{negl}(\lambda)$.

We make two brief remarks about the definition. First, the definition only considers the two party setting. Much like the informal description of proxy reencryption in Section 3.1, there is only a single uncorrupted key and a single corrupted key. Second, not only are inputs to ReEnc allowed to be malformed, but the corrupted public key pk^* can be malformed as well. The adversary outputs pk^* itself and it needs not be honestly generated. This makes RIND-CPA security as defined in [DKL$^+$18] formally incomparable to all other definitions of proxy reencryption security we know of, including the very similar $\mathsf{ind\text{-}cca}_{0,1}$ of [NAL15b]. $\mathsf{ind\text{-}cca}_{0,1}$ can be viewed as a version of RIND-CPA for the multi-party setting where malformed public keys are not allowed.

Neither of these properties of the RIND-CPA definition are relevant to our proof of Theorem 3.7.3; the theorem holds replacing RIND-CPA with $\mathsf{ind\text{-}cca}_{0,1}$.

3.7.1 Informal comparison

RIND-CPA is less suitable than HRA as a replacement for CPA security of proxy reencryption. First and most importantly, HRA better captures the intuitive guarantees of Enc-CPA security for standard encryption. Second, HRA is more useful as an intermediate notion of security.

Capturing Enc-CPA security. In Enc-CPA security for standard encryption, the adversary is able to arbitrarily affect the distribution of plaintext messages. One way of viewing this aspect of the definition is that Enc-CPA requires security while being agnostic as to the true distribution over messages (except that it is efficiently sampleable). Other than choosing the distribution over messages, the adversary is only allowed to see publicly-available information (i.e. public keys and parameters) and honestly encrypted ciphertexts. Informally, the Enc-CPA guarantee is that security should hold under normal operating conditions against eavesdropping parties without making distributional assumptions on plaintext messages. However, Enc-CPA makes no guarantees about dishonestly generated or malformed ciphertexts.

HRA security captures this intuitive guarantee better than RIND-CPA. In the course of normal operation of a proxy reencryption, an adversarial party will see reencrypted ciphertexts. These ciphertexts may come at any time—both before and after other ciphertexts whose contents should remain secret. While HRA allows reencryption both before and after the challenge, RIND-CPA restricts the reencryption oracle to the period before the challenge.

HRA makes minimal assumptions about the distribution of plaintext messages by allowing the adversary to choose messages itself, just as in Enc-CPA. RIND-CPA goes further by making requirements in the face of malformed or dishonestly generated ciphertexts.

Usefulness as an intermediate notion. For classical encryption, Enc-CCA security is strictly stronger than Enc-CPA security. In fact, there are many settings where Enc-CPA security is demonstrably insufficient. Why then does the cryptography community continue to study it? There are many answers to this question, but we mention only two. First, although insufficient for some applications, Enc-CPA is useful in others. Second, it is useful as an intermediate goal because it seems to capture a sort of hard core of the general problem of encryption and spurs the development of new techniques.

HRA security enjoys these same features; RIND-CPA does not. As for usefulness for applications, HRA is meaningful in many of the envisioned applications of proxy reencryption—many more than CPA security. Because RIND-CPA restricts the reencryption oracle to the period before the challenge ciphertext, its usefulness in applications is less clear.

The challenge of constructing CCA secure proxy reencryption is the same as the challenge of Enc-CCA secure encryption: namely, dealing with dishonestly generated, possibly malformed ciphertexts. RIND-CPA, by allowing malformed ciphertexts, presents similar challenges as full CCA security.

As for the usefulness of HRA as an intermediate goal towards CCA security, the historical development of proxy reencryption is proof itself. This sounds paradoxical: how can this be true if the notion has only just been introduced? Many of cryptographers that were targeting CPA security developed schemes that achieve HRA security with only minimal modification. The techniques developed in these constructions were later adapted to achieve CCA security. This suggests that cryptographers' intuitions for the hard core of reencryption were not flawed, only the formalization of these intuitions as CPA security. HRA security is a better formalization for these intuitions and thus an appropriate intermediate goal for reencryption research.

3.7.2 Separating RIND-CPA and HRA security

The following pair of theorems support the conclusion that HRA security and RIND-CPA security are incomparable. The theorems and proofs apply to both RIND-CPA and $\mathsf{ind\text{-}cca}_{0,1}$ security notions. For simplicity, we use RIND-CPA throughout.

Theorem 3.7.2. *If there exists an HRA secure PRE scheme, then there exists a PRE scheme that is HRA secure but not RIND-CPA (nor* $\mathsf{ind\text{-}cca}_{0,1}$*) secure.*

Proof. Suppose PRE is HRA secure, and let $\top$ be a special symbol that is not a valid ciphertext. Define a new scheme PRE' by modifying reencryption as follows:

$$\mathsf{ReEnc}'(\mathsf{rk},\mathsf{ct}) := \begin{cases} \mathsf{ReEnc}(\mathsf{rk},\mathsf{ct}) & \text{if } \mathsf{ct} \neq \top \\ \mathsf{rk} & \text{if } \mathsf{ct} = \top \end{cases}.$$

PRE' is still HRA secure: $\mathcal{O}_{\mathsf{ReEnc}'}$ is functionally equivalent to $\mathcal{O}_{\mathsf{ReEnc}}$ when restricted to honestly generated ciphertexts..

PRE' is not RIND-CPA secure: a single call to $\mathcal{O}_{\mathsf{ReEnc}'}(i,j,\top)$ (made before the challenge) allows the adversary to learn the reencryption key $\mathsf{rk}_{i\to j}$ and thereby decrypt the challenge ciphertext. □

Theorem 3.7.3. *Under the assumptions stated below, there exists a PRE scheme that is RIND-CPA secure but not HRA (nor* $\mathsf{ind\text{-}cca}_{0,1}$*) secure.*

The claim assumes the existence of pair of encryption schemes, PRE and FHE with the following properties. PRE is a RIND-CPA secure proxy reencryption scheme with a ciphertext space $\mathcal{C}_{\mathsf{inner}}$. FHE is a circuit private fully homomorphic encryption scheme with message space $\mathcal{M}_{\mathsf{fhe}} = \mathcal{C}_{\mathsf{inner}}$. The message spaces and ciphertext spaces of the two schemes are all disjoint and efficiently decidable. Finally, the additional proxy reencryption scheme $\mathsf{PRE}_{\mathsf{FHE}}$ corresponding to the FHE scheme (see Section 3.5.3) is RIND-CPA secure.[12] For simplicity, we also assume perfect correctness of reencryption (for both schemes) and of homomorphic evaluation.

The remainder of this appendix is devoted to the proof of Theorem 3.7.3.

3.7.2.1 Proof Intuition for Theorem 3.7.3

Recall that RIND-CPA security allows the adversary access to an unrestricted ReEnc oracle, but only before the challenge ciphertext is generated. The main difficulty in separating RIND-CPA and HRA security is the restriction in the HRA reencryption oracle to honestly generated ciphertexts.

The first idea in our construction is the observation that separating RIND-CPA and HRA security would be easy if it were possible to use Enc oracle to generate a fresh, honest encryption of the challenge plaintext. This fresh encryption could be reencrypted by the HRA reencryption oracle to a corrupted key, revealing the challenge plaintext.

The second idea is to have two layers of encryption, where the message space of the outer layer is equal to the ciphertext space of the inner layer. If the challenge ciphertext comes from the inner layer, then it can be used as input to the Enc oracle to generate a new *outer ciphertext* containing information about the challenge plaintext—namely, an encryption of the challenge ciphertext. The outer ciphertext is honestly generated and can be reencrypted to a corrupt party and decrypted. But it seems we are no better off; decrypting the outer ciphertext only returns the challenge ciphertext still encrypted under the key of an honest party.

The third idea is to modify ReEnc—using fully homomorphic encryption—to reencrypt both the outer ciphertext and the inner challenge ciphertext. In addition to the usual reencrypted ciphertext, we augment ReEnc to output an additional, *doubly*

[12]The proof requires that an encryption scheme be both fully homomorphic and support proxy reencryption with RIND-CPA security. For concreteness, we have chosen to assume that there exists an FHE scheme whose corresponding PRE is RIND-CPA secure, but a different construction would suffice. We do not further explore the underlying cryptographic assumptions needed to instantiate this encryption scheme.

reencrypted ciphertext, where both the outer and inner ciphertexts have been reencrypted. If the recipient of the resulting ciphertext is corrupt, the adversary can decrypt both layers and recover the challenge plaintext, violating HRA security.

We now describe the intuition for how to perform double reencryption. Suppose the proxy reencryption scheme used for the outer layer of encryption is also fully homomorphic. Such a scheme can be constructed from any FHE scheme (see Section 3.5.3). Given input an outer layer ciphertext $\mathsf{ct}_{\mathsf{outer}} = \mathsf{Enc}(\mathsf{ct}_{\mathsf{inner}})$, ReEnc will homomorphically evaluate $\mathsf{Eval}_{\mathsf{fhe}}(\mathsf{ReEnc}, \mathsf{ct}_{\mathsf{outer}})$. The result is an (non-reencrypted) outer ciphertext containing a reencrypted inner ciphertext. Then, ReEnc reencrypts that outer ciphertext. This produces a reencrypted outer ciphertext containing a reencrypted inner ciphertext.

Violating HRA security is simple: the adversary encrypts the challenge ciphertexts, reencrypts it to a corrupted key, then decrypts the doubly-reencrypted component twice to recover the challenge message.

It remains to prove that the constructed PRE scheme is RIND-CPA secure. Double reencryption can be simulated by a sequence of calls to Enc, Dec and $\mathcal{O}_{\mathsf{ReEnc}}$, allowing us to analyze the two-layered scheme without the double-reencryption modification to ReEnc. The RIND-CPA security of that scheme follows directly from the RIND-CPA security of the PRE scheme underlying the two layers.

The remainder of the section is organized as follows. First, we formalize *Dual PRE*, a conceptually simpler generalization of the layered encryption described above. We then present the PRE scheme used to separate RIND-CPA and HRA security. Finally, we show that the scheme is indeed HRA insecure and RIND-CPA secure.

3.7.2.2 Dual PRE

Let $\mathsf{PRE}_\alpha = (\mathsf{Setup}_\alpha, \mathsf{KeyGen}_\alpha, \mathsf{Enc}_\alpha, \mathsf{Dec}_\alpha, \mathsf{ReKeyGen}_\alpha, \mathsf{ReEnc}_\alpha)$ be a proxy reencryption scheme with message space $\mathcal{M}_\alpha$ and ciphertext space $\mathcal{C}_\alpha$. Similarly, let PRE_β be a proxy reencryption scheme with message space $\mathcal{M}_\beta$ and ciphertext space $\mathcal{C}_\beta$. Suppose that $\mathcal{M}_\alpha$ and $\mathcal{M}_\beta$ are disjoint and that $\mathcal{C}_\alpha$ and $\mathcal{C}_\beta$ are disjoint. Suppose also that membership is efficiently decidable for all four sets. We define the *Dual PRE scheme*

$\mathsf{PRE}_{\mathsf{dual}} = (\mathsf{Setup}_{\mathsf{dual}}, \mathsf{KeyGen}_{\mathsf{dual}}, \mathsf{Enc}_{\mathsf{dual}}, \mathsf{Dec}_{\mathsf{dual}}, \mathsf{ReKeyGen}_{\mathsf{dual}}, \mathsf{ReEnc}_{\mathsf{dual}})$ for message space $\mathcal{M}_{\mathsf{dual}} = \mathcal{M}_\alpha \cup \mathcal{M}_\beta$ and ciphertext space $\mathcal{C}_{\mathsf{dual}} = \mathcal{C}_\alpha \cup \mathcal{C}_\beta$ as follows.

$$\begin{aligned}
\mathsf{Setup}_{\mathsf{dual}}(1^\lambda) &:= (\mathsf{pp}^\alpha, \mathsf{pp}^\beta), \\
&\qquad \text{where } \mathsf{pp}^\alpha \leftarrow \mathsf{Setup}_\alpha(1^\lambda) \\
&\qquad\qquad \mathsf{pp}^\beta \leftarrow \mathsf{Setup}_\beta(1^\lambda) \\
\mathsf{KeyGen}_{\mathsf{dual}}(\mathsf{pp}^\alpha, \mathsf{pp}^\beta) &:= ((\mathsf{pk}^\alpha, \mathsf{pk}^\beta), (\mathsf{sk}^\alpha, \mathsf{sk}^\beta)), \\
&\qquad \text{where } (\mathsf{pk}^\alpha, \mathsf{sk}^\alpha) \leftarrow \mathsf{KeyGen}_\alpha(\mathsf{pp}^\alpha) \\
&\qquad\qquad (\mathsf{pk}^\beta, \mathsf{sk}^\beta) \leftarrow \mathsf{KeyGen}_\beta(\mathsf{pp}^\beta)) \\
\mathsf{ReKeyGen}_{\mathsf{dual}}(\mathsf{sk}_i^\alpha, \mathsf{sk}_i^\beta, \mathsf{pk}_j^\alpha, \mathsf{pk}_j^\beta) &= (\mathsf{rk}_{i\to j}^\alpha, \mathsf{rk}_{i\to j}^\beta), \\
&\qquad \text{where } \mathsf{rk}_{i\to j}^\alpha \leftarrow \mathsf{ReKeyGen}_\alpha(\mathsf{sk}_i^\alpha, \mathsf{pk}_j^\alpha) \\
&\qquad\qquad \mathsf{rk}_{i\to j}^\beta \leftarrow \mathsf{ReKeyGen}_\beta(\mathsf{sk}_i^\beta, \mathsf{pk}_j^\beta) \\
\mathsf{Enc}_{\mathsf{dual}}(\mathsf{pk}^\alpha, \mathsf{pk}^\beta, \mathsf{m}) &:= \begin{cases} \mathsf{Enc}_\alpha(\mathsf{pk}^\alpha, \mathsf{m}) & \text{if } \mathsf{m} \in \mathcal{M}_\alpha \\ \mathsf{Enc}_\beta(\mathsf{pk}^\beta, \mathsf{m}) & \text{if } \mathsf{m} \in \mathcal{M}_\beta \end{cases} \\
\mathsf{Dec}_{\mathsf{dual}}(\mathsf{sk}^\alpha, \mathsf{sk}^\beta, \mathsf{ct}) &:= \begin{cases} \mathsf{Dec}_\alpha(\mathsf{sk}^\alpha, \mathsf{ct}) & \text{if } \mathsf{ct} \in \mathcal{C}_\alpha \\ \mathsf{Dec}_\beta(\mathsf{sk}^\beta, \mathsf{ct}) & \text{if } \mathsf{ct} \in \mathcal{C}_\beta \end{cases} \\
\mathsf{ReEnc}_{\mathsf{dual}}(\mathsf{rk}^\alpha, \mathsf{rk}^\beta, \mathsf{ct}) &:= \begin{cases} \mathsf{ReEnc}_\alpha(\mathsf{rk}^\alpha, \mathsf{ct}) & \text{if } \mathsf{ct} \in \mathcal{C}_\alpha \\ \mathsf{ReEnc}_\beta(\mathsf{rk}^\beta, \mathsf{ct}) & \text{if } \mathsf{ct} \in \mathcal{C}_\beta \end{cases}
\end{aligned}$$

Claim 3.7.4. *If* PRE_α *and* PRE_β *are RIND-CPA secure, then* $\mathsf{PRE}_{\mathsf{dual}}$ *is RIND-CPA secure.*

Outline. For any efficient adversary $\mathcal{A}_{\mathsf{dual}}$, there exist efficient adversaries $\mathcal{A}_\alpha$ and $\mathcal{A}_\beta$ such that $\mathsf{Adv}^{\mathcal{A}_{\mathsf{dual}}}_{\mathsf{PRE}_{\mathsf{dual}}} \leq \max\{\mathsf{Adv}^{\mathcal{A}_\alpha}_{\mathsf{PRE}_\alpha}, \mathsf{Adv}^{\mathcal{A}_\beta}_{\mathsf{PRE}_\beta}\}$. The adversary $\mathcal{A}_\alpha$ simulates the $\mathsf{PRE}_{\mathsf{dual}}$ security game by calling its own oracles or simulating an instance of PRE_β, as appropriate. adv_β is analogous. By the assumption that PRE_α and PRE_β are both RIND-CPA secure, $\mathsf{Adv}^{\mathcal{A}_{\mathsf{dual}}}_{\mathsf{PRE}_{\mathsf{dual}}}$ is negligible. □

3.7.2.3 The Counterexample

For the relevant background on fully homomorphic encryption and its connection to proxy reencryption, see Section 3.5.3 and [Gen09a, Section 2].

Let $\mathsf{PRE}_{\mathsf{inner}}$ be a RIND-CPA secure proxy reencryption scheme with ciphertext space $\mathcal{C}_{\mathsf{inner}}$. Let FHE be a circuit private fully homomorphic encryption scheme with message space $\mathcal{M}_{\mathsf{fhe}} = \mathcal{C}_{\mathsf{inner}}$ and let $\mathsf{PRE}_{\mathsf{fhe}}$ be the corresponding proxy reencryption scheme. Suppose that $\mathsf{PRE}_{\mathsf{fhe}}$ is RIND-CPA secure. Note that FHE and $\mathsf{PRE}_{\mathsf{fhe}}$ are two different views of *the same encryption scheme*, with identical KeyGen, Enc, and Dec algorithms. Specifically, the homomorphic evaluation algorithm $\mathsf{Eval}_{\mathsf{fhe}}$ works on ciphertexts of $\mathsf{PRE}_{\mathsf{fhe}}$.

We construct a proxy reencryption scheme PRE by modifying the Dual PRE scheme. Let $\mathsf{PRE}_{\mathsf{dual}}$ be the Dual PRE of $\mathsf{PRE}_\alpha = \mathsf{PRE}_{\mathsf{inner}}$ and $\mathsf{PRE}_\beta = \mathsf{PRE}_{\mathsf{fhe}}$. We augment $\mathsf{ReEnc}_{\mathsf{dual}}$ to perform double reencryption as described in the proof intuition above in addition to the normal ciphertext reencryption. Enc and Dec are modified accordingly. The modified lines are marked with a $\star$.

$$\mathsf{Enc}(\mathsf{pk}^{\mathsf{inner}}, \mathsf{pk}^{\mathsf{fhe}}, \mathsf{m}) := \begin{cases} \mathsf{Enc}_{\mathsf{inner}}(\mathsf{pk}^{\mathsf{inner}}, \mathsf{m}) & \text{if } \mathsf{m} \in \mathcal{M}_{\mathsf{inner}} \\ (\mathsf{Enc}_{\mathsf{fhe}}(\mathsf{pk}^{\mathsf{fhe}}, \mathsf{m}), \bot) & \text{if } \mathsf{m} \in \mathcal{M}_{\mathsf{fhe}} \quad (\star) \end{cases}$$

$$\mathsf{ReEnc}(\mathsf{rk}^{\mathsf{inner}}, \mathsf{rk}^{\mathsf{fhe}}, \mathsf{ct}) := \begin{cases} \mathsf{ReEnc}_{\mathsf{inner}}(\mathsf{rk}^{\mathsf{inner}}, \mathsf{ct}) & \text{if } \mathsf{ct} \in \mathcal{C}_{\mathsf{inner}} \\ (\mathsf{ReEnc}_{\mathsf{fhe}}(\mathsf{rk}^{\mathsf{fhe}}, \mathsf{ct}^1), \widetilde{\mathsf{ct}}) & \text{if } \mathsf{ct} = (\mathsf{ct}^1, \mathsf{ct}^2) \text{ and } \mathsf{ct}^1 \in \mathcal{C}_{\mathsf{fhe}} \quad (\star) \\ \bot & \text{otherwise} \quad (\star) \end{cases}$$

$$\text{where } \widetilde{\mathsf{ct}} = \mathsf{ReEnc}_{\mathsf{fhe}}\Big(\mathsf{rk}^{\mathsf{fhe}}, \mathsf{Eval}_{\mathsf{fhe}}\big(\mathsf{ReEnc}_{\mathsf{inner}}(\mathsf{rk}^{\mathsf{inner}}, \cdot), \mathsf{ct}^1\big)\Big)$$

$$\mathsf{Dec}(\mathsf{sk}^{\mathsf{inner}}, \mathsf{sk}^{\mathsf{fhe}}, \mathsf{ct}) := \begin{cases} \mathsf{Dec}_{\mathsf{inner}}(\mathsf{sk}^{\mathsf{inner}}, \mathsf{ct}) & \text{if } \mathsf{ct} \in \mathcal{C}_{\mathsf{inner}} \\ \mathsf{Dec}_{\mathsf{fhe}}(\mathsf{sk}^{\mathsf{fhe}}, \mathsf{ct}^1) & \text{if } \mathsf{ct} = (\mathsf{ct}^1, \mathsf{ct}^2) \text{ and } \mathsf{ct}^1 \in \mathcal{C}_{\mathsf{fhe}} \quad (\star) \end{cases}$$

Observe that for honestly generated ciphertexts, $\mathsf{ct} = (\mathsf{ct}^1, \mathsf{ct}^2)$ if and only if $\mathsf{ct}^1 \in \mathcal{C}_{\mathsf{fhe}}$, and both hold whenever $\mathsf{ct} \notin \mathcal{C}_{\mathsf{inner}}$.

3.7.2.4 HRA Insecurity

The attack on the HRA security game uses only the correctness of the PRE and FHE schemes. It requires only two parties; to help disambiguate the various subscripts and superscripts, we will index these parties by alice and bob. Let $\mathsf{m}_0, \mathsf{m}_1 \in \mathcal{M}_{\mathsf{inner}}$ be distinct messages. The HRA adversary proceeds as follows.

- Receive pp from challenger. Call the Uncorrupted Key Generation oracle once to obtain $\mathsf{pk}_{\mathsf{alice}}$. Call the Corrupted Key Generation oracle once to obtain $(\mathsf{pk}_{\mathsf{bob}}, \mathsf{sk}_{\mathsf{bob}})$.

- Query the Challenge oracle on input $(\mathsf{alice}, \mathsf{m}_0, \mathsf{m}_1)$ to obtain the challenge $(1, \mathsf{ct}^*_{\mathsf{alice,inner}})$, an encryption of m_b for random $b \leftarrow \{0,1\}$. Recall that 1 is the index of this ciphertext which may be used as input to $\mathcal{O}_{\mathsf{ReEnc}}$.

- Query $\mathcal{O}_{\mathsf{Enc}}(\mathsf{alice}, \mathsf{ct}^*_{\mathsf{alice,inner}})$ to obtain the ciphertext $(2, \mathsf{ct}_{\mathsf{alice}})$, where $\mathsf{ct}_{\mathsf{alice}} = (\mathsf{ct}^*_{\mathsf{alice,fhe}}, \perp)$.

- Query $\mathcal{O}_{\mathsf{ReEnc}}(\mathsf{alice}, \mathsf{bob}, 2)$ to obtain $(3, \mathsf{ct}_{\mathsf{bob}})$, where $\mathsf{ct}_{\mathsf{bob}} = (\mathsf{ct}^1_{\mathsf{bob,fhe}}, \mathsf{ct}^*_{\mathsf{bob,fhe}})$. If $\mathcal{O}_{\mathsf{ReEnc}}$ outputs $\perp$, abort.

- Let $\mathsf{ct}^*_{\mathsf{bob,inner}} = \mathsf{Dec}\big(\mathsf{sk}_{\mathsf{bob}}, (\mathsf{ct}^*_{\mathsf{bob,fhe}}, \perp)\big)$. Let $\mathsf{m}' = \mathsf{Dec}(\mathsf{sk}_{\mathsf{bob}}, \mathsf{ct}^*_{\mathsf{bob,inner}})$.

- Finally, call the Decision oracle with b' such that $\mathsf{m}_{b'} = \mathsf{m}'$.

The call to $\mathcal{O}_{\mathsf{ReEnc}}$ never causes the adversary to abort, because no ciphertext output by $\mathcal{O}_{\mathsf{Enc}}$ can be in the set Deriv (see the specification of $\mathcal{O}_{\mathsf{ReEnc}}$ in Definition 3.4.1).

It remains to show only that $b' = b$. By the correctness of reencryption and homomorphic evaluation for $\mathsf{PRE}_{\mathsf{fhe}}$:

$$\begin{aligned}
\mathsf{ct}^*_{\mathsf{bob,inner}} &= \mathsf{Dec}_{\mathsf{fhe}}(\mathsf{sk}^{\mathsf{fhe}}_{\mathsf{bob}}, \mathsf{ct}^*_{\mathsf{bob,fhe}}) \\
&= \mathsf{Dec}_{\mathsf{fhe}}\left(\mathsf{sk}^{\mathsf{fhe}}_{\mathsf{bob}}, \mathsf{ReEnc}_{\mathsf{fhe}}\left(\mathsf{rk}^{\mathsf{fhe}}_{\mathsf{alice}\to\mathsf{bob}}, \mathsf{Eval}_{\mathsf{fhe}}\big(\mathsf{ReEnc}_{\mathsf{inner}}(\mathsf{rk}^{\mathsf{inner}}_{\mathsf{alice}\to\mathsf{bob}}, \cdot), \mathsf{ct}^*_{\mathsf{alice,fhe}}\big)\right)\right) \\
&= \mathsf{ReEnc}_{\mathsf{inner}}\left(\mathsf{rk}^{\mathsf{inner}}_{\mathsf{alice}\to\mathsf{bob}}, \mathsf{ct}^*_{\mathsf{alice,inner}}\right)
\end{aligned}$$

By the correctness of $\mathsf{PRE}_{\mathsf{inner}}$:

$$\begin{aligned}
\mathsf{m}_{b'} &= \mathsf{Dec}_{\mathsf{inner}}(\mathsf{sk}^{\mathsf{inner}}_{\mathsf{bob}}, \mathsf{ct}^*_{\mathsf{bob,inner}}) \\
&= \mathsf{Dec}_{\mathsf{inner}}\left(\mathsf{sk}^{\mathsf{inner}}_{\mathsf{bob}}, \mathsf{ReEnc}_{\mathsf{inner}}\left(\mathsf{rk}^{\mathsf{inner}}_{\mathsf{alice}\to\mathsf{bob}}, \mathsf{ct}^*_{\mathsf{alice,inner}}\right)\right) \\
&= \mathsf{Dec}_{\mathsf{inner}}\left(\mathsf{sk}^{\mathsf{inner}}_{\mathsf{bob}}, \mathsf{ReEnc}_{\mathsf{inner}}\left(\mathsf{rk}^{\mathsf{inner}}_{\mathsf{alice}\to\mathsf{bob}}, \mathsf{Enc}_{\mathsf{inner}}\left(\mathsf{pk}^{\mathsf{inner}}_{\mathsf{alice}}, \mathsf{m}_b\right)\right)\right) \\
&= \mathsf{m}_b
\end{aligned}$$

3.7.2.5 RIND-CPA Security

We assumed that both $\mathsf{PRE}_{\mathsf{inner}}$ and $\mathsf{PRE}_{\mathsf{fhe}}$ were RIND-CPA secure. By Claim 3.7.4, the Dual PRE scheme $\mathsf{PRE}_{\mathsf{dual}}$ is also RIND-CPA secure. It remains to show that the counterexample PRE (constructed by modifying $\mathsf{PRE}_{\mathsf{dual}}$) is RIND-CPA secure.

We reduce the RIND-CPA security of PRE to the RIND-CPA security of $\mathsf{PRE}_{\mathsf{dual}}$. Recall that the RIND-CPA security game is like the CPA security game, but with access to an unrestricted reencryption oracle $\mathcal{O}_{\mathsf{ReEnc}}$ up until the Challenge Oracle is called.

From any probabilistic, polynomial-time algorithm $\mathcal{A}$ (the PRE adversary), we construct an algorithm $\mathcal{A}' = \mathcal{A}_{\mathsf{dual}}$ such that $\mathsf{Adv}^{\mathcal{A}'}_{\mathsf{rind-cpa}}(\lambda) \geq \mathsf{Adv}^{\mathcal{A}}_{\mathsf{rind-cpa}}(\lambda) - \mathsf{negl}(\lambda)$; by the RIND-CPA security of $\mathsf{PRE}_{\mathsf{dual}}$ this advantage is negligible, completing the proof.

$\mathcal{A}_{\mathsf{dual}}$ runs $\mathcal{A}$ and simulates the RIND-CPA security game (if $\mathcal{A}$ does not follow the specification of the HRA security game, $\mathcal{A}_{\mathsf{dual}}$ simply aborts). At the outset, $\mathcal{A}_{\mathsf{dual}}$ makes one additional Corrupted Key Generation query, obtaining the key pair $(\mathsf{pk}_\star, \mathsf{sk}_\star)$. This key pair is not passed to $\mathcal{A}$, and will only be used to answer calls to $\mathcal{O}_{\mathsf{ReEnc}}$. To answer oracle calls by $\mathcal{A}$ to any oracle other than $\mathcal{O}_{\mathsf{ReEnc}}$ and $\mathcal{O}_{\mathsf{Enc}}$, $\mathcal{A}_{\mathsf{dual}}$ simply passes the calls and answers unaltered to the corresponding RIND-CPA oracles. These oracles are perfectly simulated.

To answer oracle calls to $\mathcal{O}_{\mathsf{Enc}}$: $\mathcal{A}_{\mathsf{dual}}$ calls its own encryption oracle to obtain the response (k, ct). It then returns $(k, (\mathsf{ct}, \perp))$ to $\mathcal{A}$. This perfectly simulates the encryption oracle for PRE.

To answer oracle calls to $\mathcal{O}_{\mathsf{ReEnc}}(i, j, \mathsf{ct})$: $\mathcal{A}_{\mathsf{dual}}$ uses a series of calls to its own reencryption oracle to construct the necessary ciphertext, as follows. For intuition, recall that the unrestricted reencryption oracle allows $\mathcal{A}_{\mathsf{dual}}$ to reencrypt any outer layer ciphertext to corrupted party $\star$ and thereby decrypt and learn the inner layer ciphertext. Then it can reencrypt the inner ciphertext, encrypt the result using the outer scheme using pk_i, and finally reencrypt the result to the recipient party j.

- If $\mathsf{ct} \in \mathcal{C}_{\mathsf{inner}}$, call own $\mathcal{O}_{\mathsf{ReEnc}}(i, j, \mathsf{ct})$, and return response to $\mathcal{A}$. If $\mathsf{ct} \neq (\mathsf{ct}^1, \mathsf{ct}^2)$ or $\mathsf{ct}^1 \notin \mathcal{C}_{\mathsf{fhe}}$, return $\perp$ to $\mathcal{A}$.
- Let $\mathsf{ct} = (\mathsf{ct}^1, \mathsf{ct}^2)$. Call own $\mathcal{O}_{\mathsf{ReEnc}}(i, j, \mathsf{ct}^1)$ and let ct_j^1 be the response.
- Call own $\mathcal{O}_{\mathsf{ReEnc}}(i, \star, \mathsf{ct}^1)$ and let $\mathsf{ct}_\star^1$ be the response. Note that $\mathsf{ct}_\star^1$ is the encryption under $\mathsf{pk}_\star$ of a *ciphertext.* Let $\mathsf{ct}' = \mathsf{Dec}(\mathsf{sk}_\star, \mathsf{ct}_\star^1)$.
- Call own $\mathcal{O}_{\mathsf{ReEnc}}(i, j, \mathsf{ct}')$, and let ct_j' be the response.
- Call own $\mathcal{O}_{\mathsf{ReEnc}}(i, j, \mathsf{Enc}(\mathsf{pk}_i, \mathsf{ct}_j'))$, and let $\widetilde{\mathsf{ct}}$ be the response.
- Return $(\mathsf{ct}_j^1, \widetilde{\mathsf{ct}})$ to $\mathcal{A}$.

This whole sequence of reencryptions, decryptions, evaluations, and encryptions produces a ciphertext whose output distribution is statistically close to the output distribution of $\mathcal{O}_{\mathsf{ReEnc}}$ in the real PRE game. This follows from the circuit privacy of the FHE scheme (along with the correctness of both the reencryption and FHE evaluation).

The views of $\mathcal{A}$ in the real game (using the real $\mathcal{O}_{\mathsf{ReEnc}}$) and the simulated security game (using the simulation described above) are statistically close. $\mathcal{A}_{\mathsf{dual}}$ wins the security game for $\mathsf{PRE}_{\mathsf{dual}}$ if and only if $\mathcal{A}$ wins the simulated security game for PRE.

Part II

Data Anonymization

Chapter 4

Towards Formalizing the GDPR Notion of Singling Out

4.1 Introduction

Data privacy laws—like HIPAA, FERPA, and Title 13 in the US, and the GDPR in the EU—govern the use of sensitive personal information. These laws delineate the boundaries of appropriate use of personal information and impose steep penalties upon rule breakers. To adhere to these laws, practitioners need to apply suitable controls and statistical disclosure limitation techniques. Many commonly used techniques including k-anonymity, bucketing, rounding, pseudonymization, and swapping offer privacy protections that are seemingly intuitive but only poorly understood. And while there is a vast literature of best practices, a litany of successful privacy attacks demonstrates that these techniques often fall short of the sort of privacy envisioned by legal standards [Ohm10].

A more disciplined approach is needed. However, there is a significant conceptual gap between legal and mathematical thinking around data privacy. Privacy regulations are grounded in legal concepts such as personally-identifiable information (PII), linkage, distinguishability, anonymization, risk, and inference. In contrast, much of the recent progress in data privacy technology is rooted in mathematical privacy

Based on "Towards Formalizing the GDPR's Notion of Singling Out" with Kobbi Nissim [CN19].

models such as differential privacy [DMNS06] that offer a foundational treatment of privacy, with formal privacy guarantees. And while such techniques are being actively developed in the academy, industry, and government, there is a basic disconnect between the legal and mathematical conceptions. The effect is uncertainty as to which technical offerings adequately match expectations expressed in legal standards [NW18].

Bridging between legal and technical concepts of privacy. We aim to address this uncertainty by translating between the legal and the technical. To do so, we begin with a concept appearing in the law, then model some aspect of it mathematically. With the mathematical formalism in hand, we can better understand the *requirements* of the law, their *implications*, and the *techniques* that might satisfy them.

This is part of a larger effort to bridge between legal and technical conceptions of privacy. An earlier work analyzed the privacy requirements of FERPA and modeled them in a game-based definition, as is common in cryptography. The definition was used to argue that the use of differentially private analyses suffices for satisfying a wide range of interpretation of FERPA [NBW$^+$18]. An important feature of FERPA that enabled this analysis is that FERPA and its accompanying documents contain a rather detailed description of a privacy attacker and the attacker's goals.

This chapter focuses on the concept of *singling out* from the GDPR. More specifically, we examine what it means for a data anonymization mechanism to ensure *security against singling out* in a data release. Preventing singling out attacks in a dataset is a necessary (but maybe not sufficient) precondition for a dataset to be considered effectively anonymized and thereby free from regulatory restrictions under the GDPR. Ultimately, our goal is to better understand a concept foundational to the GDPR, enabling a rigorous mathematical examination of whether certain classes of techniques (e.g., k-anonymity, differential privacy, pseudonymization) provide an important legal protection.

We are not the first to study this issue. "Opinion on Anonymisation Techniques" [A29b] provides guidance about the use of various privacy technologies—

including k-anonymity and differential privacy—as anonymization techniques. It's analysis is centered on asking whether each technology effectively mitigates three risks: "singling out, linkability, and inference." For instance, [A29b] concludes that with k-anonymity singling is no longer a risk whereas with differential privacy it "may not" be a risk. Though similar in purpose to our work, its technical analyses are informal and coarse. Reconsidering these questions with mathematical rigor, we encourage revisiting the conclusions in [A29b].

4.1.1 Singling Out in the GDPR

We begin with the text of the GDPR. It consists of *articles* detailing the obligations placed on processors of personal data as well as *recitals* containing explanatory remarks. Article 1 of the regulation delineates its scope as "lay[ing] down rules relating to the protection of natural persons with regard to the processing of personal data and rules relating to the free movement of personal data." The GDPR places no restrictions on the processing of non-personal data, even if this data is the result of *anonymizing* personal data.[1] Personal data is defined in Article 4 to mean "any information relating to an identified or identifiable natural person; an identifiable natural person is one who can be identified, directly or indirectly." What it means for a person to be "identified, directly or indirectly" is not elaborated in the articles of the GDPR. Recital 26 sheds a little more light: "To determine whether a natural person is identifiable account should be taken of all the means reasonably likely to be used, such as singling out, either by the controller or by another person to identify the natural person directly or indirectly." Singling out is one way to identify a person in data, and only data that does not allow singling out may be excepted from the regulation.[2]

For insight as to the regulation's meaning, we refer to two documents prepared

[1]Recital 26 emphasizes this point: "The principles of data protection should therefore not apply to anonymous information, namely information which does not relate to an identified or identifiable natural person or to personal data rendered anonymous in such a manner that the data subject is not or no longer identifiable."

[2]Interestingly, singling out is the only criterion for identifiability explicitly mentioned in the GDPR, the only occurrence the term being the quoted passage from Recital 26.

by the Article 29 Data Protection Working Party, an advisory body set out by the EU Data Protection Directive.[3] "Opinion on the Concept of Personal Data" [A29a] elaborates on the meaning of "identifiable, directly or indirectly." A person is identified "within a group of persons [when] he or she is distinguished from all other members of the group." One way of distinguishing a person from a group is by specifying "criteria which allows him to be recognized by narrowing down the group to which he belongs." If the group is narrowed down to an individual, that individual has been singled out.[4] Looking ahead, we will call this *isolating* an individual in the dataset and argue that not every instance of isolation should be considered a singling out attack.

We highlight three additional insights that inform our work. First, identification does not require a name or any other traditional identifier. For instance, singling out can be done with a "small or large" collection of seemingly innocuous traits (e.g., "the man wearing a black suit"). Indeed, this is what is meant by "indirectly identifiable." An example of singling out in practice cited by [A29b] showed that four locations sufficed to uniquely identify 95% of people in a pseudonymized dataset of time-stamped locations. This is considered singling out even though no method of linking such location traces to individuals' names was identified.

Second, identifiable data may come in many forms, including microdata, aggregate statistics, news articles, encrypted data, video footage, and server logs. What's important is not the form of the data, its whether the data permits an individual to be singled out. We apply this same principle to the manner in which an individual is singled out within a dataset. Most examples focus on specifying a collection of attributes (e.g., four time-stamped locations) that match a single person in the data. The collection of attributes corresponds to a *predicate*: a function that assigns to each person in the dataset a value 0 or 1 (interpreted as false or true respectively). We interpret the regulation as considering data to be personal if an individual can be

[3]Formally, Directive on the protection of individuals with regard to the processing of personal data and on the free movement of such data. 95/46/EC.

[4]The notion of "singling out" is not defined in the Opinion on the Concept of Personal Data [A29a]. It is used in [A29a] four times, each consistent with the above interpretation. Our interpretation coincides with and was initially inspired by that of [FEO+18], defining "singling out as occurring when an analyst correctly makes a statement of the form 'There is exactly one user that has these attributes.' "

distinguished within a dataset using any predicate, not only those that correspond to specifying collections of attributes. Just as "small or large" collections of attributes may be used to single out, we allow these predicates to be simple or complex.

Third, whether or not a collection of attributes identifies a person is context-dependent. "A very common family name will not be sufficient to identify someone - i.e. to single someone out - from the whole of a country's population, while it is likely to achieve identification of a pupil in a classroom." Both the *prevalence* of the name and the *size* of the group are important in the example, and will be important in our formalization.

4.1.2 Our Contributions

4.1.2.1 Defining Security Against Predicate Singling Out

We formalize and analyze *predicate singling out*, a notion which is intended to partially model the GDPR's notion of singling out. Following the discussion above, we begin with the idea that singling out an individual from a group involves specifying a predicate that uniquely distinguishes the individual, which we call *isolation*. Using this terminology, an intuitive interpretation of the GDPR's requirement is that to be considered secure against singling out, a function of the data must prevent isolation. Trying to make this idea formal, we will see that it requires some refinement.

We restrict our attention to datasets $\mathbf{x} = (x_1, \ldots, x_n)$ of size n, where each row x_i is sampled according to some underlying probability distribution D over a universe X. The dataset $\mathbf{x}$ is assumed to contain personal data corresponding to individuals, with at most one row per individual. For example, $\mathbf{x}$ might consist of home listings, hospital records, internet browsing history, or any other personal information. A mechanism M takes $\mathbf{x}$ as input and outputs some data release $M(\mathbf{x})$, be it a map of approximate addresses, aggregate statistics about disease, or pseudonymized internet histories. We call M an *anonymization mechanism* because it purportedly anonymizes the personal data $\mathbf{x}$.

An adversary $\mathcal{A}$ attempts to output a predicate $p : X \to \{0, 1\}$ that *isolates* a row in $\mathbf{x}$, i.e., there exists i such that $p(x_i) = 1$ and $p(x_j) = 0$ for all $j \neq i$. We emphasize that it is rows in the original dataset $\mathbf{x}$ on which the predicate acts, not the output $\mathbf{y}$. In part, this is a byproduct of our desire to make no assumptions on the form of M's output. While it might make sense to apply a predicate to pseudonymized microdata, it is far from clear what it would mean for a synthetic dataset or for aggregate statistics. Observe that this choice also rules out predicates p that "isolate" rows by referring to their position in $\mathbf{x}$ (i.e., "the seventh row").

M prevents isolation if there doesn't exist an adversary $\mathcal{A}$ that isolates a row in $\mathbf{x}$ except with very small probability over the randomness of sampling $\mathbf{x} \leftarrow D^n$, the mechanism $\mathbf{y} \leftarrow M(\mathbf{x})$, and the adversary $\mathcal{A}(\mathbf{y})$. Unfortunately, this is impossible to achieve by any mechanism M. To wit, there is a *trivial adversary*—one that that doesn't look at $\mathbf{y}$ and denoted by $\mathsf{T}(\perp)$—that isolates a row with probability approximately 0.37. The adversary simply outputs p that matches a $1/n$ fraction of the distribution D. For example, for a dataset of size $n = 365$ random people selected at random from the United States population, $\mathsf{T}(\perp)$ may simply output $p =$ (born on March 15th). This predicate will isolate a row with probability $(1 - 1/365)^{364} \approx 37\%$. Isolation is hence not necessarily indicative of a failure to protect against singling out, as $\mathsf{T}(\perp)$ would succeed with $\approx 37\%$ probability (for any n) even if M does not output anything at all.[5]

A trivial adversary can give us a *baseline* against which to measure isolation success. But the baseline should not simply be 37% chance of success. Consider the earlier example of a dataset of 365 random Americans. What if an adversary output predicates like $p =$ (born on March 15th $\wedge$ vegan $\wedge$ speaks Dutch $\wedge$ concert pianist), and managed to isolate 10% of the time? Though 10% is much less than 37%, the predicate is extremely specific and unlikely to isolate a person by chance. We formalize this intuition by considering the baseline risk of isolation as a function of the *weight* of p, i.e., the chance that p matches a random row sampled from the distribution D. The baseline for predicates of weight $1/n$ is 37%, but the baseline for an extremely

[5] In Section 4.3.2 we show that $\mathsf{T}(\perp)$ need not know the distribution D to isolate with probability $\approx 37\%$ if D has sufficient min-entropy.

specific predicate may be much lower. The more specific the predicate, the closer the baseline gets to zero. Our primary focus in this chapter is on the regime of predicate weights where the baseline is negligible, corresponding to predicates with negligible weight.[6] We get:

Definition 4.3.6 (informal) An adversary *predicate singles out* a row in $\mathbf{x}$ if it outputs a predicate that isolates a row with probability significantly higher than the baseline risk. A mechanism M is *secure against predicate singling out* (*PSO secure*) if no adversary can use its output to predicate single out.

4.1.2.2 Analyzing Security Against Predicate Singling Out

Having formulated security against singling out, our next goal is to understand the guarantee it offers, what mechanisms satisfy it, and how this concept relates to existing privacy concepts, including differential privacy and k-anonymity.

Two desirable properties of a privacy concept are robustness to *post-processing* and to *composition*. The former requires that if a mechanism M is deemed secure, then anything that can be computed using the outcome of M should also be deemed secure. Hence, the outcome may be reused without creating additional privacy risk. For instance, if a PSO secure mechanism M outputs microdata, then any statistics that can be computed from that microdata should also be PSO secure. It follows directly from the definition of PSO security that it is robust to post-processing.

We would like that the privacy risk of multiple data releases is not significantly greater than the accumulated risks of the individual releases. In this case, we say that the privacy concept composes. We prove that PSO security does not compose, and give two examples of this failure. First, we show that releasing aggregate statistics is PSO secure but fails to compose super-logarithmically many times. A collection of $\omega(\log(n))$ counts may allow an adversary to isolate a row with probability arbitrarily close to one using a predicate with negligible weight (and negligible baseline). Second, we construct less natural pair of mechanisms that individually are PSO secure but

[6]For completeness, we also consider in Section 4.3 predicates of weight $\omega(\log n/n)$, where the baseline is also negligible.

together allow the recovery of a row in the dataset. The first mechanism extracts and outputs a secret encryption key from one part of $\mathbf{x}$. The second extracts the same key and uses it to encrypt the last row $x_n \in \mathbf{x}$, outputting the corresponding ciphertext. The mechanisms individually prevent predicate singling out, but together completely fail.

Next, we ask whether existing privacy concepts guarantee PSO security. We already know that differential privacy is not necessary for PSO security as exact counts are PSO secure but not differentially private. However, differential privacy does provide PSO security. The proof relies on the connection between differential privacy and statistical generalization guarantees [DFH$^+$15, BNS$^+$16]. We show that predicate singling out implies a form of overfitting to the underlying dataset. If M is differentially private it prevents this form of overfitting, and hence protects against predicate singling out.

Finally, we examine k-anonymity [SS98] and show that it does not prevent predicate singling out attacks. Instead, it enables an adversary to predicate single out with probability approximately 37% using extremely low-weight predicates for which the baseline risk is negligible. Briefly, the attack begins by observing that typical k-anonymous algorithms "almost" predicate single out. They reveal predicates—usually, collections of attributes—that are satisfied by only k rows in the dataset. In an effort to make the k-anonymized data as useful as possible, these predicates are as descriptive and specific as possible. To predicate single out a row from the dataset of size n using the k-anonymous output, it roughly suffices to predicate single out a row from any grouping of k rows in the output.

4.1.3 Implications for the GDPR

Precisely formalizing predicate singling out attacks allows us to examine with mathematical rigor the extent to which specific algorithms and paradigms protect against them. In particular, we show that k-anonymity fails to prevent predicate singling out, but that differential privacy prevents predicate singling out. Our conclusions contrast with those of the Article 29 Working Party: they conclude that k-anonymity eliminates the risk of singling out while differential privacy "may not" [A29b]. These disagreements may raise a doubt about whether our modeling indeed matches the regulators' intent.

Our goal in interpreting the text of the GDPR and related documents, and in defining predicate singling out, is to provide a precise mathematical formalism to capture some aspect of the concept of personal data (as elucidated in the regulation and in [A29a]) and the associated concept of anonymization. We want to render mathematically *falsifiable* a legal claim that a given algorithmic technique anonymizes personal data by providing a *necessary* condition for such anonymizers.

We argue that predicate singling out succeeds. A number of modeling choices limit the scope of our definition, but limiting the scope poses no issue. Specifically, (i) we only consider randomly sampled datasets; (ii) we only consider an attacker who has no additional knowledge of the dataset besides the output of a mechanism; (iii) we do not require that isolation be impossible, instead comparing to a baseline risk of isolation. A technique that purports to anonymize all personal data against all attackers must at least do so against randomly sampled data and against limited attackers. And unless the idea of anonymization mechanisms is completely vacuous, one must compare against a baseline risk.

We must be careful not when narrowing our definition's scope, but when expanding it. The most significant expansion[7] is our choice to parameterize the baseline risk by the weight of a predicate. But this is a minimal expansion and only done to prevent a severe weakness. Not doing so would mean that a mechanism that published the first row of the dataset 20% of the time could be said to "prevent singling out." Any meaningful instantiation of "preventing singling out" should rule out such mechanisms. Ours is a natural way of doing so.

This does not mean that our modeling is the only one possible. As the starting point for the analysis is a description which does not use mathematical formalism, but is rather a (somewhat incomplete) description using natural language. It is certainly plausible that alternative mathematical formalizations of singling out could be

[7]We discuss additional subtleties in Section 4.3.3.

extracted from the very same text. We are looking forward to seeing such formalizations emerge.

Finally, one may still claim that the assessments made in [A29b] should be taken as ground truth and that the Article 29 WP meant for any interpretation of singling out to be consistent with these assessments. That is, the protection provided by k-anonymity implicitly defines the meaning of singling out (partially or in full). We believe, however, that such a position would be hard to justify. To the best of our knowledge, the assessments made by the Article 29 WP were not substantiated by a mathematical analysis. Furthermore, we caution against defining privacy implicitly as the guarantee provided by particular techniques; this approach is doomed to fail. In particular, the choice of defining privacy as the result of applying practices such as suppression of directly identifying information has proved a problematic choice that unfortunately pervades current legal privacy standards.

Is predicate singling out a good privacy concept? A predicate singling out attack can be a stepping stone towards a greater harm, even in settings where isolation alone may not. It may enable linking a person's record in the dataset to some external source of information [NS08], or targeting of individuals for differential treatment. As such, it is meaningful as a mode of privacy failure, both in the GDPR context and otherwise.

And, while we believe that PSO security is relevant for the GDPR as a necessary property of techniques that anonymize personal data, we do not consider it a sufficiently protective privacy concept by itself. First, singling out is a specific mode of privacy failure. It is not clear that ruling out this failure mode is sufficient for privacy (in particular, two other failure modes are mentioned in [A29b]: linkage and inference). Second, our definition considers a setting where the underlying data is chosen i.i.d. from some (unknown) underlying distribution, an assumption that is not true in many real-life contexts. PSO security may not prevent singling out in such contexts. Lastly, we believe that self-composition is an essential property of any reasonable privacy definition. However, as we show in Section 4.4.3, security against singling out does not self compose.

4.2 Preliminaries

Notation. Let $X = \{0,1\}^d$ be the data universe. A database $\mathbf{x} = (x_1, \ldots, x_n)$ consists of n elements taken from X. We often consider databases where each entry x_i is chosen i.i.d. from a probability distribution $D \in \Delta(X)$ over X.

For the purposes of asymptotic analyses, we will use the number of rows $n \in \mathbb{N}$ in a dataset as the complexity parameter. Furthermore, the parameter $d = d(n)$ is a function of n, but we typically omit the dependence.[8]

For a predicate $p : X \to \{0,1\}$ we define $\mathsf{weight}_D(p) \triangleq \mathbb{E}_{x \sim D}[p(x)]$ and for a database $\mathbf{x} \in X^n$ we define $p(\mathbf{x}) \triangleq \frac{1}{n}\sum_{i=1}^{n} p(x_i)$. A mechanism M is a Turing Machine that takes as input a database $\mathbf{x} \in X^n$. A mechanism M may be randomized and/or interactive.

We use U_ℓ for a random variable sampled from the uniform distribution over $\{0,1\}^d$. A function $f(n)$ is negligible in n if $f(n) = n^{-\omega(1)}$; this is denoted $f(n) = \mathsf{negl}(n)$.

4.2.1 Preliminaries from Randomness Extraction

Definition 4.2.1 (Min-entropy, average min entropy [DORS08]). *Let Y_1, Y_2 be two random variables. The* min-entropy *of a Y_1 is*

$$H_\infty(Y_1) = -\log\left(\max_y \Pr[Y_1 = y]\right).$$

The average min-entropy[9] *of Y_1 given Y_2 is*

$$\widetilde{H}_\infty(Y_1 \mid Y_2) = -\log\left(\mathop{\mathbb{E}}_{Y_2}\left[\max_y \Pr[Y_1 = y \mid Y_2]\right]\right).$$

[8]More formally, we can consider an ensemble of data domains $\mathcal{X} = \{X_n = \{0,1\}^{d(n)}\}_{n \in \mathbb{N}}$ and an ensemble of distributions $\mathcal{D} = \{D_n\}_{n \in \mathbb{N}}$, where $D_n \in \Delta(X_n)$.

[9]In [Smi09] this same quantity is called conditional min-entropy and denoted H_∞.

Fact 4.2.2. *For all Y_1 and Y_2: $H_\infty(Y_1) \geq \widetilde{H}_\infty(Y_1 \mid Y_2) \geq H_\infty(Y_1) - \log(|\mathsf{supp}(Y_2)|)$, where $\mathsf{supp}(Y_2)$ is the support of Y_2.*

Definition 4.2.3 (2-universal hash functions). *$H = \{h : \{0,1\}^d \to \{0,1\}^m\}$ is a 2-universal family of hash functions if $\Pr_{h \sim H}[h(x) = h(x')] = 2^{-m}$ for all $x, x' \in \{0,1\}^d$ where the probability is over the selection of h uniformly at random from H.*

As an example, for $a, b \in \{0,1\}^d$ let $h_{a,b}(x)$ be the function that returns the first m bits of $ax + b$ where the arithmetic is in the field $GF(2^d)$. Then $H = \{h_{a,b} : a, b \in \{0,1\}^d\}$ is 2-universal.

Definition 4.2.4 (Statistical distance). *The statistical distance of random variables Y_1, Y_2 with support $\{0,1\}^d$ is $\mathsf{SD}(Y_1, Y_2) = \frac{1}{2}\sum_{y \in \{0,1\}^d} \left|\Pr[Y_1 = y] - \Pr[Y_2 = y]\right|$. If $\mathsf{SD}(Y_1, Y_2) < \alpha$ we say that Y_1 and Y_2 are α-close.*

Lemma 4.2.5 (Generalized Leftover Hash Lemma [DORS08]). *Let $\lambda \in \mathbb{N}$, $\alpha > 0$, Y_1 a random variable over $\{0,1\}^d$, and Y_2 a random variable. Let $H = \{h : \{0,1\}^d \to \{0,1\}^m\}$ be a 2-universal family of hash functions where $m \leq \lambda - 2\log(1/\alpha^2) + 2$. For every Y_1 and Y_2 with $\widetilde{H}_\infty(Y_1 \mid Y_2) \geq \lambda$, $(h, h(Y_1), Y_2)$ is α^2-close to (h, U_m, Y_2) in total variation distance, where $h \in_R H$ is a uniformly random function from the hash family and U_m is uniform over $\{0,1\}^m$.*

Corollary 4.2.6. *$h(Y)$ is α-close to uniform with probability at least $1 - \alpha$ over $h \in_R H$.*

Proof. Let $H_{>\alpha} = \{h \in H : h(Y) \text{ is not } \alpha \text{ close to uniform}\}$. We have $\alpha^2 \geq \Delta((h, h(Y)), \text{unif}) > \Pr[h \in H_{>\alpha}] \cdot \alpha$. Hence $\Pr[h \in H_{>\alpha}] < \alpha$. □

4.3 Security Against Predicate Singling Out (PSO security)

We consider a setting in which a data controller has in its possession a dataset $\mathbf{x} = (x_1, \ldots, x_n)$ consisting of n rows sampled i.i.d. from a distribution $D \in \Delta(X)$.

The data controller publishes the output of an *anonymization mechanism* M applied to the dataset $\mathbf{x}$. A predicate singling out (PSO) adversary A is a non-uniform Turing machine with access to the mechanism $M(\mathbf{x})$ and produces a predicate $p : X \to \{0,1\}$.[10] We abuse notation and write $\mathcal{A}(M(\mathbf{x}))$, regardless whether M is an interactive or non-interactive mechanism. For now, we assume all adversaries have complete knowledge of D and are computationally unbounded; we reexamine these choices in Section 4.3.3 below.

Intuitively, the adversary's goal is to output predicate p that isolates a row in $\mathbf{x}$, where we associate the Article 29 WP Opinion on Anonymisation Techniques notion of "isolat[ing] some or all records which identify an individual in [a] dataset" with the production of a description that matches exactly one row in the dataset. Mathematically, the description would be in form of a predicate mapping data universe elements into $\{0,1\}$.

Definition 4.3.1 (Row isolation). *A predicate* p isolates a row in $\mathbf{x}$ *if there exists a unique* $x \in \mathbf{x}$ *such that* $p(x) = 1$. *I.e., if* $p(\mathbf{x}) = 1/n$. *We denote this event* $\mathsf{iso}(p, \mathbf{x})$.

It is tempting to require that a mechanism M only allow a negligible probability of isolating a row, but this intuition is problematic. An adversary that does not have access to M—a *trivial* adversary—can output a predicate p with $\mathsf{weight}_D(p) \approx 1/n$ and hence isolate a row in $\mathbf{x}$ with probability $\binom{n}{1} \cdot \mathsf{weight}_D(p) \cdot (1 - \mathsf{weight}_D(p))^{n-1} \approx e^{-1} \approx 37\%$. In Section 4.3.2 we will see that in many cases the trivial adversary need not know the distribution to produce such a predicate.

Instead of considering the absolute probability that an adversary outputs a predicate that isolates a row, we consider the increase in probability relative to a *baseline risk*: the probability of isolation by a trivial adversary.

Definition 4.3.2 (Trivial Adversary). *A predicate singling out adversary* T *is* trivial *if the distribution over outputs of* T *is independent of* $M(\mathbf{x})$. *That is* $\mathsf{T}(M(\mathbf{x})) = \mathsf{T}(\perp)$.

[10] As is typical in cryptography, strengthening the adversary to be non-uniform (including possibly having full knowledge of the distribution D) yields stronger security definition. See Section 4.3.3 for further discussion.

An unrestricted trivial adversary can isolate a row with probability about $1/e$. Towards a more expressive notion of the baseline risk, we restrict adversaries to output a predicate from a particular class of *admissible* predicates $P \subseteq \{p : X \to \{0,1\}\}$, i.e., a subset of predicates on X.[11]

Definition 4.3.3 (Adversarial success probability). *Let D be a distribution over X. For mechanism M, an adversary $\mathcal{A}$, a set of admissible predicates P, and $n \in \mathbb{N}$, let*

$$\mathsf{Succ}_P^{\mathcal{A},M}(n,D) \triangleq \Pr_{\substack{\mathbf{x}\leftarrow D^n \\ p\leftarrow \mathcal{A}(M(\mathbf{x}))}} [\mathsf{iso}(p,\mathbf{x}) \ \wedge \ p \in P].$$

Definition 4.3.4 (Baseline). *For $n \in \mathbb{N}$ and set of admissible predicates P,*

$$\mathsf{base}_D(n,P) \triangleq \sup_{\textit{Trivial } \mathsf{T}} \mathsf{Succ}_P^{\mathsf{T},\perp}(n,D)$$

We typically omit the parameter D when the distribution is clear from context. In this chapter, we focus on two classes of admissible predicates parameterized by the *weight* of the predicate p.

Definition 4.3.5 (Predicate families low and high). *For $0 \le w_{\mathsf{low}}(n) \le 1/n \le w_{\mathsf{high}}(n) \le 1$ we define the predicate families*

$$\mathsf{low} = \{p : \mathsf{weight}_D(p) \le w_{\mathsf{low}}(n)\} \quad \textit{and} \quad \mathsf{high} = \{p : \mathsf{weight}_D(p) \ge w_{\mathsf{high}}(n)\}.$$

We will consider the success probability of adversaries restricted to these admissible predicates and will denote them $\mathsf{Succ}_{\le w_{\mathsf{low}}}^{\mathcal{A},M}$ and $\mathsf{Succ}_{\ge w_{\mathsf{high}}}^{\mathcal{A},M}$ as shown in Figure 4-1.

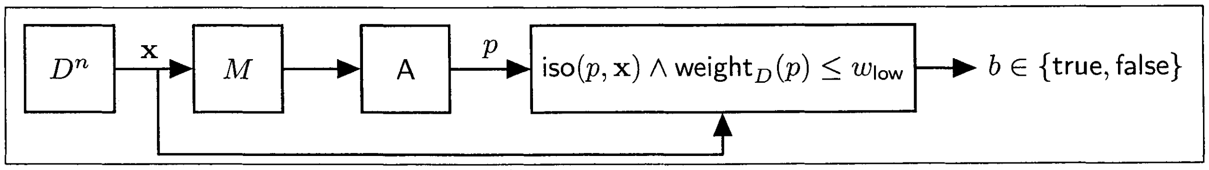

D^n
$\mathbf{x}$
M
A
p
$\mathsf{iso}(p,\mathbf{x}) \wedge \mathsf{weight}_D(p) \le w_{\mathsf{low}}$
$b \in \{\mathsf{true},\mathsf{false}\}$

Figure 4-1: $\mathsf{Succ}_{\le w_{\mathsf{low}}}^{\mathsf{A},M}(n,D) = \Pr_{D,M,\mathsf{A}}[b = \mathsf{true}]$.

[11]More formally, we restrict the adversary to an ensemble of admissible predicates $\mathcal{P} = \{P_n\}_{n\in\mathbb{N}}$, where $P_n \subseteq \{p : X_n \to \{0,1\}\}$, a subset of predicates on X_n.

4.3.1 Security Against Predicate Singling Out

We now have the tools for presenting our definition of security against singling out. We require that no adversary should have significantly higher probability of isolating a row than that of a trivial adversary, conditioned on both outputting predicates from the same class of admissible predicates.

Definition 4.3.6 (Security against predicate singling out). *For $\varepsilon(n) > 0$, $\delta(n) > 0$, $0 \leq w_{\mathsf{low}}(n) \leq 1/n \leq w_{\mathsf{high}}(n) \leq 1$, we say a mechanism M is $(\varepsilon, \delta, w_{\mathsf{low}}, w_{\mathsf{high}})$* secure against predicate singling out *($(\varepsilon, \delta, w_{\mathsf{low}}, w_{\mathsf{high}})$*-PSO secure*) if for all adversaries $\mathcal{A}$ and distributions D:*

$$\begin{aligned} \mathsf{Succ}^{\mathcal{A},M}_{\leq w_{\mathsf{low}}}(n, D) &\leq e^{\varepsilon(n)} \cdot \mathsf{base}_D(n, \mathsf{low}) + \delta(n), \\ \mathsf{Succ}^{\mathcal{A},M}_{\geq w_{\mathsf{high}}}(n, D) &\leq e^{\varepsilon(n)} \cdot \mathsf{base}_D(n, \mathsf{high}) + \delta(n). \end{aligned} \tag{4.1}$$

We often omit explicit reference to the parameter n for ε, δ, w_{low}, and w_{high}.

We say a mechanism is secure against predicate singling out *(*PSO secure*) if for all $w_{\mathsf{low}} = \mathsf{negl}(n)$, $w_{\mathsf{high}} = \omega(\frac{\log n}{n})$ there exists $\delta = \mathsf{negl}(n)$ such that M is $(0, \delta, w_{\mathsf{low}}, w_{\mathsf{high}})$-PSO secure.*

The definition is strengthened as ε and δ get smaller, and as w_{low} and w_{high} get closer to $1/n$. As shown below, when $w_{\mathsf{low}} = \mathsf{negl}(n)$ the baseline is negligible. This is probably the most important regime of Definition 4.3.6 as such predicates are likely to not only isolate a row in the database but also an individual in the entire population. The baseline is also negligible when $w_{\mathsf{high}} = \omega(\log n/n)$. It is not clear to the authors how beneficial finding a predicate in this regime may be to an attacker. The reader may decide to ignore Equation 4.1 in Definition 4.3.6 (as is depicted in Figure 4-1). We include the high weight regime in our analysis so as not to overlook potential singling out risks which rely on high weight predicates.

We also define a strong notion of predicate singling out, where an adversary can simultaneously isolate all rows of a dataset.

Definition 4.3.7 (Fully Predicate Singling Out). *An adversary $\mathcal{A}$* fully singles out *against a mechanism M and distribution D if (with high probability) it outputs a collection of n negligible-weight predicates p_i, each of which isolates a different row of the input dataset $\mathbf{x}$. More formally, if*

$$\Pr_{\substack{\mathbf{x}\leftarrow D^n \\ (p_1,\ldots,p_n)\leftarrow\mathcal{A}(M(\mathbf{x}))}}[\forall p_i, p_j : \mathsf{iso}(p_i,\mathbf{x}) \wedge \mathsf{weight}_D(p_i) = \mathsf{negl}(n) \wedge (p_i \wedge p_j)(\mathbf{x}) = 0] > 1-\mathsf{negl}(n) \tag{4.2}$$

Examples. On input $(x_1,\ldots,x_n)$ the mechanism M_f outputs $(f(x_1),\ldots,f(x_n))$ for some possibly randomized function f. Whether M_f prevents predicate singling out depends on f. On one extreme, if $f(x) = x$ and $|X| \gg n$, then M_f provides no protection. On the other extreme, if $f(x)$ is completely random, $M_f(\mathbf{x})$ contains no information about $\mathbf{x}$ and provides no benefit to the adversary. More formally, for all $\mathbf{x}$ the output of $M_f(\mathbf{x})$ is uniform; this allows us to construct a trivial adversary T that perfectly simulates any adversary $\mathcal{A}$.[12]

If f is invertible, then it offers no more protection than the identity function. However, f being many-to-one does not give an assurance. For instance, suppose the data is uniform over $\{0,1\}^n$ and $f : \{0,1\}^n \to \{0,1\}^{n/2}$ outputs the last $n/2$ bits of an input x. M_f is *not* secure. Indeed, it allows fully predicate singling out. For any $y_i = f(x_i)$ in the output, the adversary can output the predicate $p_i : (x) \mapsto \mathbb{I}(f(x) = y_i)$. $\Pr[\mathsf{iso}(p_i,\mathbf{x})] = 1 - \mathsf{negl}(n)$ and $\mathsf{weight}_{U_n}(p_i) = 2^{-n/2} = \mathsf{negl}(n)$.

4.3.2 Bounding the Baseline

In this section, we characterize the baseline over intervals in terms of a simple function $B(n,w)$. For $n \geq 2$ and a predicate p of weight w, the probability over $\mathbf{x} \sim D^n$ that p isolates a row in $\mathbf{x}$ is

$$B(n,w) \triangleq n \cdot w \cdot (1-w)^{n-1}$$

[12]Uniformity without conditioning on $\mathbf{x}$ may not be enough. For example, if the data itself is uniform, then the output of the identity function is also uniform. See also footnote 16.

$B(n, w)$ is maximized at $w = 1/n$ and strictly decreases moving away from the maximum. It is helpful to recall that $(1 - 1/n)^n \approx e^{-1}$ even for relatively small values of n. $(1 - 1/n)^{n-1}$ also approaches e^{-1} as $n \to \infty$, and does so from above.

As made formal in Claim 4.3.8, a trivial adversary maximizes its success of isolating a row by outputting a predicate p with $\mathsf{weight}_D(p)$ as close as possible to $1/n$ (the weight that maximizes $B(n, w)$). The set of possible values for $\mathsf{weight}_D(p)$ depends not only on w_{low} and w_{high}, but also on the distribution. We say that a weight $w \in [0, 1]$ is *realizable* under distribution D if there exists p such that $\mathsf{weight}_D(p) = w$. The baseline is characterized by $B(n, w)$.

Claim 4.3.8. *For every $n > 0$, w_{low}, w_{high} and D,*

$$\mathsf{base}_D(n, \mathsf{low}_n) = B(n, w^*_{\mathsf{low}}(n)) \quad \textit{and} \quad \mathsf{base}_D(n, \mathsf{high}_n) = B(n, w^*_{\mathsf{high}}(n)),$$

where

$$w^*_{\mathsf{low}}(n) = \sup\{w \le w_{\mathsf{low}}(n) : \textit{ realizable}\} \quad \textit{and} \quad w^*_{\mathsf{high}}(n) = \inf\{w \ge w_{\mathsf{high}}(n) : \textit{ realizable}\}.$$

Because $B(n, w)$ increases as w approaches $1/n$, the baseline has a simple upperbound.

Corollary 4.3.9. *For every w_{low}, w_{high}, $n \in \mathbb{N}$ and distribution D,*

$$\mathsf{base}_D(n, \mathsf{low}_n) \le B(n, w_{\mathsf{low}}(n)) \quad \textit{and} \quad \mathsf{base}_D(n, \mathsf{high}_n) \le B(n, w_{\mathsf{high}}(n)).$$

Proof of Claim 4.3.8. For $w \in [0, 1]$, let $P_w = \{p : \mathsf{weight}_D(p) = w\}$. First, we show that for all w, $\mathsf{base}(n, P_w) \le B(n, w)$. For any fixed predicate p,

$$\Pr_{\mathrm{x} \sim D^n}[\mathsf{iso}(p, \mathrm{x})] = \binom{n}{1} \cdot \mathsf{weight}_D(p) \cdot (1 - \mathsf{weight}_D(p))^{n-1} = B(n, \mathsf{weight}_D(p)).$$

For a trivial adversary T, let $\alpha_w(\mathsf{T}) = \Pr_{\mathsf{T}(\perp)}[\mathsf{weight}_D(p) = w]$.

$$\mathsf{base}(n, P_w) = B(n, w) \cdot \sup_{\text{Trivial } \mathsf{T}} \alpha_w(\mathsf{T}) \tag{4.3}$$

If w is realizable under D, then there exists a deterministic trivial adversary T with $\alpha_w(\mathsf{T}) = 1$; otherwise $\alpha_w(\mathsf{T}) = 0$ for all T.

For $W \subseteq [0,1]$, let $P_W = \{p : \mathsf{weight}_D(p) \in W\}$. By definition, $\mathsf{base}(n, P_W) \geq \sup_{w \in W} \mathsf{base}(n, P_w)$. Next, we show that in fact $\mathsf{base}(n, P_W) = \sup_{w \in W} \mathsf{base}(n, P_w)$. Suppose, towards contradiction, that $\mathsf{base}(n, P_W) > \sup_{w \in W} \mathsf{base}(n, P_w)$. Then there exists trivial T with $\mathsf{Succ}^{\mathsf{T},\perp}_{P_W}(n) > \sup_{w \in W} \mathsf{base}(n, P_w)$. There must also exist an deterministic trivial adversary T' with $\mathsf{Succ}^{\mathsf{T}',\perp}_{P_W}(n) \geq \mathsf{Succ}^{\mathsf{T},\perp}_{P_W}(n)$ but which always outputs predicates of a single weight $w' \in W$, a contradiction.

Combining with (4.3): for any W,

$$\mathsf{base}(n, P_W) = \sup_{\substack{w \in W \\ \text{realizable}}} B(n, w).$$

Because $B(n, w)$ monotonically increases as $w \to 1/n$,

$$\sup_{\substack{w \leq w_{\mathsf{low}} \\ \text{realizable}}} B(n, w) = B(n, w^*_{\mathsf{low}})$$

$$\sup_{\substack{w \geq w_{\mathsf{high}} \\ \text{realizable}}} B(n, w) = B(n, w^*_{\mathsf{high}}) \qquad \square$$

The dependence on the realizability of weights under D makes the exact baseline unwieldy. For example, the difference between the true baseline and the upper bound can be as large as $1/e$. Thankfully, the $B(n, w)$ upper bound is nearly tight when the underlying distribution has moderate min-entropy. Moreover, the corresponding lower bound is achievable by an efficient uniform trivial adversary who is oblivious of the distribution (see Section 4.3.3).

Claim 4.3.10 (Baseline Lower Bound). *Let $c > 0$ and $0 \leq w_{\mathsf{low}}(n) \leq 1/n \leq w_{\mathsf{high}}(n) \leq 1$. If D has min-entropy at least $\lambda > 5(c + \log n + 2)$, then $\mathsf{base}_D(n, \mathsf{low}_n) \geq B(n, w_{\mathsf{low}}(n)) - 2^{-c}$ and $\mathsf{base}_D(n, \mathsf{high}_n) \geq B(n, w_{\mathsf{high}}(n)) - 2^{-c}$.*

Proof of Claim 4.3.10. We prove the claim for $w_{\mathsf{low}}(n)$; the proof for $w_{\mathsf{high}}(n)$ is analogous. Let $m \geq c + \log n + 2$. Either $w_{\mathsf{low}}(n) \leq 2^{-(c+\log n)}$ or $w_{\mathsf{low}}(n) \geq 2^{-(m-1)}$. If $w_{\mathsf{low}}(n) \leq 2^{-(c+\log n)}$, then $B(n, w_{\mathsf{low}}) \leq n w_{\mathsf{low}} \leq 2^{-c}$, making the claim trivial. It remains to consider $w_{\mathsf{low}}(n) \geq 2^{-(m-1)}$.

Let P_H be the family of predicates from Lemma 4.3.11 and T_H be a trivial adversary that outputs a random $p_h \in_R P_H$. Recall that for any predicate p, $\Pr_{\mathbf{x} \sim D^n}[\mathsf{iso}(p, \mathbf{x})] = B(n, \mathsf{weight}_D(p))$. By Lemma 4.3.11

$$\begin{aligned}\mathsf{Succ}_{\mathsf{low}}^{\mathsf{T}_H,\perp}(n) &\geq \Pr_{\mathbf{x}\sim D^n, h\in_R H}[\mathsf{iso}(p_h, \mathbf{x}) \ \wedge \ \mathsf{weight}_D(p_h) \in W_{\mathsf{low}}] \\ &= \Pr_{h \in_R H}[\mathsf{weight}_D(p_h) \in W_{\mathsf{low}}] \cdot \Pr_{\mathbf{x}\sim D^n, h\in_R H}[\mathsf{iso}(p_h, \mathbf{x}) \mid \mathsf{weight}_D(p_h) \in W_{\mathsf{low}}] \\ &\geq (1 - 2^{-m}) \cdot B(n, 3 \cdot 2^{-m})\end{aligned}$$

Observing that $\left|\frac{dB}{dw}(w)\right| \leq \frac{dB}{dw}(0) = n$, $\mathsf{Succ}_{\mathsf{low}}^{\mathsf{T}_H,\perp}(n) \geq B(n, w_{\mathsf{low}}(n)) - 3 \cdot 2^{-m} n - 2^{-m} \geq B(n, w_{\mathsf{low}}(n)) - 2^{-(m - \log n - 2)} \geq B(n, w_{\mathsf{low}}(n)) - 2^{-c}$. □

Informally, the assumption that D has min-entropy λ implies two useful facts. First, the set of realizable weights is dense: for any w, there exists a realizable w' such that $|w - w'|$ is small. Second, the Leftover Hash Lemma allows us to construct an efficient uniform adversary who can find a predicate with weight w' without any knowledge of the distribution. The following lemma captures these properties:

Lemma 4.3.11. *For $m \in \mathbb{N}$ and a set X, let $H = \{h : X \to \{0,1\}^m\}$ be 2-universal family of hash functions. For any $w \geq 2^{-(m-1)}$ (respectively, $w \leq 1 - 2^{-(m-1)}$), there exists a collection of predicates $P_H = \{p_h\}_{h \in H}$ such that for all distributions D over X with min-entropy at least $\lambda = 5m$, $\mathsf{weight}_D(p_h) \in [w - 3 \cdot 2^{-m}, w]$ with probability at least $1 - 2^{-m}$ over $h \in_R H$. (respectively, $\mathsf{weight}_D(p_h) \in [w, w + 3 \cdot 2^{-m}]$).*

Proof of Lemma 4.3.11. We prove the lemma for $w \geq 2^{-(m-1)}$; the proof for $w \leq 1 - 2^{-(m-1)}$ is analogous. Identify the set $\{0,1\}^m$ with the set $\{0, 1, \dots, 2^m - 1\}$ in the natural way. For $y \in \{0,1\}^m$, define the function $r(y) \triangleq \frac{y}{2^m - 1}$, the projection of y onto the interval $[0,1]$. Let $0 \leq \Delta \leq w$ be some constant to be chosen later, and let w_m be the greatest multiple of 2^{-m} less or equal to $w - \Delta$.

$$\Pr_{y \in_R \{0,1\}^m}[r(y) \leq w - \Delta] = \Pr_{y \in_R \{0,1\}^m}[r(y) \leq w_m]$$

$$= w_m + 2^{-m}$$
$$\in [w - \Delta, w - \Delta + 2^{-m}] \tag{4.4}$$

Let $H = \{h : X \to \{0,1\}^m\}$ be 2-universal family of hash functions. For each $h \in H$ we define the predicate p_h:

$$p_h(x) = \begin{cases} 1 & r(h(x)) \leq w - \Delta \\ 0 & r(h(x)) > w - \Delta \end{cases}.$$

By the Leftover Hash Lemma, for every $\alpha > 0$ to be chosen later and every $\lambda \geq m + 2\log(1/\alpha^2)$, if D has min-entropy at least λ then

$$\big(h, h(x)\big)_{\substack{h \in_R H \\ x \sim D}}$$

is α^2-close to the uniform distribution over $H \times X_n$ in total variation distance. By Corollary 4.2.6 $h(D)$ is α-close to uniform over X with probability at least $1 - \alpha$ over $h \in_R H$. For such h, by (4.4),

$$\mathsf{weight}_D(p_h) = \Pr_{x \leftarrow D}[p_h(x) \leq w - \Delta] \in \big[w - \Delta - \alpha,\ w - \Delta + 2^{-m} + \alpha\big].$$

Set $\alpha = 2^{-m}$ and $\Delta = 2\alpha$. Then $\mathsf{weight}_D(p_h) \in [w - 3 \cdot 2^{-m}, w]$ with probability at least $1 - \alpha = 1 - 2^{-m}$ whenever $\lambda \geq m + 2\log(1/\alpha^2) = 5m$, completing the proof. □

Remark 4.3.12. The proof of Claim 4.3.10 requires only that is possible to sample a predicate such that $\mathsf{weight}_D(p_h) \in W_{\mathsf{low}}$. If we switch the order of quantifiers in the claim by allowing the trivial adversary to depend on the distribution D, then the proof (and thus the trivial adversary) can be derandomized. Indeed, for any D with sufficient min-entropy, there are many p_h that can be used. This observation is used in the proof of Theorem 4.4.6.

4.3.3 Reflections on Modelling Assumptions

In many ways, Definition 4.3.6 requires a very high level of protection, similar to what is standard in the foundations of cryptography. The definition requires a mechanism to provide security for all distributions D and against non-uniform, computationally unbounded adversaries.[13] The main weakness in the required protection is that it considers only data that is i.i.d., whereas real-life data cannot generally be modeled as i.i.d.

Any mechanism that purports to be a universal anonymizer of data under the GDPR—by transforming personal data into non-personal data—must prevent singling out. Our definition is intended to capture a necessary condition for a mechanism to be considered as rendering data sufficiently anonymized under the GDPR. Any mechanism that prevents singling out in all cases must prevent it in the special case that the data is i.i.d. from a distribution D. We view a failure to provide security against predicate singling out (Definition 4.3.6) or is fully predicate singling out (Definition 4.3.7) as strong evidence that a mechanism does not provide security against singling out; hence, it does not protect from identification, as per the analysis in Section 4.1.1.

On the other hand, satisfying Definition 4.3.6 is *not sufficient* for arguing that a mechanism renders data sufficiently anonymized under the GDPR. Singling out is only one of the many "means reasonably likely to be used" to identify a person in a data release.[14] Furthermore, the definition considers only i.i.d. data; satisfying it may not even be sufficient to conclude that a mechanism prevents singling out in all relevant circumstances.

[13] It is reasonable to limit the adversary in Definition 4.3.6 to polynomial time. If we restricted our attention to distributions with moderate min-entropy, our results would remain qualitatively the same: our trivial adversaries and lower bounds are all based on efficient and uniform algorithms; our upper bounds are against unbounded adversaries. Relatedly, restricting to min-entropy distributions would allow us to switch the order of quantifiers of D and T in the definition of the baseline without affecting our qualitative results.

[14] Article 29 Working Party Opinion on Anonymisation techniques [A29b] enumerates three criterions for identification: singling out, linkage, and inference.

4.4 Properties of PSO security

Two desirable properties of privacy concepts are that (i) immunity to post-processing, i.e., further processing of the outcome of a mechanism, without access to the data, should not increase privacy risks, and (ii) closure under composition, i.e., a combination of two or more mechanisms which satisfy the requirements of the privacy concept is a mechanism that also satisfies the requirements (potentially, with worse parameters). Differential privacy is an example of a privacy concept that is immune to post-processing and is closed under composition.

In this section we prove that PSO security withstands post-processing but not composition. We give two demonstrations for the latter. In the first we consider mechanisms which count the number of dataset rows satisfying a property. We show that releasing a count satisfies Definifion 4.3.6. However, there exists a collection of $\omega(\log(n))$ counts which allows an adversary to isolate a row with probability arbitrarily close to one using a predicate with negligible weight. For the second demonstration, we construct a (less natural) pair of mechanisms that individually satisfy Definifion 4.3.6 but together allow the recovery of a row in the dataset. This latter construction borrows ideas from [NSS+18]. An immediate conclusion is that PSO security is distinct from differential privacy. More importantly, not being closed under composition is a significant weakness of the notion of PSO security. Our constructions rely on very simple mechanisms that would likely be deemed secure against singling out under other formulations of the concept. It may well be that non-closure under composition is inherent for singling out.

From a legal or policy point of view, we believe that a privacy concept which is not closed under composition (or not immune to post-processing) should not be accepted as sufficient. Pragmatically, the fact that PSO security is not closed under composition suggests that this concept can be used for *disqualifying* privacy technology (if they are not PSO secure) but also that this concept must be combined with other requirements if it used for approving technology.

4.4.1 Post-processing

For any non-interactive mechanism M, let F be a (possibly non-uniform) algorithm taking inputs of the form $M(\mathbf{x})$. Let $F \circ M$ be the mechanism that on input $\mathbf{x}$ returns $F(M(\mathbf{x}))$.

Lemma 4.4.1 (Post-processing). *If M is $(\varepsilon, \delta, w_{\mathsf{low}}, w_{\mathsf{high}})$-PSO secure, then $F \circ M$ is too.*

Proof. We show something stronger: for all M, F, $\mathcal{A}$ there exists $\mathcal{A}_F$ such that for all n, P, D: $\mathsf{Succ}_P^{\mathcal{A}_F,M}(n) = \mathsf{Succ}_P^{\mathcal{A},F\circ M}(n)$. On input $M(\mathbf{x})$, $\mathcal{A}_F$ simulates $\mathcal{A}$ on input $F(M(\mathbf{x}))$ and returns the resulting predicate p. The distribution of $\mathcal{A}_F$'s output with mechanism M is identical to that of $\mathcal{A}$ with mechanism $F \circ M$, proving the lemma. $\square$

The definition and proof above extend to the case where the mechanism M is interactive.

4.4.2 Example PSO-secure mechanisms

This section presents two PSO-secure mechanisms. These examples are useful for developing intuition for the PSO security notion. Additionally, they are the foundation for the examples of self-composition failures in the next section.

4.4.2.1 Counting Mechanism

For any predicate $q : X \to \{0,1\}$, we define the corresponding Counting Mechanism:

```
Mechanism: Counting Mechanism $M_{\#q}$
  input: x

  return $|\{1 \le i \le n : q(x_i) = 1\}|$
```

For example, consider the least-significant bit predicate lsb, that takes as input a string $x \in \{0,1\}^*$ and outputs $x[1]$. The corresponding Counting Mechanism $M_{\#\mathsf{lsb}}$ returns the sum of the first column of $\mathbf{x}$.

The security of the Counting Mechanism is a corollary of the following proposition.

Proposition 4.4.2. *For all $\mathcal{A}$, P, $M : X^n \mapsto Y$: $\mathsf{Succ}_P^{\mathcal{A},M}(n) \leq |Y| \cdot \mathsf{base}(n, P)$, where Y is the codomain of M.*

Proof. We define a *trivial adversary* T such that for all $\mathcal{A}$, $\mathsf{Succ}_P^{\mathsf{T},\perp}(n) \geq \frac{1}{|Y|} \cdot \mathsf{Succ}_P^{\mathcal{A},M}(n)$. The proposition follows by definition of $\mathsf{base}(n, P)$. T samples a random $y \in_R Y$ and returns $p \leftarrow \mathcal{A}(y)$.

$$\mathsf{Succ}_P^{\mathsf{T},\perp}(n) = \Pr_{\substack{\mathbf{x} \leftarrow D^n \\ y \in_R Y \\ p \leftarrow \mathcal{A}(y)}} [\mathsf{iso}(p, \mathbf{x}) \ \wedge \ p \in P] \geq \frac{\mathsf{Succ}_P^{\mathcal{A},M}}{|Y|}$$

The inequality follows from the fact that for all databases $\mathbf{x}$, there exists $y^* = y^*(\mathbf{x}) \in Y$ such that

$$\Pr_{p \leftarrow \mathcal{A}(y^*)} [\mathsf{iso}(p, \mathbf{x}) \ \wedge \ p \in P] \geq \Pr_{p \leftarrow \mathcal{A}(M(\mathbf{x}))} [\mathsf{iso}(p, \mathbf{x}) \ \wedge \ p \in P],$$

and that for all $\mathbf{x}$, $\Pr_{y \in_R Y}[y = y^*] \geq \frac{1}{|Y|}$. □

Corollary 4.4.3. *$M_{\#q}$ PSO secure.*

As exact counts are not differentially private, this corollary demonstrates that differential privacy is not necessary for PSO security.

4.4.2.2 Predicate Mechanism

For any predicate $q : X \to \{0, 1\}$, we define the corresponding Predicate Mechanism:

```
Mechanism: Predicate Mechanism $M_q$
    input: $\mathbf{x}$

    return $(q(x_1), q(x_2), \ldots, q(x_n))$
```

Theorem 4.4.4. *M_q is PSO secure.*

We prove the security of M_q by showing that its output is "no more helpful" to the PSO adversary than the counts returned by $M_{\#q}$.

Proposition 4.4.5 (Permutation Proposition). *For a permutation $\sigma : [n] \to [n]$ of n elements and a dataset $\mathbf{x} = (x_1, \ldots, x_n)$, define $\sigma(\mathbf{x}) = (x_{\sigma(1)}, x_{\sigma(2)}, \ldots, x_{\sigma(n)})$. For any mechanism M, let $M \circ \sigma$ be the mechanism that on input $\mathbf{x}$ returns $M(\sigma(\mathbf{x}))$. For all $\mathcal{A}$, P, D, and σ: $\mathsf{Succ}_P^{\mathcal{A},M}(n) = \mathsf{Succ}_P^{\mathcal{A},M\circ\sigma}(n)$.*

Proof. For all σ, the distributions D^n and $\sigma(D^n)$ are identical. For all p and $\mathbf{x}$, $\mathsf{iso}(p, \mathbf{x})$ if and only if $\mathsf{iso}(p, \sigma(\mathbf{x}))$. Using these two observations:

$$\begin{aligned}
\mathsf{Succ}_P^{\mathcal{A},M}(n) &= \Pr_{\substack{\mathbf{x}\leftarrow D^n \\ p\leftarrow \mathcal{A}(M(\mathbf{x}))}} [\mathsf{iso}(p, \mathbf{x}) \ \wedge \ p \in P] \\
&= \Pr_{\substack{\mathbf{x}\leftarrow D^n \\ p\leftarrow \mathcal{A}(M\circ\sigma(\mathbf{x}))}} [\mathsf{iso}(p, \sigma(\mathbf{x})) \ \wedge \ p \in P] \\
&= \Pr_{\substack{\mathbf{x}\leftarrow D^n \\ p\leftarrow \mathcal{A}(M\circ\sigma(\mathbf{x}))}} [\mathsf{iso}(p, \mathbf{x}) \ \wedge \ p \in P] \\
&= \mathsf{Succ}_P^{\mathcal{A},M\circ\sigma}(n)
\end{aligned}$$

□

Proof of Theorem 4.4.4. Consider M_1 that on input $\mathbf{x}$ samples a random permutation σ and returns $M_q \circ \sigma(\mathbf{x})$. By the Permutation Proposition, $\mathsf{Succ}_P^{\mathcal{A},M_1}(n) = \mathsf{Succ}_P^{\mathcal{A},M_q}(n)$. Next, consider the randomized algorithm F that on input $m \in [n]$ outputs a uniformly random bitstring $y \in \{0,1\}^n$ of Hamming weight m. By post-processing and the security of $M_{\#q}$, the mechanism $M_2 = F \circ M_{\#q}$ is PSO secure.

M_1 and M_2 are the same mechanism: on every input $\mathbf{x}$, the output distributions are identical. Therefore M_q is PSO secure. □

4.4.3 Failure to Compose

4.4.3.1 Failure to Compose $\omega(\log n)$ Times

The security of a single count (Corollary 4.4.3) easily extends to $O(\log n)$-many counts (even adaptively chosen), as the size of the codomain grows polynomially. However, our next theorem states that a fixed set of $\omega(\log(n))$ counts suffices to predicate single out with probability close to e^{-1} (which can be amplified to $1 - \mathsf{negl}(n)$).

Theorem 4.4.6. *For a collection of predicates* $Q = (q_0, \dots, q_m)$*, let* $M_{\#Q}(\mathbf{x}) \triangleq (M_{\#q_0}(\mathbf{x}), \dots, M_{\#q_m}(\mathbf{x}))$*. Let* $X = \{0,1\}^m$ *and* $D = U_m$ *the uniform distribution over* X*. There exists* Q *and an adversary* $\mathcal{A}$ *such that*

$$\mathsf{Succ}_{\leq 2^{-m}}^{\mathcal{A}, M_{\#Q}}(n) \geq B(n, 1/n) - \mathsf{negl}(n).$$

Choosing $m = \omega(\log(n))$ yields $2^{-m} = \mathsf{negl}(n)$.

Proof. Let q_0 be any predicate such that $\mathsf{weight}_{U_m}(q_0) \leq 1/n$ and $\Pr_{\mathbf{x} \leftarrow U_m^n}[\mathsf{iso}(q_0, \mathbf{x})] \geq B(n, 1/n) - \mathsf{negl}(n)$. For instance, $q_0(x) = 1$ iff $x < 2^m/n$ (where in the last inequality we treat x as a number written in binary).[15]

For $i \in \{1, \dots, m\}$, define the predicate $q_i(x) \triangleq (q_0(x) \ \wedge \ x[i])$, and let $y_i = M_{\#q_i}(\mathbf{x})$. Consider the deterministic adversary $\mathcal{A}$ that on input $M_{\#Q}(\mathbf{x}) = (y_0, \dots, y_m)$ outputs the predicate

$$p(x) = q_0(x) \ \wedge \ \left(\bigwedge_{i=1}^{m} (x[i] = y_i) \right).$$

Observe that $\mathsf{iso}(q_0, \mathbf{x}) \implies \mathsf{iso}(p, \mathbf{x})$ and that by construction $\mathsf{weight}_{U_m}(p) = 2^{-m}$. Thus

$$\begin{aligned} \mathsf{Succ}_{\leq 2^{-m}}^{\mathcal{A}, M_{\#Q}}(n) &= \Pr_{\substack{\mathbf{x} \leftarrow U_m^n \\ p \leftarrow \mathcal{A}(M_{\#Q}(\mathbf{x}))}}[\mathsf{iso}(p, \mathbf{x})] \\ &\geq \Pr_{\substack{\mathbf{x} \leftarrow U_m^n \\ p \leftarrow \mathcal{A}(M_{\#Q}(\mathbf{x}))}}[\mathsf{iso}(q_0, \mathbf{x})] \\ &\geq B(n, 1/n) - \mathsf{negl}(n) \end{aligned}$$

□

Remark 4.4.7. When the attack succeeds, all the predicates q_i match 0 or 1 rows in $\mathbf{x}$. It may seem that an easy way to counter the attack is by masking low counts, a common measure taken e.g., in contingency tables. However, it is easy to modify the attack to only use predicates matching $\Theta(n)$ rows using one extra query. This means that restricting the mechanism to suppress low counts cannot prevent this type of

[15] Or use Claim 4.3.10 with $w_{\mathsf{low}}(n) = 1/n$, and Remark 4.3.12.

attack. Let q^* be a predicate with $\mathsf{weight}_{U_m}(q^*) = 1/2$ (e.g., parity of the bits), and let $q_i^* = q_i \vee q^*$. The attack succeeds whenever $q^*(\mathbf{x}) = q_0^*(\mathbf{x}) + 1$. If $q^*(x)$ and $q_0(x)$ are independent, then this occurs with probability at least $\frac{1}{2} \cdot B(n, 1/n) - \mathsf{negl}(n)$. As before, the probability can be amplified to $1 - \mathsf{negl}(n)$.

While a single count is PSO secure for *any* data distribution, the above attack against $\omega(\log(n))$ counts applies only to the uniform distribution U_m. Using the Leftover Hash Lemma, we can generically extend the attack to general distributions D with moderate min-entropy, at the cost of randomizing the attacked mechanism (i.e., set of counts). Informally, we hash the data to a smaller domain where its image will be almost uniformly distributed, and adapt the attack appropriately.

Given a 2-universal family of functions $H = \{h : X \to \{0,1\}^m\}$, a mechanism M and adversary $\mathcal{A}$, we construct a new randomized mechanism M_H and new adversary $\mathcal{A}_H$. The following lemma relates the success probability of the modified $\mathcal{A}_H$ with respect to D to that of $\mathcal{A}$ with respect to U_m.

```
Mechanism: $M_H : X \to Y$
  fixed: $H = \{h : X \to \{0,1\}^m\}$, and
         $M : \{0,1\}^m \to Y$
  input: $\mathbf{x} \in X$
  sample $h \in_R H$;
  return $(h, M(h(\mathbf{x})))$
```

Lemma 4.4.8. *For any $\mathcal{A}$ there exists $\mathcal{A}_H$ such that for all M, w_{low}, w_{high}, $\alpha > 0$ and $D \in \Delta(X)$ with min-entropy $\lambda > m + 2\log(1/\alpha^2)$:*

$$\left| \mathsf{Succ}^{\mathcal{A}_H, M_H}_{\leq w_{\mathsf{low}} + \alpha}(n, D) - \mathsf{Succ}^{\mathcal{A}, M}_{\leq w_{\mathsf{low}}}(n, U_m) \right| \leq n\alpha$$

where $\mathsf{Succ}(n, D)$ (respectively, $\mathsf{Succ}(n, U_m)$) denotes the PSO success probability with respect to the distribution D (respectively, U_m) as in Definition 4.3.3.

Proof. For a predicate p on $\{0,1\}^m$ we define a corresponding predicate on X: $p_h(x) \triangleq p(h(x))$. On input $(h, y) \leftarrow M_H(\mathbf{x})$, A_H simulates $p \leftarrow A(y)$ and outputs p_h.

We call $h \in H$ *good* if $h(D)$ is α-close to U_m. By Corollary 4.2.6, $(1-\alpha)$-fraction of h are good. By the goodness of h, if $\mathsf{weight}_{U_m}(p) \leq w_{\mathsf{low}}$ then $\mathsf{weight}_D(p_h) \leq w_{\mathsf{low}} + \alpha$.

$$\left|\mathsf{Succ}^{\mathcal{A}_H, M_H}_{\leq w_{\mathsf{low}}+\alpha}(n, D) - \mathsf{Succ}^{\mathcal{A},M}_{\leq w_{\mathsf{low}}}(n, U_m)\right| \leq \mathsf{SD}(U_m^n, h(D^n)) \leq n\alpha \quad \square$$

Corollary 4.4.9. *Let $\alpha = \mathsf{negl}(n)$ and $\lambda = m + 2\log(1/\alpha^2)$. For any $m = \omega(\log(n))$, there exists a distribution over $O(m)$-many predicates Q_h, a negligible function $w_{\mathsf{low}}(n)$, and an adversary $\mathcal{A}$ such that for all D with min-entropy at least λ:*

$$\mathsf{Succ}^{\mathcal{A}, M_{\#Q_h}}_{\leq w_{\mathsf{low}}}(n) \geq 1 - \mathsf{negl}(n).$$

Proof. The success probability in Theorem 4.4.6 is easily amplified from $1/e$ to $1 - \mathsf{negl}(n)$ by repetition. Applying Lemma 4.4.8 to the result almost completes the proof; it remains to verify that the resulting mechanism M_H can be written as $M_{\#Q_h}$ for some $Q_h = (q_0^h, \ldots, q_m^h)$. To do so, take $q_i^h(x) = q_i(h(x))$, where q_i is from the proof of Theorem 4.4.6. $\square$

Remark 4.4.10. Corollary 4.4.9 is only meaningful as an example of a failure of composition if each $M_{\#q_i^h}$ taken in isolation is PSO secure, something that is *not* provided by Lemma 4.4.8. However, $M_{\#q_i^h}$ is an instance of the counting mechanism and thus secure.

Theorem 4.4.6 can be extended to the predicate mechanism M_Q; this follows from the observation that $M_{\#Q}$ can be implemented by post-processing M_Q. But in fact a much stronger attack is possible.

Claim 4.4.11. *For a collection of predicates $Q = (q_1, \ldots, q_m)$, let $M_Q(\mathbf{x}) \triangleq (M_{q_1}(\mathbf{x}), \ldots, M_{q_m}(\mathbf{x}))$. Let $X = \{0,1\}^m$ and $D = U_m$ the uniform distribution over X. For $m = \omega(\log(n))$, there exists Q and an adversary $\mathcal{A}$ such that $\mathcal{A}$ fully predicate singles out against M_Q and D.*

Proof Outline. For $i \in [m]$, define the predicate $q_i(x) = x[i]$, the ith bit of x. Let $Q_{\mathsf{bits}} = (q_1, \ldots, q_m)$. For each row $j \in [n]$ and column $i \in [m]$, $M_{Q_{\mathsf{bits}}}(\mathbf{x})$ outputs the bit $x_i[j]$. The adversary outputs the collection of predicates $\{p_j\}_{j\in[n]}$ where

$$p_j(x) = \bigwedge_{i=1}^{m} (x[i] = x_j[i]). \quad \square$$

4.4.3.2 Failure to Compose Twice

Borrowing ideas from [NSS+18], we construct two mechanisms M_{ext} and M_{enc} which are individually secure against singling out (for arbitrary distributions), but which together allow an adversary to single out with high probability when the data is uniformly distributed over the universe $X = \{0,1\}^m$. With more work, the composition attack can be extended to more general universes and to distributions with sufficient min-entropy.

We divide the input dataset into three parts: a source of randomness $\mathbf{x}_{\mathsf{ext}} = (x_1, \ldots, x_{\frac{n}{2}})$, a message x_n, and a holdout set $\mathbf{x}_{\mathsf{hold}} = (x_{\frac{n}{2}+1}, \ldots, x_{n-1})$ used in the proof. $M_{\mathsf{ext}}(\mathbf{x})$ outputs an encryption secret key sk based on the rows in $\mathbf{x}_{\mathsf{ext}}$, using the von Neumann extractor.

```
Mechanism: $M_{\mathsf{ext}}$
  input: $\mathbf{x}$

  $\mathsf{sk} \leftarrow \emptyset$, the empty string;
  for $i \leftarrow 1$ to $\frac{n}{2}$ by 2 do
      if $\mathsf{lsb}(x_i) = 0 \wedge \mathsf{lsb}(x_{i+1}) = 1$ then
          $\mathsf{sk} \leftarrow s \| 0$
      if $\mathsf{lsb}(x_i) = 1 \wedge \mathsf{lsb}(x_{i+1}) = 0$ then
          $\mathsf{sk} \leftarrow s \| 1$
  end
  if $|\mathsf{sk}| \geq m$ then
      return $\mathsf{sk}[1:m]$, the first $m$ bits of $\mathsf{sk}$
  else
      return $\perp$
```

$M_{\mathsf{enc}}(\mathbf{x})$ runs $\mathsf{sk} \leftarrow M_{\mathsf{ext}}$. If $\mathsf{sk} \neq \perp$, it outputs $\mathsf{sk} \oplus x_n$ (using sk as a one-time pad to encrypt x_n); otherwise, it outputs $\perp$. Alone, neither sk nor $\mathsf{sk} \oplus x_n$ allows the adversary to single out, but using both an adversary can recover x_n and thereby single it out.

Theorem 4.4.12. *M_{ext} and M_{enc} are secure against predicate singling out (Definition 4.3.6). For $m = \omega(\log(n))$ and $m \leq n/8$, $X = \{0,1\}^m$, and $D = U_m$ the uniform distribution over X, there exists an adversary $\mathcal{A}$ such that*

$$\mathsf{Succ}^{\mathcal{A}, M_{\mathsf{ExtEnc}}}_{\leq 2^{-m}}(n) \geq 1 - \mathsf{negl}(n),$$

where $M_{\mathsf{ExtEnc}} = (M_{\mathsf{ext}}, M_{\mathsf{enc}})$.

Proof. Let $\mathbf{x} = (\mathbf{x}_{\mathsf{ext}}, \mathbf{x}_{\mathsf{hold}}, x_n)$ as described above.

Security of M_{ext}. This is a special case of the security of the predicate mechanism M_q (Theorem 4.4.4) and post-processing, with $q = \mathsf{lsb}$. [16]

In fact, M_{ext} is even $(\ln(2), 0, 1/n, 1/n)$-PSO secure. We provide a brief outline of the proof. Consider a related mechanism $M^{\top}_{\mathsf{ext}}$ that outputs $\top$ if $|\mathsf{sk}| \geq m$ and $\perp$ otherwise. By Proposition 4.4.2, $M^{\top}_{\mathsf{ext}}$ is $(\ln(2), 0, 1/n, 1/n)$-PSO secure. The security of M_{ext} can be reduced to that of $M^{\top}_{\mathsf{ext}}$ using a generalization of Proposition 4.4.5 to distributions of permutations. $\square$

Security of M_{enc}. For $\mathcal{A}$, $w_{\mathsf{low}}(n) < \mathsf{negl}(n)$, and $w_{\mathsf{high}}(n) = \omega(\log(n)/n)$, let

$$\gamma_{\mathsf{low}} = \mathsf{Succ}^{\mathcal{A}, M_{\mathsf{enc}}}_{\leq w_{\mathsf{low}}}(n) \quad \text{and} \quad \gamma_{\mathsf{high}} = \mathsf{Succ}^{\mathcal{A}, M_{\mathsf{enc}}}_{\geq w_{\mathsf{high}}}(n).$$

We must show that $\gamma_{\mathsf{low}}, \gamma_{\mathsf{high}} < \mathsf{negl}(n)$. It is easy to bound γ_{high} using the holdout set $\mathbf{x}_{\mathsf{hold}}$, which is independent of the output M_{enc}:

$$\gamma_{\mathsf{high}} \leq \Pr_{\mathbf{x}, M_{\mathsf{enc}}, \mathcal{A}}[p(\mathbf{x}_{\mathsf{hold}}) \leq 1 \mid \mathsf{weight}_D(p) \geq w_{\mathsf{high}}] = (1 - w_{\mathsf{high}})^{n-m-2} = o(1 - \log(n)/n)^{\Omega(n)}.$$

[16] The security of M_{ext} does not follow from the mere fact that its output is nearly uniform. For example, the mechanism that outputs x_1 may be uniform, but it trivially allows singling out. Security would follow if the output was nearly uniform *conditioned* on $\mathbf{x}$.

To bound γ_{low}, we consider the two possible values of $p(x_n)$. Write $\gamma_{\mathsf{low}} = \gamma_{\mathsf{low}}^0 + \gamma_{\mathsf{low}}^1$ where

$$\gamma_{\mathsf{low}}^b \triangleq \Pr\left[\mathsf{iso}(p, \mathbf{x}) \ \wedge \ \mathsf{weight}_D(p) \leq w_{\mathsf{low}} \ \wedge \ p(x_n) = b\right]$$

If $\mathcal{A}$ singles out and $p(x_n) = 1$, then the it must have gleaned information about x_n from the ciphertext $\mathsf{sk} \oplus x_n$, which should be impossible. The von Neumann extractor guarantees that either $\mathsf{sk} = \perp$ or sk is uniformly distributed in $\{0,1\}^{\frac{n}{2}}$. Either way, the output of $M_{\mathsf{enc}}(\mathbf{x})$ is information-theoretically independent of x_n. Therefore

$$\gamma_{\mathsf{low}}^1 \leq \Pr_{\mathbf{x}, M_{\mathsf{enc}}, \mathcal{A}}[p(x_n) = 1 \mid \mathsf{weight}_D(p) \leq w_{\mathsf{low}}] \leq w_{\mathsf{low}} = \mathsf{negl}(n).$$

If $\mathcal{A}$ singles out and $p(x_n) = 0$, then it is effectively singling out against the sub-dataset $\mathbf{x}_{-n} = (x_1, \dots, x_{n-1})$ That is

$$\begin{aligned}
\gamma_{\mathsf{low}}^0 &= \Pr_{\mathbf{x}, M_{\mathsf{enc}}, \mathcal{A}}[\mathsf{iso}(p, \mathbf{x}) \ \wedge \ \mathsf{weight}_D(p) \leq w_{\mathsf{low}} \ \wedge \ p(x_n) = 0] \\
&= \Pr_{\mathbf{x}, M_{\mathsf{enc}}, \mathcal{A}}[\mathsf{iso}(p, \mathbf{x}_{-n}) \ \wedge \ \mathsf{weight}_D(p) \leq w_{\mathsf{low}} \ \wedge \ p(x_n) = 0]
\end{aligned}$$

We construct B that tries to single out against mechanism M_{ext} using $\mathcal{A}$. We assume that B can sample from D.[17] On input sk, B samples $x'_n \sim D$ and runs $p \leftarrow \mathcal{A}(\mathsf{sk} \oplus x'_n)$.

$$\begin{aligned}
\mathsf{Succ}_{\leq w_{\mathsf{low}}}^{\mathsf{B}, M_{\mathsf{ext}}}(n) &\geq \Pr\left[\mathsf{iso}(p, \mathbf{x}_{-n}) \ \wedge \ \mathsf{weight}_D(p) \leq w_{\mathsf{low}} \ \wedge \ p(x'_n) = 0 \ \wedge \ p(x_n) = 0\right] \\
&\geq \Pr\left[\mathsf{iso}(p, \mathbf{x}_{-n}) \ \wedge \ \mathsf{weight}_D(p) \leq w_{\mathsf{low}} \ \wedge \ p(x'_n) = 0\right] \\
&\quad \cdot \Pr[p(x_n) = 0 \mid \mathsf{weight}_D(p) \leq w_{\mathsf{low}}] \\
&\geq \gamma_{\mathsf{low}}^0 \cdot (1 - w_{\mathsf{low}}) \\
&\geq \gamma_{\mathsf{low}}^0 \cdot (1 - \mathsf{negl}(n))
\end{aligned}$$

Therefore γ_{low}^0 is negligible.

[17] It is tempting to try to remove this assumption by picking x'_n arbitrarily, say $x'_n = 0^m$. Because sk is uniform, the ciphertexts $\mathsf{sk} \oplus x_n$ and $\mathsf{sk} \oplus x'_n$ are identically distributed and perfectly indistinguishable. This intuition is misleading (see also footnote 16).

$\square$

Insecurity of M_{ExtEnc} for $D = U_m$. The output of $M_{\mathsf{ExtEnc}}(\mathbf{x})$ is a pair $(\mathsf{sk}, \mathsf{ct})$. If $(\mathsf{sk}, \mathsf{ct}) = (\perp, \perp)$, $\mathcal{A}$ aborts. The for-loop in M_{ext} extracts $n/4$ uniform bits in expectation. By a Chernoff Bound, for $m \leq n/8$, $\Pr_{\mathbf{x}}[\mathsf{sk} = \perp] \leq e^{-n/16} = \mathsf{negl}(n)$.

If $(\mathsf{sk}, \mathsf{ct}) \neq (\perp, \perp)$, $\mathcal{A}$ recovers $x_n = \mathsf{ct} \oplus \mathsf{sk}$ and outputs the predicate

$$p(x) = \big(x = x_n\big).$$

By the choice of $m = \omega(\log(n))$, $\mathsf{weight}_{U_m}(p) = 2^{-m} < \mathsf{negl}(n)$. $\Pr[\mathsf{iso}(p, \mathbf{x}) \mid \mathsf{sk} \neq \perp] = 1 - \Pr[\exists j \neq n : x_j = x_n] = 1 - n \cdot 2^{-m} > 1 - \mathsf{negl}(n)$. The bound on $\mathsf{Succ}^{\mathcal{A}, M_{\mathsf{ExtEnc}}}_{\leq 2^{-m}}$ follows, completing the proof of the claim and the theorem. $\square$

$\square$

4.4.3.3 Singling Out and Failure to Compose

The failure to compose demonstrated in Section 4.4.3.1 capitalizes on the use of multiple counting queries. Such queries underlie a large variety of statistical analyses and machine learning algorithms. We expect that other attempts to formalize security against singling out would also allow counting queries. If so, our negative composition results may generalize beyond the notion of PSOsecurity.

The failure to compose demonstrated in Section 4.4.3.2 is more contrived. We expect that other attempts to formalize security against singling out would allow mechanisms like M_{ext}, where the output is uniform even conditioned on the input. It is less clear to us whether a mechanism like M_{Enc} would be allowed under other possible formalizations of security against singling out. If an alternate formalization is to compose, it likely must forbid M_{Enc}.

4.5 Differential Privacy, Generalization and PSO Security

4.5.1 Preliminaries from Differential Privacy

For $\mathbf{x}, \mathbf{x}' \in X^n$, we write $\mathbf{x} \sim \mathbf{x}'$ if the two datasets differ on exactly one element x_i.

Definition 4.5.1 (Differential Privacy [DMNS06, DKM+06]). *A randomized mechanism $M : X^n \to T$ is (ε, δ)-differentially private if for all $\mathbf{x} \sim \mathbf{x}' \in X^n$ and for all events $S \subseteq T$,*

$$\Pr[M(\mathbf{x}) \in S] \leq e^{\varepsilon} \Pr[M(\mathbf{x}') \in S] + \delta,$$

where the probability is taken over the randomness of the mechanism M.

Lemma 4.5.2 (Basic and Parallel Composition [DKM+06, McS09]). *Let M (ε, δ)-differentially private and M' (ε', δ')-differentially private. The mechanism $M' \circ M : \mathbf{x} \mapsto M'(M(\mathbf{x}), \mathbf{x})$ is $(\varepsilon + \varepsilon', \delta + \delta')$-differentially private. Let $(\mathbf{x}_1, \ldots, \mathbf{x}_\ell)$ be a partition of $\mathbf{x}$ into disjoint datasets. The mechanism $M^\ell : (\mathbf{x}_1, \ldots, \mathbf{x}_\ell) \mapsto (M(\mathbf{x}_1), \ldots, M(\mathbf{x}_\ell))$ is (ε, δ)-differentially private.*

Theorem 4.5.3 (Exponential Mechanism [MT07]). *For domain X^n and outcome space $\mathcal{R}$, let $u : X^n \times \mathcal{R} \to \mathbb{R}$ be a utility function. The sensitivity of u is $\Delta u = \max_{r \in \mathcal{R}} \max_{\mathbf{x} \sim \mathbf{x}'} |u(\mathbf{x}, r) - u(\mathbf{x}', r)|$. For a dataset $\mathbf{x}$, let $\mathrm{opt}_u(\mathbf{x}) = \max_{r \in \mathcal{R}} u(\mathbf{x}, r)$ and let $\mathcal{R}_{\mathrm{opt}} = \{r \in \mathcal{R} : u(\mathbf{x}, r) = \mathrm{opt}_u(\mathbf{x})\}$. For any $\varepsilon > 0$, there exists a mechanism $M^{\varepsilon}_{\mathsf{Exp}} : X^n \times \mathcal{R} \times u \to \mathcal{R}$ that is $(\varepsilon, 0)$-differentially private such that for all $\mathbf{x}$ and all $t > 0$:*

$$\Pr\left[u(M^{\varepsilon}_{\mathsf{Exp}}(\mathbf{x}, u, \mathcal{R}) \leq \mathrm{opt}_u(\mathbf{x}) - \frac{2\Delta u}{\varepsilon}\left(\ln\left(\frac{|\mathcal{R}|}{|\mathcal{R}_{\mathrm{opt}}|}\right) + t\right)\right] \leq e^{-t}.$$

Our analysis of how PSO security relates to differential privacy is through a connection of both concepts to statistical generalization. For differential privacy, this connection was established in [DFH+15, BNS+16]. We will also use a variant of the latter result from [NS15]:[18]

Lemma 4.5.4 (Generalization lemma). *Let $\mathcal{A} : (X^n)^\ell \to 2^X \times [\ell]$ be an (ε, δ)-differentially private algorithm that operates on ℓ sub-databases and outputs a predicate $p : X \to \{0,1\}$ and an index $i \in [\ell]$. Let $\mathbf{x} = (\mathbf{x}_1, \dots, \mathbf{x}_\ell)$ where every $\mathbf{x}_i \sim D^n$ is a database containing i.i.d. elements from D, and let $(p,i) \leftarrow \mathcal{A}(\mathbf{x})$. Then*

$$\mathop{\mathbb{E}}_{\mathbf{x}\sim(D^n)^\ell}\left[\mathop{\mathbb{E}}_{(p,i)\leftarrow\mathcal{A}(\mathbf{x})}[p(\mathbf{x}_i)]\right] \leq e^\varepsilon \cdot \mathop{\mathbb{E}}_{\mathbf{x}\sim(D^n)^\ell}\left[\mathop{\mathbb{E}}_{(p,i)\leftarrow\mathcal{A}(\mathbf{x})}[\mathsf{weight}_D(p)]\right] + \ell\delta \tag{4.5}$$

$$\mathop{\mathbb{E}}_{\mathbf{x}\sim(D^n)^\ell}\left[\mathop{\mathbb{E}}_{(p,i)\leftarrow\mathcal{A}(\mathbf{x})}[p(\mathbf{x}_i)]\right] \geq e^{-\varepsilon} \left(\mathop{\mathbb{E}}_{\mathbf{x}\sim(D^n)^\ell}\left[\mathop{\mathbb{E}}_{(p,i)\leftarrow\mathcal{A}(\mathbf{x})}[\mathsf{weight}_D(p)]\right] - \ell\delta\right). \tag{4.6}$$

4.5.2 Differential Privacy Implies PSO Security

Theorem 4.5.5. *For all $\varepsilon = O(1)$, $\delta = \mathsf{negl}(n)$, $w_{\mathsf{low}} \leq 1/n$, and $w_{\mathsf{high}}(n) = \omega(\log n/n)$, if M is (ε,δ)-differentially private, then M is $(\varepsilon', \delta', w_{\mathsf{low}}, w_{\mathsf{high}})$-PSO secure for*

$$\varepsilon' = \varepsilon + (n-1)\ln\left(\frac{1}{1 - w_{\mathsf{low}}}\right) \quad \textit{and} \quad \delta' = \mathsf{negl}(n).$$

For $w_{\mathsf{low}} = o(1/n)$, $\varepsilon' = \varepsilon + o(1)$.[19]

Proof. The theorem consists of Claims 4.5.6 and 4.5.7, each using one part of the generalization lemma. That lemma holds even when the distribution D is known, a fact used in both proofs.

Claim 4.5.6. *If M is (ε,δ)-d.p., then for all $\mathcal{A}$ and $w_{\mathsf{low}} \in [0, 1/n]$*

$$\mathsf{Succ}^{\mathcal{A},M}_{\leq w_{\mathsf{low}}}(n) \leq e^{\varepsilon'} \cdot \mathsf{base}(n, w_{\mathsf{low}}) + n\delta.$$

Claim 4.5.7. *For $\varepsilon = O(1)$ and $\delta = \mathsf{negl}(n)$, if M is (ε,δ)-d.p., then for all $\mathcal{A}$ and all $w_{\mathsf{high}} = \omega(\log n/n)$,*

$$\alpha \triangleq \mathsf{Succ}^{\mathcal{A},M}_{\geq w_{\mathsf{high}}}(n) \leq \mathsf{negl}(n).$$

[18]The proof of Equation 4.5 of Lemma 4.5.4 is identical to that of Lemma 3.3 in [NS15], skipping the last inequality in the proof. The proof of Equation 4.6 is analogous.

[19]For all $w_{\mathsf{low}} \leq 1/n$ and n, $\varepsilon' < \varepsilon + 1$ by the fact that $(1 - w_{\mathsf{low}})^{n-1} \geq (1 - 1/n)^{n-1} > e^{-1}$.

Proof of Claim 4.5.6. Let $w^* = \max\{w \leq w_{\mathsf{low}} : w \text{ realizeable under } D\}$. Given $p \leftarrow \mathcal{A}(M(\mathbf{x}))$, w_{low}, and D, define the predicate p^*:

$$p^*(x) \equiv \begin{cases} p(x) & \text{if } \mathsf{weight}_D(p) \leq w_{\mathsf{low}} \\ 0 & \text{if } \mathsf{weight}_D(p) > w_{\mathsf{low}} \end{cases}$$

Observe that $\mathsf{weight}_D(p^*) \leq w^*$. The predicate p^* can be computed from p, D, and w_{low} without further access to $\mathbf{x}$. Because differential privacy is closed under post-processing, if M is (ε, δ)-differentially private, then the computation that produces p^* is as well.

$$\begin{aligned} \mathsf{Succ}^{\mathcal{A},M}_{\leq w_{\mathsf{low}}}(n) &\leq \Pr_{\mathbf{x},p}[p(\mathbf{x}) \geq 1/n \ \wedge \ \mathsf{weight}_D(p) \leq w^*] \\ &\leq n \cdot \mathop{\mathbb{E}}_{\mathbf{x},p}[p^*(\mathbf{x})] \\ &\leq n \cdot (e^{\varepsilon} w^* + \delta) && \text{by Lemma 4.5.4, } \ell = 1 \\ &= e^{\varepsilon} \frac{\mathsf{base}(n, w^*)}{(1-w^*)^{n-1}} + n\delta && \text{by Claim 4.3.8} \\ &\leq e^{\varepsilon} \frac{\mathsf{base}(n, w^*)}{(1-w_{\mathsf{low}})^{n-1}} + n\delta \\ &= e^{\varepsilon'} \mathsf{base}(n, w_{\mathsf{low}}) + \delta' && \text{by Claim 4.3.8} \end{aligned}$$

□

Proof of Claim 4.5.7. Fix an adversary $\mathcal{A}$. The figure below defines algorithm B which will violate the Generalization Lemma for $\ell = O(\frac{\log n}{\alpha})$. M is (ε, δ)-differentially private, and $M^{\varepsilon}_{\mathsf{Exp}}$ (Theorem 4.5.3) is $(\varepsilon, 0)$-differentially private. By basic and paraelel composition, B is $(2\varepsilon, \delta)$-differentially private.

Define the event PSO to be the event that $\mathcal{A}$ successfully predicate singles out on one of the sub-databases with a high-weight predicate: $\mathsf{PSO} = \{\exists i \in [\ell] : \mathsf{iso}(p_i, \mathbf{x}_i) \wedge \mathsf{weight}_D(p_i) \geq w_{\mathsf{high}}\}$. By the choice of ℓ, $\Pr[\mathsf{PSO}] = 1 - (1-\alpha)^{\ell} \geq 1 - \frac{1}{n}$. Conditioned on PSO, $\max_{i \in I} u(i) \geq -1/n$. $\Delta u = 1/n$, and $|I| \leq \ell$. The Exponential Mechanism guarantees that

$$\Pr_{\mathbf{x};(p_{i^*}, i^*) \leftarrow \mathsf{B}(\mathbf{x})}\left[p_{i^*}(\mathbf{x}_{i^*}) \geq \frac{1}{n} + \frac{2}{n\varepsilon}(\ln \ell + t) \mid \mathsf{PSO}\right] \leq e^{-t}.$$

Choosing $t = \ln n$ and using the fact that $p_{i^*}(\mathbf{x}_{i^*}) \leq 1$,

$$\mathop{\mathbb{E}}_{\mathbf{x};(p_{i^*},i^*)\leftarrow \mathsf{B}(\mathbf{x})}[p_{i^*}(\mathbf{x}_{i^*}) \mid \mathsf{PSO}] \leq \frac{1}{n} + \frac{2}{n\varepsilon}(\ln \ell + \ln n) + \frac{1}{n}.$$

```
Input: $D$, $\mathbf{x} \sim (D^n)^\ell$
$I \leftarrow \emptyset$, the empty set;
for $i \leftarrow 1, \dots, \ell$ do
    $p_i \leftarrow \mathcal{A}(M(\mathbf{x}_i))$;
    $u_i = -p_i(\mathbf{x}_i)$;
    if $\mathsf{weight}_D(p_i) \geq w_{\mathsf{high}}$ then
        $I \leftarrow I \cup \{i\}$
end
Let $u : i \mapsto -p_i(\mathbf{x}_i)$ for $i \in I$;
$i^* \leftarrow M^{\varepsilon}_{\mathsf{Exp}}(\mathbf{x}, I, u)$;
return $(i^*, p_{i^*})$
```
Algorithm: B for Proof of Claim 4.5.7.

$$\begin{aligned}
&\mathop{\mathbb{E}}_{\mathbf{x};(p_{i^*},i^*)\leftarrow \mathsf{B}(\mathbf{x})}[p_{i^*}(\mathbf{x}_{i^*})] \\
&\qquad = \Pr[\neg\mathsf{PSO}]\,\mathbb{E}[p_{i^*}(\mathbf{x}_{i^*}) \mid \neg\mathsf{PSO}] + \Pr[\mathsf{PSO}]\,\mathbb{E}[p_{i^*}(\mathbf{x}_{i^*}) \mid \mathsf{PSO}] \\
&\qquad \leq \Pr[\neg\mathsf{PSO}] + \mathbb{E}[p_{i^*}(\mathbf{x}_{i^*}) \mid \mathsf{PSO}] \\
&\qquad < \frac{3}{n} + \frac{2}{n\varepsilon}(\ln \ell + \ln n).
\end{aligned}$$

$$\begin{aligned}
&\mathop{\mathbb{E}}_{\mathbf{x};(p_{i^*},i^*)\leftarrow \mathsf{B}(\mathbf{x})}[\mathsf{weight}_D(p_{i^*})] \\
&\qquad = \Pr[\neg\mathsf{PSO}]\,\mathbb{E}[\mathsf{weight}_D(p_{i^*}) \mid \neg\mathsf{PSO}] + \Pr[\mathsf{PSO}]\,\mathbb{E}[\mathsf{weight}_D(p_{i^*}) \mid \mathsf{PSO}] \\
&\qquad \geq \Pr[\mathsf{PSO}] \cdot \mathbb{E}[\mathsf{weight}_D(p_{i^*}) \mid \mathsf{PSO}] \\
&\qquad \geq (1 - \frac{1}{n}) \cdot w_{\mathsf{high}} \\
&\qquad > \frac{3w_{\mathsf{high}}}{4}
\end{aligned}$$

Applying the Lemma for the $(2\varepsilon, \delta)$-d.p. mechanism B,

$$\frac{3}{n} + \frac{2}{n\varepsilon}(\ln \ell + t) \geq e^{-2\varepsilon}\left(\frac{3w_{\mathsf{high}}}{4} - \ell\delta\right).$$

If $\delta = \omega(\frac{\alpha}{n})$, then by the assumption that δ is negligible, $\alpha = \mathsf{negl}(n)$. Otherwise $\delta = O(\frac{\alpha}{n}) = O(\frac{\log n}{n\ell})$ and

$$\frac{2}{\varepsilon}(\ln \ell + \ln n) \geq \frac{3nw_{\mathsf{high}} - O(\log n)}{4e^{2\varepsilon}}.$$

For $\varepsilon = O(1)$ and $w_{\mathsf{high}} = \omega(\frac{\log n}{n})$, $\ln \ell + \ln n = \omega(\log n)$. By the choice of $\ell = O(\frac{\log n}{\alpha})$, $\alpha = \mathsf{negl}(n)$. □

□

4.6 Does k-Anonymity Provide PSO Security?

k-anonymity [SS98, Swe02] is a strategy intended to help a data holder "release a version of its private data with scientific guarantees that the individuals who are the subjects of the data cannot be re-identified while the data remain practically useful" [Swe02]. It is achieved by making each individual in a data release indistinguishable from at least $k - 1$ individuals. Typically, a k-anonymized dataset is produced by subjecting it to a sequence of generalization and suppression operations.

The Article 29 Working Party Opinion on Anonymisation Techniques concludes that k-anonymity prevents singling out [A29b]. In this section, we analyze the extent to which k-anonymity provides PSO security. We show that k-anonymized dataset typically provides an attacker information which is sufficient to predicate singling out with constant probability. This result challenges the determination of the Article 29 Working Party.[20]

[20]Our results hold equally for ℓ-diversity [MKGV07] and t-closeness [LLV07a] which the Article 29 Working Party also concludes prevent singling out.

4.6.1 Preliminaries

Let $(A_1, \dots, A_m)$ be *attribute domains.* A dataset $\mathbf{x} = (x_1, \dots, x_n)$ is collection of rows $x_i = (a_{i,1}, \dots, a_{i,m})$ where $a_{i,j} \in A_j$. For subsets $\widehat{a}_{i,j} \subseteq A_j$, we view $y_i = (\widehat{a}_{i,1}, \dots, \widehat{a}_{i,m})$ as a set in the natural way, writing $x_i \in y_i$ if $\forall j \in [m]$, $a_{i,j} \in \widehat{a}_{i,j}$. We say that a dataset $\mathbf{y} = (y_1, \dots, y_n)$ is derived from $\mathbf{x}$ by generalization and suppression if $\forall i \in [n]$, $x_i \in y_i$. For example, if (A_1, A_2, A_3) correspond to "5 digit ZIP Code," "Gender," and "Year of Birth," then it may be that $x_i = (91015, F, 1972)$ and $y_i = (91010\text{–}91019, F, 1970\text{–}1975)$.

k-anonymity aims to capture a sort of anonymity of a crowd: a data release $\mathbf{y}$ is k-anonymous if any individual row in the release cannot be distinguished from $k-1$ other individuals. Let $\mathsf{count}(\mathbf{y}, y) \triangleq |\{i \in [n] : y_i = y\}|$ be the number of rows in $\mathbf{y}$ which agree with y.[21]

Definition 4.6.1 (k-Anonymity (rephrased from [Swe02])). *For $k \geq 2$, a dataset $\mathbf{y}$ is* k-anonymous *if* $\mathsf{count}(\mathbf{y}, y_i) \geq k$ *for all $i \in [n]$. An algorithm is called a* k-anonymizer *if on an input dataset $\mathbf{x}$ its output is a k-anonymous $\mathbf{y}$ which is derived from $\mathbf{x}$ by generalization and suppression.*

Our goal is to relate k-anonymity and PSO security. It will be convenient to define a generalization of k-anonymity—*predicate k-anonymity* which captures the core property of k-anonymity but relaxes its strict syntactic requirements.

For a predicate $\phi : X \to \{0,1\}$ and dataset $\mathbf{x}$, let $\mathbf{x}_\phi = \{x \in \mathbf{x} : \phi(x) = 1\}$. We assume that $|\mathbf{x}_\phi|$ is computable given the output of the k-anonymizer, but this does not qualitatively affect the results in this section.

Definition 4.6.2 (Predicate k-Anonymity). *Let* Anon *be an algorithm mapping a dataset $\mathbf{x} \in X^n$ to a collection of predicates $\Phi = \{\phi : X \to \{0,1\}\}$. For $k \geq 2$ we call* Anon predicate k-anonymous *if for all $\phi \in \Phi$, $|\mathbf{x}_\phi| \geq k$.*

k-anonymity is a special case of predicate k-anonymity that considers only specific

[21] Often count is paramaterized by a subset Q of the attribute domains called a *quasi-identifier.* This parameterization does not affect our analysis and we omit it for simplicity.

collections of predicates Φ induced by a dataset $\mathbf{y}$:

$$\Phi = \{\phi_y(x) = 1 \iff x \in y\}_{y \in \mathbf{y}}.^{22}$$

Definition 4.6.3. *A predicate k anonymizer is* k_{max}-bounded *if* $\forall \mathbf{x}, \exists \phi \in \Phi$ *such that* $|\mathbf{x}_\phi| \leq k_{\mathsf{max}}$.

4.6.2 Illustrative Examples

Before presenting a formal technical analysis, we provide two illustrative examples of very simple k-anonymizers that fail to provide security against predicate singling out. For both examples, let $D = U_\ell$ be the uniform distribution over $\{0,1\}^n$. The dataset $\mathbf{x}$ consists of n records sampled i.i.d. from D.

Bit suppression. This k-anonymizer processes groups of k rows in index order and suppresses all bit locations where the k rows disagree. Namely, for each group g of k rows $(x_{gk+1}, \ldots, x_{gk+k)})$ it outputs k copies of the string $y_g \in \{0,1,\star\}^n$ where $y_g[j] = b \in \{0,1\}$ if $x_{gk+1}[j] = \cdots = x_{gk+k}[j] = b$ (i.e., all the k rows in the group have b as their jth bit) and $y_g[j] = \star$ otherwise.

In the terminology of Definition 4.6.2, the predicate $\phi_g(x)$ evaluates to 1 if $y_g[j] \in \{x[j], \star\}$ for all $j \in [n]$ and evaluates to 0 otherwise. Namely, $\phi_g(x)$ checks whether x agrees with y_g (and hence with all of $x_{gk+1}, \ldots, x_{gk+k)})$ on all non-suppressed bits.

In expectation, $n/2^k$ positions of y_g are not suppressed. For large enough n, with high probability over the choice of $\mathbf{x}$, at least $\frac{n}{2 \cdot 2^k}$ positions in y_g are not suppressed. In this case, $\mathsf{weight}_D(\phi_g) \leq 2^{-\frac{n}{2 \cdot 2^k}}$ which is a negligible function of n for any constant k.

We now show how ϕ_g can be used adversarially. In expectation $n(1 - 2^{-k}) \geq 3n/4$ positions of y_g are suppressed. For large enough n, with high probability over the choice of $\mathbf{x}$ at least $n/2$ of the positions in y_g are suppressed. Denote these

[22] See also the definition of k-anonymity for face images [NSM05, Definition 2.10]. Using the notation of that paper, it is also special case of predicate k-anonymity, with $\Phi = \{\phi_{\Gamma_d}(\Gamma) = 1 \iff f(\Gamma) = \Gamma_d\}_{\Gamma_d \in H_d}$

positions $i_i, \ldots, i_{n/2}$. Define the predicate $p_k(x)$ that evaluates to 1 if the binary number resulting from concatenating $x[i_1], x[i_2], \ldots, x[i_{n/2}]$ is greater than $2^{n/2}/k$ and 0 otherwise. Note that $\mathsf{weight}_D(p_k) \approx 1/k$ and hence p_k isolates within the group g with probability $\approx 1/e \approx 0.37$, as was the case with the trivial adversary described at the beginning of Section 4.3.

An attacker observing ϕ_g can now define a predicate $p(x) = \phi_g(x) \wedge p_k(x)$. By the analysis above, $\mathsf{weight}(p)$ is negligible (as it is bounded by $\mathsf{weight}(\phi_g)$) and $p(x)$ isolates a row in $\mathbf{x}$ with probability ≈ 0.37. Hence, the k-anonymizer of this example fails to protect against singling out.

Theorem 4.6.4 below captures the intuition from our bit suppression example and generalizes it, hence demonstrating that k-anonymity would not typically protect against predicate singling out. We note that Theorem 4.6.4 does not capture all possible ways in which the outcome of a k-anonymizer can be exploited, in particular, the following simple example.

Interval Buckets. This k-anonymizer sorts the rows in lexicographic order and outputs the intervals $[a_g, b_g] = [x_{gk+1}, x_{gk+k}]$ (where the indices are after sorting and renaming). The corresponding predicate $\phi_{a_g,b_g}(x) = 1$ if $x \in [a_g, b_g]$.

Observe that any of the endpoints a_g or b_g reveal a row in $\mathbf{x}$ and hence an adversary can predicate single out with probability 1 using predicates of weight 2^{-n}.

4.6.3 k-Anonymity Enables Predicate Singling Out

Theorem 4.6.4. *For any $k_{\mathsf{max}} \geq 2$, there exists an (efficient, uniform, randomized) algorithm $\mathcal{A}$ such that for all D with min-entropy $\lambda \geq m + 2\log(1/\alpha^2) + k_{\mathsf{max}} \log n$ (for $m \in \mathbb{N}$, α), and all predicate anonymizers* Anon *that are k_{max}-bounded, and all $w_{\mathsf{low}} > 0$:*

$$\mathsf{Succ}^{\mathcal{A},\mathsf{Anon}}_{\leq w_{\mathsf{low}}}(n) \geq \eta \cdot (e^{-1} - 2^{-m}n - k\alpha^2)$$

where

$$\eta \triangleq \Pr_{\substack{\mathbf{x} \leftarrow D^n \\ \phi \leftarrow \mathsf{Anon}(\mathbf{x})}} [\mathsf{weight}_D(\phi) \leq w_{\mathsf{low}}(n)] .$$

For distributions with sufficient min-entropy ($m = \omega(\log n)$, $\alpha = \mathsf{negl}(n)$), the adversary's success probability is approximately $\eta/e \approx \eta \cdot B(k, 1/k)$. To predicate single out, the adversary must output a predicate that both isolates $\mathbf{x}$ and has low weight. The theorem shows that these two requirements essentially decompose: η is the probability that the predicate k-anonymizer outputs a low-weight predicate and $B(k, 1/k)$ is the probability that a trivial adversary predicate singles out a dataset of size k. Algorithms for k-anonymity generally try to preserve as much information in the dataset as possible. We expect such algorithms to typically yield low-weight predicates and correspondingly high values of η.

Proof of Theorem 4.6.4. On input $\Phi \leftarrow \mathsf{Anon}(\mathbf{x})$, $\mathcal{A}$ selects $\phi \in \Phi$ such that $2 \leq |\mathbf{x}_\phi| \leq k_{\mathsf{max}}$. $\mathcal{A}$ will construct some predicate q and output the conjunction $p \triangleq \phi \wedge q$. Noting that $\mathsf{weight}_D(p) \leq \mathsf{weight}_D(\phi)$, and that $\mathsf{iso}(q, \mathbf{x}_\phi) \implies \mathsf{iso}(p, \mathbf{x})$,

$$\begin{aligned}\mathsf{Succ}^{\mathcal{A},\mathsf{Anon}}_{\leq w_{\mathsf{low}}}(n) &\geq \Pr\left[\mathsf{iso}(q, \mathbf{x}_\phi) \ \wedge\ \mathsf{weight}_D(\phi) \leq w_{\mathsf{low}}\right] \\ &= \eta \cdot \Pr\left[\mathsf{iso}(q, \mathbf{x}_\phi) \,\middle|\, \mathsf{weight}_D(\phi) \leq w_{\mathsf{low}}\right] \end{aligned} \tag{4.7}$$

Claim 4.6.5. *There exists $\mathcal{A}$ such that for all $k_\phi \geq 2$*

$$\Pr_{\substack{\mathbf{x}\leftarrow D^n \\ \phi\leftarrow\mathsf{Anon}(\mathbf{x}) \\ p\leftarrow\mathcal{A}(\phi,w)}}\left[\mathsf{iso}(q, \mathbf{x}_\phi) \,\middle|\, |\mathbf{x}_\phi| = k_\phi \ \wedge\ \mathsf{weight}_D(\phi) \leq w_{\mathsf{low}}\right] \geq B(k_\phi, 1/k_\phi) - 2^{-m}n - k\alpha^2.$$

The claim is proved below. Using the claim we get:

$$\begin{aligned}&\Pr\left[\mathsf{iso}(q, \mathbf{x}_\phi) \,\middle|\, \mathsf{weight}_D(\phi) \leq w_{\mathsf{low}}\right] \\ &\quad= \sum_{k_\phi = k}^{k_{\mathsf{max}}} \Pr\left[|\mathbf{x}_\phi| = k_\phi\right] \cdot \Pr\left[\mathsf{iso}(q, \mathbf{x}_\phi) \,\middle|\, |\mathbf{x}_\phi| = k_\phi \ \wedge\ \mathsf{weight}_D(\phi) \leq w_{\mathsf{low}}\right] \\ &\quad\geq \sum_{k_\phi} \Pr\left[|\mathbf{x}_\phi| = k_\phi\right] \cdot \left(B(k_\phi, 1/k_\phi) - 2^{-m}n - k\alpha^2\right) \\ &\quad= \mathop{\mathbb{E}}_{k_\phi}\left[B(k_\phi, 1/k_\phi)\right] - 2^{-m}n - k\alpha^2 \\ &\quad\geq e^{-1} - 2^{-m}n - k\alpha^2 \end{aligned} \tag{4.8}$$

The last inequality follows from the fact that for all $k_\phi \geq 2$, $(1 - 1/k_\phi)^{k_\phi - 1} > e^{-1}$. Combining (4.8) with (4.7) completes the proof. □

The proof of Claim 4.6.5 uses the Leftover Hash Lemma in a manner closely resembling Lemma 4.3.11, but with an additional challenge. The earlier application of LHL proved that a random hash function selected appropriately isolates a row with probability close to e^{-1}. It relied on the fact that each row was sampled i.i.d. from a distribution with min-entropy. In contrast, the rows in $\mathbf{x}_\phi$ are a function of Anon and the whole dataset $\mathbf{x}$. They are not independently distributed and even their marginal distributions may be different than D.

We can use the LHL to prove the claim if we can show that the rows in $\mathbf{x}_\phi$ still have sufficient (conditional) min-entropy. The following lemma does exactly that.

Lemma 4.6.6. *Let $Y_1, \dots, Y_n$ be i.i.d. random variables and let F be a (randomized) function mapping $(Y_1, \dots, Y_n)$ to (j, I) where $j \in [n]$ and $I \subseteq [n] \setminus \{j\}$ of size $|I| = k - 1$. Let $Y_I = \{Y_i\}_{i \in I}$.*

$$\widetilde{H}_\infty(Y_j \mid Y_I) \geq H_\infty(Y_j) - (k-1)\log n \geq H_\infty(Y_1) - k \log n.$$

Proof of Lemma 4.6.6. We prove the second inequality first. The idea in used in (4.9) is used in (4.10).

$$\begin{aligned} 2^{-H_\infty(Y_j)} &= \max_y \Pr[Y_j = y] \\ &\leq \max_y \Pr[\exists \ell \in [n], Y_\ell = y] \qquad (4.9) \\ &\leq n \cdot \max_y \Pr[Y_1 = 1] \\ &= 2^{\log n - H_\infty(Y_j)} \end{aligned}$$

The first inequality:

$$2^{-\widetilde{H}_\infty(Y_j \mid Y_I)} = \mathop{\mathbb{E}}_{Y_I}\left[\max_y \Pr[Y_j = y \mid Y_I]\right]$$

$$
\begin{aligned}
&= \sum_{y_I} \Pr[Y_I = y_I] \cdot \left(\max_y \Pr[Y_j = y \mid Y_I = y_I] \right) \\
&= \sum_{y_I} \max_y \left(\Pr[Y_I = y_I] \cdot \Pr[Y_j = y \mid Y_I = y_I] \right) \\
&= \sum_{y_I} \max_y \left(\Pr[Y_j = y] \cdot \Pr[Y_I = y_I \mid Y_j = y] \right) \\
&\leq \sum_{y_I} \max_y \left(\Pr[Y_j = y] \cdot \Pr[\forall i \in I, \exists \ell \in [n] \setminus \{j\}, Y_\ell = y_i \mid Y_j = y] \right) \qquad (4.10) \\
&= \sum_{y_I} \max_y \left(\Pr[Y_j = y] \cdot \Pr[\forall i \in I, \exists \ell \in [n] \setminus \{j\}, Y_\ell = y_i] \right) \\
&= \sum_{y_I} \Pr[\forall i \in I, \exists \ell \in [n] \setminus \{j\}, Y_\ell = y_i] \cdot \max_y \Pr[Y_j = y] \\
&= \max_y \Pr[Y_j = y] \cdot \sum_{y_I} 1 \\
&\leq \binom{n}{k-1} \cdot 2^{-H_\infty(Y_j)} \\
&\leq 2^{(k-1)\log n - H_\infty(Y_j)}
\end{aligned}
$$ □

Proof of Claim 4.6.5. We us a corollary of the Leftover Hash Lemma.

Corollary 4.6.7 (Corollary to Leftover Hash Lemma (4.2.5)). *For random variables $Y_1, \ldots, Y_k$, and $j \in [n]$, let $Y_{-j} = \{Y_i : i \neq j\}$. If for all $j \in [n]$, $\widetilde{H}_\infty(Y_j \mid Y_{-j}) = \lambda \geq m + 2\log(1/\alpha^2)$, then $(h(Y_1), \ldots, h(Y_k))_{h \in_R H}$ is $k\alpha^2$-close to uniform over $(\{0,1\}^m)^k$ in total variation distance.*

The construction of q uses the Leftover Hash Lemma and is very similar to the construction of the predicates in Lemma 4.3.11. Identify the set $\{0,1\}^m$ with the set $\{0, 1, \ldots, 2^m - 1\}$ in the natural way. For $y \in \{0,1\}^m$, define the function $r(y) \triangleq \frac{y}{2^m - 1}$, the projection of y onto the interval $[0,1]$.

Let w_ϕ be the multiple of 2^{-m} closest to $1/k_\phi$. Observe that $|B(k_\phi, w_\phi) - B(k_\phi, 1/k_\phi)| \leq \left| w_\phi - \frac{1}{k_\phi} \right| \cdot \max_{w' \in [0,1]} \left| \frac{dB}{dw}(w') \right| \leq 2^{-m} n$.

Let $H = \{h : X \to \{0,1\}^m\}$ be a 2-universal family of hash functions. For each

$h \in H$ define the predicate q_h:

$$q_h(x) = \begin{cases} 1 & r(h(x)) < w_\phi \\ 0 & r(h(x)) \geq w_\phi \end{cases}.$$

Because w_ϕ is a multiple of 2^{-m},

$$\Pr_{y \in_R \{0,1\}^m}[r(y) < w_\phi] = w_\phi$$

By Lemma 4.6.6, $\mathbf{x}_\phi$ (viewed as a k_ϕ-tuple of random variables) satisfies the average min-entropy hypothesis of Corollary 4.6.7. Applying that Corollary:

$$\begin{aligned} &\Pr_{\mathbf{x}_\phi, q_h}\left[\mathsf{iso}(q, \mathbf{x}_\phi) \;\middle|\; |\mathbf{x}_\phi| = k_\phi \;\wedge\; \mathsf{weight}_D(\phi) \leq w_{\mathsf{low}}\right] \\ &\geq \Pr_{y_1, \ldots, y_{k_\phi} \in_R \{0,1\}^m}[\exists \text{ unique } j \in [k_\phi] : r(y_j) < w_\phi] - k_\phi \alpha^2 \\ &= B(k_\phi, w_\phi) - k_\phi \alpha^2 \\ &\geq B(k_\phi, 1/k_\phi) - 2^{-m}n - k_\phi \alpha^2. \end{aligned}$$

□

Chapter 5

Theoretical Attacks in Practice

5.1 Introduction

Responding to public and legislation pressures, companies are seeking practical and usable technological solutions to their data privacy problems. Larger corporations often employ Chief Privacy Officers and teams of privacy engineers to develop custom data privacy solutions. Some of these companies, including Google, Apple, and Uber, are recently experimenting with provable approaches to privacy using cryptography and differential privacy, which—at their current stage of development—require significant research and engineering efforts.

But not all companies have the means and technological sophistication to adopt this sort of bespoke approach to privacy. This void is being filled by a growing industry of companies selling off-the-shelf data privacy solutions, many of which aim to *anonymize* or *de-identify* sensitive data. These companies often advertise their anonymization products as not only preventing the disclosure of sensitive data and but also ensuring compliance with relevant privacy laws, including HIPAA, FERPA, and GDPR. Lack of transparency surrounds some of these technologies. Even when disclosed, the technical underpinnings of many privacy protection claims are heuristic and hence hard to evaluate.

Heuristic approaches to data privacy are not typically ruled out by data privacy

Based in part on "Linear Program Reconstruction in Practice" with Kobbi Nissim [CN18].

regulations. Furthermore, these regulations are not typically interpreted to require a strong level of protection from data privacy technology. For example, the EU's General Data Protection Regulation limits its scope to those "means reasonably likely to be used" to re-identify data [Eur16]. A report from the UK's Information Commissioner's Office interprets similar language in earlier legislation as requiring protection against "motivated intruders"; alas, these intruders are assumed to lack both "any prior knowledge" and "specialist expertise" [Off12]. This policy approach allows practitioners to argue that data privacy technology can be deployed even when they can be theoretically demonstrated to be vulnerable to attacks, as *purely theoretical* attacks plausibly fall outside the scope of the relevant regulations

We believe that this is an unhealthy state of affairs. However, one can hope to affect the legal interpretation of existing regulations by implementing theoretical attacks and demonstrating their practicality. As a striking example, the decision to use differential privacy for the 2020 Decennial Census in the US was largely motivated by the Census Bureau's realization that traditional statistical disclosure limitation techniques may be vulnerable to practical reconstruction attacks [Abo18].

Reasoning about privacy. Academics have developed paradigms to reason formally about certain aspects of data privacy. Among these is differential privacy: a privacy notion that has attracted significant attention since its introduction in 2006 by Dwork, McSherry, Nissim, and Smith [DMNS06]. A computation on a collection of data is differentially private if the outcome is essentially statistically independent of any individual datum.

Work leading to differential privacy demonstrated fundamental tradeoffs between privacy and utility when computing statistics on sensitive information. In 2003, Dinur and Nissim considered executing noisy statistical queries on a database $\mathbf{x}$ of n entries [DN03]. They showed that if the noise magnitude is $o(\sqrt{n})$, then there exists a simple linear program reconstructing all but a small fraction of $\mathbf{x}$, a result that was further generalized and strengthened in [DMT07, DY08]. This work helped establish robustness to composition as a fundamental privacy desideratum and provided useful guidance in the theoretical development of a rigorous approach to privacy.

Reconstruction and composition attacks in practice. In this chapter, we describe two attacks on data privacy implementations in the wild. Both systems were created by experts specifically motivated by legal compliance concerns. To the best of our knowledge, both attacks are the first of their kind to be carried out against systems deployed in the wild.

Our first attack is a linear reconstruction attack on a statistical query system, successfully reconstructing data from a commercially-available statistical database system. The attack was performed on the production system called Diffix [FEO+18] using a real dataset deployed in test environment in the course of a bug bounty program by Aircloak. The goal of Diffix is to allow data analysts to perform an unlimited number of statistical queries on a sensitive database while protecting the underlying data and while introducing only minimal error. It is being advertised as an off-the-shelf, GDPR-compliant privacy solution, and the company reports that "CNIL, the French national data protection authority, has already evaluated Diffix against the GDPR anonymity criteria, and have stated that Aircloak delivers GDPR-level anonymity" [AIR18b]. As we show, by answering unlimited, highly accurate statistical queries Diffix is vulnerable to linear reconstruction attacks.

Our second attack is against a highly publicized k-anonymous dataset consisting of online educational student records. While it was known that in principle k-anonymity does not compose, to the best of our knowledge ours is the first example of a real-world failure of composition.

5.2 Linear Reconstruction in Practice

5.2.1 Diffix

Diffix is a system that sits between a data analyst and a dataset. The data analyst issues counting queries using a restricted subset of SQL. For example,

```
SELECT count(*) FROM loans
WHERE status = 'C' AND client-id BETWEEN 2000 and 3000
```

Diffix executes a related query on the underlying dataset and computes the answer to the query along with some additional random error. The noisy answer is returned to the analyst.

A primary focus of Diffix's design is noise generation, which is a function of both the text of the query and the subset of the data included in that query. The noise is sampled from a zero-mean normal distribution and, depending on the data, rounded to the nearest integer. The standard deviation of the noise depends on the complexity of the query; each additional condition in the query introduces an additional layer noise of standard deviation 1.

In addition to adding Gaussian noise, Diffix employs a number of heuristic techniques to promote privacy. To protect against an attacker who may try to average noise out by issuing many logically equivalent but syntactically distinct queries, Diffix restricts the use of certain SQL operators, especially math operators. Other techniques include suppressing small counts, modify extreme values, and disallowing many SQL operators (including OR).

In order to accelerate development and testing of our linear reconstruction attack, we simulated Diffix's noise addition and small-count suppression in MATLAB. The results in Section 5.2.2 use real query responses from Diffix, while those in Section 5.2.3 use the simulation.

The Aircloak Challenge. From December 2017 to May 2018, Aircloak ran "the first bounty program for anonymized data re-identification," offering prizes of up to \$5,000 for successful attacks [AIR18a]. The company granted researchers access to five datasets through Diffix, along with documentation of the design and implementation of Diffix and complete versions of the datasets for analysis. Researchers were allowed to use auxiliary information gleaned directly from the datasets in order to carry out their attacks. We commend Aircloak for making Diffix available to privacy researchers and for their support throughout.

Aircloak measured the success of an attack using an effectiveness parameter α

and a confidence improvement parameter κ. They have verified our attack to achieve the best possible parameters. In a recent blog post, Aircloak reported that "Only two attack teams formulated successful attacks. ... Fixes for both attacks have been implemented" [AIR18c] At the time of writing, we have not examined the new restrictions on the query language introduced to by Aircloak to counter these attacks.

5.2.2 Implementing the Linear Reconstruction Attack

The attack targets a dataset $\mathbf{x}$ of size n database entries indexed by a set of unique identifiers ID. Each entry has an associated value of a Boolean target attribute, x_{id}. Each query $q \subseteq [n]$ specifies a subset of entries, and the response $a_q = q(\mathbf{x}) + e_q$ is the sum of true value $q(\mathbf{x}) = \sum_{\mathsf{id} \in q} x_{\mathsf{id}}$ and an error term e_q. The errors are sampled from a zero-mean Gaussian distribution of standard deviation σ, then rounded to the nearest integer. Each query q is a uniformly random subset of $[n]$. The set of all queries is denoted Q and is of size m.

We implemented a linear reconstruction attack following the approach of [DMT07] to find a candidate database $\mathbf{x}'$ minimizing the total error. [DMT07] was designed for a setting when some errors may be very significant, but typical errors are small. In contrast, the linear program of [DN03] is suitable when there is a bound on the maximum error magnitude. Although we use the linear program of [DMT07], we deviate by using subset queries. That work analyzes a number of other types of queries, including ± 1 queries of the form $q^{\pm}(\mathbf{x}) = \sum_{\mathsf{id} \in q} x_{\mathsf{id}} - \sum_{\mathsf{id} \notin q} x_{\mathsf{id}}$ for subsets $q \subseteq [n]$. While these queries can be implemented using subset queries,[1] the standard deviation of the resulting noise would be larger. In contrast, subset queries were directly implementable in Diffix with less noise and proved effective. Section 5.2.3 reports on additional experiments testing the accuracy of these three contrasting approaches in the face of Gaussian noise.

We solve the following linear program over $n + m$ variables $\mathbf{x}' = (x'_{\mathsf{id}})_{\mathsf{id} \in \mathsf{ID}}$ and $(e'_q)_{q \in Q}$:

[1] $q^{\pm}(\mathbf{x}) = q(\mathbf{x}) - q^c(\mathbf{x})$

variables: $\mathbf{x}' = (x'_{\mathsf{id}})_{\mathsf{id}\in\mathsf{ID}}$ and $(e'_q)_{q\in Q}$

minimize: $\sum_{q\in Q} |e'_q|$

subject to:

$$\forall q \in Q, \quad e'_q = a_q - q(\mathbf{x}')$$

$$\forall \mathsf{id} \in \mathsf{ID}, \quad 0 \le x'_{\mathsf{id}} \le 1$$

There is a standard linearizing of the above nonlinear objective function by introducing m additional variables. To compute the final output, we round the real-valued x'_{id} to the nearest value in $\{0,1\}$.

The results described in this section are from a reanalysis of data gathered during the Aircloak Challenge using the linear program described above. During the course of the actual challenge, we used a slightly modified linear program as described in Appendix 5.2.4.

Querying "random" subsets. The main hurdle in implementing the attack was specifying queries for random subsets of the rows of the dataset. Diffix determines the error magnitude per query depending on the description of the query. It increases the noise magnitude for each additional condition in the query string. Random queries would require lengthy description and Diffix would hence introduce large noise that would reduce the reconstruction accuracy. We needed to find a way to specify a random—or "random" enough—subset of the data using as few conditions as possible.

Our approach, ad hoc yet ultimately effective, was to use the unique user identifier id as the source of "randomness." For each "random" query we used a predicate p_q and let $q = \{\mathsf{id} : p_q(\mathsf{id}) = 1\}$. Concretely, each query was specified by a prime p, an offset j, an exponent $e \in \{0.5, 0.6, \ldots, 1.9\} \setminus \{1\}$, and a modulus $m \in \{2, 5\}$. Row number id was included in the query $q = (p, j, e, m)$ if the jth digit in the decimal representation of $(p \cdot \mathsf{id})^e$ was congruent to $0 \mod m$. For example, the following query corresponds to $p = 2$, $j = 2$, $e = 0.7$ and $m = 5$.

```
SELECT count(clientId) FROM loans
WHERE floor(100 * ((clientId * 2)^0.7) + 0.5)
    = floor(100 * ((clientId * 2)^0.7))
```

The exact form of the query depended on the various syntactic restrictions included in Diffix. By modifying the ranges of p and j, we were able to tune the total number of queries. We restricted p to the first 25 primes and $j \in [5]$, resulting in a total of 3500 queries.

Results. Our target was the loans table in the banking dataset, consisting of real data of 827 loans from a bank in the Czech Republic. The rows are indexed by the clientId attribute, a unique number between 2 and 13971. Each row has an associated loanStatus attribute, a letter from 'A' to 'D'.

Our goal was to determine which loans had loanStatus = 'C', given only knowledge of the clientIds. In order to minimize the total number of queries, we restricted our attention to the subset of clientIds in the range [2000, 3000], which contained 73 entries. Ultimately, our queries were of the form:

```
SELECT count(clientId) FROM loans
WHERE floor(100 * ((clientId * 2)^0.7) + 0.5)
    = floor(100 * ((clientId * 2)^0.7))
AND clientId BETWEEN 2000 and 3000
AND loanStatus = 'C'
```

Diffix added error of standard deviation 4 to the output of these queries. We applied the same attack on different ranges of clientIds with 110, 130, and 142 entries (and in the last case, targeting the loanStatus value 'A'). In each case, we performed 3500 queries.

The linear program reconstructed the data for all four clientId ranges perfectly.

5.2.3 Simulated Experiments

In addition to the above results using the actual Diffix system, we performed additional experiments using a simulation of a noisy statistical query mechanism. The simulated mechanism answers counting queries with zero-mean, normally-distributed noise with standard deviation σ (and rounds to the nearest integer). It also suppresses low counts in the same way as the Diffix system, though that was only be relevant for the first experiment. All experiments described below were implemented in MATLAB on a personal laptop, and all linear programs were solved in less than 4 seconds.

Removing auxiliary information. One drawback of our original attack on Diffix was the need for complete knowledge of the clientIds as a prerequisite to performing the attack. Our first experiment sought to infer these clientIds. First, we identified a range of 100 possible clientIds that had a large number of present clientIds (relative to the other possible ranges). We want a large number of present clientIds to minimize the effect of Diffix's low-count suppression. We settled on the range $[2500, 2600]$ with 12 clientIds. While we identified this range using exact counts, we believe such a range could be found by querying Diffix itself.[2]

We simulated responses to 3500 queries of the following form:

```
SELECT count(clientId) FROM loans
WHERE floor(100 * ((clientId * 2)^0.7) + 0.5)
    = floor(100 * ((clientId * 2)^0.7))
AND clientId BETWEEN 2500 and 2600
```

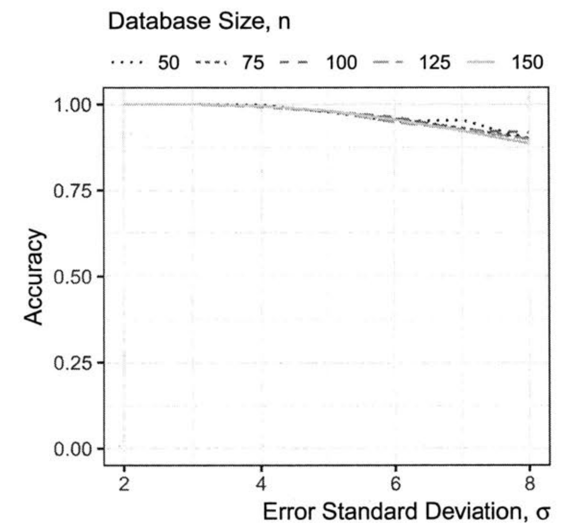
The [DMT07] linear program was used to infer which clientIds are present in the range. There was 1 false negative among the 12 present clientIds and 0 false positives among the 88 absent clientIds.

How accuracy varies with size, queries, and error. The accuracy of the linear reconstruction attack depends on the size of the dataset, the magnitude of the error, and the number of queries. When implementing our attack on Diffix, we used many more queries than seemed necessary for the level of noise used. The next experiment illustrates how the accuracy of [DMT07] varies with each of these parameters against a system using Gaussian noise to answer counting queries.

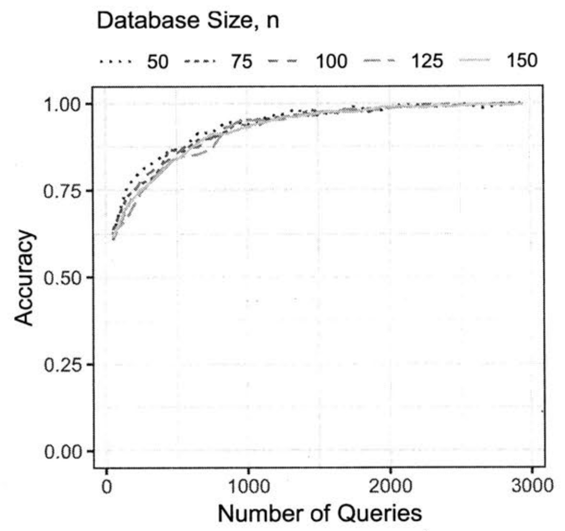
[2] *E.g.*, by issuing the query `SELECT count(*) FROM loans WHERE clientid BETWEEN` a `and` b to approximate the number of present clientIds in the range $\{a, \dots, b\}$.

The results are summarized in Figure 5-1. The plots display the average accuracy over 10 simulated runs of our [DMT07] reconstruction algorithm as the error magnitude, database size, and number of queries were varied. Each run resampled the Gaussian noise while the underlying dataset remained fixed. It is interesting to observe that the size of the dataset does not seem to significantly affect the effectiveness of reconstruction.

As described in Section 5.2.2, the analysis in [DMT07] applies to ± 1 queries but not to subset queries. To compare the effectiveness of these two query types, we ran the same simulations using ± 1 queries. The results are summarized in Figure 5-2. The plots are nearly indistinguishable from the corresponding plots in Figure 5-1.



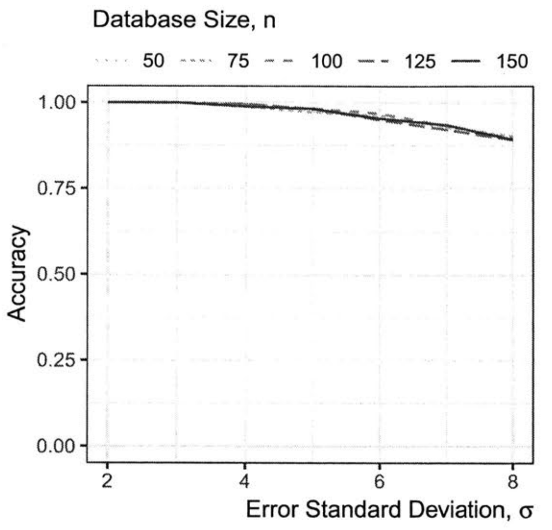
(a) Noise standard deviation varied from 1 to 20, in increments of 1, with 2550 queries. For $n = 100$, the mean accuracy falls below 0.99 at $\sigma = 5$ and below 0.95 at $\sigma = 7$.



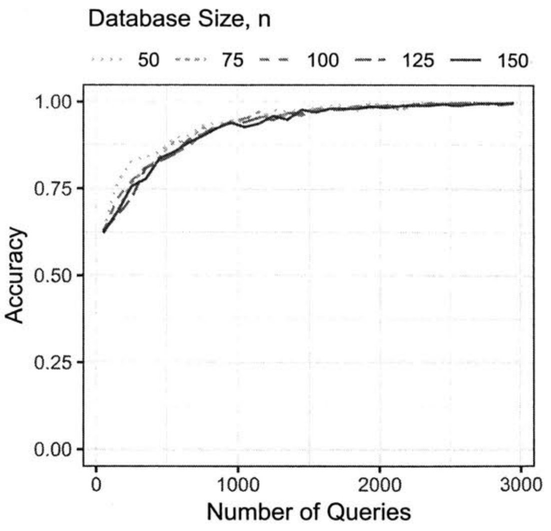
(b) The number of queries varied from 50 to 2950, in increments of 100, with noise magnitude $\sigma = 4$. For $n = 100$, the mean accuracy surpasses 0.95 at 1150 queries and surpasses 0.99 at 2050 queries.

Figure 5-1: Reconstruction accuracy as a function of the (5-1a) noise magnitude and (5-1b) number of queries, for various database sizes. The data is averaged over 10 trials of the [DMT07] linear program using subset queries.

Comparing [DN03] and [DMT07]. The original linear reconstruction attack for noisy counting queries comes from [DN03]. In contrast to [DMT07], [DN03] makes the additional assumption that each error e_q is bounded by a maximum error $\mathcal{E}$. In



(a) Noise standard deviation varied from 1 to 20, in increments of 1, with 2550 queries. For $n = 100$, the mean accuracy falls below 0.99 at $\sigma = 5$ and below 0.95 at $\sigma = 8$.



(b) The number of queries varied from 50 to 2950, in increments of 100, with noise magnitude $\sigma = 4$. For $n = 100$, the mean accuracy surpasses 0.95 at 1050 queries and surpasses 0.99 at 2050 queries.

Figure 5-2: Reconstruction accuracy as a function of the (5-2a) noise magnitude and (5-2b) number of queries, for various database sizes. The data is averaged over 10 trials of the [DMT07] linear program using ± 1 queries.

our experiments, we write $\mathcal{E} = B\sigma$, where B is the *error bound multiplier* and σ is the standard deviation of the Gaussian errors. The [DN03] linear program reflects the bounded-error assumption with an additional constraint and uses a trivial objective function.

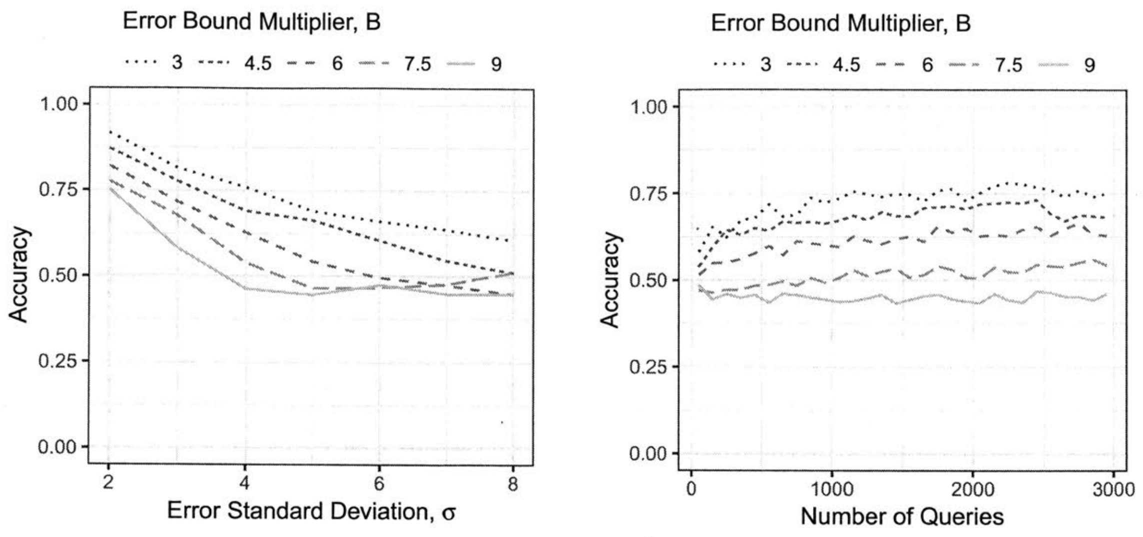
variables: $\mathbf{x}' = (x'_{\mathsf{id}})_{\mathsf{id} \in \mathsf{ID}}$ and $(e'_q)_{q \in Q}$

minimize: 0

subject to:

$$\begin{aligned} \forall q \in Q, \quad & e'_q = a_q - q(\mathbf{x}') \\ & e'_q \leq B\sigma \\ \forall \mathsf{id} \in \mathsf{ID}, \quad & 0 \leq x'_{\mathsf{id}} \leq 1 \end{aligned}$$

Our final experiment illustrates how the accuracy of the above [DN03]-based linear program varies as a function of the error magnitude, number of queries, and the error bound multiplier. The results are summarized in Figure 5-3. The plots display the



(a) Noise standard deviation varied from 1 to 20, in increments of 1, with 2550 queries and dataset size $n = 100$.

(b) The number of queries varied from 50 to 2950, in increments of 100, with noise magnitude $\sigma = 4$ and dataset size $n = 100$.

Figure 5-3: Accuracy as a function of the (5-3a) noise magnitude and (5-3b) number of queries, for various values of the DiNi multiplier B. The data is averaged over 10 trials using a dataset of size $n = 100$.

average accuracy over 10 simulated runs of our [DN03] reconstruction algorithm as the parameters were varied. Each run resampled the Gaussian noise while the underlying dataset remained fixed.

Observe that as the error bound multiplier B increases, the accuracy of reconstruction degrades. Because the linear program terminates once any feasible points is found, it is not surprising that expanding the set of feasible points by increasing B.

Note however that the pattern extends to $B = 3$. One would expect a few queries (in expectation about 4.6 queries per 2550 for $\sigma = 4$) to have rounded error greater than 3σ. Nevertheless, in each of the 240 trials run with $B = 3$ and at least 2550 queries, a feasible solution was found. In contrast, for $B = 2.5$ and $\sigma = 4$ half of all executions with 1850 queries were infeasible (dropping to $\geq 90\%$ infeasible at 2250 or more queries).

5.2.4 Additional Information on the Aircloak Challenge Attack

The results described in Section 5.2.2 are from a reanalysis of data gathered during the Aircloak Challenge. During the course of the Aircloak Challenge, we used a modified version of the [DMT07] linear program. For transparency, this section describes the modified linear program and its effectiveness.

The only difference between the linear program originally used and the one described in Section 5.2.2 is the addition of constraints upper bounding the magnitude of any error term.

variables: $\mathbf{x}' = (x'_{\mathsf{id}})_{\mathsf{id} \in \mathsf{ID}}$ and $(e'_q)_{q \in Q}$

minimize: $\sum_{q \in Q} |e'_q|$

subject to:

$$\begin{aligned} \forall q \in Q, \quad & e'_q = a_q - q(\mathbf{x}') \\ & e'_q \leq 5\sigma \\ \forall \mathsf{id} \in \mathsf{ID}, \quad & 0 \leq x'_{\mathsf{id}} \leq 1 \end{aligned}$$

where $\sigma = 4$ is the standard deviation of the true error distribution. Note that if the true errors were distributed according to $N(0, \sigma)$ and rounded to the nearest integer, an error of magnitude greater than 5σ would be expected once in every 1.7 million queries.

We first implemented the linear reconstruction solver using data from the clientId range $[2000, 3000]$, for which it achieved perfect reconstruction. Together with researchers at the Max Planck Institute for Software Systems, we verified the attack on three additional ranges of clientIds containing 110, 130, and 142 clientIds. The results are summarized in Table 5.1. In two of the three ranges, the attack again inferred whether each loanStatus was 'C' with high accuracy (1 and .9538). We were surprised, therefore when the final validation (this time targeting loanStatus 'A' rather than 'C') achieved accuracy of only 75.4%. Our confusion compounded when the accuracy degraded after increasing the number of queries, suggesting that we were not accounting for some source of error.

After further investigation, we realized that performing the queries for clientIds in [10000, 12000] required more numerical precision than seemed to be supported by Diffix. The larger clientId values in this range and the larger constants required for additional queries introduced errors that had not affected our earlier tests. Ultimately, high accuracy was recovered by ignoring the results from queries with larger values of e which seemed to require greater (but making no other changes to the linear program solver).

clientIds Range	# Entries (n)	# Queries	Target Status	Accuracy
2000-3000	73	3500	'C'	1
3000-5000	110	3500	'C'	1
5000-7000	130	3500	'C'	.9538
10000-12000	142	3500	'A'	.7535
10000-12000	142	2000, $e \leq 1.4$	'A'	1
10000-12000	142	1000, $e \leq 0.8$	'A'	.9930

Table 5.1: Summary of reconstruction tests performed against Diffix using the modified [DMT07] linear program. The queries in the final two resulted from restricting the exponent e to the indicated range.

5.3 Composition Attacks for k-Anonymity

5.3.1 Introduction to k-Anonymity

k-anonymity [SS98, Swe02] is a concept intended to help a data holder "release a version of its private data with scientific guarantees that the individuals who are the subjects of the data cannot be re-identified while the data remain practically useful" [Swe02]. The approach is to make each individual in a data release indistinguishable from at least $k-1$ individuals. Typically, a k-anonymized dataset is produced by subjecting it to a sequence of generalization and suppression operations.

Let $(A_1, \ldots, A_m)$ be *attribute domains*. A dataset $\mathbf{x} = (x_1, \ldots, x_n)$ is collection of rows $x_i = (a_{i,1}, \ldots, a_{i,m})$ where $a_{i,j} \in A_j$. For subsets $\widehat{a}_{i,j} \subseteq A_j$, we view $y_i = (\widehat{a}_{i,1}, \ldots, \widehat{a}_{i,m})$ as a set in the natural way, writing $x_i \in y_i$ if $\forall j \in [m]$, $a_{i,j} \in \widehat{a}_{i,j}$. We say that a dataset $\mathbf{y} = (y_1, \dots, y_n)$ is derived from $\mathbf{x}$ by generalization and suppression if $\forall i \in [n]$, $x_i \in y_i$. For example, if (A_1, A_2, A_3) correspond to "5 digit ZIP Code," "Gender," and "Year of Birth," then it may be that $x_i = (91015, F, 1972)$ and $y_i = (91010\text{–}91019, F, 1970\text{–}1975)$.

k-anonymity aims to capture a sort of anonymity of a crowd: data release $\mathbf{y}$ is k-anonymous if any individual row in the release cannot be distinguished from $k-1$ other individuals. This guarantee is typically parameterized by a subset $Q \subset \{A_j\}_{j \in [m]}$ of the attribute domains called a *quasi-identifier*.[3] Let $y(Q)$ be the restriction of y to those attribute domains $A_j \in Q$. Let $\mathsf{count}(\mathbf{y}, y, Q) \triangleq |\{y' \in \mathbf{y} : y'(Q) = y(Q)\}|$ be the number of rows in $\mathbf{y}$ which agree with y (including y itself). We call this the *effective anonymity* of y in $\mathbf{y}$ with respect to Q.

Definition 5.3.1 (k-Anonymity (rephrased from [Swe02])). *For $k \geq 2$, a dataset $\mathbf{y}$ is* k-anonymous *with respect to quasi-identifier Q if for all $y \in \mathbf{y}$, $\mathsf{count}(\mathbf{y}, y, Q) \geq k$. An algorithm is called a* k-anonymizer *if on an input dataset $\mathbf{x}$ its output is a k-anonymous $\mathbf{y}$ which is derived from $\mathbf{x}$ by generalization and suppression.*

Whatever privacy k-anonymity guarantees rests on the assumption that the "data holder can identify attributes in his private data that may also appear in external information and therefore, can accurately identify quasi-identifiers" [Swe02]. This assumption has been the subject of much criticism generally [NS10] and for specific datasets [PPC17]. Even when quasi-identifiers are properly identified, k-anonymity does not necessarily prevent sensitive inferences. Various refinements of k-anonymity aiming to provide greater inferential privacy have been proposed— including ℓ-diversity [MGKV06] and t-closeness [LLV07b].

5.3.1.1 Intersection attacks

Two desirable properties of a privacy concept are (i) immunity to post-processing (i.e., further processing of the outcome of a mechanism, without access to the data, should

[3]This definition of quasi-identifier, as the collection of multiple attributes, is from [Swe02]. Many, including the authors of the HarvardX-MITx dataset discussed below [MH14], use quasi-identifier to mean one of the constituent attributes. So each [Swe02] quasi-identifier is comprised of multiple [MH14] qausi-identifers.

not increase privacy risks); and (ii) closure under composition (i.e., a combination of two or more mechanisms which satisfy the requirements of the privacy concept is a mechanism that also satisfies the requirements, potentially, with worse parameters).

k-anonymity as defined above enjoys neither property. First, the result of removing rows from a k-anonymous dataset may not satisfy k-anonymity as defined. This fragility to post-processing is not so much a privacy failure as a syntactic weakness of the definition itself. Composition failure poses a more serious threat to privacy. Consider two $k = 3$-anonymous versions of an unknown dataset $\mathbf{x}$: $\mathbf{y}$ uses the quasi-identifier $Q = \{\text{ZIP Code}\}$ and $\mathbf{y}'$ uses the quasi-identifier $Q' = \{\text{Year of Birth}\}$. Both treat baldness and income as sensitive variables that cannot be learned from external sources.

$\mathbf{y} =$

ZIP	Income	Bald
9101$\star$	$\geq$ \$100k	Yes
9101$\star$	$\geq$ \$100k	No
9101$\star$	\$50k to \$75k	No
20037	\$75k to \$100k	No
20037	$\leq$ \$25k	No
20037	$\leq$ \$25k	Yes

$\mathbf{y}' =$

YoB	Income	Bald
Odd	$\geq$ \$100k	Yes
Odd	\$75k to \$100k	No
Odd	$\leq$ \$25k	No
Even	$\geq$ \$100k	No
Even	\$50k to \$75k	No
Even	$\leq$ \$25k	Yes

Given the ZIP Code and Year of Birth of a data subject, it is impossible to determine their tonsorial state or income using only one of $\mathbf{y}$ or $\mathbf{y}'$. But using both, we can join $\mathbf{y}$ and $\mathbf{y}'$ using income and baldness and deduce that the data subject who lives in 91011 and was born in 1971 must be bald and makes at least \$100k.

[GKS08] calls this an *intersection attack*. The example intersection attack stems from the use of two different quasi-identifiers for the same dataset.[4] It is easy to dismiss the example as a misapplication of k-anonymity—something that an expert wouldn't do. Because quasi-identifiers are supposed to capture that information that may be available from external data sources, it stands to reason that the union of quasi-identifiers should also be considered a quasi-identifier except with very good

[4]One can construct a similar example using only a single quasi-identifier anonymized in two different ways.

reason (or accident). Indeed, though the possibility of intersection attacks has been understood for over a decade [GKS08], to the best of our knowledge it was not previously known to have ever manifested in practice. This is despite the fact that foundational work on k-anonymity explicitly considers the possibility of multiple quasi-identifiers [SS98, Swe98].

In this thesis, we show that exactly this type of quasi-identifier mismatch occurs in a widely publicized dataset, allowing tens of thousands of data subjects to be distinguished within a supposedly k-anonymous dataset (Section 5.3.3). The same dataset seems to also exemplify the syntactic post-processing issue described above (Section 5.3.3.3)

5.3.2 The Harvard-MIT edX Dataset

597,692 individuals registered for 17 online courses offered by Harvard and MIT through the edX platform [HRN+14]. edX collected data about these students' demographics (e.g., date of birth, location) and engagement with course content (e.g., number of forum posts, grade). In order to enable outside researchers to study the data, its custodians determined to make it public.

However, edX considered this data to be subject to the Family Educational Rights and Privacy Act (FERPA), a data privacy law restricting the disclosure of certain educational records [MH14]. "To meet these privacy specifications, the HarvardX and MITx research team (guided by the general counsel, for the two institutions) opted for a k-anonymization framework" [ABW15]. A value of $k = 5$ "was chosen to allow legal sharing of the data."

We summarize the raw edX dataset, the chosen quasi-identifiers, the implementation of k-anonymization, and the resulting published dataset $\mathbf{x}_{\mathsf{ed}}$ based on documentation included with the dataset [MH14] and in two articles describing the creation of the dataset itself [DRW+14, ABW15]. The final published result $\mathbf{x}_{\mathsf{ed}}$ includes 641,138 course registrations by 476,532 students across 16 courses. $\mathbf{x}_{\mathsf{ed}}$ has been downloaded 12,532 times.

The raw dataset consisted of 841,687 rows for 597,692 students. Each row corresponded to the registration of a single student in a single course; it contained the student's username and IP address, the course name, demographic information about the student (including self-reported level of education, gender, and year of birth), and information about the student's interaction with the course (including number of forum posts, final course grade, and whether they received a certificate of completion). Most students chose not to report level of education, gender, and year of birth at all, and these data are missing. The researchers considered username and IP address to be identifying. Each username was replaced by a random 7-digit ID. A username appearing in multiple rows was replaced by the same random ID in each. The IP address was used to infer a student's country (along with self-reported address if available) but was otherwise redacted.

Five variables were considered to form a quasi-identifier, $Q = \{$gender, year of birth, country, course, number of forum posts$\}$. "The last one was chosen as a quasi-identifier because the edX forums are somewhat publicly accessible and someone wishing to re-identify the dataset could, with some effort, compile the count of posts to the forum by username" [MH14]. Separately, the set of courses that each student enrolled in were considered to form a quasi-identifier: $Q' = \{$enrolled in course 1, ..., enrolled in course 16$\}$. The data was k-anonymized first according to Q' and then according to Q.[5] The result contained 641,138 rows and 476,532 students.

5.3.3 Violating k-Anonymity of the HarvardX-MITx dataset

k-anonymizing a dataset according to multiple different quasi-identifiers may compromise privacy. The edX dataset proves that this possibility is a reality. The dataset $\mathbf{x}_{\mathsf{ed}}$ is formatted so that a row corresponds to a student-course pair. But the object of our privacy concern is not a student-course pair, but the student himself. Each row related to an individual student contains the same random ID. Our first step is to reorganize the dataset to consist of 476,532 rows, each corresponding to a single student's ID.

[5]Additionally, ℓ-diversity was enforced for the final course grade, with $\ell = 2$.

In the new pivoted dataset $\mathbf{x}'_{\mathsf{ed}}$, rows contain demographic information about a student (including gender, year of birth, and country[6]) and information about the student's involvement in each of the 16 courses (including whether the student enrolled or received a certificate, grade, and number of forum posts).

The original dataset was k-anonymized according to two different quasi-identifiers Q and Q'.[7]Let $Q_{\mathsf{all}} = Q \cup Q' = \{$gender, year of birth, country, all course enrollments, number of forum posts in all courses$\}$. $\mathbf{x}'_{\mathsf{ed}}$ is very far from 5-anonymous with respect to Q_{all}. 7.1% of students (33,925 students) in $\mathbf{x}'_{\mathsf{ed}}$ have effective anonymity 1 with respect to Q_{all} (i.e., $\mathsf{count}(\mathbf{x}'_{\mathsf{ed}}, y, Q_{\mathsf{all}}) = 1$). 15.3% have effective anonymity less than 5. Recall that 5-anonymity was edX's FERPA compliance target.

We emphasize that the creators of $\mathbf{x}_{\mathsf{ed}}$ never intended or claimed to provide 5-anonymity with respect to Q_{all}. But since quasi-identifiers are intended to capture the information that may be publicly available, the union of quasi-identifiers also be considered a quasi-identifier. Any exceptions must be justified, but no justification is given for $\mathbf{x}_{\mathsf{ed}}$.

	Q_{all}	Q_{cls}	Q_{acq}	$Q_{\mathsf{acq+loe}}$	Q_{emp}	$Q_{\mathsf{emp+loe}}$
EA 1 in $\mathbf{x}'_{\mathsf{ed}}$	7.1	.025 (1.7)	6.7	8.7	.79 (23.2)	1.2 (34.2)
EA < 5 in $\mathbf{x}'_{\mathsf{ed}}$	15.3	.045 (3.0)	14.6	20.6	1.8 (54.3)	2.3 (67.4)
EA 1 in $\mathbf{x}'_{\mathsf{ed,clean}}$	6.9	-	6.6	8.8	.82 (26.1)	1.2 (38.9)
EA < 5in $\mathbf{x}'_{\mathsf{ed,clean}}$	15.2	-	14.7	20.8	1.9 (60.9)	2.4 (75.9)
UEA 1 in $\mathbf{x}'_{\mathsf{ed}}$	1.9	-	1.5	1.6	.14 (4.2)	.15 (4.5)
UEA < 5 in $\mathbf{x}'_{\mathsf{ed}}$	4.7	-	4.0	4.3	.42 (12.5)	.48 (14.2)

Table 5.2: Percent of students with effective anonymity (EA) or unambiguous effective anonymity (UEA) 1 and < 5 in edX-derived datasets with respect to various choices of attacker knowledge Q, as described in this section. Numbers in parentheses are the value as a percentage of a relevant subset of the full dataset: for Q_{cls}, students with at least one forum post (7,251); for Q_{emp} and $Q_{\mathsf{emp+loe}}$, students with at least one certificate (16,224).

[6]Different rows of the same student often listed different countries. Almost always there were only two different values, one of which was "Unknown/Other." In this case, we used the other value for the student's unified record. In all other cases, we used "Unknown/Other."

[7]The effective number of different quasi-identifiers in the pivoted dataset $\mathbf{x}'_{\mathsf{ed}}$ is 17: Q' as before and $Q_1, \ldots, Q_{16}$, where $Q_i = \{$gender, year of birth, country, enrolled in course i, number of forum posts in course $i\}$.

5.3.3.1 Limiting the Attacker's Knowledge

One might still argue that an attacker who knows Q_{all} is too powerful, and that only protection against weaker attackers is necessary. We disagree, but it raises a nonetheless important question: Does $\mathbf{x}_{\mathsf{ed}}$ provide effective anonymity against more 'realistic' attackers? We consider three hypothetical attackers: edX classmates, casual acquaintances, and prospective employers. Table 5.2 summarizes the results of all analyses described in this section.

edX classmate. Consider an attacker who knows $Q_{\mathsf{cls}} =$ {number of forum posts in courses 1–16} $\subseteq Q_{\mathsf{all}}$. 120 students in $\mathbf{x}'_{\mathsf{ed}}$ have effective anonymity 1 with respect to Q_{cls}, and 216 have effective anonymity less than 5. These numbers may seem minute, but they constitute 1.7% and 3.0% of the 7251 students in the dataset that made any forum posts whatsoever.

Why might somebody know Q_{cls}? Each edX course had an online forum for student discussions. Because these posts were public to all students enrolled in a given course, the number of forum posts made by any user was considered by edX to be publicly available information. But Ignoring composition, they did not consider the combination of forum post counts made by a user across courses. This is despite the fact that 20 students *in the dataset itself* enrolled in all 16 courses and could have compiled forum post counts across courses for all other edX students.

In fact, the situation is worse than this. For every one of the 120 students with uniquely identifying forum post counts, forum post counts in a strict subset of the 16 courses suffices to uniquely distinguish them. This enables more classmates to act as attackers than just the 20 who took all courses. Indeed, the 120 vulnerable students can be distinguished by 23–70 classmates each; 60 can be distinguished by 40–49 classmates each.

Casual acquaintance. Consider an attacker who knows $Q_{\mathsf{acq}} =$ {gender, year of birth, location, enrollment in courses 1–16} $\subseteq Q_{\mathsf{all}}$. 6.7% of students in $\mathbf{x}'_{\mathsf{ed}}$ have effective anonymity 1 with respect to Q_{acq}, and 14.6% have effective anonymity less than 5.

Why might somebody know Q_{acq}? Imagine two casual acquaintances who, in the course of normal conversation, discuss their experiences edX. It is no stretch to assume that they would tell each other which courses they enrolled in. It is also reasonable to assume they know each other's ages, genders, and locations, as many acquaintances do.

Indeed, acquaintances would likely know each other's level of education too, even though this is not included in any of the quasi-identifiers (namely, LoE $\notin Q_{\mathsf{all}}$). If we augment the attacker's knowledge with level of education $Q_{\mathsf{acq+loe}} = Q_{\mathsf{acq}} \cup \{LoE\}$, then things become worse. 8.7% students in $\mathbf{x}'_{\mathsf{ed}}$ have effective anonymity 1 with respect to $Q_{\mathsf{acq+loe}}$, and 20.6% have effective anonymity less than 5.

Prospective employer. Consider an attacker who knows Q_{emp} ={gender, year of birth, location, certificates earned in courses 1–16} $\not\subseteq Q_{\mathsf{all}}$. Q_{emp} only includes those certificates actually earned, but omits courses in which a student enrolled but did not earn a certificate. 3772 students in $\mathbf{x}'_{\mathsf{ed}}$ have effective anonymity 1 with respect to Q_{acq}, and 8808 have effective anonymity less than 5. These numbers may seem small, but they constitute 23.2% and 54.3% of the 16,224 students in the dataset that earned any certificates whatsoever.

Why might somebody know Q_{emp}? Imagine a student applying for a job who lists on their resume the certificates they earned in edX. It is no stretch to assume the prospective employer also knows the applicant's gender, age, and location. While certifications were not included in any of the quasi-identifiers originally considered, there are reasonable scenarios in which they may become known.

If a resume includes edX certifications, it would likely also include level of education too. If we augment the attacker's knowledge with level of education $Q_{\mathsf{emp+loe}} = Q_{\mathsf{emp}} \cup \{LoE\}$, then again things become worse. 5546 students in $\mathbf{x}'_{\mathsf{ed}}$ have effective anonymity 1 with respect to $Q_{\mathsf{emp+loe}}$, and 10942 have effective anonymity less than 5. These constitute 34.2% and 67.4% of the 16,224 students in the dataset that earned any certificates whatsoever.

5.3.3.2 Inference and Unambiguous Effective Anonymity

What inferences can our attackers make about individual students? 7.1% of students are uniquely distinguishable using Q_{all}. A naive interpretation is that an attacker who knows Q_{all} would be able to definitively learn the grades of 7.1% of the students. However, two sources of ambiguity in $\mathbf{x}_{\mathsf{ed}}$ deserve discussion.[8]

Missing data. The first source of ambiguity is information missing from the raw dataset. Gender, year of birth, and level of education were voluntarily self-reported by students. Many students chose not to provide this information: 14.9% of students records are missing at least one of these attributes. This information is missing in the raw data, not just the published data.[9] Thus, a female Italian student born in 1986 might appear in the dataset with any or all of these three attributes missing.

This makes the 7.1% result difficult to interpret. From an inferential standpoint, the relevant question is not how many students have unique quasi-identifiers, but how many have *unambiguously* unique quasi-identifiers. This number may be much lower than 7.1%. But this ambiguity comes from the data, not from k-anonymity. The missing information may be masking the implementation's weaknesses. It is similarly difficult to interpret the results of all our hypothetical attackers (except the classmate using Q_{cls} which contains none of these attributes).

Table 5.2 summarizes the results of two additional sets of analyses on the dataset which address the problem of missing data. The first analysis examines the effectiveness of edX's implementation of k-anonymity among those students who volunteered their personal information. We remove all the students who did not self-report gender, year of birth, or level of education. Let $\mathbf{x}'_{\mathsf{ed,clean}} \subset \mathbf{x}'_{\mathsf{ed}}$ be the subset with those students removed. We determine the effective anonymity of students in $\mathbf{x}'_{\mathsf{ed,clean}}$ with respect to Q_{all}, Q_{acq}, $Q_{\mathsf{acq+loe}}$, Q_{emp}, and $Q_{\mathsf{emp+loe}}$. The results are qualitatively similar

[8]A third source of ambiguity does not deserve discussion: erroneous data. We are critiquing a privacy technique; erroneous data is not a feature of the privacy technique, it is an undesirable feature of the data.

[9]"The general strategy chosen was to prioritize integrity of the original values over the number of records preserved. As a result, 10–20% of the dataset will be deleted, but the remaining records will be virtually unaltered" [MH14].

to the results using $\mathbf{x}'_{\mathsf{ed}}$.

The second analysis considers what inferences our attackers can make about individuals using the dataset $\mathbf{x}'_{\mathsf{ed}}$ as is. First we define the *unambiguous effective anonymity* of $y \in \mathbf{y}$ with respect to Q. Unambiguous effective anonymity generalizes effective anonymity and helps us reason about the inferences an attacker can make. Recall that for attribute domains $\{A_j\}_{j\in[m]}$ and subsets $\{\widehat{a}_j \subset A_j\}_{j\in[m]}$, we view $y = (\widehat{a}_1, \dots, \widehat{a}_m)$ as a set in the natural way. For $Q \subseteq \{A_j\}_{j\in[m]}$, $y(Q)$ is the restriction of y to those attribute domains $A_j \in Q$. The *unambiguous effective anonymity* of $y \in \mathbf{y}$ with respect to Q is $\mathsf{count}_{\mathsf{u}}(\mathbf{y}, y, Q) \triangleq |\{y' \in \mathbf{y} : y'(Q) \cap y(Q) \neq \emptyset\}|$.

We reperform our analyses using unambiguous effective anonymity: $\mathsf{count}_{\mathsf{u}}$ instead of count. To do so, we interpret the attribute values present in $\mathbf{x}_{\mathsf{ed}}$ as sets of attribute values that are consistent with the value. For example, the gender "female' is interpreted as the set {'female', 'unknown'} and the gender unknown' is interpreted as the set {'female', 'male', 'other', 'unknown'}. Year of birth and level of education are handeled analogously.

1.9% of students have unambiguous effective anonymity 1 with respect to Q_{all}. As expected, this is much smaller than the 7.1% with effective anonymity 1. But these 9,125 students are unambiguously identifiable in the dataset to anybody who knows all the quasi-identifiers, without knowing whether the students chose to self-report their gender, year of birth, or level of education. This allows an attacker to draw meaningful inferences about them.

Take, for example, a prospective employer who is interested in discovering whether a job applicant failed (or failed to complete) edX courses. 4.5% of the 16,224 students (732 students) who earned any certifications have unambiguous effective anonymity 1 with respect to $Q_{\mathsf{emp+loe}}$. 333 of these students failed at least one course, and 38 failed three or more courses.

Omitted data. The second source of ambiguity is omissions in the published data. Of the 597,692 students enrolled in edX courses over the relevant period, only 476,532 appear in the published dataset. 20.3% of students are completely omitted. Moreover, some students had only subset of their courses removed from the dataset. One might argue that the 20.3% omission rate imbues $\mathbf{x}_{\mathsf{ed}}$ with additional anonymity—even for those 7.1% of included records uniquely distinguishable using Q_{all}.

This argument is unconvincing. First, it implicitly assumes that whether a record is omitted or included is a random event, independent of its contents. In reality, the omitted records are "outliers and highly active users because these users are more likely to be unique and therefore easy to re-identify" [MH14]. We believe it unlikely that a large fraction of the 7.1% of uniquely distinguishable users $\mathbf{x}_{\mathsf{ed}}$ might have a matching record among those omitted, though we cannot test this hypothesis. Second, the argument suggests a natural way to anonymize a dataset: release a random fraction of the data, say $1/k$ of the rows. A randomly subsampled version of the edX dataset would be much more useful than $\mathbf{x}_{\mathsf{ed}}$. The data would be unredacted and ungeneralized. Unlike the actual released dataset, statistics computed from the subsample would easily generalize to the underlying raw data, enabling better social science research. But the edX curators did no such thing, suggesting that they may not have considered subsampling to be sufficient to meet their anonymity requirements.

5.3.3.3 Post-processing

The edX dataset aims to provide 5-anonymity with respect to two quasi-identifiers: $Q = \{$gender, year of birth, country, course, number of forum posts$\}$ and $Q' = \{$enrolled in course 1, ..., enrolled in course 16$\}$. However, $\mathbf{x}_{\mathsf{ed}}$ is not actually 5-anonymous with respect to Q'. There are 245 students with effective anonymity 1 (i.e., enrolled in a unique set of courses). A total of 753 students have effective anonymity less than five with respect to Q'.

We suspect this error is due to k-anonymity's formal fragility with respect to post-processing. The raw data was first 5-anonymized with respect to Q' and then with respect to Q. In the course of anonymizing with respect to Q, some rows in the dataset were deleted. These deletions may have caused the violations of 5-anonymity.

Bibliography

[A29a] *Article 29 Data Protection Working Party Opinion 04/2007 on the Concept of Personal Data.*

[A29b] *Article 29 Data Protection Working Party Opinion 05/2014 on Anonymisation Techniques.*

[ABC$^+$07] Giuseppe Ateniese, Randal Burns, Reza Curtmola, Joseph Herring, Lea Kissner, Zachary Peterson, and Dawn Song. Provable data possession at untrusted stores. In *Proceedings of the 14th ACM conference on Computer and communications security*, pages 598–609. Acm, 2007.

[ABH09] Giuseppe Ateniese, Karyn Benson, and Susan Hohenberger. Key-private proxy re-encryption. In *CT-RSA*, volume 5473, pages 279–294. Springer, 2009.

[Abo18] John Abowd. Staring-down the database reconstruction theorem, July 2018. https://www.census.gov/content/dam/Census/newsroom/press-kits/2018/jsm/jsm-presentation-database-reconstruction.pdf.

[ABW$^+$13] Yoshinori Aono, Xavier Boyen, Lihua Wang, et al. Key-private proxy re-encryption under lwe. In *International Conference on Cryptology in India*, pages 1–18. Springer, 2013.

[ABW15] Olivia Angiuli, Joe Blitzstein, and Jim Waldo. How to de-identify your data. *Communications of the ACM*, 58(12):48–55, 2015.

[ACJ17] Prabhanjan Ananth, Aloni Cohen, and Abhishek Jain. Cryptography with updates. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 445–472. Springer, 2017.

[AFGH05] Giuseppe Ateniese, Kevin Fu, Matthew Green, and Susan Hohenberger. Improved proxy re-encryption schemes with applications to secure distributed storage. In *NDSS*. The Internet Society, 2005.

[AFGH06] Giuseppe Ateniese, Kevin Fu, Matthew Green, and Susan Hohenberger. Improved proxy re-encryption schemes with applications to secure distributed storage. *ACM Transactions on Information and System Security (TISSEC)*, 9(1):1–30, 2006.

[AGVW13] Shweta Agrawal, Sergey Gorbunov, Vinod Vaikuntanathan, and Hoeteck Wee. Functional encryption: New perspectives and lower bounds. In *Advances in Cryptology - CRYPTO 2013 - 33rd Annual Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2013. Proceedings, Part II*, pages 500–518, 2013.

[AIK07] Benny APPLEBAUM, Yuval ISHAI, and Eyal KUSHILEVITZ. Cryptogaphy in nc0. *SIAM journal on computing*, 36(4):845–888, 2007.

[AIR18a] Aircloak attack challenge, 2018. https://aircloak.com/solutions/attack-challenge-en/.

[AIR18b] Data anonymisation: What it is and why it matters. 2018. https://aircloak.com/wp-content/uploads/Data-Anonymisation-What-it-is-and-Why-it-Matters.pdf.

[AIR18c] Data compliance in the gdpr - how anonymisation allows you to stay compliant in your data analysis, Aug 2018. https://aircloak.com/data-compliance-in-the-gdpr/.

[AJ15] Prabhanjan Ananth and Abhishek Jain. Indistinguishability obfuscation from compact functional encryption. In *Advances in Cryptology–CRYPTO 2015*, pages 308–326. Springer, 2015.

[AJS15a] Prabhanjan Ananth, Abhishek Jain, and Amit Sahai. Indistinguishability obfuscation from functional encryption for simple functions. Technical report, Cryptology ePrint Archive, Report 2015/730, 2015.

[AJS15b] Prabhanjan Ananth, Abhishek Jain, and Amit Sahai. Patchable indistinguishability obfuscation: io for evolving software. *IACR Cryptology ePrint Archive*, 2015:1084, 2015.

[AKV03] André Adelsbach, Stefan Katzenbeisser, and Helmut Veith. Watermarking schemes provably secure against copy and ambiguity attacks. In Moti Yung, editor, *Proceedings of the 2003 ACM workshop on Digital rights management 2003, Washington, DC, USA, October 27, 2003*, pages 111–119. ACM, 2003.

[BBS98a] Matt Blaze, Gerrit Bleumer, and Martin Strauss. Divertible protocols and atomic proxy cryptography. In *EUROCRYPT'98*, volume 1403 of *LNCS*, pages 127–144, 1998.

[BBS98b] Matt Blaze, Gerrit Bleumer, and Martin Strauss. Divertible protocols and atomic proxy cryptography. In *Advances in Cryptology—EUROCRYPT'98*, pages 127–144. Springer, 1998.

[BF11] Dan Boneh and David Mandell Freeman. Homomorphic signatures for polynomial functions. In *Annual International Conference on the Theory* *and Applications of Cryptographic Techniques*, pages 149–168. Springer, 2011.

[BFP$^+$15] Abhishek Banerjee, Georg Fuchsbauer, Chris Peikert, Krzysztof Pietrzak, and Sophie Stevens. Key-homomorphic constrained pseudorandom functions. In Dodis and Nielsen [DN15], pages 31–60.

[BGG94] Mihir Bellare, Oded Goldreich, and Shafi Goldwasser. Incremental cryptography: The case of hashing and signing. In *Advances in Cryptology—CRYPTO'94*, pages 216–233. Springer, 1994.

[BGG95] Mihir Bellare, Oded Goldreich, and Shafi Goldwasser. Incremental cryptography and application to virus protection. In *Proceedings of the twenty-seventh annual ACM symposium on Theory of computing*, pages 45–56. ACM, 1995.

[BGI$^+$01a] Boaz Barak, Oded Goldreich, Russell Impagliazzo, Steven Rudich, Amit Sahai, Salil P. Vadhan, and Ke Yang. On the (im)possibility of obfuscating programs. In *Advances in Cryptology - CRYPTO 2001, 21st Annual International Cryptology Conference, Santa Barbara, California, USA, August 19-23, 2001, Proceedings*, pages 1–18, 2001.

[BGI$^+$01b] Boaz Barak, Oded Goldreich, Russell Impagliazzo, Steven Rudich, Amit Sahai, Salil P. Vadhan, and Ke Yang. On the (im)possibility of obfuscating programs. In Joe Kilian, editor, *Advances in Cryptology - CRYPTO 2001, 21st Annual International Cryptology Conference, Santa Barbara, California, USA, August 19-23, 2001, Proceedings*, volume 2139 of *Lecture Notes in Computer Science*, pages 1–18. Springer, 2001.

[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.

[BGI13] Elette Boyle, Shafi Goldwasser, and Ioana Ivan. Functional signatures and pseudorandom functions. Cryptology ePrint Archive, Report 2013/401, 2013. `https://eprint.iacr.org/2013/401`.

[BGL$^+$15] Nir Bitansky, Sanjam Garg, Huijia Lin, Rafael Pass, and Siddartha Telang. Succinct randomized encodings and their applications. In *STOC*, 2015.

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

[BHR12] Mihir Bellare, Viet Tung Hoang, and Phillip Rogaway. Foundations of garbled circuits. In *Proceedings of the 2012 ACM conference on Computer and communications security*, pages 784–796. ACM, 2012.

[BKS17] Foteini Baldimtsi, Aggelos Kiayias, and Katerina Samari. Watermarking public-key cryptographic functionalities and implementations. In *Information Security - 20th International Conference, ISC 2017, Ho Chi Minh City, Vietnam, November 22-24, 2017, Proceedings*, pages 173–191, 2017.

[BKY01] Enrico Buonanno, Jonathan Katz, and Moti Yung. Incremental unforgeable encryption. In *Fast Software Encryption*, pages 109–124. Springer, 2001.

[BLMR13a] Dan Boneh, Kevin Lewi, Hart Montgomery, and Ananth Raghunathan. Key homomorphic prfs and their applications. In *Advances in Cryptology-CRYPTO 2013*, pages 410–428. Springer, 2013.

[BLMR13b] 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.

[BLW17] Dan Boneh, Kevin Lewi, and David J. Wu. Constraining pseudorandom functions privately. In *Public-Key Cryptography - PKC 2017 - 20th IACR International Conference on Practice and Theory in Public-Key Cryptography, Amsterdam, The Netherlands, March 28-31, 2017, Proceedings, Part II*, pages 494–524, 2017.

[BM97] Mihir Bellare and Daniele Micciancio. A new paradigm for collision-free hashing: Incrementality at reduced cost. In *Advances in Cryptology—EUROCRYPT'97*, pages 163–192. Springer, 1997.

[BNPW16] Nir Bitansky, Ryo Nishimaki, Alain Passelègue, and Daniel Wichs. From cryptomania to obfustopia through secret-key functional encryption. volume TCC, 2016.

[BNS$^+$16] Raef Bassily, Kobbi Nissim, Adam D. Smith, Thomas Steinke, Uri Stemmer, and Jonathan Ullman. Algorithmic stability for adaptive data analysis. In Daniel Wichs and Yishay Mansour, editors, *Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2016, Cambridge, MA, USA, June 18-21, 2016*, pages 1046–1059. ACM, 2016.

[BP13] Nir Bitansky and Omer Paneth. On the impossibility of approximate obfuscation and applications to resettable cryptography. In *Symposium on Theory of Computing Conference, STOC'13, Palo Alto, CA, USA, June 1-4, 2013*, pages 241–250, 2013.

[BPR12] Abhishek Banerjee, Chris Peikert, and Alon Rosen. Pseudorandom functions and lattices. In *Advances in Cryptology - EUROCRYPT 2012 -*

31st Annual International Conference on the Theory and Applications of Cryptographic Techniques, Cambridge, UK, April 15-19, 2012. Proceedings, pages 719–737, 2012.

[BPR$^+$17] Cristian Borcea, Yuriy Polyakov, Kurt Rohloff, Gerard Ryan, et al. Picador: End-to-end encrypted publish–subscribe information distribution with proxy re-encryption. *Future Generation Computer Systems*, 71:177–191, 2017.

[BSW11] Dan Boneh, Amit Sahai, and Brent Waters. Functional encryption: Definitions and challenges. In *Theory of Cryptography*, pages 253–273. Springer, 2011.

[BV11] Zvika Brakerski and Vinod Vaikuntanathan. Fully homomorphic encryption from ring-lwe and security for key dependent messages. In *Annual cryptology conference*, pages 505–524. Springer, 2011.

[BV15a] Nir Bitansky and Vinod Vaikuntanathan. Indistinguishability obfuscation from functional encryption. In *Foundations of Computer Science (FOCS), 2015 IEEE 56th Annual Symposium on*, pages 171–190. IEEE, 2015.

[BV15b] Zvika Brakerski and Vinod Vaikuntanathan. Constrained key-homomorphic prfs from standard lattice assumptions - or: How to secretly embed a circuit in your PRF. In Dodis and Nielsen [DN15], pages 1–30.

[BW07] Dan Boneh and Brent Waters. Conjunctive, subset, and range queries on encrypted data. In *Theory of Cryptography Conference*, pages 535–554. Springer, 2007.

[BW13] Dan Boneh and Brent Waters. Constrained pseudorandom functions and their applications. In *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*, pages 280–300, 2013. Full version available from http://eprint.iacr.org/2013/352.

[CCL$^+$14] Nishanth Chandran, Melissa Chase, Feng-Hao Liu, Ryo Nishimaki, and Keita Xagawa. Re-encryption, functional re-encryption, and multi-hop re-encryption: A framework for achieving obfuscation-based security and instantiations from lattices. In *PKC*, pages 95–112. Springer, 2014.

[CH07] Ran Canetti and Susan Hohenberger. Chosen-ciphertext secure proxy re-encryption. In *Proceedings of the 14th ACM conference on Computer and communications security*, pages 185–194. ACM, 2007.

[CHJV15] Ran Canetti, Justin Holmgren, Abhishek Jain, and Vinod Vaikuntanathan. Indistinguishability obfuscation of iterated circuits and RAM programs. In *STOC*, 2015.

[CHN$^+$16] Aloni Cohen, Justin Holmgren, Ryo Nishimaki, Vinod Vaikuntanathan, and Daniel Wichs. Watermarking cryptographic capabilities. In *Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2016, Cambridge, MA, USA, June 18-21, 2016*, pages 1115–1127, 2016.

[CHN$^+$18] Aloni Cohen, Justin Holmgren, Ryo Nishimaki, Vinod Vaikuntanathan, and Daniel Wichs. Watermarking cryptographic capabilities. *SIAM J. Comput.*, 47(6):2157–2202, 2018.

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

[CIJ$^+$13] Angelo De Caro, Vincenzo Iovino, Abhishek Jain, Adam O'Neill, Omer Paneth, and Giuseppe Persiano. On the achievability of simulation-based security for functional encryption. In *Advances in Cryptology - CRYPTO 2013 - 33rd Annual Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2013. Proceedings, Part II*, pages 519–535, 2013.

[CKLM12] Melissa Chase, Markulf Kohlweiss, Anna Lysyanskaya, and Sarah Meiklejohn. Malleable proof systems and applications. In *Advances in Cryptology-EUROCRYPT 2012*, pages 281–300. Springer, 2012.

[CKLM13] Melissa Chase, Markulf Kohlweiss, Anna Lysyanskaya, and Sarah Meiklejohn. Succinct malleable nizks and an application to compact shuffles. In *Theory of Cryptography*, pages 100–119. Springer, 2013.

[CKN03] Ran Canetti, Hugo Krawczyk, and Jesper B Nielsen. Relaxing chosen-ciphertext security. In *Annual International Cryptology Conference*, pages 565–582. Springer, 2003.

[CKO14] Nishanth Chandran, Bhavana Kanukurthi, and Rafail Ostrovsky. Locally updatable and locally decodable codes. In *Theory of Cryptography*, pages 489–514. Springer, 2014.

[CN18] Aloni Cohen and Kobbi Nissim. Linear program reconstruction in practice. *CoRR*, abs/1810.05692, 2018.

[CN19] Aloni Cohen and Kobbi Nissim. Towards formalizing the gdpr's notion of singling out. *CoRR*, abs/1904.06009, 2019.

[Coh19] Aloni Cohen. What about bob? the inadequacy of CPA security for proxy reencryption. In *Public-Key Cryptography - PKC 2019 - 22nd*

IACR International Conference on Practice and Theory of Public-Key Cryptography, Beijing, China, April 14-17, 2019, Proceedings, Part II, pages 287–316, 2019.

[CS03] Ronald Cramer and Victor Shoup. Design and analysis of practical public-key encryption schemes secure against adaptive chosen ciphertext attack. *SIAM Journal on Computing*, 33(1):167–226, 2003.

[CWYD10] Sherman SM Chow, Jian Weng, Yanjiang Yang, and Robert H Deng. Efficient unidirectional proxy re-encryption. In *International Conference on Cryptology in Africa*, pages 316–332. Springer, 2010.

[DFH$^+$15] Cynthia Dwork, Vitaly Feldman, Moritz Hardt, Toniann Pitassi, Omer Reingold, and Aaron Leon Roth. Preserving statistical validity in adaptive data analysis. In Rocco A. Servedio and Ronitt Rubinfeld, editors, *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing, STOC 2015, Portland, OR, USA, June 14-17, 2015*, pages 117–126. ACM, 2015.

[DKL$^+$18] David Derler, Stephan Krenn, Thomas Lorünser, Sebastian Ramacher, Daniel Slamanig, and Christoph Striecks. Revisiting proxy re-encryption: Forward secrecy, improved security, and applications. In *IACR International Workshop on Public Key Cryptography*, pages 219–250. Springer, 2018.

[DKM$^+$06] Cynthia Dwork, Krishnaram Kenthapadi, Frank McSherry, Ilya Mironov, and Moni Naor. Our data, ourselves: Privacy via distributed noise generation. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 486–503. Springer, 2006.

[DMNS06] Cynthia Dwork, Frank McSherry, Kobbi Nissim, and Adam Smith. Calibrating noise to sensitivity in private data analysis. In *Theory of cryptography conference*, pages 265–284. Springer, 2006.

[DMT07] Cynthia Dwork, Frank McSherry, and Kunal Talwar. The price of privacy and the limits of LP decoding. In *STOC*, pages 85–94. ACM, 2007.

[DN03] Irit Dinur and Kobbi Nissim. Revealing information while preserving privacy. In *Proceedings of the twenty-second ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 202–210. ACM, 2003.

[DN15] Yevgeniy Dodis and Jesper Buus Nielsen, editors. *Theory of Cryptography - 12th Theory of Cryptography Conference, TCC 2015, Warsaw, Poland, March 23-25, 2015, Proceedings, Part II*, volume 9015 of *Lecture Notes in Computer Science*. Springer, 2015.

[DN18] Nico Döttling and Ryo Nishimaki. Universal proxy re-encryption. Cryptology ePrint Archive, Report 2018/840, 2018. https://eprint.iacr.org/2018/840.

[DORS08] Yevgeniy Dodis, Rafail Ostrovsky, Leonid Reyzin, and Adam Smith. Fuzzy extractors: How to generate strong keys from biometrics and other noisy data. *SIAM journal on computing*, 38(1):97–139, 2008.

[DRW$^+$14] Jon P Daries, Justin Reich, Jim Waldo, Elise M Young, Jonathan Whittinghill, Andrew Dean Ho, Daniel Thomas Seaton, and Isaac Chuang. Privacy, anonymity, and big data in the social sciences. 2014.

[DSLSZ15] Dana Dachman-Soled, Feng-Hao Liu, Elaine Shi, and Hong-Sheng Zhou. Locally decodable and updatable non-malleable codes and their applications. In *Theory of Cryptography*, pages 427–450. Springer, 2015.

[DY08] Cynthia Dwork and Sergey Yekhanin. New efficient attacks on statistical disclosure control mechanisms. In *CRYPTO*, volume 5157 of *Lecture Notes in Computer Science*, pages 469–480. Springer, 2008.

[EPRS17] Adam Everspaugh, Kenneth Paterson, Thomas Ristenpart, and Sam Scott. Key rotation for authenticated encryption. In *Annual International Cryptology Conference*, pages 98–129. Springer, 2017.

[Eur16] Regulation (EU) 2016/679 of the European Parliament and of the Council of 27 April 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing Directive 95/46/EC (General Data Protection Regulation). *Official Journal of the European Union*, L119:1–88, May 2016.

[FEO$^+$18] Paul Francis, Sebastian Probst Eide, Pawel Obrok, Cristian Berneanu, Sasa Juric, and Reinhard Munz. Extended diffix. *CoRR*, abs/1806.02075, 2018.

[Fis97] Marc Fischlin. Incremental cryptography and memory checkers. In *Advances in Cryptology—EUROCRYPT'97*, pages 393–408. Springer, 1997.

[FKKP18] Georg Fuchsbauer, Chethan Kamath, Karen Klein, and Krzysztof Pietrzak. Adaptively secure proxy re-encryption. Cryptology ePrint Archive, Report 2018/426, 2018. https://eprint.iacr.org/2018/426.

[FL17] Xiong Fan and Feng-Hao Liu. Proxy re-encryption and re-signatures from lattices. 2017.

[Gen09a] Craig Gentry. *A fully homomorphic encryption scheme*. Stanford University, 2009.

[Gen09b] Craig Gentry. Fully homomorphic encryption using ideal lattices. In *STOC*, pages 169–178. ACM, 2009.

[GGG$^+$14] Shafi Goldwasser, S. Dov Gordon, Vipul Goyal, Abhishek Jain, Jonathan Katz, Feng-Hao Liu, Amit Sahai, Elaine Shi, and Hong-Sheng Zhou. Multi-input functional encryption. In Phong Q. Nguyen and Elisabeth Oswald, editors, *Advances in Cryptology - EUROCRYPT 2014 - 33rd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Copenhagen, Denmark, May 11-15, 2014. Proceedings*, volume 8441 of *Lecture Notes in Computer Science*, pages 578–602. Springer, 2014.

[GGH$^+$13a] Sanjam Garg, Craig Gentry, Shai Halevi, Mariana Raykova, Amit Sahai, and Brent Waters. Candidate indistinguishability obfuscation and functional encryption for all circuits. In *54th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2013, 26-29 October, 2013, Berkeley, CA, USA*, pages 40–49, 2013. Full version available from http://eprint.iacr.org/2013/451.

[GGH$^+$13b] Sanjam Garg, Craig Gentry, Shai Halevi, Mariana Raykova, Amit Sahai, and Brent Waters. Candidate indistinguishability obfuscation and functional encryption for all circuits. In *54th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2013, 26-29 October, 2013, Berkeley, CA, USA*, pages 40–49. IEEE Computer Society, 2013.

[GGHZ14] Sanjam Garg, Craig Gentry, Shai Halevi, and Mark Zhandry. Fully secure functional encryption without obfuscation. *IACR Cryptology ePrint Archive*, 2014:666, 2014.

[GGM84] Oded Goldreich, Shafi Goldwasser, and Silvio Micali. How to construct random functions (extended abstract). In *25th Annual Symposium on Foundations of Computer Science, West Palm Beach, Florida, USA, 24-26 October 1984*, pages 464–479, 1984.

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

[GGSW13] Sanjam Garg, Craig Gentry, Amit Sahai, and Brent Waters. Witness encryption and its applications. In *STOC*, pages 467–476. ACM, 2013.

[GHL$^+$14] Craig Gentry, Shai Halevi, Steve Lu, Rafail Ostrovsky, Mariana Raykova, and Daniel Wichs. Garbled ram revisited. In *Advances in Cryptology–EUROCRYPT 2014*, pages 405–422. Springer, 2014.

[GHV10] Craig Gentry, Shai Halevi, and Vinod Vaikuntanathan. i-hop homomorphic encryption and rerandomizable yao circuits. In *Advances in Cryptology–CRYPTO 2010*, pages 155–172. Springer, 2010.

[GKP$^+$13] Shafi Goldwasser, Yael Tauman Kalai, Raluca A. Popa, Vinod Vaikuntanathan, and Nickolai Zeldovich. Reusable garbled circuits and succinct functional encryption. In *STOC*, pages 555–564. ACM, 2013.

[GKS08] Srivatsava Ranjit Ganta, Shiva Prasad Kasiviswanathan, and Adam Smith. Composition attacks and auxiliary information in data privacy. In *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 265–273. ACM, 2008.

[GLO15] Sanjam Garg, Steve Lu, and Rafail Ostrovsky. Black-box garbled ram. In *Foundations of Computer Science (FOCS), 2015 IEEE 56th Annual Symposium on*, pages 210–229. IEEE, 2015.

[GLOS15] Sanjam Garg, Steve Lu, Rafail Ostrovsky, and Alessandra Scafuro. Garbled ram from one-way functions. In *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing*, pages 449–458. ACM, 2015.

[GLSW14] Craig Gentry, Allison B. Lewko, Amit Sahai, and Brent Waters. Indistinguishability obfuscation from the multilinear subgroup elimination assumption. *IACR Cryptology ePrint Archive*, 2014:309, 2014.

[GLW14] Craig Gentry, Allison Lewko, and Brent Waters. Witness encryption from instance independent assumptions. In *International Cryptology Conference*, pages 426–443. Springer, 2014.

[GP15] Sanjam Garg and Omkant Pandey. Incremental program obfuscation. *IACR Cryptology ePrint Archive*, 2015:997, 2015.

[GPSW06] Vipul Goyal, Omkant Pandey, Amit Sahai, and Brent Waters. Attribute-based encryption for fine-grained access control of encrypted data. In *Proceedings of the 13th ACM Conference on Computer and Communications Security, CCS 2006, Alexandria, VA, USA, Ioctober 30 - November 3, 2006*, pages 89–98, 2006.

[GVW15a] Sergey Gorbunov, Vinod Vaikuntanathan, and Hoeteck Wee. Predicate encryption for circuits from lwe. In *Annual Cryptology Conference*, pages 503–523. Springer, 2015.

[GVW15b] Sergey Gorbunov, Vinod Vaikuntanathan, and Daniel Wichs. Leveled fully homomorphic signatures from standard lattices. In *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing*, pages 469–477. ACM, 2015.

[HHY11] Yi-Jun He, Lucas CK Hui, and Siu Ming Yiu. Avoid illegal encrypted drm content sharing with non-transferable re-encryption. In *Communication Technology (ICCT), 2011 IEEE 13th International Conference on*, pages 703–708. IEEE, 2011.

[HJO$^+$15] Brett Hemenway, Zahra Jafargholi, Rafail Ostrovsky, Alessandra Scafuro, and Daniel Wichs. Adaptively secure garbled circuits from one-way functions. *IACR Cryptology ePrint Archive*, 2015:1250, 2015.

[HLP11] Shai Halevi, Yehuda Lindell, and Benny Pinkas. Secure computation on the web: Computing without simultaneous interaction. In *Advances in Cryptology–CRYPTO 2011*, pages 132–150. Springer, 2011.

[HMW07] Nicholas Hopper, David Molnar, and David Wagner. From weak to strong watermarking. In *Theory of Cryptography, 4th Theory of Cryptography Conference, TCC 2007, Amsterdam, The Netherlands, February 21-24, 2007, Proceedings*, pages 362–382, 2007.

[HRN^{+}14] Andrew Ho, Justin Reich, Sergiy Nesterko, Daniel Seaton, Tommy Mullaney, Jim Waldo, and Isaac Chuang. Harvardx and mitx: The first year of open online courses, fall 2012-summer 2013. *Ho, AD, Reich, J., Nesterko, S., Seaton, DT, Mullaney, T., Waldo, J., & Chuang, I.(2014). HarvardX and MITx: The first year of open online courses (HarvardX and MITx Working Paper No. 1)*, 2014.

[HRSV07] Susan Hohenberger, Guy Rothblum, Abhi Shelat, and Vinod Vaikuntanathan. Securely obfuscating re-encryption. *Theory of Cryptography*, pages 233–252, 2007.

[ID03] Anca-Andreea Ivan and Yevgeniy Dodis. Proxy cryptography revisited. In *NDSS*, 2003.

[IK00] Yuval Ishai and Eyal Kushilevitz. Randomizing polynomials: A new representation with applications to round-efficient secure computation. In *Foundations of Computer Science, 2000. Proceedings. 41st Annual Symposium on*, pages 294–304. IEEE, 2000.

[Jak99] Markus Jakobsson. On quorum controlled asymmetric proxy re-encryption. In *International Workshop on Public Key Cryptography*, pages 112–121. Springer, 1999.

[Kei70] Thomas Keightley. *The Fairy Mythology: Illustrative of the Romance and Superstition of Various Countries.* 1870. Retrieved from: http://www.sacred-texts.com/neu/celt/tfm/: 2 November 2015.

[KHP06] Himanshu Khurana, Jin Heo, and Meenal Pant. From proxy encryption primitives to a deployable secure-mailing-list solution. In *International Conference on Information and Communications Security*, pages 260–281. Springer, 2006.

[KL93] Michael Kearns and Ming Li. Learning in the presence of malicious errors. *SIAM Journal on Computing*, 22(4):807–837, 1993.

[KPTZ13] Aggelos Kiayias, Stavros Papadopoulos, Nikos Triandopoulos, and Thomas Zacharias. Delegatable pseudorandom functions and applications. In *2013 ACM SIGSAC Conference on Computer* *and Communications Security, CCS'13, Berlin, Germany, November 4-8, 2013*, pages 669–684, 2013. Full version available from http://eprint.iacr.org/2013/379.

[KSW08] Jonathan Katz, Amit Sahai, and Brent Waters. Predicate encryption supporting disjunctions, polynomial equations, and inner products. In *EUROCRYPT*, volume 4965 of *Lecture Notes in Computer Science*, pages 146–162. Springer, 2008.

[KVH00] M. Kutter, S. Voloshynovskiy, and A. Herrigel. The watermark copy attack. In *Proceedings of the SPIE, Security and Watermarking of Multimedia Contents II*, volume 3971, pages 371–379, 2000.

[KW17] Sam Kim and David J. Wu. Watermarking cryptographic functionalities from standard lattice assumptions. In *Advances in Cryptology - CRYPTO 2017 - 37th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 20-24, 2017, Proceedings, Part I*, pages 503–536, 2017.

[LLV07a] Ninghui Li, Tiancheng Li, and Suresh Venkatasubramanian. t-closeness: Privacy beyond k-anonymity and l-diversity. In *Proceedings of the 23rd International Conference on Data Engineering, ICDE 2007, The Marmara Hotel, Istanbul, Turkey, April 15-20, 2007*, pages 106–115, 2007.

[LLV07b] Ninghui Li, Tiancheng Li, and Suresh Venkatasubramanian. t-closeness: Privacy beyond k-anonymity and l-diversity. In *2007 IEEE 23rd International Conference on Data Engineering*, pages 106–115. IEEE, 2007.

[LO13] Steve Lu and Rafail Ostrovsky. How to garble ram programs? In *Advances in Cryptology–EUROCRYPT 2013*, pages 719–734. Springer, 2013.

[LP09] Huijia Lin and Rafael Pass. Non-malleability amplification. In *STOC*, pages 189–198. ACM, 2009.

[LPK10] Sangho Lee, Heejin Park, and Jong Kim. A secure and mutual-profitable drm interoperability scheme. In *Computers and Communications (ISCC), 2010 IEEE Symposium on*, pages 75–80. IEEE, 2010.

[LPR13] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. A toolkit for ring-lwe cryptography. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 35–54. Springer, 2013.

[LPST16a] Huijia Lin, Rafael Pass, Karn Seth, and Sidharth Telang. Indistinguishability obfuscation with non-trivial efficiency. In *Public-Key Cryptography - PKC 2016 - 19th IACR International Conference on Practice and Theory in Public-Key Cryptography, Taipei, Taiwan, March 6-9, 2016, Proceedings, Part II*, pages 447–462, 2016.

[LPST16b] Huijia Lin, Rafael Pass, Karn Seth, and Sidharth Telang. Output-compressing randomized encodings and applications. In *Theory of Cryptography*, pages 96–124. Springer, 2016.

[LPZ19] Zhen Liu, Yanbin Pan, and Zhenfei Zhang. Cryptanalysis of an ntru-based proxy encryption scheme from asiaccs'15. Cryptology ePrint Archive, Report 2019/083, 2019. `https://eprint.iacr.org/2019/083`.

[LV08] Benoît Libert and Damien Vergnaud. Unidirectional chosen-ciphertext secure proxy re-encryption. In *International Workshop on Public Key Cryptography*, pages 360–379. Springer, 2008.

[McS09] Frank D McSherry. Privacy integrated queries: an extensible platform for privacy-preserving data analysis. In *Proceedings of the 2009 ACM SIGMOD International Conference on Management of data*, pages 19–30. ACM, 2009.

[MGKV06] Ashwin Machanavajjhala, Johannes Gehrke, Daniel Kifer, and Muthuramakrishnan Venkitasubramaniam. l-diversity: Privacy beyond k-anonymity. In *22nd International Conference on Data Engineering (ICDE'06)*, pages 24–24. IEEE, 2006.

[MH14] MITx and HarvardX. HarvardX-MITx Person-Course Academic Year 2013 De-Identified dataset, version 2.0, 2014.

[Mic97] Daniele Micciancio. Oblivious data structures: applications to cryptography. In *Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*, pages 456–464. ACM, 1997.

[MKGV07] Ashwin Machanavajjhala, Daniel Kifer, Johannes Gehrke, and Muthuramakrishnan Venkitasubramaniam. L-diversity: Privacy beyond k-anonymity. *TKDD*, 1(1):3, 2007.

[MPRS] Ilya Mironov, Omkant Pandey, Omer Reingold, and Gil Segev. Incremental deterministic public-key encryption. In *EUROCRYPT 2012*.

[MT07] Frank McSherry and Kunal Talwar. Mechanism design via differential privacy. In *Proceedings of the 48th Annual IEEE Symposium on Foundations of Computer Science*, pages 94–103. IEEE Computer Society, 2007.

[NAL15a] David Nuñez, Isaac Agudo, and Javier Lopez. Ntrureencrypt: An efficient proxy re-encryption scheme based on ntru. In *Proceedings of the 10th ACM Symposium on Information, Computer and Communications Security*, pages 179–189. ACM, 2015.

[NAL15b] David Nunez, Isaac Agudo, and Javier Lopez. A parametric family of attack models for proxy re-encryption. In *Computer Security Foundations Symposium (CSF), 2015 IEEE 28th*, pages 290–301. IEEE, 2015.

[NBW$^+$18] Kobbi Nissim, Aaron Bembenek, Alexandra Wood, Mark Bun, Marco Gaboardi, Urs Gasser, David R. O'Brien, Thomas Steinke, and Salil Vadhan. Bridging the gap between computer science and legal approaches to privacy. *Harvard Journal of Law & Technology*, 31(2):687–780, Spring 2018.

[Nis13] Ryo Nishimaki. How to watermark cryptographic functions. In *Advances in Cryptology - EUROCRYPT 2013, 32nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Athens, Greece, May 26-30, 2013. Proceedings*, pages 111–125, 2013. Full version available from http://eprint.iacr.org/2014/472.

[NS08] Arvind Narayanan and Vitaly Shmatikov. Robust de-anonymization of large sparse datasets. In *Security and Privacy, 2008. SP 2008. IEEE Symposium on*, pages 111–125. IEEE, 2008.

[NS10] Arvind Narayanan and Vitaly Shmatikov. Myths and fallacies of "personally identifiable information". *Commun. ACM*, 53(6):24–26, June 2010.

[NS15] Kobbi Nissim and Uri Stemmer. On the generalization properties of differential privacy. *CoRR*, abs/1504.05800v1, 2015.

[NSM05] Elaine M Newton, Latanya Sweeney, and Bradley Malin. Preserving privacy by de-identifying face images. *IEEE transactions on Knowledge and Data Engineering*, 17(2):232–243, 2005.

[NSS99] David Naccache, Adi Shamir, and Julien P. Stern. How to copyright a function? In *Public Key Cryptography, Second International Workshop on Practice and Theory in Public Key Cryptography, PKC '99, Kamakura, Japan, March 1-3, 1999, Proceedings*, pages 188–196, 1999.

[NSS$^+$18] Kobbi Nissim, Adam D. Smith, Thomas Steinke, Uri Stemmer, and Jonathan Ullman. The limits of post-selection generalization. In Samy Bengio, Hanna M. Wallach, Hugo Larochelle, Kristen Grauman, Nicolò Cesa-Bianchi, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, 3-8 December 2018, Montréal, Canada.*, pages 6402–6411, 2018.

[NW15] Ryo Nishimaki and Daniel Wichs. Watermarking cryptographic programs against arbitrary removal strategies. *IACR Cryptology ePrint Archive*, 2015:344, 2015.

[NW18] Kobbi Nissim and Alexandra Wood. Is privacy *privacy?* *Philosophical Transactions of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*, 376(2128), 2018.

[Off12] UK Information Commissioner's Office. *Anonymisation: managing data protection risk code of practice.* Nov 2012.

[Ohm10] Paul Ohm. Broken promises of privacy: Responding to the surprising failure of anonymization. *UCLA Law Review*, 57:1701–1777, 2010.

[O'N10] Adam O'Neill. Definitional issues in functional encryption. *IACR Cryptology ePrint Archive*, 2010:556, 2010.

[PPC17] Ferdinando Pennarola, Luca Pistilli, and Michael Chau. Angels and daemons: Is more knowledge better than less privacy? an empirical study on a k-anonymized openly available dataset. 2017.

[PRSV17] Yuriy Polyakov, Kurt Rohloff, Gyana Sahu, and Vinod Vaikuntanathan. Fast proxy re-encryption for publish/subscribe systems. *ACM Transactions on Privacy and Security (TOPS)*, 20(4):14, 2017.

[PS17] Chris Peikert and Sina Shiehian. Privately constraining and programming prfs, the lwe way. *IACR Cryptology ePrint Archive*, 2017:1094, 2017.

[PWA$^+$16] L Phong, L Wang, Y Aono, M Nguyen, and X Boyen. Proxy re-encryption schemes with key privacy from lwe. Technical report, Cryptology ePrint Archive, Report 2016/327, 2016. http://eprint. iacr. org/2016/327, 2016.

[Rom90] John Rompel. One-way functions are necessary and sufficient for secure signatures. In *STOC*, pages 387–394. ACM, 1990.

[SBC$^+$07] Elaine Shi, John Bethencourt, TH Hubert Chan, Dawn Song, and Adrian Perrig. Multi-dimensional range query over encrypted data. In *2007 IEEE Symposium on Security and Privacy (SP'07)*, pages 350–364. IEEE, 2007.

[Smi09] Geoffrey Smith. On the foundations of quantitative information flow. In *International Conference on Foundations of Software Science and Computational Structures*, pages 288–302. Springer, 2009.

[SS98] Pierangela Samarati and Latanya Sweeney. Generalizing data to provide anonymity when disclosing information (abstract). In Alberto O. Mendelzon and Jan Paredaens, editors, *Proceedings of the Seventeenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, June 1-3, 1998, Seattle, Washington, USA*, page 188. ACM Press, 1998.

[SW05] Amit Sahai and Brent Waters. Fuzzy identity-based encryption. In *EUROCRYPT'05*, volume 3494 of *Lecture Notes in Computer Science*, pages 457–473. Springer, 2005.

[SW14] Amit Sahai and Brent Waters. How to use indistinguishability obfuscation: deniable encryption, and more. In *Symposium on Theory of Computing, STOC 2014, New York, NY, USA, May 31 - June 03, 2014*, pages 475–484, 2014. Full version available from http://eprint.iacr.org/2013/454.

[Swe98] Latanya Sweeney. Datafly: A system for providing anonymity in medical data. In *Database Security XI*, pages 356–381. Springer, 1998.

[Swe02] Latanya Sweeney. k-anonymity: A model for protecting privacy. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 10(05):557–570, 2002.

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

[Wat15a] Brent Waters. A punctured programming approach to adaptively secure functional encryption. In *Advances in Cryptology - CRYPTO 2015 - 35th Annual Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2015, Proceedings, Part II*, pages 678–697, 2015.

[Wat15b] Brent Waters. A punctured programming approach to adaptively secure functional encryption. In *CRYPTO 2015*, 2015.

[Yao86] Andrew Chi-Chih Yao. How to generate and exchange secrets (extended abstract). In *FOCS'86*, pages 162–167. IEEE, 1986.

[YF11] Maki Yoshida and Toru Fujiwara. Toward digital watermarking for cryptographic data. *IEICE Transactions*, 94-A(1):270–272, 2011.

[Zha16] Mark Zhandry. How to avoid obfuscation using witness prfs. In *Theory of Cryptography - 13th International Conference, TCC 2016-A, Tel Aviv, Israel, January 10-13, 2016, Proceedings, Part II*, pages 421–448, 2016.